The VGSpartans Handbook

VISTA GRANDE HIGH SCHOOL

THE VGSPARTANS HANDBOOK

How the campus web platform works: guides for students, runbooks for admins, and the architecture behind all of it.

VERSION	GENERATED	PAGES	ONLINE
1.0.1-alpha	2026-08-04	159	vgspartans.org/docu

This document is generated from the platform's own repository on every release, so it never describes a version of VGSpartans that was not shipped. The online edition is searchable and always current; this one is for reading offline, and for the day the site itself is the thing that is broken.

The VGSpartans Handbook

CONTENTS

Part 0. Welcome

Part 1. Getting started

The VGSpartans Handbook

Part 2. Platform guides

The VGSpartans Handbook

Part 3. Administration

Part 4. Security

The VGSpartans Handbook

Part 5. Architecture

Part 6. Operations

The VGSpartans Handbook

Part 7. Contributing

Part 8. Reference

The VGSpartans Handbook

PART 0

WELCOME

What this handbook is, how it is organised, and how to read it.

PART 0. WELCOME

THE VGSPARTANS HANDBOOK

Everything about how VGSpartans works, written down in one place, for students, admins and developers alike.

Explanation . For everyone, admins, developers . Checked 2026-07-25 . /documentation

---

IN PLAIN TERMS

This is the manual for VGSpartans. If something on the platform confuses you, breaks, or you have been handed a job on it and nobody explained how, the answer should be in here. It is written so that a person with no coding background can follow it, and it says so plainly when a page is not finished yet.

VGSpartans is the campus web platform for Vista Grande High School. It is a handful of separate sites that share one account: club websites, personal pages, a wiki, a writing space, a discussion board, a newsroom, and the consoles that run all of it. This handbook covers every one of them, plus the machinery underneath.

WHO THIS IS FOR

There are three kinds of reader here, and most pages say at the top which ones they are written for.

- **Everyone.** You have a school account and you want to use the platform. You never need to know what a Worker is.
- **Admins.** You moderate content, manage people, or run a club console. You need to know exactly which button does what, and what happens after you press it.
- **On-call and developers.** You keep it running, or you change it. You need the architecture, the runbooks, and the reasons behind the design.

WHERE TO START

- New to the platform? Start with Part 1, **Getting started**. It has one walkthrough per kind of reader.
- Something is broken right now? Go straight to Part 6, **Operations**, and open the incident index. It is a table of symptoms: find yours, follow the runbook.
- Want to know how a specific platform works? Part 2, **Platform guides**, has a chapter for each one.
- Want to change the code? Part 7, **Contributing**, and Part 5, **Architecture**.

The VGSpartans Handbook

WHAT THIS HANDBOOK PROMISES

Two things, and they are the reason it is worth trusting during a bad day.

1. **Nothing here is guessed.** Every claim about how the platform behaves was checked against the code that implements it, and the pages name the files. Where something could not be verified, the page says so instead of inventing an answer.
2. **A page that is not written says it is not written.** Unfinished pages carry a notice and a pointer to whatever the real source of truth is today. None of them are padded out to look complete.

How to use this handbook explains the page types, the badges, and the conventions in more detail.

WHERE IT LIVES

The handbook is markdown files in the platform's own repository, under `docs/handbook`. The site you are reading renders them, and every page has an "Edit this page on GitHub" link at the bottom that opens the exact file. Fixing a wrong sentence is a pull request, the same as fixing a bug.

That placement is the point. Documentation kept somewhere else drifts from the code within a month. Documentation kept beside the code gets reviewed in the same pull request as the change it describes.

The VGSpartans Handbook

PART 0. WELCOME

HOW TO USE THIS HANDBOOK

The page types, the audience labels, the callouts, and the conventions this handbook follows so you can read it quickly.

Explanation . For everyone, admins, developers . Checked 2026-07-26 . /documentation/how-to-use-this-handbook

IN PLAIN TERMS

Every page in here says what kind of page it is and who it is for, so you can tell in one glance whether it will teach you something, walk you through a job, or get you out of trouble. This page explains those labels and the handful of other conventions the handbook uses.

THE FOUR KINDS OF PAGE

Under every page's title is a line of small type. The first thing on it is what that page is trying to do for you. Mixing these up is the most common way documentation becomes useless, so the handbook keeps them apart.

Kind	What it is for	What it looks like
Tutorial	Learning by doing, when you are new	A single path from start to finish, with nothing optional in it
How-to	Getting one specific job done	A short list of steps that assumes you already know the basics
Runbook	Fixing something that is broken, under pressure	A fixed skeleton: symptom, causes, fix, verify
Explanation	Understanding how or why something works	Prose, diagrams, and the reasoning behind a decision
Reference	Looking up an exact value	Tables and lists, kept boring on purpose

WHO EACH PAGE IS FOR

Next along that same line is who the page assumes you are.

- **Everyone** means the page assumes no role and no technical background.
- **Admins** means it assumes you can open an admin console.
- **Developers** means it assumes you have the code checked out and can run it.

If a page is not written for you, you can still read it, but it will probably refer to a screen you cannot open or a command you cannot run.

The rest of the line is the date the page's claims were last checked against the code, and, on a page that has not been written yet, the word "unfinished".

The VGSpartans Handbook

RUNBOOKS HAVE A FIXED SHAPE

Every runbook uses the same six headings, in the same order, always:

1. **Symptom.** What you are seeing. Match this against your situation first.
2. **Before you touch anything.** What to capture or check so you do not destroy the evidence, and so a wrong guess is reversible.
3. **Likely causes.** Ordered by how often they are actually the cause.
4. **Fix, step by step.** Each step names the exact screen, button, command or file. No step says “check the config” without saying which one.
5. **Verify it worked.** How to prove the fix landed, rather than hoping.
6. **If this did not fix it.** Who to escalate to, and what evidence to bring.

Under pressure you do not want to read prose. You want to know that step 4 is always the fix and step 6 is always the exit.

CALLOUTS

Four kinds of aside show up throughout.

> IN PLAIN TERMS
>
> **In plain terms** opens most pages. Two to four sentences that a non-technical reader can follow, sometimes with an analogy. If you only read one box on a page, read this one.

> WHY IT'S THIS WAY
>
> **Why it’s this way** explains a design decision. These exist because “that is just how it is” is the fastest way to get a system broken by someone trying to tidy it. Longer ones become their own pages under Part 5, Architecture.

> WARNING
>
> **Warning** marks something that can lose data, lock people out, or cost money if you get it wrong.

> NOTE
>
> **Note** is a detail worth knowing that does not fit the flow of the page.

HOW CLAIMS ARE BACKED UP

When a page says the platform does something, it names the file that does it. A repo path in `code formatting` like `docs/REPO-RULES.md` is a real path in the repository, and a check in the build fails if it is not. When a code block quotes the repository, it carries the file’s path as a caption above it.

The VGSpartans Handbook

If a page could not verify something, it says so in the open, marked `TODO(verify)`, rather than filling the gap with a plausible sentence. During an incident, a confidently wrong step costs more than a missing one.

FINDING THINGS

- **Search** is in the header, or press `Ctrl` + K. It reads the full text of every page, and it matches page titles and headings on fragments as short as two letters, so a half-remembered word still finds the page.
- **The sidebar** is the full contents. Each part opens and shuts where it stands, without loading anything; the first entry inside one is its Overview.
- **Filter sidebar**, at the top of that list, narrows it to titles that match what you type. Press / to jump into it. It only looks at titles, which is what makes it instant; search is the one that reads the pages.
- **The table of contents** on the right jumps around inside the page you are on, and the marker beside it tracks how far through you are.
- **The directory** lists every page in the handbook on one screen, in reading order.
- **The PDF** is the entire handbook in one file, linked from **the documentation home**. It is regenerated on every production release, so it is never more than one release behind the site.

IF A PAGE IS WRONG

Every page has an “Edit this page on GitHub” link in its footer, which opens that page’s markdown file. Corrections go through the same pull request process as code, which is written up in **Contributing**.

If you would rather just tell someone, open a support ticket at `/support` and say which page it was.

The VGSpartans Handbook

PART 0. WELCOME

WORDS YOU WILL SEE EVERYWHERE

The handful of VGSpartans terms that turn up on every page, and where the full glossary lives.

Reference . For everyone, admins, developers . Checked 2026-07-25 . /documentation/glossary-pointer

---

IN PLAIN TERMS

This handbook uses some words in a specific way. Here are the dozen that turn up most, in one screen, so you are not stopped on the first paragraph of every page. The full alphabetical list is in the reference part at the back.

The complete list, including every acronym and internal code name, is the **glossary** in Part 8. This page is only the short version: the words you cannot avoid.

THE WORDS

Platform. One of the sites that make up VGSpartans: VGClubs, VGSites, VGWiki, VGWrites, VGSynergy, VGAgora, VGNews. Each has its own address and its own purpose, and they all share one sign-in.

Core. The part of VGSpartans that owns your account. It runs the main site, the sign-in flow, and the admin and developer consoles. Every other platform asks core who you are rather than deciding for itself.

Console. An authenticated screen for managing something, as opposed to a public page you can just read. The admin console, the developer console, and each club's officer consoles are all consoles.

Subdomain. The part before the dot in an address. Each platform and each console lives on its own, so `wiki.` and `admin.` are different places even though they are the same account.

Session. The record that you are signed in on this device right now. Signing out ends it; so does the platform ending it for you, which is a thing it does deliberately when something looks wrong.

MFA. Multi-factor authentication. A second proof that you are you, beyond having the email address. Covered in Part 4.

Moderation. The automatic and human review that content goes through. "Advisory" moderation flags something for a person to look at; it does not delete anything by itself.

Take-down. The deliberate removal of a piece of content by an admin, which really deletes it rather than hiding it. Evidence is captured first.

Audit log. The record of who did what in a console. It is written for every privileged action, and it is not editable from the interface.

The VGSpartans Handbook

Advisor, webmaster, treasurer, secretary. The four club roles that get their own console. An advisor is a staff member; the rest are students the advisor seats.

Migration. A numbered file that changes the shape of a database. They run in order and each one runs exactly once, ever.

Deploy. Publishing the current code to the live site. On this platform a deploy happens automatically when code reaches the production branch.

Preview stack. A complete second copy of the whole platform, on its own domains and its own data, used to check a release before it becomes real. Nothing in it touches production.

Runbook. A page in Part 6 that fixes one specific broken thing, step by step.

WORDS THIS HANDBOOK AVOIDS

Some words get used loosely elsewhere and are kept out of here on purpose, because they mean two different things depending on who is talking:

- “The server”, when what is meant is one specific platform’s worker.
- “The database”, when the platform has several and they hold different things.
- “The admin”, when what is meant is either a school admin or the developer.

Where a page needs to be exact, it names the thing.

The VGSpartans Handbook

PART 0. WELCOME

CHANGELOG

What changed in VGSpartans, release by release, and what it means for the people using it.

Reference . For everyone, admins, developers . Checked 2026-07-26 . /documentation/changelog

---

Nothing has been recorded here yet.

This page is the place where each release will be written down: what changed, who it affects, and anything a reader has to do differently afterwards. Until the first entry lands, the honest answer is that the record starts here rather than that it exists somewhere else.

Until then, the repository's own history is the complete account of every change ever made to the platform.

The VGSpartans Handbook

PART 1

GETTING STARTED

Three first-day walkthroughs: one for students, one for admins, one for developers.

PART 1. GETTING STARTED

GETTING STARTED

Three first-day walkthroughs, one for each kind of person who uses VGSpartans.

Explanation . For everyone, admins, developers . Checked 2026-07-25 . Unfinished . /documentation/getting-started

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover which of the three tracks to pick and what each one covers.

The VGSpartans Handbook

PART 1. GETTING STARTED

FOR ADMINS

Your first day with an admin account: getting in, finding the console, and handling your first case.

Tutorial . For admins . Checked 2026-07-25 . Unfinished . /documentation/getting-started/for-admins

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover an overview of the admin track.

The VGSpartans Handbook

PART 1. GETTING STARTED

YOUR FIRST DAY AS AN ADMIN

Getting into the admin console for the first time, including the browser check that guards it.

Tutorial . For admins . Checked 2026-07-25 . Unfinished . /documentation/getting-started/for-admins/first-day

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

The source of truth today is `platforms/core/src/pages/console-check.astro`.

When this page is written it will cover the attestation gate, the sign-in path for admins, and what to do when the gate refuses you.

The VGSpartans Handbook

PART 1. GETTING STARTED

A TOUR OF THE ADMIN CONSOLE

Every panel in the admin console and what it is for.

Tutorial . For admins . Checked 2026-07-25 . Unfinished . /documentation/getting-started/for-admins/admin-console-tour

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

The source of truth today is `platforms/core/src/components/console/AdminConsole.tsx`.

When this page is written it will cover each tab in the console, written from the component that renders it.

The VGSpartans Handbook

PART 1. GETTING STARTED

YOUR FIRST MODERATION CASE

Working one item from the moderation queue end to end.

Tutorial . For admins . Checked 2026-07-25 . Unfinished . /documentation/getting-started/for-admins/first-moderation-case

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

The source of truth today is `docs/CONTENT-MODERATION.md`.

When this page is written it will cover opening a case, reading the evidence, and choosing an outcome.

The VGSpartans Handbook

PART 1. GETTING STARTED

FOR DEVELOPERS

From a fresh machine to a running local copy of the whole platform.

Tutorial . For developers . Checked 2026-07-25 . Unfinished . /documentation/getting-started/for-developers

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover an overview of the developer track.

The VGSpartans Handbook

PART 1. GETTING STARTED

SETTING UP YOUR MACHINE

One script installs Node, pnpm, the workspace and the browsers the tests need.

Tutorial . For developers . Checked 2026-07-25 . Unfinished . /documentation/getting-started/for-developers/machine-setup

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

The source of truth today is `scripts/setup-env.sh` and `CONTRIBUTING.md`.

When this page is written it will cover what the setup script installs, what it leaves you to fill in, and the per-distro traps.

The VGSpartans Handbook

PART 1. GETTING STARTED

RUNNING IT LOCALLY

What `pnpm dev` starts, on which addresses, and how to boot more of the platform.

Tutorial . For developers . Checked 2026-07-25 . Unfinished . /documentation/getting-started/for-developers/run-it-locally

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

The source of truth today is `scripts/dev-all.mjs`.

When this page is written it will cover the multi-worker dev stack, the local hostnames, and the reset commands.

The VGSpartans Handbook

PART 1. GETTING STARTED

MAKING YOUR FIRST CHANGE

A small change, from branch to pull request, with every check that has to pass.

Tutorial . For developers . Checked 2026-07-25 . Unfinished . /documentation/getting-started/for-developers/first-change

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

The source of truth today is `CONTRIBUTING.md`.

When this page is written it will cover a worked example of the full contribution loop.

The VGSpartans Handbook

PART 1. GETTING STARTED

A TOUR OF THE CODEBASE

Where everything lives in the workspace, and why it is split the way it is.

Tutorial . For developers . Checked 2026-07-25 . Unfinished . /documentation/getting-started/for-developers/codebase-tour

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

The source of truth today is `docs/MULTI-WORKER-SPLIT.md`.

When this page is written it will cover a guided walk through platforms/, packages/, schools/ and scripts/.

The VGSpartans Handbook

PART 1. GETTING STARTED

FOR STUDENTS

Your first hour on VGSpartans: signing in, finding your way around, and knowing who to ask.

Tutorial . For everyone . Checked 2026-07-25 . Unfinished . /documentation/getting-started/for-students

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover an overview of the student track.

The VGSpartans Handbook

PART 1. GETTING STARTED

YOUR FIRST SIGN-IN

How to sign in to VGSpartans for the first time with your school email.

Tutorial . For everyone . Checked 2026-07-25 . Unfinished . /documentation/getting-started/for-students/first-sign-in

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

The source of truth today is `platforms/core/src/pages/sign-in.astro`.

When this page is written it will cover the sign-in screen, the emailed code, and what happens the first time.

The VGSpartans Handbook

PART 1. GETTING STARTED

A TOUR OF THE HUB

What is on the hub board after you sign in, and what each card takes you to.

Tutorial . For everyone . Checked 2026-07-25 . Unfinished . /documentation/getting-started/for-students/tour-of-the-hub

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

The source of truth today is `platforms/core/src/pages/hub.astro`.

When this page is written it will cover every card on the hub and where it leads.

The VGSpartans Handbook

PART 1. GETTING STARTED

SET UP YOUR ACCOUNT

Your profile, your display name, your avatar, and the security settings worth turning on.

Tutorial . For everyone . Checked 2026-07-25 . Unfinished . /documentation/getting-started/for-students/set-up-your-account

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

The source of truth today is `platforms/core/src/pages/settings/`.

When this page is written it will cover the settings screens and what each control changes.

The VGSpartans Handbook

PART 1. GETTING STARTED

GETTING HELP

Where to ask when something is confusing, broken, or wrong.

Tutorial . For everyone . Checked 2026-07-25 . Unfinished . /documentation/getting-started/for-students/getting-help

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

The source of truth today is `platforms/core/src/pages/support.astro`.

When this page is written it will cover the support ticket flow, the report flow, and what happens after you send one.

The VGSpartans Handbook

PART 2

PLATFORM GUIDES

One chapter per platform: what it is, how to use it, who can do what, and what breaks.

PART 2. PLATFORM GUIDES

PLATFORM GUIDES

One chapter for each platform that makes up VGSpartans.

Explanation . For everyone, admins, developers . Checked 2026-07-25 . Unfinished . /documentation/platforms

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover what each platform is for and how the chapters are laid out.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

CORE

The apex site, the account system, and the admin and developer consoles.

Explanation . For everyone, admins, developers . Checked 2026-07-25 . Unfinished . /documentation/platforms/core

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover what it is, who it is for, and why it exists.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

USING CORE

What you can do in Core and how to do it.

How-to . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/core/using-it

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the day-to-day tasks, written from the screens that exist.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

ROLES IN CORE

Who can do what in Core, and how those permissions are decided.

Reference . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/core/roles

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover every role and the exact capability each one carries.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

CORE UNDER THE HOOD

How Core is built: its worker, its data, and its moving parts.

Explanation . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/core/under-the-hood

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the worker, the database, the storage, and the request path.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

TROUBLESHOOTING CORE

Things that go wrong in Core, what causes them, and how to fix them.

Runbook . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/core/troubleshooting

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover a symptom table and the fix for each row.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

VGADVISOR

The console a member of staff opens for whatever the school has given them, assembled from modules the platforms contribute.

Explanation . For everyone, admins, developers . Checked 2026-07-31 . /documentation/platforms/vgadvisor

IN PLAIN TERMS

VGAdvisor is the one console a teacher opens. What is inside it depends entirely on what they have been given: someone who advises a club finds that club's governance, someone who runs the school paper finds the newsroom, and someone who has both finds both, in one rail.

VGAdvisor lives at `advisor.<appDomain>` and is served by `vgcore`.

On its own it is nothing. It has a rail, a header and no rows. Every section in it is contributed by a platform, and a person sees a platform's sections only if they hold that platform's grant. Hold none and the console does not open at all.

WHAT A MODULE IS

A module is data, not an interface: an id, a heading for the rail, a predicate over the signed-in person's resolved identity, and the rows it contributes. They live in `packages/services/src/advisor/modules.ts`.

Module	Held by	What it carries
Clubs	Anyone advising at least one club	Members, content review, reported content, club information
Newsroom	The paper's adviser, and any editor and above	Review queue, budget, tips, desks, shows, staff, outlet settings

Support sits outside every module. It is how a member of staff reaches a person about any of this, so it is there whichever module let them in.

WHY THE DOOR AND THE RAIL COME FROM ONE LIST

`panelOk(identity, "advisor")` is derived from the very same registry the rail is built from. That is what makes two failures impossible rather than merely unlikely: a console that opens onto nothing, and a grant that opens no door.

It matters because the second one is a real trap. A teacher running the school paper is exactly the person a club-shaped console would have locked out, and the only surface that could have seated them would have been behind the door the seat was supposed to unlock.

The VGSpartans Handbook

WHICH CLUB, AND WHICH PAPER

The Clubs module is scoped by a club switcher in the header. An advisor with more than one club picks between them, and every read and write carries that choice, so nothing is ever silently done to the wrong club.

The Newsroom module needs no switcher. There is one school paper.

WHAT IS NOT HERE

The club website's page builder is not in VGAdvisor. It is on the club site console at `webmaster.<appDomain>`, and an advisor reaches it there with the same access a webmaster has, on the club they advise. The Club site row in the rail is a door to it rather than a second copy of it.

The writing of a story is not here either. That is the newsroom desk at `news.<appDomain>/newsroom`, where a reporter works on their own pieces. VGAdvisor carries the decisions about everyone's pieces. See **Roles in VGNews** for where the line falls.

WHERE THINGS LIVE

Piece	Path
The module registry	`packages/services/src/advisor/modules.ts`
The console	`platforms/core/src/components/console/AdvisorConsole.tsx`
The newsroom panels	`platforms/core/src/components/console/advisor/`
The page	`platforms/core/src/pages/advisor/index.astro`
The endpoint	`platforms/core/src/pages/api/v1/advisor.ts`
Club governance	`packages/services/src/clubs/governance.ts`
Newsroom write policy	`packages/services/src/newsroom/actions.ts`

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

VGSITES

Personal pages for students and staff, built and published from a dashboard.

Explanation . For everyone, admins, developers . Checked 2026-07-25 . Unfinished . /documentation/platforms/vgsites

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover what it is, who it is for, and why it exists.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

USING VGSITES

What you can do in VGSites and how to do it.

How-to . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vgsites/using-it

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the day-to-day tasks, written from the screens that exist.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

ROLES IN VGSITES

Who can do what in VGSites, and how those permissions are decided.

Reference . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vgsites/roles

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover every role and the exact capability each one carries.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

VGSITES UNDER THE HOOD

How VGSites is built: its worker, its data, and its moving parts.

Explanation . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vgsites/under-the-hood

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the worker, the database, the storage, and the request path.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

TROUBLESHOOTING VGSITES

Things that go wrong in VGSites, what causes them, and how to fix them.

Runbook . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vgsites/troubleshooting

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover a symptom table and the fix for each row.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

VGCLUBS

A real website for every club, run by elected student webmasters.

Explanation . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vgclubs

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover what it is, who it is for, and why it exists.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

USING VGCLUBS

What you can do in VGClubs and how to do it.

How-to . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vgclubs/using-it

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the day-to-day tasks, written from the screens that exist.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

ROLES IN VGCLUBS

Who can do what in VGClubs, and how those permissions are decided.

Reference . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vgclubs/roles

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover every role and the exact capability each one carries.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

VGCLUBS UNDER THE HOOD

How VGClubs is built: its worker, its data, and its moving parts.

Explanation . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vgclubs/under-the-hood

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the worker, the database, the storage, and the request path.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

TROUBLESHOOTING VGCLUBS

Things that go wrong in VGClubs, what causes them, and how to fix them.

Runbook . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vgclubs/troubleshooting

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover a symptom table and the fix for each row.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

ADVISING A CLUB

What a club advisor can see and do, where each of those controls lives, and how they seat the student officers.

How-to . For admins . Checked 2026-07-31 . /documentation/platforms/vgclubs/advisor-console

IN PLAIN TERMS

A club advisor is the teacher accountable for a club. They decide who holds which job in it, whether the club's website is public, and whether member content needs their approval first. Those controls are in the advisor console; the club's actual website is edited on the club site console.

WHERE THE CONTROLS ARE

What you want to do	Where
Seat or re-role an officer	`advisor.<appDomain>`
Approve or take down member content	`advisor.<appDomain>`
Triage a report about the club's site	`advisor.<appDomain>`
Set the club's name, meeting time, contact	`advisor.<appDomain>`
Launch or pause the club website	`advisor.<appDomain>`
Build the pages, upload files, set colours	`webmaster.<appDomain>`

The advisor console is **VGAdvisor**, and the Clubs rows are one module in it. If you advise more than one club, the header carries a switcher and every action applies to the club named in it.

You reach the page builder with the same access a webmaster has, on the clubs you advise. The Club site row in the rail takes you there.

SEATING THE OFFICERS

Add a member by school email; they need to have signed in once before. Each role is capped per club, and the caps are set platform-wide by an admin rather than per club. A **custom role** lets you assemble a permission set by hand when none of the presets fit.

Seating, re-roling and removing all grant or revoke a console, so all three need a step-up code and all three are written to the audit trail.

CONTENT APPROVAL

Content approval required holds new member content for you to read before it can go live. Turn it off and members publish directly. Either way, the platform-wide safety checks still run: your approval and

The VGSpartans Handbook

the safety scan are separate gates, and content needs both.

LAUNCHING THE SITE

A club’s website returns a 404 to the public until you launch it. Members can build the whole thing first. Pausing puts it back behind the 404 and deletes nothing.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

THE WEBMASTER CONSOLE

The club site builder: pages, events, news, officers and photos.

How-to . For everyone . Checked 2026-07-25 . Unfinished . /documentation/platforms/vgclubs/webmaster-console

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the builder, written from the screens that exist.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

THE TREASURER CONSOLE

The club's money screens and what a treasurer is allowed to record.

How-to . For everyone . Checked 2026-07-25 . Unfinished . /documentation/platforms/vgclubs/treasurer-console

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover every control on the treasurer console.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

THE SECRETARY CONSOLE

The club's minutes and membership screens.

How-to . For everyone . Checked 2026-07-25 . Unfinished . /documentation/platforms/vgclubs/secretary-console

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover every control on the secretary console.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

VGWIKI

The campus encyclopedia, written and kept current by Spartans.

Explanation . For everyone, admins, developers . Checked 2026-07-25 . Unfinished . /documentation/platforms/vgwiki

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover what it is, who it is for, and why it exists.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

USING VGWIKI

What you can do in VGWiki and how to do it.

How-to . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vgwiki/using-it

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the day-to-day tasks, written from the screens that exist.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

ROLES IN VGWIKI

Who can do what in VGWiki, and how those permissions are decided.

Reference . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vgwiki/roles

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover every role and the exact capability each one carries.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

VGWIKI UNDER THE HOOD

How VGWiki is built: its worker, its data, and its moving parts.

Explanation . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vgwiki/under-the-hood

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the worker, the database, the storage, and the request path.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

TROUBLESHOOTING VGWIKI

Things that go wrong in VGWiki, what causes them, and how to fix them.

Runbook . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vgwiki/troubleshooting

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover a symptom table and the fix for each row.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

THE VGWIKI EDITOR

Writing and editing an article, and what happens to your text when you save.

How-to . For everyone . Checked 2026-07-25 . Unfinished . /documentation/platforms/vgwiki/editor

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the editor, the stored format, and the version history.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

SEARCHING VGWIKI

How VGWiki search works and what it can and cannot find.

Explanation . For everyone . Checked 2026-07-25 . Unfinished . /documentation/platforms/vgwiki/search

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the full-text index and its limits.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

VGWRITES

A private writing space for students, with a shared word editor.

Explanation . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vgwrites

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover what it is, who it is for, and why it exists.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

USING VGWRITES

What you can do in VGWrites and how to do it.

How-to . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vgwrites/using-it

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the day-to-day tasks, written from the screens that exist.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

ROLES IN VGWRITES

Who can do what in VGWrites, and how those permissions are decided.

Reference . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vgwrites/roles

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover every role and the exact capability each one carries.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

VGWRITES UNDER THE HOOD

How VGWrites is built: its worker, its data, and its moving parts.

Explanation . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vgwrites/under-the-hood

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the worker, the database, the storage, and the request path.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

TROUBLESHOOTING VGWRITES

Things that go wrong in VGWrites, what causes them, and how to fix them.

Runbook . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vgwrites/troubleshooting

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover a symptom table and the fix for each row.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

VGSYNERGY

Private circles for collaboration, gated by member approval.

Explanation . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vgsynergy

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover what it is, who it is for, and why it exists.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

USING VGSYNERGY

What you can do in VGSynergy and how to do it.

How-to . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vgsynergy/using-it

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the day-to-day tasks, written from the screens that exist.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

ROLES IN VGSYNERGY

Who can do what in VGSynergy, and how those permissions are decided.

Reference . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vgsynergy/roles

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover every role and the exact capability each one carries.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

VGSYNERGY UNDER THE HOOD

How VGSynergy is built: its worker, its data, and its moving parts.

Explanation . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vgsynergy/under-the-hood

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the worker, the database, the storage, and the request path.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

TROUBLESHOOTING VGSYNERGY

Things that go wrong in VGSynergy, what causes them, and how to fix them.

Runbook . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vgsynergy/troubleshooting

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover a symptom table and the fix for each row.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

VGAGORA

The campus discussion board, organised into fixed categories.

Explanation . For everyone, admins, developers . Checked 2026-07-25 . Unfinished . /documentation/platforms/vgagora

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover what it is, who it is for, and why it exists.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

USING VGAGORA

What you can do in VGAgora and how to do it.

How-to . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vgagora/using-it

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the day-to-day tasks, written from the screens that exist.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

ROLES IN VGAGORA

Who can do what in VGAgora, and how those permissions are decided.

Reference . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vgagora/roles

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover every role and the exact capability each one carries.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

VGAGORA UNDER THE HOOD

How VGAgora is built: its worker, its data, and its moving parts.

Explanation . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vgagora/under-the-hood

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the worker, the database, the storage, and the request path.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

TROUBLESHOOTING VGAGORA

Things that go wrong in VGAgora, what causes them, and how to fix them.

Runbook . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vgagora/troubleshooting

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover a symptom table and the fix for each row.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

VGNEWS

The student newsroom on its own subdomain, with bylines, section desks, and an editor's approval in front of everything that publishes.

Explanation . For everyone, admins, developers . Checked 2026-07-31 . /documentation/platforms/vgnews

---

> **IN PLAIN TERMS**
>
> VGNews is the school newspaper. Reporters write stories, an editor reads them and decides whether they run, and only then do they appear. It has named bylines and sections the way a real paper does, which is exactly what makes it different from a club posting an update on its own site. It also carries video segments, photo essays and briefs, because the class that runs it does more than write.

VGNews lives at `news.<appDomain>`. It is a curated masthead the whole school reads, with articles across section desks (News, Sports, Opinion, Features, Multimedia), named bylines, and photo and video journalism.

WHAT IT IS, AND WHAT IT IS NOT

VGNews is a school-of-record newsroom: advisor-managed, read by the whole campus. It is deliberately not a club site. The media-arts club already has its own site in VGClubs, with text, a blog, galleries and the video pipeline.

VGNews earns its own subplatform because its governance and its data model are genuinely different:

- **Bylines**, not “the club posted”. A named reporter wrote this.
- **Section desks**, and a masthead front page aggregating many authors.
- **An editorial workflow**, where a reporter drafts, an editor approves, and only then does it publish. That is separate from, and stacked on top of, the platform-wide safety checks.

> **WARNING**
>
> If a request would be satisfied by “a club posting content”, it belongs in VGClubs, not here. Do not widen VGNews toward a general content pond. The editorial gate is the whole point, and it stops meaning anything the moment anyone can post.

Anyone in the school can send the newsroom a tip, and that is not an exception to the paragraph above. A tip is its own thing in its own table with no path to an article: it is read by the desk and by nobody else, it never appears on the site, and the only thing that can come of it is an editor opening an assignment. Readers suggest; staff publish.

The VGSpartans Handbook

THE FIVE FORMATS

A media-arts and broadcast-journalism class does not only write prose, so a piece declares what shape it is. The choice drives what the console offers and how the page is laid out, and nothing else: every format walks the same editorial ladder and takes the same safety checks.

Format	What it is
Story	Words first, with a lead photo and pictures through the text
Video segment	A clip at the top, with the writeup under it
Photo essay	A set of photographs in order, each with its own caption
Single photo	One photograph and a line about it
Brief	A few sentences. No photo, no video

Video segments can belong to a **show**, a named series with its own page. The broadcast wall at `/watch` gathers every cleared segment, grouped by show.

THE BUDGET

The paper is run off an assignment board, the same way a real newsroom runs off a budget meeting and a class period runs off a plan. An editor opens an assignment; a reporter takes it and starts the draft from it; submitting the piece marks the assignment filed on its own, because the draft and the assignment are joined.

Lateness is worked out from the due date whenever the board is read, so it is never a stale flag waiting on something to run.

WHO SEES WHAT

Every article carries an audience setting. `school` is the default and means signed-in readers only. `public` means world-readable and indexable.

A signed-out visitor, which includes every search engine crawler, is served only public pieces. A signed-out reader who follows a direct link to a real school story is bounced to sign-in rather than told it does not exist, so a genuine missing page stays distinguishable from a sign-in bounce.

STATUS

VGNews is built, typechecks clean, and its unit suite passes. It has not yet been provisioned or deployed to production; the deploy gate is held for an explicit go. The steps are at the end of **VGNews under the hood**.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

USING VGNEWS

Filing a story in one of the five formats, running the budget, and what happens to a reader's tip.

How-to . For everyone, admins, developers . Checked 2026-08-02 . /documentation/platforms/vgnews/using-it

IN PLAIN TERMS

VGNews has two sides. The paper itself is open to the school, and anyone can read it. The newsroom console is where the paper is made, and it opens only for the people an adviser has put on staff. This page walks the console section by section, plus the one thing anyone in the school can do: send the newsroom a tip.

THE TWO SIDES

Side	Where	Who gets it
The paper	`news.<appDomain>`	Everyone. Public stories are open to anyone at all
The newsroom	`news.<appDomain>/newsroom`	The people an adviser has seated on staff

The console is not something you request or find by guessing the address. A **Newsroom console** card appears on your **hub** the moment an adviser seats you, and it disappears again if they take the seat away. Without a seat the address itself is closed, so a stale link goes back to the hub with a note rather than to a console.

A teacher who advises the paper also reaches the same newsroom from **VGAdvisor**, beside whatever else the school has given them.

THE RAIL

What you see on the left depends on your seat, and every section listed there is one you can actually use. See **Roles in VGNews**.

The VGSpartans Handbook

Section	What it is
The desk	What the paper has in hand, what is about to run, and what is holding it
Stories	Your writing surface: the story list and the editor
Story budget	Who is writing what, as a board and as a list
Copy desk	Stories waiting on an editor
Photo desk	Every photograph the paper is carrying, and what each one still needs
Front page	Which story leads, and who can see what has run
Story tips	What readers have sent in
Sections and beats	The desks a piece can be filed to, and the shows a segment can air on
Staff	The masthead
Corrections	What the paper got wrong, and the record of saying so
Archive	Everything that has run
Settings	The masthead's name, the adviser, and the publishing rules

The box at the top of the console jumps between the sections on this rail. The surfaces that hold rows — the story desk, the copy desk and the archive — carry their own search for the stories in them. **View the site** opens the paper itself.

THE DESK

The desk is what opens first, and it is a summary of things you can go and check rather than a set of figures on their own. Four tiles say what is being written, what is waiting on an editor, how many photographs still need describing, and what is live.

The publish gate is the piece closest to running, with the checks it has to clear: filed to a desk, bylined, its photographs described, its clip checked, and — when the newsroom asks for review — an editor's sign-off. The same rules are applied by the server when somebody presses Publish, so a gate that reads clear and a publish that refuses cannot mean different things.

On the wire is every move on every story: who filed what, who sent something back, and the note they sent it back with.

FILING A STORY

Under **Stories**, **New story** asks first what shape the piece is. That choice decides what the rest of the screen offers you, and it can be changed later.

The VGSpartans Handbook

Format	What it is
Story	Words first, with a lead photo and pictures through the text
Video segment	A clip at the top, with your writeup under it
Photo essay	A set of photographs in order, each with its own caption
Single photo	One photograph and a line about it
Brief	A few sentences. No photo, no video

Write the body in the editor, the same writing surface the rest of the platform uses. Photos you drop into the text are uploaded as you place them.

Describe your lead photo. The description field next to it is what a reader using a screen reader hears in place of the picture, and it is read by the safety check like any other words on the page. A photograph with no description is a photograph some of your readers do not get.

Save first, then add media. A clip or a photo has to be attached to something, so the upload panel appears once the draft exists.

Read it through swaps the editor for the story as a reader will see it, with the headline and standfirst above it. It is the same renderer the article page uses, so what you see is what runs.

When it is ready, **Submit** puts it in the editor's queue. If it came off the budget board, the board marks itself filed at the same moment. If it came off an assignment, what the desk asked for sits above the editor while you write.

Throw away removes a draft nobody has seen. It is only offered on a draft, for exactly that reason: once an editor has it, taking it back is the copy desk's job. If the draft came off the budget board, the assignment goes back on it as open.

THE STORY BUDGET

Story budget opens on a board of five columns — pitched, reporting, drafting, copy desk, ready — and every card on it is worked out from the stories and assignments themselves. Nothing has to be dragged from one column to another to keep it true. **List** is the same work as a table, where an editor can change an assignment's state or take it off the board.

Above the board is your own queue: what you have been handed, and what is still unclaimed. **Start writing** opens the draft for you, already titled and filed to the right desk, and marks the assignment as yours. If two people press it on the same story, the first one gets the assignment; the second is told so, and keeps their draft.

Opening an assignment takes a working title, what you want covered, a desk, and optionally a reporter and a due date. **Edit** on any row changes the same five things afterwards, so a story can move to a different reporter or a different week without being taken off the board and opened again. Late is worked out from the due date each time the board loads, so it is right whenever you look.

The VGSpartans Handbook

THE COPY DESK

Copy desk is what waits on an editor. **Approve** sends a story to the safety check and then to the site. **Send back** returns it to the reporter with a note, and the note is the point of the thing, so it is written in the console rather than in a one-line pop-up.

The Safety column is separate from the status. A story reading Published whose scan has not finished is not on the site yet.

THE PHOTO DESK

Photo desk is every photograph the paper is carrying, lead images and photo-essay frames together, with the ones that still need something first.

What it asks for is a **description**: the sentence a reader who cannot see the picture is left with. It is a different sentence from the headline almost every time — the headline describes the story, and a description says what is actually in the frame. Write it here and it saves when you click away.

THE FRONT PAGE

Front page is which story leads. Only a story the safety check has cleared can take the lead, because the lead is the one story a reader cannot miss. The same screen flips a piece between school-only and public, and lists what is still coming.

CORRECTIONS

Corrections is the record of what the paper got wrong.

Pick the story, say whether it is a **correction** (we printed something untrue) or a **clarification** (we printed something true that could be read the wrong way), and write what was wrong and what is right. Filing it does not publish it: **Put it on the story** does, and from that moment it is on the public page, dated, under the article.

While it is still a draft you can **edit** it, because a correction with a typo in it helps nobody. Once it has been on the story it cannot be edited or deleted. If it was filed in error it can be **withdrawn**, which takes it off the page and leaves the record that it happened. That asymmetry is the whole point of keeping one.

THE ARCHIVE AND THE MASTHEAD

Archive is everything that has run, searchable, with what each piece carried and who could see it. It is a read: the controls that change a published story live on the copy desk and the front page. Anything the safety check has cleared carries an **on the site** link straight to the page a reader gets.

Sections and beats are the buckets a piece is filed into — a desk is a section of the paper, a show is a series that video segments belong to. Both are managed on the same screen. **Edit** changes the name and the line readers see under the heading on its own page; the arrows change the order they run in on

The VGSpartans Handbook

the site's navigation. Hiding either takes it off that navigation and leaves everything already filed to it alone, which is also why there is no way to delete one. The address is set from the name when it is created and does not move, so a link that has been shared keeps working.

Staff seats reporters and editors by school email. They need to have signed in once before.

SETTINGS

Settings holds the masthead's name and tagline, the adviser's email, and the paper's publishing rules. Both rules are applied when somebody presses Publish, not just drawn here:

- **Editors review stories before they publish.** On by default.
- **Hold a story until every photograph on it has a description.** Off until you turn it on, so switching it on does not pull anything already up. With it on, the publish button refuses a story whose pictures are undescribed and says how many.

> ⚠ WARNING
>
> The adviser email is a key, not a label. Whoever it belongs to gets the whole newsroom and can do everything in it, including changing that field. Clearing it takes the newsroom away from them.

SENDING THE NEWSROOM A TIP

Any signed-in Spartan can use **Pitch a story** on the news site. It is a title and a few sentences.

A tip is only ever a tip. It is read by the desk and nobody else, it never appears on the site, and the only thing that can come of it is an editor choosing to open an assignment. **Turn into an assignment** opens the same form the budget uses, prefilled with what the reader wrote, so the desk decides the working title, the desk it is filed to, who writes it and by when before it reaches the board. You can have a handful waiting at once; past that, the desk gets a chance to read them before you send more.

WHAT THE SCHOOL SEES

Every piece carries an audience. **School only** is the default and means signed-in readers. **Public** means anyone, including search engines.

A signed-out visitor is served only public pieces. Following a direct link to a school-only story sends them to sign in rather than telling them it does not exist.

READING IT

The front page is the masthead. **Watch** is the broadcast wall, where cleared video segments are grouped by show, and each show has its own page.

A story that has been corrected carries its corrections under the article, dated, for as long as the story is up. They are put there deliberately, by a person, and they stay: the reader a correction is written for is the one who saw the wrong version and came back.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

ROLES IN VGNEWS

Reporter, editor, editor in chief and adviser, what each one can do, and how a story moves between them.

Reference . For everyone, admins, developers . Checked 2026-07-31 . /documentation/platforms/vgnews/roles

IN PLAIN TERMS

Four kinds of person work in the newsroom. A reporter writes and submits. An editor decides whether it runs. An editor in chief also decides who is on the staff and what the sections are. The adviser is the teacher, who can do all of it and is the person ultimately accountable for what the paper prints.

THE ROLES

Newsroom roles extend the platform-wide permission system in `packages/services/src/permissions.ts`. They are not a parallel scheme with their own rules.

The area is `news`, and the verbs are `news.write`, `news.edit`, `news.publish`, `news.sections` and `news.seats`. Three presets bundle them:

Role	Can do
Reporter	Create and edit their own drafts, and submit one for review
Editor	Everything a reporter can, plus kick a story back or approve it
Editor in chief	Everything an editor can, plus manage the sections and the staff seats

The **adviser** is not seated. It is derived: the signed-in email matching the adviser email in the newsroom's settings, the same way a club advisor works. The adviser holds every news permission and is the accountable backstop.

`hasNewsPerm(identity, perm)` is the one check. The API route is the real authorization gate, combining an action-to-permission map with the platform's mutation guard, never the client.

WHAT EACH ROLE SEES

A seat opens the newsroom console. What is on its rail once you are inside is your seat's own permissions, because a section whose every button refuses you is worse than no section.

The VGSpartans Handbook

Section	Opens for
The desk, Story budget, Archive	Any seat
Stories	`news.write`
Copy desk, Photo desk, Story tips, Corrections	`news.edit`
Front page	`news.publish`
Sections and beats	`news.sections`
Staff, Settings	`news.seats`

So a reporter gets the desk, their own stories, the budget they are working from and the archive. An editor also gets everything that is a decision about somebody else's work. The editor in chief and the adviser also get Staff and Settings, because `news.sections` and `news.seats` only exist on the preset that already carries `news.edit`.

The rail is presentation. Every action is re-checked against the real identity on the server, so the sections a rail happens to draw grant nothing.

A teacher who advises the paper reaches the same newsroom from the advisor console instead, beside everything else the school has given them; see **VGAdvisor**. Both read one payload and post to one policy in `packages/services/src/newsroom/actions.ts`, so an approval means the same thing whichever console asked for it.

Writing a piece is the one thing that stays in the newsroom console's own code, for a reason that is not layout: those actions ask a second question, *is this piece theirs*, and that check needs the resolved identity of the person asking. Everything shared asks only *may this person do this at all*.

HOW A STORY MOVES

Every article has an editorial status: Draft, Pending, Published or Removed.

Move	Who	What happens
Draft to Pending	The reporter, by submitting	The story enters the editor's queue
Pending to Draft	An editor, by kicking it back	A reason is required and is logged
Pending to Published	An editor, by approving	The story runs, once its safety checks clear
Published to Pending	Anyone editing a published piece	Editing pulls it back for re-approval

Every status move appends a row to `articleStatusLog`: who moved it, from what, to what, and why. That log is not editable from the interface.

The VGSpartans Handbook

> **NOTE**
>
> Editing a published piece pulls it back to Pending on purpose. A story that has changed since an editor read it has not been approved in its current form, and treating an edit as a minor amendment is how an unreviewed paragraph ends up on the front page.

PUBLISHING IS TWO GATES, NOT ONE

An editor approving a story is necessary but not sufficient. Every article also carries a separate safety state set by the platform's moderation pipeline, and a reader sees the piece only when both are good.

That means an editor pressing Publish can never reveal a piece the safety scan is still holding, and a clean scan can never surface a piece no editor approved.

The mechanics are in **VGNews under the hood**.

REVIEW-REQUIRED

Whether an approval can publish directly, or has to pass through review, is a platform setting rather than something hardcoded. When review is on, the API route refuses a direct publish.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

VGNEWS UNDER THE HOOD

The two-axis state model, the scan-first ordering that stops an unscanned story publishing, and the one function that decides who can read what.

Explanation . For developers . Checked 2026-07-31 . /documentation/platforms/vgnews/under-the-hood

---

IN PLAIN TERMS

Two separate switches have to be on before anyone can read a story: an editor approved it, and the safety checks cleared it. This page is about keeping those two switches genuinely separate, and about the ordering problem underneath them, where a video that passed its check last week must not be allowed to smuggle out an article whose words were never checked at all.

THE SHAPE

One worker, one database (`VGNEWS_DB`), one bucket (`STORAGE_NEWS`) and one namespace (`NEWS_KV`). The established per-subplatform shape, mirroring `platforms/vgagora`.

Two workers read this newsroom, though. The newsroom console runs in vgnews; the newsroom module inside VGAdvisor runs in vgcore, which binds `VGNEWS_DB` already. Cloudflare runs exactly one worker per request and routes are dispatched by host, so a console spanning two platforms' databases has to live in the worker that binds both, and a console cannot be served half from each.

That is why the editorial state machine, the read model and the write policy sit in `packages/services/src/newsroom/` rather than inside either console, and why the PANELS sit in `packages/ui/src/newsroom/NewsroomPanels.tsx` — the same reason `ContentReportsPanel` and `SupportPanel` live there. Two copies of an editorial state machine is how an approve in one console and a publish in the other drift apart; two copies of a review queue is how they start looking like two different products.

The derived views the desk draws — its tiles, the publish gate, the board's columns, the archive — are pure functions in `desk.ts` over that one payload, so the summary and the table under it cannot disagree. Nothing there invents a figure the database does not hold: there is no view counter in VGNews and so no audience tile.

The split within that package is deliberate and load-bearing:

- `actions.ts` holds every action that asks only *may this person do this*. Both routes dispatch through it.
- Anything that also asks *is this piece theirs* — creating, editing and submitting a draft, and every photo-essay plate edit — stays in the vgnews route, where the resolved identity is the point. An ownership check that drifts into a shared service stops being obvious to a reviewer.

The permission table and the step-up tiers live in `actions.ts` too, and both routes read them. An action missing from the permission table is refused rather than waved through, on both sides.

The VGSpartans Handbook

- **No identity database, no auth secret, no video-host secret here.** VGNews binds the `CORE` service binding and reaches identity, session, step-up, the moderation triggers and video signing over it. `vgcore` is the single auth authority, so this console cannot drift from it.
- `vgcore` **owns all migrations**, including `platforms/core/migrations-vgnews/`. VGNews binds the database and reads and writes, but never migrates it.
- **The gateway publishes vgnews before itself** in CI, because the gateway's service binding targets the vgnews worker.

THE TWO-AXIS STATE MODEL

Every row in `articles` carries two independent state axes. Both have to be good for a reader to see the piece.

Axis	Values	Who moves it
`status`	`Draft`, `Pending`, `Published`, `Removed`	An editor
`modStatus`	`pending`, `visible`, `held`	The moderation pipeline. Defaults to `pending`

> **DANGER**
>
> Keep these two axes separate, and never let one imply the other. Collapsing them is the single most dangerous change anyone could make to VGNews: the moment an editorial state implies a safety state, an editor pressing Publish reveals a piece that a scan is still holding.

FORMATS, SHOWS, THE BUDGET AND TIPS

`articles.format` is one of five values and is **presentation only**. Every format walks the same editorial ladder and takes the same scan, so nothing in this page's safety reasoning branches on it. `showId` and `episodeNo` are only meaningful on a segment, and the service clears them whenever a piece is re-formatted away from one, so stale chrome cannot linger on a story that used to be a segment.

A photo essay is the ordered `plate` set on its own article. Plates are rows in `news_media` with a `role` and a `sortOrder`, which means they need no new storage path and no new moderation surface: each one is already gated on its own row's safety status, exactly as an inline body image is. A published essay whose plates are still checking renders an empty run, and the console says so rather than leaving the reporter guessing.

Three tables carry the rest: `news_shows`, `news_assignments` and `news_pitches`.

A pitch is publish-first, and every other user-text surface here is scan-first. That is a considered difference, not an inconsistency. An article is broadcast to the whole school and there is no cheap way for a person to notice a bad one already live, so it scans before anyone sees it. A pitch is read by the desk and by nobody else, so the exposure of letting it through while it scans is one editor seeing it, while holding it invisible would mean the desk silently loses tips. A held verdict is withdrawn by the

The VGSpartans Handbook

read's own filter rather than by deleting the row, the same as every other surface. A separate profanity gate at submit rejects outright with a 422, so the obvious case never reaches a person at all.

There is no path from a pitch to an article. Accepting one opens an *assignment*, and an assignment is not a thing that can go live. A row that can be published must never be one edit away from a row a reader wrote.

SCAN-FIRST: THE ORDERING PROBLEM

The hard problem VGNews solves is ordering. A video that cleared moderation during drafting must not publish an article whose text was never scanned. The resolution has three interlocking parts. Change one and it leaks.

1. Video scans at upload, and can only hold. Finishing a video upload writes the video's storage key, sets its playback id to null, sets `modStatus` to `pending`, and fires a video moderation scan in scan-first mode. The core-side video gate is hold-only: a bad verdict holds the article, and a clear verdict writes the playback id back but never flips the article visible. Only the text path can make an article visible.

2. Publishing resets the safety state and re-scans, behind a video guard. Approving or publishing loads the row and runs a video guard first, refusing if a video key is set but the playback id is still null, which means an attached clip that has not cleared. Then it sets `status` to `Published` AND resets `modStatus` to `pending`, logs the move, and triggers a fresh scan of the text and the hero image.

The reset is safe precisely because the video guard already refused any article with an uncleared clip.

3. The reader enforces the media gate. An attached clip is only playable when the playback id is not null.

THE VISIBILITY CONTRACT

Getting this wrong publishes the school's private reporting to the open web, so it lives in exactly one function in `platforms/vgnews/src/services/newssite.ts`, and every read path reuses it.

```
platforms/vgnews/src/services/newssite.ts
status      = 'Published'                  -- an editor approved it
mod_status  = 'visible'                    -- the pipeline cleared the body and hero
deleted_at IS NULL                         -- not taken down
video_key IS NULL OR video_uid IS NOT NULL  -- no clip, or a cleared one
```

On top of that sits the audience gate. A signed-out viewer, which includes every crawler, is served only `public` pieces. Whether the reader is signed in is resolved from the cookie through `CoreRPC.verifySession`, and it fails closed to signed-out, which is the safer, public-only audience, on any error. This module never trusts anything the client can forge.

Worst wins on re-publish. The outcome applier keeps a condition that excludes held rows from an ordinary re-publish, so a flagged article can never be un-held by one.

The VGSpartans Handbook

MEDIA

Two distinct models, both gated.

Hero image. One per article, keyed on the row, served from a hero route and gated on the article's own visibility contract. It is scanned at publish time, with the text, not at upload.

Inline body images. Each gets its own row in `news_media`, following the VGWiki media model, served from a media route, and each gated on its own row's safety status. Every inline photo is a scan-first surface with its own verdict, triggered on upload. The article body renders through the shared editor's markup renderer with images enabled and the image prefix pointed at the isolated content domain.

Hero and inline both ride the cookieless content domain, gated on the content's publish and safety state. A school piece's hero is protected by its unguessable article id; the PAGE is where the school-versus-public audience gate lives.

MODERATION REGISTRATION

Core imports the vgnews schema directly and binds its database so the gate can write back. Three surface keys are registered, in the same change that gave the tables their status and deletion columns and a read path that filters on them. That ordering is the doctrine: no moderation theatre.

- In `platforms/core/src/moderation/apply-outcome.ts`: the article surface, the hold-only video gate, and the media surface.
- In `platforms/core/src/moderation/content-registry.ts`: the same three keys, so take-down and evidence capture work.

UPLOADS

Uploads ride the device-proof funnel, not a plain multipart fetch.

- **Images**, hero and inline: base64 inside JSON, with the server applying a body cap and the shared upload gate.
- **Video**: chunked byte posts for the parts, and JSON posts for begin, finish and abort, with the step-up tier applied to those rather than to every part. The multipart mechanics are the shared `packages/services/src/video-upload.ts` module, which vgclubs also delegates to. The part size is 10 MiB; the local simulator's minimum multipart part size is 5 MiB.

CONFIGURATION

Policy values live in `platform_config` with a console control, never in source. Video limits come from `packages/services/src/video-limits.ts`. Review-required is a config key, and the API route refuses a direct publish when review is on.

The masthead fetch window is a fetch bound, not a policy cap: a low-activity desk's full river still renders on its section page.

The VGSpartans Handbook

WHERE THINGS LIVE

Piece	Path
Schema	`packages/db/src/schema/vgnews.ts`
Editorial state machine and publish-time scan	`packages/services/src/newsroom/editorial.ts`
The shared write policy	`packages/services/src/newsroom/actions.ts`
The budget board and the tip queue	`packages/services/src/newsroom/budget.ts`, `packages/services/src/newsroom/pitches.ts`
The correction record	`packages/services/src/newsroom/corrections.ts`
The desk's derived views	`packages/services/src/newsroom/desk.ts`
The shapes both consoles render	`packages/services/src/newsroom/types.ts`
The reader and the visibility contract	`platforms/vgnews/src/services/newssite.ts`
Video and media services	`platforms/vgnews/src/services/newsvideo.ts`, `platforms/vgnews/src/services/newsmedia.ts`
Pages and API	`platforms/vgnews/src/pages/`
The newsroom console	`platforms/vgnews/src/components/console/NewsroomConsole.tsx`
The panels both consoles render	`packages/ui/src/newsroom/NewsroomPanels.tsx`
The guard and host router	`platforms/vgnews/src/middleware.ts`
Surface registration	`platforms/core/src/moderation/apply-outcome.ts`, `platforms/core/src/moderation/content-registry.ts`
Styles	`packages/ui/src/styles/spartan/vgnews.css`
The marketing page	`platforms/core/src/pages/[...locale]/news.astro`

The public masthead is never guarded. The newsroom console and its API gate on the `newsroom` panel, and fail closed with a 503 if core is unreachable.

TESTING

`platforms/vgnews/test/` binds only the database, the bucket and the namespace, with no core binding. The moderation triggers catch an undefined core and fail safe to `pending`, so state transitions are testable while a fresh scan simply never clears.

Two suites: the editorial state machine with the publish-time video-cleared guard, the safety-state reset and the permissions; and the read filter with both gates, the audience gate, the media gate, and the two servability helpers.

The VGSpartans Handbook

TRAPS

> ⚠ WARNING
>
> `packages/kernel/src/hosts.ts` is hand-maintained, not generated. `vgnews` has to be in the content-platform list and its type, and `"news"` in the app-platform list. Miss either and the host silently returns 404.

- **A brand-new `news.lvh.me` is DNS-filtered until `pnpm dev:hosts` runs on the machine running the browser.** The symptom is "cannot access news.lvh.me" while a curl with an explicit Host header to the local port returns 200. It is DNS, not the worker.
- **The reader stays dark until a remote deploy runs the moderation workflows.** Development skips the AI scan and fails closed to `pending`, so no article shows publicly in local development unless you hand-set its safety status to visible.

THE DEPLOY GATE

Held for an explicit go. When it is given:

1. `pnpm infra:provision`, which patches real resource ids into `platforms/vgnews/wrangler.jsonc`.
2. `pnpm db:migrate:remote`, which applies `migrations-vgnews/`.
3. Deploy, publishing vgnews before the gateway. CI syncs five secrets: `BETTER_AUTH_SECRET`, `CONTENT_BASE_DOMAIN`, `APP_DOMAIN`, `DETAIL_PEPPER` and `DEVELOPER_EMAIL`.
4. Add the storage lifecycle rule on the video prefix, infrequent-access after 30 days, as with clubs. The bucket is already in the backup workflow.

No video-host secret is needed on vgnews. It signs embeds and moderates video through core, exactly like vgclubs.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

TROUBLESHOOTING VGNEWS

Things that go wrong in VGNews, what causes them, and how to fix them.

Runbook . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vgnews/troubleshooting

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover a symptom table and the fix for each row.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

VGGALLERY

The campus photo and art gallery. Designed, not yet built.

Explanation . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vggallery

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

The source of truth today is `docs/VGGALLERY.md`.

When this page is written it will cover what it is, who it is for, and why it exists.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

USING VGGALLERY

Putting up your own work on the arts wall, shooting and publishing a campus album, and what the crew reviews.

How-to . For everyone, admins, developers . Checked 2026-08-02 . /documentation/platforms/vggallery/using-it

IN PLAIN TERMS

VGGallery has two wings under one roof. The **arts wall** is open to every Spartan: you put up your own drawings, paintings, digital work, or photographs of things you made, and no seat is needed. The **campus wing** is the school's visual record, shot and published by a seated crew. Both are run from one console, the darkroom, and which wings you see there depends on whether anyone has given you a seat.

THE TWO WINGS

Wing	What it holds	Who can post
Campus	Event and campus-life albums, shot by the crew	Seated photographers and curators
Arts	Your own work, standing on its own	Any signed-in Spartan

Everything defaults to **Spartans only**, because these are photographs of identifiable students. Only a curator can make a campus album public, and only the artist can make their own piece public.

THE RAIL

The darkroom is at `gallery.<appDomain>/darkroom`. What you see on the left depends on your seat, and every section listed is one you can actually use. See **Roles in VGGallery**.

Section	What it is	Who sees it
Campus	The albums, and the surface for building one	The crew
Arts	Your own work, and the review queue if you curate	Everyone signed in
Crew	Who is on the campus crew	A lead curator
Settings	The gallery's name, the accountable adult, and the review rule	A lead curator

An unseated Spartan opens straight into Arts and sees nothing else, which is the whole of what the arts wall needs.

The **View the gallery** button at the top opens the public site in a new tab. The search box in the header jumps between these sections; the albums table and the arts wall carry their own filters for finding a

The VGSpartans Handbook

particular album or piece.

PUTTING UP YOUR OWN WORK

1. Open **Arts** and choose your files. Your browser shrinks each one and strips the location data out of it before it leaves the page, and the server strips it again on the way in.
2. Give it a title, a line about it, and a medium.
3. If review is off, press **Put it up**. It appears once the safety checks clear.
4. If a curator reviews the wall first, press **Send for review** instead. You can press **Pull it back** while it is waiting if you change your mind.

A piece you have put up can be made visible to **anyone on the internet**, or pulled back to Spartans only, from the same row. Think about that one: public means search engines too.

If a curator sends your piece back, **their note is on the row**, above the buttons, along with who wrote it and when. That note is the reason it moved, so it is the first thing to read.

BUILDING A CAMPUS ALBUM

1. In **Campus**, start an album: a title, a line about it, and the day it happened. The date is the day of the event, not the day you post it, so a homecoming set put up a week late still reads as homecoming night.
2. Open it and add the photographs. The panel header shows how many the album holds and the cap it is working to.
3. Caption each one. **No names unless they said yes.** Changing the words on a photo sends it back through the safety checks, which is why an edited caption goes quiet for a moment.
4. Put them in the order you want them seen with the ↑ and ↓ buttons. This is the order the album page shows.
5. Pick a **cover** — the photo the album card leads with. The one currently chosen is marked. If it ever comes down, the card falls back to the newest photo that has cleared, so a card never goes blank.
6. Press **Send for review**. A curator takes it from there.

Adding a photo to an album that is already published pulls the whole album back for review, so nothing new ever reaches the wall unseen. The console says so when it happens.

REVIEWING

A curator sees the queue at the top of **Arts** and the albums table in **Campus**.

- **Approve** puts a piece on the wall. **Publish** does the same for an album.
- **Send back** returns it with a note. The note is required, and it is what the person who made it reads. It appears only on things actually waiting on you — something already on the wall is taken down or unpublished instead.
- **Take it off the wall** hides a piece without destroying it.

The VGSpartans Handbook

- **Take down** is the last resort: the picture itself is deleted, not hidden.

Published rows carry an **On the wall** link straight to the public page, and it only appears once the piece is genuinely there — a row that is published but still being checked has no link, because there would be nothing at the other end of it.

SETTINGS

A lead curator sets the gallery's name and tagline, names the **accountable adult** whose address makes them the adviser, and decides whether a curator reviews arts pieces before they go up. Upload caps and size limits live in the admin console with every other platform limit.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

ROLES IN VGGALLERY

Who can do what in VGGallery, and how those permissions are decided.

Reference . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vggallery/roles

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

The source of truth today is `docs/VGGALLERY.md`.

When this page is written it will cover every role and the exact capability each one carries.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

VGGALLERY UNDER THE HOOD

How VGGallery is built: its worker, its data, and its moving parts.

Explanation . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vggallery/under-the-hood

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

The source of truth today is `docs/VGGALLERY.md`.

When this page is written it will cover the worker, the database, the storage, and the request path.

The VGSpartans Handbook

PART 2. PLATFORM GUIDES

TROUBLESHOOTING VGGALLERY

Things that go wrong in VGGallery, what causes them, and how to fix them.

Runbook . For everyone, admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/platforms/vggallery/troubleshooting

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

The source of truth today is `docs/VGGALLERY.md`.

When this page is written it will cover a symptom table and the fix for each row.

The VGSpartans Handbook

PART 3

ADMINISTRATION

The role model, the consoles, moderation, and the settings that govern the school.

PART 3. ADMINISTRATION

ADMINISTRATION

The role model behind VGSpartans, and the consoles that act on it.

Explanation . For admins . Checked 2026-07-25 . Unfinished . /documentation/administration

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover every role on the platform and what separates them.

The VGSpartans Handbook

PART 3. ADMINISTRATION

THE CONSOLES

Every authenticated management surface on the platform, and who reaches it.

Explanation . For admins, developers . Checked 2026-07-25 . Unfinished . /documentation/administration/consoles

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the full list of consoles and their access rules.

The VGSpartans Handbook

PART 3. ADMINISTRATION

THE ADMIN CONSOLE

The school admin console, panel by panel.

Reference . For admins . Checked 2026-07-25 . Unfinished . /documentation/administration/consoles/admin-console

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover each panel, written from the component that renders it.

The VGSpartans Handbook

PART 3. ADMINISTRATION

THE DEVELOPER CONSOLE

The highest-privilege console on the platform, the two independent locks in front of it, and the manual setup one of them needs.

Explanation . For developers . Checked 2026-07-27 . /documentation/administration/consoles/developer-console

---

> **IN PLAIN TERMS**
>
> The developer console can change almost anything about the platform. It is protected by two separate locks that were built to be independent: an identity check at Cloudflare's edge, before the request ever reaches our code, and a second check inside the application. Either one alone would be enough on a good day. Both together are what makes a bad day survivable.

LOCK 1: CLOUDFLARE ACCESS

Cloudflare Access authenticates the operator at the edge, before any request reaches the worker. It is configured once in the Cloudflare dashboard, not in the repository, so there is no Access configuration or secret committed here.

1. Zero Trust, Access, Applications, Add an application, Self-hosted.
2. Application domain: `developer.vgspartans.org`. Leave the path blank so the policy covers the whole host.
3. Session duration: keep it short, for example 24 hours, to match a high-privilege surface.
4. Add a policy:
   - Action: Allow.
   - Include: Emails, listing only the operator addresses that should ever reach this console. Keep it to the smallest set possible.
   - Optionally also require a login method (one-time PIN or an identity provider) under Authentication.
5. Save, then add a second policy with Action: Block, Include: Everyone, below the allow policy, as an explicit default-deny backstop.

For DNS, `developer.vgspartans.org` needs a proxied (orange-cloud) `AAAA 100::` record in the `vgspartans.org` zone, the same as the other console subdomains. The existing wildcard route already sends its traffic to the worker, so no new worker route is needed.

The host-scoping trap

An Access policy is scoped to a HOST. That only protects the console if the console's API is reachable on that host and nowhere else.

The VGSpartans Handbook

> **DANGER**
>
> It used not to be. `/api/` was a plain shared path, so a POST to `https://vgspartans.org/api/v1/developer` hit the same handler on the apex, where no Access policy sits. Every developer mutation was one session cookie away from anyone who had one.
>
> `CONSOLE_API_OWNER` in `packages/kernel/src/hosts.ts` now pins each console API to the one console subdomain that owns it, and `routeRequest` returns 404 for it anywhere else in production. Development and CI keep path-based access, since they have no subdomains. Do not move that check back below the shared-path branch.

LOCK 2: THE SESSION AND PANEL GUARD

Defence in depth, so a misconfigured or deleted Access policy still cannot expose the console.

- `platforms/core/src/pages/developer/index.astro` sets `prerender = false`, which makes the route render on demand so the middleware runs on every request.
- The middleware maps `/developer` to the `developer` panel, and `panelOk` in `packages/services/src/identity.ts` requires an admin-level identity: a master-admin email, or the `developer` or `it_admin` profile role. A non-privileged session is redirected to the sign-in path.
- The API is guarded the same way, mapping `/api/v1/developer` to the `developer` panel, and its mutations are step-up verified and audited.

Break-glass is a door, not a role

`panelOk(identity, "developer")` admits a master admin. That is deliberate break-glass for the DOOR, and it must not be read as “an admin is a developer”.

Anything that mints or manages an administrator goes through `managementError` and `setUserRole` in `platforms/core/src/services/usermgmt.ts`, which requires `actor.isDeveloper` and refuses self-targeting.

`grantAdminRole` once skipped both, which let a master admin write itself the `developer` role from inside the break-glass door. It delegates now, and `platforms/core/test/grant-admin.test.ts` holds that line.

> **WARNING**
>
> Keep both locks. Do not weaken `prerender = false`, do not remove the `/developer` panel rule, and do not treat Access alone as sufficient.

THE SIGN-IN LOCKDOWN

Flags & config carries one switch that closes the platform’s front door: **Sign-in lockdown**. It is the control to reach for during a security incident, a migration, or a window before launch when the school should not be signing in at all.

The VGSpartans Handbook

While it is on:

- No login code is sent to anyone outside the allowlist. A school `@cguhsd.org` address is refused exactly like a stranger's; the domain rule stops applying.
- No new account can be created, from the sign-up form or any other route. A code that was issued before the switch was flipped will not open a session either.
- Anybody already signed in who is not on the allowlist is signed out on their next click, server-side. Without that, a member's session would keep working for another fortnight and the lockdown would be a sign-up freeze in disguise.

The person being refused sees the ordinary "check your email" screen and no code arrives, which is the same uniform answer every other withheld code gets. The sign-up form is the exception and says plainly that new accounts are closed: a lockdown is a property of the platform rather than of the address typed in, so saying so there gives nothing away.

Who is still admitted

The allowlist is the union of three GitHub-managed values: `DEVELOPER_EMAIL`, `MASTER_ADMIN_EMAILS` and `LOGIN_EMAIL_WHITELIST`. The panel lists them, masked, so you can see who is still admitted before you flip anything.

It is read from the deploy's environment and never from the database, which is the point rather than an implementation detail. The way back in must not depend on the table, the console or the network path that the lockdown itself lives in, which is the standing rule for emergency access: an account you may need during an incident has to be excluded from the policy that could otherwise shut it out (NIST SP 800-53r5 AC-2(2)). `MASTER_ADMIN_EMAILS` rides along for the same reason `panelOk` admits a master admin to this console at all, so a single mistyped `DEVELOPER_EMAIL` cannot lock out every human on the deployment.

Because of that, the allowlist is deliberately **not** editable here. Widening it means editing the secret or variable in GitHub and redeploying.

> **DANGER**
>
> Turning this on with an empty allowlist locks every account out of the platform, including yours, and no console can undo it. Set `DEVELOPER_EMAIL` first. The panel says so when the list is empty.

What happens if the flag cannot be read

The two halves fail in opposite directions, on purpose.

- **Sign-in and sign-up fail closed.** If neither the config table nor its cache can be read, the request is refused. "We could not read the lockdown flag" must never resolve to "so let them in", and the cost of being wrong is one refused sign-in. An allowlisted address is answered before any of those reads happen, so this cannot strand the person who would fix it.
- **Session teardown fails soft.** If the flag cannot be read there, live sessions are left alone. A wrong refusal costs somebody a retry; a wrong mass sign-out is a platform-wide logout that no toggle takes back.

The VGSpartans Handbook

The switch is stored as `auth.lockdown` in `platform_config`, and writing it goes through the same step-up verification and audit entry as every other key on that tab. The behaviour lives in `packages/services/src/lockdown.ts`, and `platforms/core/test/auth-lockdown.test.ts` holds the line on all of it.

For the incident-response side of this, see **Users cannot sign in**.

MANUAL SETUP CHECKLIST

- [ ] Create the proxied `AAAA 100::` DNS record for `developer.vgspartans.org`.
- [ ] Create the same record for `webmaster.vgspartans.org`. The webmaster console was renamed from `manage.*`; the old host keeps working through a 308 redirect, but the new host needs its own record.
- [ ] Configure the Cloudflare Access application and its allow and block policies, as above.
- [ ] Confirm the operator emails are also admin-eligible, through the master-admin list or the `developer` profile role, so lock 2 lets them in.

The VGSpartans Handbook

PART 3. ADMINISTRATION

THE CLUB CONSOLES

The four club officer consoles and what separates them.

Reference . For admins, everyone . Checked 2026-07-25 . Unfinished . /documentation/administration/consoles/club-consoles

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the advisor, webmaster, treasurer and secretary consoles side by side.

The VGSpartans Handbook

PART 3. ADMINISTRATION

THE BUG BOUNTY CONSOLE

The invite-only, time-boxed security research programme, its access model, and the runbook for provisioning and ending access.

How-to . For developers . Checked 2026-07-25 . /documentation/administration/consoles/bounty-console

IN PLAIN TERMS

A small number of outside security researchers are given written permission to go looking for holes in VGSpartans, for a fixed period, with a private line to the developer to report what they find. This page is how that permission is granted, how it ends by itself, and why every part of it is built to close rather than to stay open.

The public-facing terms live at `/bug-bounty`. This page is the access model and the runbook.

THE THREE SURFACES

Surface	Who	Where
`bounty.<appDomain>`	The researcher	`platforms/core/src/pages/bounty/index.astro`, rendering `platforms/core/src/components/console/BountyConsole.tsx`
Developer console, Bug bounty tab	The developer	`platforms/core/src/components/console/developer/BountyPanel.tsx`
`/bug-bounty`	Anyone	The public page, where people ask to join

THE ACCESS MODEL

Nobody enrols themselves. There is no sign-up endpoint anywhere in this feature. A person emails to ask, and a developer provisions their email in the developer console with an explicit scope and an explicit expiry. That row is the authorization.

Four rules, each load-bearing.

1. Never a school address

`provisionResearcher` refuses any `@cguhsd.org` address, checked again at the endpoint so the error is readable.

The VGSpartans Handbook

WHY IT'S THIS WAY

This is what makes rule 2 safe. A closed enrolment blocks sign-in outright, so if a Spartan were enrolled on their school address, the day their bounty window lapsed they would lose their own account: profile, personal site, club console, everything. A bounty account is a separate identity, deliberately.

It also keeps the blast radius honest in the other direction. A bounty account is the one credential on the platform that an attacker knows is authorized to probe production. Off the district domain, compromising it yields no school identity.

2. Time-boxed, two months maximum, and it closes itself

`BOUNTY_MAX_DAYS` is 62: two consecutive 31-day months, so “two calendar months” is never rejected for being a day over. It is enforced on the first grant and on every renewal, and the renewal ceiling is measured from now. Chaining renewals off the old deadline would let a grant creep arbitrarily far forward while every step looked compliant.

Expiry is derived, never stored. The status column holds only what a human decides: active, suspended, or revoked. Whether the grant has run out is read off `expires_at` against the clock on every request. There is no cron that disables anything, no “expired” status anyone has to remember to write, and no cached authorization to invalidate. A grant nobody touched still closes itself.

That follows NIST SP 800-53r5 AC-2(2): temporary accounts are disabled automatically after a defined period, not at an administrator’s convenience.

3. Closure blocks sign-in, not just the console

When a grant closes:

- `panelOk(id, "bounty")` fails, so the console returns 403 to the hub with a denial notice that carries the renewal address.
- `isEligible` returns false, so `sendVerificationOTP` withholds the login code entirely. They cannot sign in at all.
- Suspend and revoke additionally call `revokeAllUserSessions`, so a live session dies at the point of the write. Natural expiry has no write event, so the guard chain’s navigation gate catches it on the next request, narrowly: only for an account whose sole eligibility was the closed grant. A researcher who is also, say, a whitelisted community member keeps their session and merely loses the console.

Because this is severe, it is never a surprise. See rule 4.

4. Mandatory MFA, and mandatory warnings

MFA. An open enrolment puts the account in the same strong-factor tier as an admin. The mandate rides the enrolment, not the console: gating only `bounty.<appDomain>` would leave an account that by definition attracts attackers unprotected everywhere else.

Strong means an authenticator app, a passkey, or a security key. The two WebAuthn types are the phishing-resistant pair; NIST SP 800-63B-4 puts high-risk workflows at AAL2 with phishing-resistant

The VGSpartans Handbook

authenticators where practical. An authenticator app stays admissible so the mandate is satisfiable without buying hardware.

The developer's table shows a readiness column per row, because provisioning happens before the person has ever signed in. "Provisioned" and "actually protected" are different states.

Warnings at 7 days and 3 days, in the console banner and by email. The mail says the exact date, says that sign-in stops working, and gives an address to write to, because once access ends they cannot use the console to ask for a renewal.

Each stage is stamped only after a send succeeds, so a mail failure retries tomorrow and a second run the same day sends nothing. Both stamps clear on renewal.

WHERE THE DATA LIVES

Three homes, and no cross-database join, because the database engine has none. They are stitched in application code, the same way identity resolution is.

Concern	Table	Database
The access grant	`bounty_researchers`	`USER_DB`, because identity resolution reads it every request
Programme content	`bounty_posts`	`CORE_DB`. Rules, instructions, tasks, notices
The conversation	`report_submissions` where `channel = 'bounty'`	`CORE_DB`

There is no second messaging system. Bounty threads are ordinary support tickets on a restricted channel, so threading, attachments (a proof of concept, a screen recording), claiming, and the advisory moderation scan all came free.

The bounty channel is developer-only

This is the strictest read rule on the platform, stricter than the admin-to-developer channel. What travels here is unfixed vulnerabilities in this platform, so:

- `listTicketQueue` subtracts the developer-only channels from a non-developer's queue outright. An admin sees no bounty rows at all, and the count reads zero.
- `getTicketDetail` returns nothing for an admin even when they own the row. Unlike the developer channel, there is no ownership escape hatch. The restriction is about who may read a working exploit, not about who wrote it down.
- The support endpoint rejects a `channel: "bounty"` request from a non-researcher with a 403 rather than downgrading it to an ordinary support ticket. A silent downgrade would take a message the sender believed was private and file it where every admin can read it, and the window is real: an enrolment can lapse between the page loading and the send.

The VGSpartans Handbook

RUNBOOK

Provisioning someone

1. Developer console, Bug bounty, “Add a researcher”.
2. Their **personal** email. A school address is refused.
3. **What they may test.** This is the authorization half of the safe harbour, so write it as though it will be quoted back at you. Name what is in bounds; anything unnamed is out.
4. **Access ends.** Pre-filled at 30 days, hard-capped at 62.

They then create their account at the sign-in page (the open grant is what makes them eligible for a login code), set up a second factor, and the Bug bounty card appears on their hub.

Renewing

Developer console, Bug bounty, Manage, then a new end date. Renewal reopens a lapsed or suspended enrolment and re-arms both warnings.

It will not reopen a revoked one. Revocation is terminal, and bringing someone back goes through provisioning, so the scope is restated deliberately rather than reinstated by a button.

Ending access

- **Pause** (suspend) is reversible and keeps the scope. Use it when something needs checking.
- **End** (revoke) is terminal.

Both kill live sessions immediately and take effect on the next request.

Publishing to the programme

Developer console, Bug bounty, “Post to the program”. Four kinds:

- **rule**: the programme’s terms. Cover the disclose.io four elements: Authorization, good-faith Acceptance, Anti-retaliation, Reciprocity.
- **instruction**: how to do the work, and what a good report contains.
- **task**: a specific piece of work, with an open or closed state.
- **notice**: time-sensitive. Pin it.

A post is programme-wide, or addressed to one researcher by email.

Retracting is a soft delete. It disappears from every console immediately, but the record survives, because a rule that was in force while somebody was testing under it is evidence about what the programme authorized at the time. ISO/IEC 29147 expects a policy whose history can be established.

OPERATIONAL NOTES

- **Step-up.** All six lifecycle verbs are marked privileged; the two noticeboard verbs are writes. Every one is audited.

The VGSpartans Handbook

- **No Cloudflare Access on `bounty.<appDomain>`**, unlike the developer host. A researcher may be an outsider with no district identity, and locking the host behind an external single-sign-on perimeter would make the console unreachable by exactly the people it exists for. Its locks are the enrolment grant, mandatory MFA, and the ordinary console chain: device fingerprint, rate limiting, CSRF.
- **Post bodies render as plain text, never as markup.** This console's readers are, by definition, people who probe for injection.
- **The daily sweep rides the existing 07 UTC cron**, inside the worker's scheduled handler, with its own error handling so a mail outage cannot take down the nightly database backup sharing the tick.

> **NOTE**
>
> Cron triggers do not follow daylight saving, so that tick drifts an hour in summer. That is irrelevant for a 7-day warning, but do not tighten the windows to hours on the assumption it is punctual.

Test it locally:

```
wrangler dev --test-scheduled
curl "localhost:8787/__scheduled?cron=0+7+*+*+*"
```

MIGRATIONS

- `platforms/core/migrations-user/0005_bounty_researchers.sql`, the grant.
- `platforms/core/migrations-core/0008_bug_bounty.sql`, the noticeboard.

The second is hand-written on purpose. Core migrations 0005 to 0007 shipped without snapshots, so `drizzle-kit generate` diffs against the 0004 snapshot and emits a destructive recreate of `report_submissions` and the moderation tables. The 0008 snapshot is kept as the re-baseline so later generates diff correctly.

REFERENCES

- ISO/IEC 29147:2018: published policy, defined scope, structured intake, stated acknowledgment timelines, versioned policy.
- **disclose.io** (https://disclose.io/docs/recipients/): the four-element safe harbour model.
- NIST SP 800-53r5 AC-2(2) and AC-2(3): automatic disabling of temporary accounts.
- NIST SP 800-63B-4: AAL2 and phishing-resistant authenticators for high-risk workflows.

The VGSpartans Handbook

PART 3. ADMINISTRATION

MODERATION

One pipeline checks every piece of text, every image and every video on every platform. This is how it decides, and what an admin's part in it is.

Explanation . For admins, developers . Checked 2026-07-25 . /documentation/administration/moderation

---

IN PLAIN TERMS

Everything anyone posts on VGSpartans goes past an automatic check before or just after it appears: what it says, what is in the picture, what is said out loud in a video. Most of it passes and nobody ever knows the check happened. Anything the check is unsure about goes into a queue for a person to look at. Anything it is certain about, in a small list of categories that are never acceptable, is stopped outright.

One pipeline serves every platform. Text, images and video from every surface pass through the same stages, the same category set, the same decision engine and the same review console. Classifiers, thresholds and policy live in exactly one place, so there is one thing to maintain and one thing to tune.

The shared engine is in `packages/services/src/moderation/`. It is modality-agnostic and reused by all three pipelines. The orchestration that runs it lives in `platforms/core/src/moderation/`.

THE PRINCIPLES

1. **Cheap first, escalate rarely.** A free synchronous gate, then Workers AI, decide the vast majority. Paid third parties are called only on ambiguity, not on every item.
2. **Store raw, not just verdicts.** Every classifier's raw scores are written to the ledger, so thresholds can be re-tuned and replayed against history without re-running the models. The same philosophy as the audit log.
3. **One home.** The whole pipeline runs on `vgcore`, which already binds every database, runs the workflows, and answers every platform over `CoreRPC`. Platforms hand off a reference and read back a verdict. That is all they do.
4. **Extensible by construction.** Classifiers are stages in an ordered chain, not hardcoded calls. Adding a model, a backup, or a whole new vendor is appending one stage entry, never rewiring the pipeline.

The VGSpartans Handbook

THE SPINE

```
[platform]  someone submits text, an image, or a video
     |
     v
  STAGE 0  synchronous ingest gate, before anything is stored
     |  profane text, contact-info and URL patterns, magic-byte sniff
     |  a hard hit is refused inline and the bytes are never stored
     |
     |  passes -> persisted, then handed off
     v
  CoreRPC.moderateText / moderateImage / moderateVideo
     |
     +--> text and image  -> a Workflow running the classifier chain
     |
     +--> video           -> a Workflow: sample frames into the image path,
     |                       transcribe audio into the text path
     v
  DECISION ENGINE (shared, pure)
     |
     v
  ledger write (raw signals + verdict)  ->  flip the publish gate in the owning database
```

STAGE 0, THE SYNCHRONOUS GATE

Runs inline on the request, before anything is stored. Hard hits are refused on the spot with a clear reason. This is the cheapest possible filter and it catches the obvious cases for free.

- `gateUpload()` checks a profane filename, the size cap, and the file's magic bytes, so a renamed executable is refused. Every upload path runs it.
- The obscenity matcher is obfuscation-aware, with a tuned false-positive allowlist.
- Contact information and bare URLs are treated as a first-class refusal in text. For a platform serving minors, unsolicited contact sharing is not a soft signal.

TEXT IS DE-OBFUSCATED BEFORE ANYTHING READS IT

Before any classifier sees text, `normalize.ts` decodes it: zero-width and format characters stripped, Unicode homoglyphs and diacritics folded to ASCII, leetspeak decoded, and run-together phrases re-spaced by a conservative dictionary segmenter, so `youareanidiot` becomes `you are an idiot`.

Each stage then classifies both the original and the decoded copy, so an evasion is exposed without discarding the original reading. The same helper feeds the image OCR text and the video transcript, so every modality inherits it.

That is grounded in Unicode UAX #15 (normalize then strip combining marks for diacritics) and UTS #39 (homoglyph skeletons; no normalization form folds scripts on its own).

The VGSpartans Handbook

THE CATEGORIES

Every classifier maps onto one internal set, and the engine takes the maximum severity per category across whatever ran. The set is Llama Guard's stock hazard taxonomy plus three additions that a youth platform needs:

- `bullying_harassment`, because the stock set under-covers peer harassment.
- `pii_sharing`, for contact details, addresses and schedules.
- `csam`, handled on a separate mandatory track.

Severity per category is none, low, mid or high. The mapping from each vendor's labels to this set is a lookup table in `packages/services/src/moderation/taxonomy.ts`, so a vendor renaming a label is a one-file change.

HOW A VERDICT IS REACHED

Every signal carries a category and a severity. Severity maps to points, the content risk is the highest weighted category, and trust is 100 minus that risk.

A per-user reputation score then shifts the risk: trusted accounts get a small discount, new or previously-flagged ones a small penalty. Reputation never overrides a category floor.

The decision, in trust units where higher is safer:

- A high-severity hit in a **floor** category is **blocked**, regardless of score or reputation.
- Effective trust below the block threshold is **blocked**, and admins are notified.
- Effective trust below the review threshold is **flagged**: it publishes, and it is filed for review.
- A flagged verdict whose evidence is a **hide-on-flag** category at mid or high severity is downgraded to **held**, hidden until a person clears it.
- Otherwise, **approved**.
- A **degraded** run, where a primary stage failed and a backup carried the verdict, is logged in full and never approved outright. An otherwise-approved degraded item becomes flagged.
- If every stage failed, the on-error policy decides: fail-open, the default, makes it flagged; fail-closed makes it held.

`decide()` in `packages/services/src/moderation/trust-score.ts` is the single source of truth for all of that.

> ⚠ **WARNING**
>
> Five floor categories are mandatory and cannot be unset from the console: child exploitation, sex crimes, sexual content, indiscriminate weapons, and violent crimes. The settings box may add more. It may never remove these.

The VGSpartans Handbook

> **WHY IT'S THIS WAY**
>
> Hide-on-flag defaults to `bullying_harassment` and deliberately NOT to `hate`. A student writing about racism, or about their own identity, must not be silenced by a borderline score. Hide-on-flag is narrower than a floor: a floor blocks outright on a high-severity hit, while this only hides a borderline flag, and it is aimed at targeted harm that a youth platform should not broadcast unreviewed. An explicit empty list turns it off.

THE SETTINGS THAT TUNE IT

These are `platform_config` keys, editable from the admin console's Settings tab. The names are read from `packages/services/src/moderation/config.ts`.

Key	What it does
`mod.text.block_at`	The block threshold
`mod.text.review_at`	The review threshold
`mod.text.on_error`	`fail_open`, the default, or `fail_closed`
`mod.text.floor_categories`	The always-block categories, on top of the five mandatory ones
`mod.text.hide_on_flag`	Categories where a flag becomes held. Empty turns it off
`mod.text.escalate_band`	The score window in which a second opinion is called
`mod.text.w.<category>`	A per-category weight override
`mod.user.enabled`	Whether the reputation layer runs
`mod.user.default`	The score a new account starts at
`mod.user.weight`	How much reputation is allowed to shift the risk
`mod.user.penalty_block`, `mod.user.penalty_upheld`	Reputation cost of a block, and of an upheld flag
`mod.user.reward_clean`, `mod.user.reward_cleared`	Reputation restored by a clean streak, and by a cleared flag

SCAN FIRST, OR PUBLISH FIRST

Not every surface waits for the verdict. Which one a surface uses is decided by its shape, not by how risky it feels.

The VGSpartans Handbook

Surface	Mode	What a blocked verdict does
A VGSynergy post or VGAgora idea image	scan first	Holds and removes the parent row; the image rides the same entity
VGWiki media	scan first	Holds the media row's status, preserves the picture, soft-deletes the row
A VGWrites cover	publish first	Clears the cover key and deletes the bytes; the work itself is untouched
A VGCore avatar	publish first	Clears the profile field and deletes the bytes

WHY IT'S THIS WAY

The split is not arbitrary. A surface is scan-first when the image is, or hangs off, a ROW with a status column that every read path filters. A cover and an avatar are FIELDS whose bytes are served straight from storage by a deterministic key, with no row read, so there is nothing per-request to gate on. Those are published behind the synchronous gate and retracted on a block.

Two costs come with that and are accepted deliberately. A failed trigger on a publish-first surface leaves an unscanned image up. And because a cover's key is derived from the work id, a late verdict applies to whatever is at that key, so a re-upload during the scan window can be taken down along with the image it replaced.

Every upload endpoint re-derives the content type from the file's magic bytes rather than trusting the browser's declared type, and stores and classifies the sniffed type. OWASP's file-upload guidance is explicit that the declared type is spoofable and that signature validation belongs alongside an extension allowlist.

CSAM IS A SEPARATE, MANDATORY TRACK

This platform serves students, so this is not optional and it is not just another category.

- Detection is hash matching against known-hash lists, on every image and every sampled video frame, independent of the AI classifiers.
- On a hit, the content is frozen: never served, and never deleted out from under an investigation. There is a legal reporting obligation in the United States to report to the NCMEC CyberTipline.

DANGER

Build the report path. Do not improvise it during an incident.

There is an open verification item that blocks the video and object-storage paths: confirm with Cloudflare whether their CSAM Scanning Tool covers frames derived from video and media served from object storage, or only images served through the zone cache. Until that is confirmed, the pre-

The VGSpartans Handbook

publication human review gate on video is the backstop for this category, which is a reason to keep that gate strict from day one.

REMOVAL, EVIDENCE, AND PERMANENT DELETION

Setting a status to "held" is a visibility flag, not a removal. Anything reached through a path that does not filter on it, an author's own view, a cached list, a direct link, a query that forgot the clause, stays up. Content the classifiers scored at the floor is exactly the content that must not survive on a technicality, so a blocked verdict and a reviewer's "Take it down" both really delete.

Deleting honestly needs a copy first. Three pieces do that.

`moderation_evidence`**, the exact copy.** Written the moment content first draws attention, whether that is a flag, a block, a report, a take-down or the author's own delete, and kept whatever happens to the original. It stores the whole source row as JSON, not a summary, plus a SHA-256 of that JSON, plus a copy of any media pulled out of the subplatform's bucket into `STORAGE_EVIDENCE`. It is unique per content and reason, so a retried workflow step or a double-clicked button reuses the first capture.

`moderation_deletions`**, the ledger and the clock.** One row per piece of content actually removed, by its author, a reviewer, or the pipeline. This is the console's Deleted tab. A permanent delete is armed, not executed: the purge lands 72 hours out and any admin can cancel until the sweep passes it. Only then are the evidence row, the copied bytes and the original bytes destroyed. The deletion row survives as a tombstone, because "this existed and was destroyed on this date" holds no content.

Order is the safety property. The removal captures evidence first, and a failure there aborts the removal, so there is no state where content is gone and no copy of it exists. The media copy is the one best-effort step: a storage hiccup must not keep harmful content up, and because the source object is only deleted at purge time, the bytes still exist even when the copy fails.

> ⚠ WARNING
>
> `held` deliberately does not delete. `held` is what a fail-closed pipeline returns when the classifiers could not run at all, for example a Workers AI outage. An outage must never delete a student's work. It stays hidden and waits for a person, which is what the queue is for.

The wiring:

- `platforms/core/src/moderation/content-registry.ts` holds one entry per removable surface: how to read the full row, how to soft-delete it, where its media lives, and which counter to re-sync. It is separate from the visibility gates in `apply-outcome.ts` on purpose. That map is about visibility and runs on every verdict; this one is about removal. A surface can be gated long before anyone is willing to let a classifier delete from it, and listing it here is the explicit consent.
- A VGWiki article is deliberately not listed. Its moderation entity is a revision, and deleting a page over one bad edit is exactly the vandalism-by-moderation attack the wiki gate exists to prevent. Rolling back to the last good revision stays the correct removal there.
- `CoreRPC.preserveBeforeDelete` is how an author's own delete reaches the locker, since the subplatforms bind no core database. It is a no-op for content that never drew a complaint, so it only

The VGSpartans Handbook

bites when a record already exists, which is what stops “delete it before anyone looks” from working.

- The hourly cron runs the purge sweep, and the console sweeps again when it reads the Deleted tab. Hourly rather than daily, so 72 hours is a real deadline rather than somewhere between 72 and 96.

THE CONSOLE

Three tabs:

- **Moderation**: what is pending, showing the preserved copy rather than the live row.
- **History**: everything already decided, across flags, reports and report-form messages.
- **Deleted**: what is gone, with the purge control.

Every item shows the raw per-classifier signals, not just the verdict, so a reviewer sees why it landed there. Reviewer decisions are logged as ground truth, which is what later threshold tuning is calibrated against.

Working an item is covered in **Working the moderation queue**.

COMPLIANCE

Subprocessors that touch student content: Cloudflare, AWS (already a subprocessor through the mail path, so Rekognition adds no new vendor), Azure, and OpenAI.

> ⚠ WARNING
>
> Azure AI Content Safety and OpenAI moderation both have to be named in the privacy policy before any student content is sent to them. Both run only on the fallback or escalation path, but an item on that path is still student content leaving to a new vendor, so the policy update gates those calls going live, not just a full rollout.

The ledger keeps raw classifier output. A retention window for that raw output on blocked items needs to be decided in line with the privacy policy and any legal hold requirements.

WHAT IS NOT WIRED YET

- **Video ingest on the submit endpoints**, and the same pattern repeated for comments, wiki articles and VGWrites chapters.
- **VGSites and VGClubs file uploads.** Both tables already have the status column; what is missing is the trigger and a read-path filter.
- **An evidence copy for the field-shaped surfaces**, the avatar and the cover. Preservation runs through the row-shaped content registry, so a blocked avatar or cover deletes its bytes with no copy kept. Every row-shaped surface does preserve.
- **The Rekognition transport.** The parser is done; the call is a guarded stub.
- **The CSAM hash matcher.** The hook exists. The zone scanner is enabled separately.

The VGSpartans Handbook

Every classifier no-ops or fails safe when its secret is absent, so nothing breaks in a deployment that has not configured one.

The VGSpartans Handbook

PART 3. ADMINISTRATION

WORKING THE MODERATION QUEUE

Taking a case, reading the evidence, and recording an outcome.

How-to . For admins . Checked 2026-07-25 . Unfinished . /documentation/administration/moderation/working-the-queue

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the queue screen and every action on it.

The VGSpartans Handbook

PART 3. ADMINISTRATION

TAKE-DOWNS

Removing content for real, what is preserved first, and how to undo it.

How-to . For admins . Checked 2026-07-25 . Unfinished . /documentation/administration/moderation/take-downs

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the take-down flow, the evidence capture, and the cancellation window.

The VGSpartans Handbook

PART 3. ADMINISTRATION

TROUBLESHOOTING MODERATION

When the pipeline flags the wrong thing, or nothing at all.

Runbook . For admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/administration/moderation/troubleshooting

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover a symptom table for the moderation pipeline.

The VGSpartans Handbook

PART 3. ADMINISTRATION

MANAGING PEOPLE

Finding an account, changing what it can do, and ending its access.

How-to . For admins . Checked 2026-07-25 . Unfinished . /documentation/administration/people-management

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the people screens in the admin console.

The VGSpartans Handbook

PART 3. ADMINISTRATION

PLATFORM SETTINGS

The settings that govern the whole school, where they live, and how to change one.

How-to . For admins, developers . Checked 2026-07-25 . Unfinished . /documentation/administration/platform-config

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the platform_config table and the console controls that write it.

The VGSpartans Handbook

PART 3. ADMINISTRATION

THE SCHOOL REGISTRY

The one file that says which platforms a school runs and on which domains.

Explanation . For developers . Checked 2026-07-25 . Unfinished . /documentation/administration/school-registry

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the registry, the generated mirrors, and the drift check.

The VGSpartans Handbook

PART 4

SECURITY

How sign-in, sessions, devices, bots, privacy and content safety actually work.

PART 4. SECURITY

SECURITY

How VGSpartans protects accounts, content and privacy, and what it deliberately does not try to protect against.

Explanation . For admins, developers . Checked 2026-07-25 . Unfinished . /documentation/security

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover a map of the security part.

The VGSpartans Handbook

PART 4. SECURITY

THE THREAT MODEL

Who this platform is defending against, and what it accepts it cannot stop.

Explanation . For admins, developers . Checked 2026-07-25 . Unfinished . /documentation/security/threat-model

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the named threats and the defences that answer each one.

The VGSpartans Handbook

PART 4. SECURITY

SIGNING IN

What happens between typing your email and being signed in.

Explanation . For admins, developers . Checked 2026-07-25 . Unfinished . /documentation/security/signing-in

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the full sign-in path, step by step.

The VGSpartans Handbook

PART 4. SECURITY

MFA AND PASSKEYS

The second factor: when it is required, and the forms it can take.

Explanation . For admins, developers . Checked 2026-07-25 . Unfinished . /documentation/security/mfa-and-passkeys

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the factor policy and each supported factor.

The VGSpartans Handbook

PART 4. SECURITY

SESSIONS AND DEVICES

What a signed-in session is, how long it lasts, the three layers that tie it to the device that created it, and every event that ends one.

Explanation . For admins, developers . Checked 2026-07-26 . /documentation/security/sessions-and-devices

IN PLAIN TERMS

Signing in gives your browser a small piece of proof it shows on every request afterwards, so you are not typing a code all day. The obvious way to steal an account is to steal that piece of proof and use it somewhere else, which is why it is not enough on its own here. It is tied to the device that created it, in three separate ways, and a copy that turns up on a different machine is asked to prove something a copy cannot prove.

WHAT A SESSION IS

One cookie, minted by better-auth when the emailed one-time code is accepted, and signed with the application secret.

Attribute	Value
Name	`__Secure-better-auth.session_token` in production, unprefixed in development
`Secure`	Yes, which the `__Secure-` prefix requires
`HttpOnly`	Yes, so page JavaScript cannot read it
`SameSite`	`Lax`
`Path`	`/`
`Domain`	The application domain, so every console subdomain sees it

The domain-wide cookie is what makes one sign-in open every console somebody is entitled to. It does not decide what they may open: `panelOk` in `packages/services/src/identity.ts` does that, per request, per console. The user-content domain is a separate registrable domain, so uploaded content is out of cookie reach entirely regardless.

The VGSpartans Handbook

> **NOTE**
>
> A `Domain` attribute is what extends a cookie to subdomains, and it is only settable on a dotted host. A bare `localhost` build cannot carry one, so `platforms/core/src/lib/auth.ts` omits `Domain` there and mints a host-only cookie instead. That is why the dev stack serves consoles on `lvh.me` rather than `localhost`: it is a real dotted domain resolving to loopback, so one local sign-in shares a session across every local console exactly as production does. A stale host-only cookie left over from an older configuration is repaired in place by `/api/v1/session-rebind`, which broadens the domain of a cookie the caller already holds a valid session for and never creates or extends one.

Stored as a hash, not as itself

The cookie carries the raw token. The database stores SHA-256 of it, behind an `h_` marker, through a wrapper over the database adapter in `platforms/core/src/lib/session-hash-adapter.ts`. Every lookup hashes the incoming cookie value and matches the stored hash.

What that buys, stated exactly: a database dump on its own does not yield directly replayable session tokens. It does not make a dump useless to somebody who also holds `BETTER_AUTH_SECRET`, because the cookie is signed and the signature is verified before the token ever reaches a lookup, so a forged cookie needs the secret too. The point is that the two failure domains stay apart: the secret lives in the worker environment, the tokens live in the database. **Cryptography** covers the rest of what is stored as a hash rather than as itself.

HOW LONG A SESSION LASTS

The lifetime is set once, when the session is created, from the role the account holds at that moment. Higher privilege means a shorter window.

Account	Lifetime
Admin or developer	3 days
Security researcher with an open bounty enrolment	3 days
Club advisor	7 days
Newsroom editor (holding `news.publish`)	7 days
Gallery curator (holding `gallery.curate`)	7 days
Everybody else, including webmasters, students and teachers	14 days

That figure is a hard ceiling, not an idle timeout. Rolling refresh is switched off (`disableSessionRefresh`), which is load-bearing: without it, better-auth would rewrite the expiry from the global default on the first request and quietly promote a 3 day admin session to 14 days. The **OWASP Session Management Cheat Sheet** (https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html) asks for exactly this, an absolute timeout regardless of activity, so a hijacked session has a bounded life.

The VGSpartans Handbook

> **NOTE**
>
> A role change does not re-scope an existing session, because nothing rewrites the expiry after creation. That is why every privilege write drops the target's live sessions rather than editing them. The next request re-resolves the identity and mints a correctly scoped session.

BINDING A SESSION TO ITS DEVICE

Three layers, each with its own switch, each judged independently.

Layer	What it proves	Switch
Freshness cookie	This session collected a real device fingerprint recently	Always on for console APIs
Device anchor	The TLS and browser identity matches the one it enrolled on	`DEVICE_ANCHOR_MODE`
Device key	This browser holds the private key the session is bound to	`DPOP_MODE`

The two mode variables take `off`, `shadow` or `enforce`. Production runs both at `enforce`, and the production deploy refuses to run otherwise.

The freshness cookie

`fp_ok` is a signed, HttpOnly, host-only cookie carrying an expiry and the session's device anchor, minted by the fingerprint ingest endpoint whenever a collection carried enough real signal. It is valid for ten minutes.

> **WHY IT'S THIS WAY**
>
> A cookie rather than a key-value entry, for two reasons. It cannot be forged client-side, because it is an HMAC over the session id and expiry under the server secret, so a browser cannot fake "fingerprinting is on". And it is read-consistent immediately: a cookie set on the ingest response is present on the very next request, whereas the key-value store is eventually consistent and the lag would false-block a cooperating user.

The gate applies to console API calls only, never to the HTML shell, because a brand-new session has to be able to load the page that does the collecting. A call with no fresh cookie is answered **428**, a strike is recorded for observability, and the client renders it as a message asking the person to re-verify. Reads get no data and writes still face the step-up gate, so refusing the call is the enforcement.

The VGSpartans Handbook

> **DANGER**
>
> This gate must never sign anybody out. It did once, and the result was a production outage: an operator whose browser blocked the fingerprint was looped back through sign-in forever, while a stolen cookie that could not fingerprint was already getting nothing from the refusal. Keeping the session alive means allowlisting the site and reloading is enough to recover. If you see sign-out loops here, that is the bug this design exists to prevent.

Local development soft-enforces, with a warning, because Miniflare has no Cloudflare edge and the ingest often cannot reach the signal quality bar. That branch is compiled out of the production build.

The device anchor

A twelve-byte digest of JA4, Cloudflare's TLS ClientHello fingerprint, plus a version-stripped user agent. Every version number in the user agent is collapsed before hashing, so a routine browser update keeps the same anchor while a different browser family or operating system does not.

Enrolment is **write-once**. The first request of a session that carries a JA4 records that session's anchor, and a later request cannot overwrite it. Every freshness cookie then embeds the stored enrolled anchor rather than the caller's current one, so a cookie replayed from another machine mints a token still carrying the victim's anchor, and the gate, which recomputes the anchor from the live request, sees the mismatch.

Until a JA4-bearing request arrives, the token is minted unanchored and the device check is skipped for that session. This is trust-on-first-use, which is why the anchor is not the primary guard.

> **NOTE**
>
> The key-value store has no atomic create-if-absent, so two concurrent first requests can both see no record and each write one. The enrolment re-reads afterwards and prefers whatever is stored, so concurrent requests converge on one anchor. That narrows the window without closing it, which is acceptable only because the device key below is the independent primary guard.

The device key

The browser generates a **non-extractable** ECDSA P-256 keypair with Web Crypto, keeps it in IndexedDB where it is unusable outside its origin, and signs a short proof on every console request. The server binds the session to the public half and verifies each proof. This follows **RFC 9449** (https://www.rfc-editor.org/rfc/rfc9449.html): the proof carries `htm` (the method), `htu` (the path), `iat` (issued at) and `jti` (a per-proof nonce), so a captured proof cannot be reused on a different endpoint, and the window it can be reused at all is 2 minutes of clock skew.

Storage is per session **and** per host. Each console origin has its own IndexedDB and therefore its own keypair, while the session cookie is domain-wide.

Registration is trust-on-first-use, and it is the only unauthenticated bind. Re-registering the same key is a no-op. Registering a **different** key while one exists is a rotation, and a rotation needs a one-time

The VGSpartans Handbook

authorisation that is minted only by completing an emailed `device_key` step-up, and is burned on use.

> **WHY IT'S THIS WAY**
>
> Rotation deliberately does not accept the session's shared six-hour step-up immunity. That immunity is inherited by a stolen cookie, so keying rotation on it would let a thief riding an unrelated step-up swap in their own device key and take over the binding. Requiring a step-up raised for the rebind itself means the attacker needs the victim's inbox, and burning the grant on use means it authorises exactly one rotation.

Reads and mutations both bind to the device, and they differ only in how:

- **A mutation** runs the full gate. It spends the single-use `jti` write, and it fails closed when no device key is bound at all, so a write that out-races key registration steps up once rather than sailing past.
- **A read** skips the `jti` write, because replaying a captured read proof only re-reads data the capturer already saw, and a key-value write on every console poll is not worth it. RFC 9449 section 11.1 treats the replay store as an additional strengthening on top of the mandatory method, path and issued-at binding, and says outright that it "may not always be feasible in practice". A read also stays lenient while no key is bound yet, so a normal first console load does not spuriously step up.

There is one forgiveness rule, and it is narrower than it looks. A live step-up grant excuses an **anchor** mismatch, because a genuine browser update can rotate JA4 and the anchor is write-once, so without this a real device would loop. It never excuses a device-key failure, and it only applies while `DPOP_MODE` is `enforce`. If the key layer is off or in shadow, nothing else would catch the replayed cookie, so an inherited grant must not wave the mismatch through.

The VGSpartans Handbook

> ⚠ **WARNING**
>
> The ceiling of this whole scheme, stated plainly: it stops a cookie being exfiltrated and replayed from another machine. Two things get past it, and they are not the same thing.
>
> The first is code running inside the enrolling browser's own origin. It cannot export the key, but it can ask the key to sign, which is just as good. **RFC 9449** (https://www.rfc-editor.org/rfc/rfc9449.html) puts that case out of scope explicitly, and every browser key-binding scheme shares that ceiling, **DBSC** (https://developer.chrome.com/docs/web-platform/device-bound-session-credentials) included.
>
> The second is anything that can read the browser profile off disk, and here DBSC is genuinely stronger rather than equivalent. `extractable: false` is a guarantee about the Web Crypto API surface and nothing more: the **specification's security considerations** (https://w3c.github.io/webcrypto/#security-considerations) state that it carries no guarantee that "the underlying cryptographic key material will not be persisted to disk, possibly unencrypted", nor that it will be out of reach of anything running with the same privileges as the browser. So the key sits in the profile under whatever at-rest protection the browser gives it, and an attacker who lifts the whole profile rather than just the cookie lifts the key along with it. DBSC keeps the private half in a TPM or secure enclave, where taking the profile does not take the key. That is the gap between this and the platform feature, and it is the reason to keep watching DBSC rather than treating this as its equal.

THE DEVICES PANEL

Everybody can see the devices seen on their own account, on the personal dashboard under Devices. It is backed by `packages/services/src/devices.ts` and the `device.*` actions of the dashboard API.

A device is named for you from its platform and graphics renderer, something like "Windows · Intel UHD", until you rename it. Each row carries a trust state:

State	Meaning	Written by
Unconfirmed	Seen, not vouched for. The default, and where a revoke puts it back	Device ingest, and Revoke
Confirmed	You proved it was you, with an emailed code	The step-up confirm flow
Admin-trusted	Reserved for a device vouched for by an administrator	Nothing yet
Flagged	Reserved for a device marked suspicious	Nothing yet

The VGSpartans Handbook

> **NOTE**
>
> The last two are real states with no writer today. They are rendered, and the step-up gate already forces a re-authentication on a flagged device, but no code path currently sets either one, so in practice a device is Unconfirmed or Confirmed. Treat them as wiring that exists ahead of the feature rather than as something you can look for in the panel.

Three actions:

- **Rename**, up to 60 characters.
- **Confirm**, which goes through the step-up flow with an emailed code and is offered only on the device you are currently using. You cannot confirm a device you are not on.
- **Revoke**, which downgrades the trust state **and** deletes every session ever seen on that device. Downgrading trust alone would sign nothing out, leaving a potentially compromised device holding a live session, which is the opposite of what somebody clicking Revoke is asking for.

Collection is paced. The console shell collects at most once every 30 minutes per browser session, and console API responses can ask for a fresh collection with an `X-FP-Collect` header when the gate needs one. Signed-out visitors are never fingerprinted.

BEING TOLD ABOUT A NEW DEVICE

The first time an account is seen on a device that matches nothing already known, the account gets an email: the device name, the rough location, the time, and a link straight to the Devices panel so revoking it is one click rather than a hunt.

The mail is capped at three per account per day.

> **WHY IT'S THIS WAY**
>
> “New device” is decided by a fingerprint match, and a browser that randomizes its canvas and WebGL output matches nothing, so it mints a brand-new device profile on every single collection. Without a cap, that account would be mailed in a loop, which is a bad experience and, since we are the sender, an outbound mail amplifier. Three covers a genuinely busy day, phone plus laptop plus library machine, and goes quiet quickly for a randomizing browser.

Alongside it, every account gets a soft browser check that can add advisory notices to the response and can never refuse anything. It carries only two: a browser that has stopped updating (eight majors behind current Chrome Stable, and only for genuine Chrome, because a fork’s `Chrome/NNN` token names the Chromium it was built on rather than its own release), and a browser that looks driven rather than merely private.

The VGSpartans Handbook

> ⚠ **WARNING**
>
> Do not port the admin policy to members. It demands genuine Google Chrome, and a school runs on iPhones, Chromebooks, Android and personal Firefox installs. It also refuses a browser that blocks canvas readback, which is a legitimate privacy choice for a minor. Browser brand, fork detection, canvas and WebGL blocking, fingerprint randomization and Turnstile are all deliberately absent from the member tier, and every one of them would refuse a legitimate student for a choice they are entitled to make. The stricter tier is **Admin browser attestation**, and it applies to admins only.

WHAT ENDS A SESSION

Event	Scope
Signing out	That session
Revoking a device	Every session seen on that device
Adding, replacing or removing a sign-in factor	Every **other** session on the account
A role, privilege or advisor-seat change	Every session on the account
Suspending or deleting an account	Every session on the account
A bounty enrolment lapsing	Every session, then sign-in withholds the code
Reaching the lifetime above	That session
Failing the device fingerprint gate	Nothing. It answers 428 and keeps the session

A credential change keeps the acting session deliberately, so the person doing the work is not signed out of their own security settings mid-flow, while a parallel session on a shared machine or a stolen cookie is evicted.

A lapsed bounty enrolment is the one that has to be revoked rather than redirected. `/sign-in` forwards an already-valid session onward, so bouncing a still-live session to the front door would loop forever, and leaving it usable until its cookie happened to expire would mean “we closed your access” was untrue for up to three more days. OWASP is explicit that ending a session on the client while the server-side record stays valid is the classic mistake.

The session purges are best-effort by design, since the privilege write is the record of truth and must not roll back on a purge failure, but they are not silent: each retries once and then logs loudly with the user id, because sessions left live after a suspend is a security event rather than a shrug.

WHAT THIS DOES NOT PROTECT AGAINST

- **Malware inside the enrolling browser.** Covered above. The device key raises the cost of stealing a session to “compromise the actual machine”, and stops there.
- **A missing edge signal.** JA4 and the Cloudflare bot score come from Bot Management and **can be absent** (https://developers.cloudflare.com/bots/concepts/ja3-ja4-fingerprint/): on unencrypted traffic, when bot

The VGSpartans Handbook

management is skipped, or in local development. The anchor degrades to the coarse user agent and the device key carries the binding.

- **Shared networks.** OWASP notes that binding a session to client properties is a detection mechanism with real limits, since NAT and shared proxies flatten whole populations onto one address. That is exactly why nothing here binds to an IP address, and why the school's shared network is never treated as a signal. See also **Rate limiting**, which keys on a signed client id for the same reason.

When somebody who should have access cannot get in, the diagnosis path is **Locked out of a console**.

The VGSpartans Handbook

PART 4. SECURITY

CRYPTOGRAPHY

Which algorithm protects which secret, the parameters it runs at, where every key comes from, and why each choice went the way it did.

Explanation . For admins, developers . Checked 2026-07-26 . /documentation/security/cryptography

---

IN PLAIN TERMS

Almost nothing about your account is kept as itself. Your password is stored as a number that can be checked against what you type but never read back into a password. The secret behind your authenticator app is locked with a key that lives outside the database. Your session is stored as a fingerprint of the thing your browser holds, not as the thing itself. This page says which method protects which secret, so nobody has to reconstruct it from the code during an incident.

The VGSpartans Handbook

WHAT PROTECTS WHAT

Secret	How it is stored	Implemented in
Password second factor	scrypt at N 32768, r 8, p 3. 32 MiB of memory per hash, a 16-byte random salt, a 32-byte output	`packages/services/src/mfa.ts`
Sign-in one-time code	Hashed before it reaches the database, by better-auth's own `storeOTP: "hashed"`	`platforms/core/src/lib/auth.ts`
Recovery codes	HMAC-SHA-256 under the app signing secret, with a label mixed into the message. Single use	`packages/services/src/mfa.ts`
Step-up confirmation codes	HMAC-SHA-256 under the app signing secret, with the challenge id mixed into the message so it acts as a salt	`packages/services/src/stepup.ts`
Authenticator app shared secret	AES-256-GCM under a 96-bit random nonce. The key is derived with HKDF-SHA-256 from the app signing secret	`packages/services/src/mfa.ts`
Passkey and security key	Only a public key, a credential id and a counter. The private key never leaves the device	`platforms/core/src/lib/mfa-webauthn.ts`
Session token	SHA-256 of the token the cookie carries, stored behind an `h_` marker. The cookie keeps the raw value	`platforms/core/src/lib/session-hash-adapter.ts`
Device binding proof	ECDSA on P-256 with SHA-256. The browser holds a non-extractable private key it cannot export	`packages/services/src/dpop.ts`, `packages/ui/src/console/dpop.ts`
IP, JA4, user agent and location in the forensic trail	HMAC-SHA-256 under a separate pepper, truncated, carrying a key version	`packages/services/src/detail-privacy.ts`
Signed cookies and tokens	HMAC-SHA-256 under the app signing secret, one derived key per purpose	`packages/services/src/secret.ts`

Everything in that table is verified by comparing against a value the server already holds. Those comparisons never use `===`. They go through `timingSafeEqualStr` in `packages/services/src/secret.ts`, which folds a length mismatch into the accumulator instead of returning early, so neither the length nor the position of the first difference leaks through how long the check took.

PASSWORDS

A password is a second factor here, never a first one. It is only ever offered after the emailed one-time code has already been accepted. That does not make its storage less important: it is the one credential a member chooses themselves, so it is the one a database leak would otherwise be worth attacking.

The VGSpartans Handbook

It is also for ordinary members only. An account that carries elevated access — an administrator, a developer, an open bug bounty researcher — can neither set a password nor sign in with one, and a password found on such an account is removed. Those roles sign in with a passkey, a security key or an authenticator app.

> **WHY IT'S THIS WAY**
>
> Because a sign-in chooser is only as strong as its cheapest branch. Requiring a strong factor and then leaving a weaker one beside it means the requirement is met once, at enrolment, and the weaker credential is what opens the door every day after. The rule the requirement was written for is only true if the weaker branch is not there at all. **OWASP's guidance on multi-factor authentication** (https://cheatsheetseries.owasp.org/cheatsheets/Multifactor_Authentication_Cheat_Sheet.html) states it as: if an application provides multiple ways for a user to authenticate, they should all require multi-factor authentication or carry other protections.
>
> An ordinary member is in a different position and keeps the option. They carry no standing authority, they are under no strong-factor requirement, and for them a password is a real second factor on top of the emailed code rather than a way around a stronger one they were required to hold.

The parameters

packages/services/src/mfa.ts

```
const SCRYPT_N = 32_768; // 2^15 => 128 * N * r = 32 MiB per hash
const SCRYPT_R = 8;
const SCRYPT_P = 3;
```

Those are one of the profiles the **OWASP Password Storage Cheat Sheet** (https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html) names, and the point of them is the memory, not the time. Cost in memory is what a graphics card or an FPGA cannot copy thousands of times over, so an attacker holding a stolen database pays 32 MiB for every single guess rather than running tens of thousands of guesses in parallel out of a few hundred bytes each.

It runs natively, through `node:crypto`, which reaches the runtime's own implementation rather than a library compiled to JavaScript. That needs the `nodejs_compat` compatibility flag, which every application worker sets in its `wrangler.jsonc`. The gateway worker does not set it and does not need it: it never imports this module.

> **WHY IT'S THIS WAY**
>
> 32 MiB per hash, and not the 128 MiB row from the same OWASP table, because a Worker isolate gets 128 MiB in total and shares it with the whole rendering application. A single hash at the larger setting would exhaust the isolate. 32 MiB leaves three quarters of it alone and still costs an attacker 32 MiB per guess.

The VGSpartans Handbook

WHY IT'S THIS WAY

Not Argon2id, which OWASP ranks first, because it is not reachable on this runtime at a price worth paying. That was measured rather than assumed: scrypt at the settings above takes about 190 ms, while Argon2id at m 19 MiB, t 2, p 1 in pure JavaScript takes about 1421 ms, all of it synchronous, which blocks the whole isolate and turns a handful of concurrent sign-ins into a self-inflicted outage. The WebAssembly Argon2 builds are worse than slow, they simply do not run: they instantiate their module from embedded bytes at runtime, and the runtime refuses with “Wasm code generation disallowed by embedder”. A static `.wasm` import wired through nine worker builds is the only route that would work, for a marginal gain over a memory-hard function the runtime already provides natively. That same WebAssembly ban comes up again in **Bot protection**, although the binding constraint for the proof-of-work gate is the browser end rather than the server.

Reading a stored hash

The stored value is self-describing. Every parameter travels with the hash:

```
scrypt$32768$8$3$<salt>$<hash>
```

Two things follow from that. The parameters can be raised later without a migration, because an old hash still carries the settings it was made with. And any stored hash that is not at the current algorithm and parameters is re-derived in place, using the plaintext, on the owner’s next successful sign-in. That is the only moment the plaintext exists to re-derive from, which is exactly why OWASP prescribes this path rather than a backfill script. `passwordNeedsRehash` decides, and `verifyPasswordFactor` does the write.

The parameters that come out of the database are bounded before they are handed to the key derivation function. `N` has to be a power of two and `128 * N * r` has to fit under a 96 MiB ceiling. Without that, a single tampered row could ask the isolate for an allocation large enough to take the worker down.

Normalising before hashing

The password is put into Unicode Normalization Form C before any of this happens.

The character “é” can arrive as one code point or as two, “e” followed by a combining accent. They look identical on screen, nobody can tell which one their keyboard or their phone produced, and the two byte strings hash to completely different values. Without normalisation, somebody who sets a password on a phone and signs in from a laptop is locked out of their own account by a difference they cannot see.

NFC specifically, and not NFKC. **NIST SP 800-63B section 3.1.1.2** (https://pages.nist.gov/800-63-4/sp800-63b.html) asks for “the normalization process for stabilized strings using the Normalization Form Canonical Composition (NFC)”. The K forms are compatibility mappings: they fold “fi” into “fi”, full-width characters into ASCII, and a superscript “²” into “2”. In a document that is helpful. In a secret it quietly collapses distinct passwords onto the same hash and throws away entropy the member believed they had.

The VGSpartans Handbook

What a password has to pass first

Before anything is hashed, the candidate has to clear both halves of the policy. The synchronous half is `password()` in `packages/kernel/src/validate.ts`:

- 12 to 64 characters
- at least one letter and at least one digit
- no run of four or more characters going the same way, so `1234` and `aaaa` are out

The other half needs the network. `packages/services/src/password-breach.ts` screens the candidate against Have I Been Pwned's breached-password corpus, and it fails closed: if that service cannot be reached, the password is refused rather than accepted unchecked. An outage stops somebody adding a password factor, which they can retry in a minute or skip by enrolling an authenticator or a passkey instead. Letting the check lapse would admit a known-breached password as a standing credential that outlives the outage by years.

The corpus is only consulted for an account that may hold a password at all. A role that signs in with a passkey, a security key or an authenticator app is turned away before the candidate reaches either half of the policy, so nothing about it leaves the worker.

Verification accepts up to 200 characters, more than the 64 a new password may be set to. A correct long passphrase must never be turned away at the door, and the bound exists only so an unbounded request body can never be fed to the key derivation function.

Guessing it online

Offline cost is only half the problem. The other half is somebody guessing against the live sign-in form, where the derivation cost is the only thing slowing them down.

Eight consecutive failed second-factor verifications in a rolling fifteen minute window lock that account's second-factor step. A success clears the counter outright, so somebody who fumbles a code and then gets it right is never locked. The count lives in D1, incremented by one guarded `UPDATE`, because a read, add, write in application code loses increments under a concurrent burst, which is precisely the shape of the attack it is there to stop.

WHERE THE KEYS COME FROM

There are two independent secrets, and the split is deliberate.

`BETTER_AUTH_SECRET` is the application signing secret. It is never used raw for two different jobs. Each use derives its own key from it, either through the HKDF `info` label (the authenticator secret key) or by mixing a fixed label into the HMAC message (recovery codes, step-up codes, the proof-of-work signing key). One key per purpose means a weakness in one use cannot be carried across to another. `requireAuthSecret` in `packages/services/src/secret.ts` fails closed: in a production build a missing secret throws rather than falling back to the development constant published in this repository.

`DETAIL_PEPPER` keys the forensic-trail tokens and is a separate secret on purpose. Rotating the signing secret is routine and its blast radius is "everybody signs in again". Rotating the pepper silently re-keys every token, so rows written before the rotation stop correlating with rows written after it,

The VGSpartans Handbook

forever. Those two things must not share a rotation schedule. **Detail privacy** covers what that buys and what it costs, and **Secrets management** covers setting both.

THE TWO PLACES SHA-1 APPEARS

Neither is a security choice, and neither is protecting anything. Both are the wire format of a protocol we do not control.

Time-based codes. RFC 6238 computes a TOTP code with HMAC-SHA-1, and that is the default every authenticator app assumes when the key URI omits an algorithm. Changing it would break Microsoft Authenticator, Google Authenticator, Authy and 1Password for every member who already enrolled. It is also the case that the collision attacks that retired SHA-1 for certificates and signatures do not apply here, because HMAC rests on the hash behaving as a pseudorandom function rather than on collision resistance, which is why **NIST SP 800-131A** (https://csrc.nist.gov/pubs/sp/800/131/a/r2/final) still accepts HMAC-SHA-1 while rejecting SHA-1 signatures. NIST does intend to **transition away from SHA-1 for all applications by 31 December 2030** (https://csrc.nist.gov/news/2022/nist-transitioning-away-from-sha-1-for-all-apps), so this is worth revisiting when the authenticator apps move.

Breach screening. Have I Been Pwned keys its corpus by SHA-1 and its range API takes the first five hex characters of one. That is an index, not a protection. The password itself never leaves the worker, and the service learns a five-character prefix shared by tens of thousands of passwords, never which one it was or whose.

RAISING A PARAMETER LATER

Everything needed for this is already in place, so the change is small and the work is in confirming it, not in writing it.

1. Raise `SCRYPT_N` (or `r`, or `p`) in `packages/services/src/mfa.ts`. Keep `128 * N * r` well under the isolate's 128 MiB, and keep `SCRYPT_MAXMEM` above it.
2. Nothing else changes. New hashes carry the new numbers, existing hashes keep verifying under the numbers they were written with, and each one is re-derived at the new setting the next time its owner signs in successfully.
3. Check the cost. The tests in `platforms/core/test/mfa.test.ts` run in the real runtime, so a setting that is too slow shows up there before it reaches anybody's sign-in.

> ⚠ **WARNING**
>
> Never lower a parameter to make the tests faster. Every account that has already been re-derived at the higher setting keeps its stronger hash, so the only accounts a lower setting reaches are the ones that have not signed in yet. You would be spending the upgrade on nobody and weakening every new password from that point on.

The VGSpartans Handbook

PART 4. SECURITY

ADMIN BROWSER ATTESTATION

The browser check in front of the highest-privilege consoles, what each of its four tests actually looks at, and the levers that stop it locking the team out.

Explanation . For admins, developers . Checked 2026-07-26 . /documentation/security/admin-attestation

IN PLAIN TERMS

An admin account can change anything on the platform, so it is held to a stricter standard than everybody else, and part of that standard is the browser it is used from. After an admin signs in, one extra screen appears before anything else loads. It runs four checks, narrates them as they go, and either lets the person through or tells them exactly what to change. It is never a sign-out and never a dead end.

WHO IT APPLIES TO

Admin and developer identities only. Every other account on the platform, which is nearly all of them, is untouched by this and keeps the ordinary device checks described in **Sessions and devices**.

It gates **all** authenticated navigation for those accounts, not only the console paths. The console is one unified surface spanning every platform, so locking `/admin` and `/developer` while leaving `/hub`, `/settings` and the rest of the signed-in site open would be a door with no wall around it.

WHERE IT SITS IN THE SIGN-IN SEQUENCE

```
sign-in  ->  /console-check  ->  /mfa  ->  wherever they were headed
```

The browser check runs **before** the second factor, and that order is deliberate.

WHY IT'S THIS WAY

Putting the browser first means a one-time code or a passkey tap only ever happens inside a browser that has already been verified. Entering a second factor into a tampered, randomizing or automated browser is precisely the thing worth avoiding, and the reverse order does exactly that.

Clearing the browser check early costs nothing, because the session it writes its grant against is still worthless without MFA. An attacker holding only inbox access is stopped one step later, having proven only that their own browser is genuine.

The two exempt lists in `platforms/core/src/lib/guard-chain.ts` encode this order. The attestation list holds only `/console-check`, `/api/v1/attest` and the shared plumbing; the MFA list is a superset of it. `/mfa` and `/settings/security` are deliberately absent from the attestation list, which is what puts the browser first. Flipping that back would silently restore the old order.

The VGSpartans Handbook

THE QUESTION IT ASKS

> *Is this an unmodified, current Google Chrome that is not hiding from us?*

The policy is a **positive allowlist** for genuine Chrome. It is not a blocklist of forks, and it must never become one.

WHY IT'S THIS WAY

Hardened Chromium forks (Brave, Cromite, Bromite, ungoogled-chromium) freeze or spoof their version and strip Client Hints. Enumerating them by name is whack-a-mole against an unbounded list, and a frozen version string means a name that works today identifies nothing next year.

Requiring the full genuine-Chrome signature inverts that. A fork that strips a signal in order to hide fails the check by the absence of the signal. Adding fork names to a blocklist would be strictly weaker than the rule already in place.

THE SCREEN

`/console-check` renders five rows and fills them in as the check runs. Four are the policy tests; the fifth is the round trip to the server that decides.

Row	Passes when
Secure browser	The browser advertises the genuine `Google Chrome` brand
Browser version	Its major release is inside the supported window
Browser integrity	Nothing is blocking, randomizing, patching or driving the browser
Human verification	A Cloudflare Turnstile challenge passed
Device handshake	The server accepted the whole verdict and wrote the grant

The check is visible rather than a silent redirect on purpose. Somebody who gets refused has to understand why and what to change, so the page narrates each step and, on a refusal, lists every failed check with a concrete fix beside it.

The page decides nothing. It collects signals with the same shared collector every other device check uses, posts them to `/api/v1/attest`, and renders what comes back. The policy itself is `evaluateBrowser` in `packages/services/src/browser-attest.ts`, and the guard chain and the endpoint both call it, so “what counts as an acceptable admin browser” has exactly one definition.

Secure browser

Genuine Chrome always advertises the `Google Chrome` brand in its **User-Agent Client Hints** (https://wicg.github.io/ua-client-hints/). The check refuses, in order: Brave (by its own `isBrave()` probe or the `Brave` brand), Microsoft Edge, Opera, and then anything that does not carry the `Google Chrome` brand at all.

The VGSpartans Handbook

That last case is the one doing the real work. A Chromium-only brand set, or no brands at all on a Chromium user agent because an extension stripped Client Hints, both land there. Hiding a signal is itself the failure.

Browser version

The window is the current Stable major, the previous one, and anything newer. The current major is read from **Google's version-history API** (https://versionhistory.googleapis.com/v1/chrome/platforms/win/channels/stable/versions), cached in the key-value store for twelve hours, with a 2.5 second timeout so Google's uptime is never on the critical path of an admin login.

Two properties of this check are load-bearing:

- **Newer always passes.** The rule is `major >= latest - 1`, not the exact pair. A browser that auto-updated ahead of our cached "latest" would otherwise be refused every single time Chrome shipped a release before our cache caught up.
- **It fails open.** If the version API is unreachable, the check falls back to a pinned constant rather than refusing everybody. An outdated but genuine Chrome is far less dangerous than a fork or a randomizer, and those are caught by the other three tests.

> **WHY IT'S THIS WAY**
>
> "Supported LTS or latest" cannot be expressed as a channel test, because the channel is not detectable from the client. Chrome's enterprise long-term option is the **Extended Stable channel** (https://support.google.com/chrome/a/answer/9027636), which carries the same version numbering as Stable and simply lags it. So the policy is a version window, and it has to be.

> **WARNING**
>
> Chrome moves to a two-week release cadence from 2026-09-08, roughly doubling the number of majors per year. A window of "current and previous" that felt comfortable at the four-week cadence is half as much wall-clock time afterwards. If this gate starts refusing people every fortnight, that is why, and the fix is to widen the accepted range in `packages/services/src/browser-attest.ts` rather than to weaken any other check.

Browser integrity

Five independent tests, any one of which refuses:

The VGSpartans Handbook

Signal	What it means
Canvas hashes to the empty digest, eight or more probes return null, or no WebGL renderer	The browser is refusing to produce identifying output at all
Canvas, audio or WebGL differ between two identical renders	A randomizer is altering the output, so this is not a device
Native functions wrapped, or navigator accessors patched onto the instance	Something is driving or rewriting the browser
The navigator claims contradict the `Sec-CH-UA*` headers the request actually sent	A spoofing extension
A Cloudflare bot score below 15, with no verified-bot flag	The edge scored the request as automation

The last row is skipped entirely when the edge has no opinion. An absent bot score means “no data”, not “suspicious”.

One observation is recorded and deliberately never blocks: reduced timer precision. Genuine Chrome clamps `performance.now()` to 100 microseconds without cross-origin isolation, so the probe false-positives on real Chrome. It rides along in the verdict’s `warnings` and refuses nobody.

Human verification

A Cloudflare Turnstile token, proving a real interactive browser environment rather than a scripted one. If the widget cannot load, the gate does not hang: a 20 second timeout settles the token as null, which fails this one check and leaves the operator a retry button. The same Turnstile primitive is described in **Bot protection**.

WHAT THE CLIENT CANNOT LIE ABOUT

Every claim the browser makes is corroborated against a signal it cannot forge from JavaScript:

- **JA4**, Cloudflare’s TLS ClientHello fingerprint, read server-side off `request.cf`. A fork cannot reproduce a genuine Chrome handshake.
- **The Cloudflare bot score**, from the same place.
- **The `Sec-CH-UA*` request headers**, cross-checked against the navigator’s own claims by `deriveUaConsistent`, which is shared with the ordinary fingerprint ingest so there is one rule rather than two.
- **The Turnstile verdict**, which is checked against Cloudflare, not asserted by the page.

> **NOTE**
>
> JA4 and the bot score come from Cloudflare Bot Management and **can be absent** (https://developers.cloudflare.com/bots/concepts/ja3-ja4-fingerprint/), on a plan without it, on unencrypted traffic, or in local development. The checks that depend on them are skipped when they are missing rather than guessed at, so the gate degrades to the client-side tests plus Turnstile instead of refusing everybody.

The VGSpartans Handbook

WHEN IT REFUSES

Nothing is written. The session simply stays in the “required” state, the page keeps showing why, and the operator fixes their browser and presses “Check again”.

Every refusal carries a title and an actionable fix, and every independent failure is reported at once rather than one at a time, so somebody with two problems does not discover the second one after fixing the first.

> ⚠ WARNING
>
> This gate never signs anybody out. A refusal is a recoverable wall, not a lockout: the session stays alive, `/sign-in` stays reachable, and switching to a supported browser is enough to get through. If you ever see it revoking sessions, that is a bug, not the design.

THE GRANT

A pass writes `attest:ok:<sessionId>` into the key-value store, and the guard chain reads it on subsequent requests.

- **Server-authoritative.** A cookie cannot forge a key-value entry. It mirrors how the MFA grant works.
- **Keyed to the session**, which is never reused, so an orphaned grant after a revoke is harmless.
- **Three day lifetime**, the admin session ceiling, so the grant dies with the session and a fresh sign-in always re-attests.

Cost matters here, because this lookup runs on every authenticated admin request. The grant is checked first, since it is the steady-state hit; the break-glass lookup only runs when the grant is missing. An attested admin never pays for both. The gate also exits before any I/O for every non-privileged identity, and whenever the mode is off, which is almost all traffic.

TURNING IT ON

One variable, `ADMIN_ATTEST_MODE`, mirroring the other rollout switches:

Value	Behaviour
unset (default)	Off. The gate exits immediately and evaluates nothing.
`shadow`	Evaluates and logs what it would have refused, blocks nothing.
`enforce`	Bounces a page to `/console-check`, answers an API with `{ require_attest }`.

The value is carried as a repository variable and pushed to the `vgcore` runtime by the deploy’s secret sync, alongside `DPOP_MODE` and `DEVICE_ANCHOR_MODE`.

A production deploy **fails** unless it is exactly `enforce`. The check is “Require admin browser attestation enforce (production)” in `.github/workflows/ci.yml`.

The VGSpartans Handbook

> **WHY IT'S THIS WAY**
>
> This gate shipped silently disabled once. The policy, the `/console-check` page and both guard-chain call sites were complete, but no environment ever set the switch, so the gate returned on its first line every time. An unset variable reads as "off", and a protection that regresses to off by omission is worse than one that was never built, because everything about it looks present.
>
> That is why the deploy refuses rather than warns, and why a deliberate shadow rollout means relaxing that one reviewed check rather than quietly unsetting the variable. Running `shadow` first and reading the logs is still the right way to find out whether a real admin's real browser trips something, but it is a change somebody has to approve.

BREAK-GLASS

Refusing privacy browsers for admins genuinely can lock a real admin out, and the fingerprint gate did exactly that in production once, so there are two levers that work without a deploy.

1. `ADMIN_ATTEST_BREAKGLASS`, a comma-separated list of email addresses that bypass the gate.
2. **The global kill-switch**, a single key-value write:

```
wrangler kv key put --binding=CACHE attest:breakglass:all 1   # arm
wrangler kv key delete --binding=CACHE attest:breakglass:all  # disarm
```

Both are checked **before** any evaluation runs, because the whole point of the lever is to work when the evaluation itself is what is locking somebody out. The global switch logs loudly on every single use, so it can never sit armed and forgotten.

> **WARNING**
>
> The global kill-switch turns the gate off for every admin at once. It is an incident tool, not a configuration setting. If it is armed, arming it is now a thing that has to be undone, and the log line is the only reminder anybody gets.

IN DEVELOPMENT

Local development has no Cloudflare edge, so there is no JA4 and no bot score, Miniflare serves over plain HTTP, and Turnstile is off without a secret. A real verdict would refuse every local admin and make the consoles unopenable, so the endpoint soft-passes in development, warns with the reasons production would have refused on, and still returns the full verdict so it stays inspectable. That branch compiles out of the production build entirely.

WHAT IT DELIBERATELY IS NOT

- **Not a fork blocklist.** Covered above, and the single most likely thing for a well-meaning change to break.

The VGSpartans Handbook

- **Not the member policy.** The soft check every other account gets is almost the opposite rule and never blocks anything. Porting this one to students would be an outage rather than a hardening: a school runs on iPhones, Chromebooks, Android and personal Firefox installs, and refusing a browser that blocks canvas readback would refuse a legitimate privacy choice made by a minor. See **Sessions and devices**.
- **Not device-bound session credentials.** DBSC (https://developer.chrome.com/docs/web-platform/device-bound-session-credentials) is the stronger primitive here, hardware-backed by the TPM or the Secure Enclave, and it would exclude non-Chrome by construction. It is a server-driven protocol and reached general availability only on Chrome 146 for Windows, so it is deliberately left as a layer to add on top of this gate rather than a replacement for it. The existing per-session device key described in **Sessions and devices** is the interim binding.

CHECKING A CHANGE TO THE POLICY

`evaluateBrowser` is pure, with no I/O, so the whole rule is unit-testable without a session or an edge. The tests in `platforms/core/test/browser-attest.test.ts` pin it, including the property that matters most: every test that expects a refusal does so because a signal is absent or contradictory, never because a fork was named. That is what makes the policy hold against forks that have not shipped yet.

If the gate is refusing somebody it should not, the diagnosis path is **Locked out of a console**.

The VGSpartans Handbook

PART 4. SECURITY

BOT PROTECTION

The human checks a public form runs before it accepts a submission, how they stack, and how each one fails.

Explanation . For admins, developers . Checked 2026-07-26 . /documentation/security/bot-protection

IN PLAIN TERMS

Anyone on the internet can reach the sign-in form and the report form without an account, which means a script can too. Two independent checks sit in front of those forms: one that watches how the browser behaves, and one that makes the browser do a small piece of pointless arithmetic before it is allowed to submit. Both are cheap for a person and expensive at the scale a bot works at. Either can be switched off, and in production at least one has to be on.

WHICH FORMS ARE GATED

The public, pre-session forms:

- sign-in and sign-up, including the one-time-code send and verify behind them
- the abuse report form
- club sign-up

THE TWO GATES

They stack. A request has to clear every gate that is switched on.

Gate	What it is	Switch
Cloudflare Turnstile	An interactive and behavioural check, verified against Cloudflare	`TURNSTILE_SECRET_KEY`, plus the build-time `PUBLIC_TURNSTILE_SITE_KEY`
ALTCHA proof-of-work	An invisible per-submit CPU cost the browser pays, verified locally	`ALTCHA_MODE=enforce`, plus the build-time `PUBLIC_ALTCHA=1`

Neither, either, or both may be on:

- **Both.** Turnstile and a solved proof-of-work are required. Defence in depth: a universal CPU cost even on traffic that clears Turnstile.
- **Turnstile only.** The hosted default.
- **ALTCHA only.** A self-hosted deployment with no Cloudflare account. Turnstile cannot run there, but the forms are still gated.
- **Neither.** Open in development, refused in production. A public write with no human check in production is a misconfiguration, not a valid state.

The VGSpartans Handbook

There is a single decision point, `verifyHuman` in `packages/services/src/human-check.ts`. Every endpoint calls it, so the precedence lives in one place and no endpoint can quietly implement its own.

THE PROOF-OF-WORK

The official ALTCHA libraries, on the v1 SHA-256 wire format.

- **Server.** `altcha-lib` (`altcha-lib/v1`), its `createChallenge` and `verifySolution`, wrapped in `packages/services/src/altcha.ts`. It is pure `crypto.subtle`, so it runs natively on the edge runtime.
- **Client.** The official `altcha` widget v3 in `display="invisible"` mode, driven programmatically by `packages/ui/src/console/altcha-solver.ts` so it fits the app's fetch-based forms. It bundles its own solver, makes no external call beyond the same-origin challenge fetch, and renders nothing.

> **WHY IT'S THIS WAY**
>
> v1 SHA-256, not v2's memory-bound algorithms, because a proof-of-work challenge has to run the same function at both ends, and the browser end is the one that cannot. `altcha-lib` ships browser implementations for only two of its algorithms, and neither of them is memory-hard. Choosing scrypt or Argon2id would mean shipping a WebAssembly build of it to every visitor before they are allowed to submit a public form.
>
> Argon2id does not run on the server either. `altcha-lib` derives it through Node's `crypto.argon2`, which the Workers runtime does not implement. Native Argon2 packages are `.node` add-ons Workers do not support, and the WebAssembly fallback is blocked by the runtime's ban on compiling WebAssembly at runtime ("Wasm code generation disallowed by embedder").
>
> The server side is not the constraint. `node:crypto` carries native scrypt, and that is what protects the password second factor (see **Cryptography**). It is a server-side key derivation function though, so it gives the widget nothing to solve. SHA-256 runs on Web Crypto at both ends with no WebAssembly at all, which is what this gate actually needs.
>
> ("v3" is the widget's version. The memory-bound protection it advertises is the v2 challenge format. Invisible mode is available on either.)

The flow

1. On submit, right after Turnstile hands over its token, the client requests a signed challenge from `/api/v1/altcha` (`platforms/core/src/pages/api/v1/altcha.ts` and `platforms/vgclubs/src/pages/api/v1/altcha.ts`).
2. The invisible widget brute-forces the number `n` where `sha256hex(salt + n)` equals the challenge, in its own web worker. Nothing renders. The only visible effect is the submit button's busy state while the deliberately light proof-of-work solves.
3. It posts the base64 solution in the `altcha` field, or the `x-altcha-solution` header. That is a separate channel from the Turnstile token, so both travel together.

The VGSpartans Handbook

4. The server recomputes the hash and the signature, checks the salt's expiry, and claims the challenge once in the key-value store, so a solved token cannot be replayed.

Security posture

- **Fail closed.** Any parse, shape, algorithm, expiry or signature error is a refusal, never a thrown exception. A challenge that cannot be minted, for example because the auth secret is missing in production, returns 503, and the solver blocks the submit rather than posting nothing.
- **No new secret.** The signing key is derived from `BETTER_AUTH_SECRET` under a fixed label, so it is cryptographically separate from that secret's other uses and a self-hosting operator sets nothing extra. Leaking it lets an attacker forge proof-of-work solutions, which bypasses this gate, but never recovers the session secret.
- **Salt and expiry are bound to the signature.** The challenge is `hash(salt + number)` and the signature signs the challenge, so editing the salt to extend its expiry invalidates the hash check unless the attacker re-runs the work.

Difficulty

`maxnumber` is the top of the range the secret number is drawn from, so the client solves about `maxnumber / 2` hashes on average.

It is a policy value, not a constant: `platform.altcha_max_number` in `platform_config`, default 50000, which is a few tens of milliseconds on weak hardware. It is editable from the developer console's "Flags and config" tab and clamped to the range 1000 to 1000000 on read. See `packages/services/src/altcha-limits.ts`.

> ⚠ **WARNING**
>
> Keep it light. Proof-of-work is regressive on slow hardware. A bot farm runs faster cores than the Chromebooks this is actually used on, so raising the difficulty punishes real low-end visitors more than it punishes attackers. It is a volume speed-bump, never a wall.

TURNING IT ON

Set both halves or neither, the same way the Turnstile pair works. The server enforcing while the client does not solve would refuse every submit.

- Worker variable: `ALTCHA_MODE=enforce`
- Build and client variable: `PUBLIC_ALTCHA=1`, in `.env` or the shell environment, not `.dev.vars`, because it is read at build time.

Difficulty stays at its default unless it is changed in the developer console.

On the hosted stack

Both are repository Variables, not secrets. Neither carries anything sensitive: a mode string and a public build flag. They live in Settings, Secrets and variables, Actions, Variables, in the same place as `DPOP_MODE`.

The VGSpartans Handbook

- `ALTCHA_MODE` is synced to the `vgcore` and `vgclubs` workers. Both run `verifyHuman` locally, because the club sign-up form is not proxied through `CoreRPC`. The deploy writes an explicit `off` when the variable is cleared, so turning the gate off really disables it instead of stranding a stale `enforce`. The secret sync skips blank values, which would otherwise leave every form refusing submissions.
- `PUBLIC_ALTCHA` is read by the core build step and the vgclubs build step.

The production deploy fails closed on a half-configured pair. The “Require ALTCHA switches consistent” gate in `.github/workflows/ci.yml` rejects `ALTCHA_MODE=enforce` without `PUBLIC_ALTCHA=1` (every form would be refused) and `PUBLIC_ALTCHA=1` without `ALTCHA_MODE=enforce` (the client would solve a proof the server never checks). It does not force the gate on. Off is a valid state.

Preview stays Turnstile-only, matching the other mode flags.

The VGSpartans Handbook

PART 4. SECURITY

RATE LIMITING

What is limited, by how much, and what a limited request sees.

Reference . For admins, developers . Checked 2026-07-25 . Unfinished . /documentation/security/rate-limiting

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover every limit and the reasoning behind its shape.

The VGSpartans Handbook

PART 4. SECURITY

DETAIL PRIVACY

Why the platform stores one-way tokens instead of raw addresses and devices for one account, and exactly what that buys.

Explanation . For admins, developers . Checked 2026-07-25 . /documentation/security/detail-privacy

IN PLAIN TERMS

To spot a stolen account, the platform records details about the connection behind every sign-in: the network address, the browser, roughly where in the world it was. For most accounts those are stored as they are. For the developer's own account they are replaced with a scrambled code that cannot be turned back, because that one account's records would otherwise amount to a home address and a device list for the person who can change anything on the platform.

WHAT IS REPLACED

The forensic pipeline logs directly identifying material about whoever is using the platform: the raw client IP, the JA4 TLS fingerprint, the User-Agent, the Cloudflare ray id, and Cloudflare's city, region, latitude and longitude.

Those columns are readable by anything that can read the database. A leaked database export. A compromised admin console. A subpoena. A careless query.

For the developer account, each of those values stops being stored as itself. It is replaced with a keyed one-way token: HMAC-SHA-256 under a dedicated secret, truncated to 160 bits, prefixed with a key version.

```
203.0.113.44  ->  v1.NeeCvjCab4ReNskthrYC9Hsh3Jw
```

The implementation is `packages/services/src/detail-privacy.ts`.

WHAT STILL WORKS

The token is deterministic, so everything the investigation console actually does with these columns is unaffected.

- **Correlate.** "Same token" still means "same address", across users and across time. Cross-user correlation, the concurrent-session view and the repeat-address detectors all keep working.
- **Confirm.** An investigator holding a suspect value can still ask "did this account come from 1.2.3.4?" by tokenising the candidate and comparing, with `detailMatches`.

What stops working, deliberately, is reading an identifier back out of the database. There is no decrypt.

The VGSpartans Handbook

> **WHY IT'S THIS WAY**
>
> That is why this is a keyed hash and not reversible encryption. An encryption key sitting in the worker environment can be used by anything that reaches the worker environment, so "encrypted at rest" would still hand the plaintext to whoever compromises the console. A one-way token cannot be turned back into an address by anyone, including us.

WHY KEYED, AND NOT JUST HASHED

This is the part that is easy to get wrong, and it is why the old `ip_hash` column was not good enough.

IPv4 is a 32-bit space. An unkeyed SHA-256 over all four billion addresses is a few minutes of GPU time, so a plain digest of an address is a reversible encoding, not a pseudonym. `fingerprint_snapshots.ip_hash` held exactly that.

The secret pepper is what makes the search infeasible, which is also why it has to live in worker secrets and never in the same database as the tokens.

FIELD POLICY

Field	Treatment	Why
`ip_address`	Tokenised into `ip_token`, column set to empty	Only ever compared for equality
`ja4_fingerprint`, `user_agent`, `cf_ray`	Tokenised	Identify a device or connection; equality only
`city`, `region`	Dropped	They name a place, and nothing computes on them
`latitude`, `longitude`	Coarsened to 2 decimal places	See below
`asn`, `asn_org`, `country`, `colo`, the VPN, Tor and hosting flags, TLS version and cipher, HTTP protocol, client hints, `Accept-*`, bot score	Kept as they are	They describe a network, not a household, and the anomaly detectors need them

The coordinates are the one case where the obvious privacy move is wrong. The impossible-travel detector in `anomaly.ts` reads latitude and longitude off previous snapshots to compute a distance, so clearing them would silently disable that detector for exactly the accounts it protects most. That is a large security regression for a small privacy gain.

They are rounded instead, and the numbers make it free. The detector fires at 500 km, and 2 decimal places is about 1.1 km of error per point, about 2.2 km on a pair, which is under half a percent of the threshold. Cloudflare reports 5 decimals, about a metre, which is spurious precision: Cloudflare's own documentation puts city-level accuracy at 50 to 80 percent with no street-level resolution by design. Rounding discards digits that never carried information but did make a stored row look like a household coordinate.

The VGSpartans Handbook

SCOPE

Protection applies to the `DEVELOPER_EMAIL` allowlist and nothing else, the same repository secret that gates the developer console. Two exclusions are deliberate and load-bearing.

Admins are not protected. An admin is a member of staff operating on other people's accounts, and abuse triage needs a readable address for them the same way it does for anyone else. The developer is a different case: one person, who runs the programme, whose home address would otherwise sit in a column every admin console query can reach.

The profile `role_key` is not consulted. `role_key` is mutable application data, seated from the admin console. Keying a privacy control on it would mean anyone who can grant a role can also decide whose address stops being logged, including their own. The allowlist is deployment configuration: changing it takes a repository secret and a deploy, which is a higher and audited bar.

That is why the check reads the allowlist directly instead of calling `privilegeClass()`, which deliberately folds in the role and the admin list. That is the right answer for authorization and the wrong one here.

`DETAIL_PROTECT_SCOPE`, a plain variable rather than a secret, can widen it:

- `developer`, the default. The `DEVELOPER_EMAIL` allowlist only.
- `all`. Every account. Nothing in the pipeline requires a readable address, so this is supported, and it is the privacy-maximal setting.
- `off`. A rollback lever. Store everything raw.

Widening the scope is a deploy, not a refactor. `ip_token` is written for every row regardless of scope, so a protected and an unprotected sighting from the same address produce the same token and cross-user correlation never goes half-blind.

Because the allowlist is the whole policy, every worker needs `DEVELOPER_EMAIL`. All eight platforms that serve `/api/v1/context` run the ingest: `vgcore`, `vgsites`, `vgclubs`, `vgwiki`, `vgwrites`, `vgsynergy`, `vgagora` and `vgnews`. CI syncs it alongside the pepper. This is not a privilege grant on the subplatforms: nothing there authorizes off that variable, it only selects raw-versus-tokenised storage.

OPERATIONS

Generate and set the pepper. It is required: the production deploy refuses to run without it.

```
openssl rand -hex 32
wrangler secret put DETAIL_PEPPER
```

Every worker needs it, not just `vgcore`, for the same reason as the allowlist above. CI syncs it to all of them.

Fail-closed direction. With no pepper configured, the pipeline redacts rather than falling back to storing raw values. For a privacy control, a missing secret has to degrade what is kept, not what is

The VGSpartans Handbook

protected. It deliberately does not throw the way `requireAuthSecret` does: throwing would take down fingerprint ingest, and an outage of the device-recognition pipeline is a worse failure than a gap in the forensic detail. The deploy gate is where that mistake gets caught instead.

Locally, leaving it unset is fine. Development falls back to a fixed insecure pepper, the same pattern `BETTER_AUTH_SECRET` uses, so tokens stay stable across restarts. That fallback is compiled out of production builds.

Rotation

> ⚠ **WARNING**
>
> Rotating `DETAIL_PEPPER` re-keys every future token, so rows written before the rotation stop correlating with rows written after, permanently. Bump `KEY_VERSION` in `packages/services/src/detail-privacy.ts` in the same change, so a reader can tell which era a token belongs to instead of silently drawing a wrong conclusion from a mismatch.

This is why the pepper is a separate secret from `BETTER_AUTH_SECRET`. Rotating the auth secret is routine and low-consequence: everyone signs in again. These two must not share a rotation schedule.

RESIDUAL RISK

Pseudonymization is not anonymization, and it is worth being precise about what this does and does not buy.

- **Anyone who obtains the pepper can brute-force the tokens back to plaintext**, because the input spaces are small. IPv4 is 2^32; User-Agent strings are drawn from a short list. The security of the whole scheme rests on the pepper's secrecy, not on the hash. Treat a pepper disclosure as equivalent to disclosing the raw columns.
- **Pseudonymized data is still personal data** under GDPR and CCPA. This reduces risk and demonstrates data protection by design. It does not take these rows out of scope.
- **Tokens are stable and global**, so they remain a linkable identifier by construction. That is the feature, and it is also the limit.

KNOWN GAPS

- **Historical `ip_hash` values are still brute-forceable.** New rows write an empty string, but rows written before this change still hold the unkeyed SHA-256. Clearing them is a destructive call on existing forensic history, so it is left as a deliberate decision rather than done silently:

  ```
  wrangler d1 execute vgspartans-audit --remote \
    --command "UPDATE fingerprint_snapshots SET ip_hash = '' WHERE ip_hash != '';"
  ```

  There is no backfill of `ip_token` for those rows. The raw address was never stored in a recoverable form for protected accounts, and for unprotected ones `ip_address` is still present if you want to derive tokens later.

The VGSpartans Handbook

- **`fingerprint_snapshots` is at 99 of the database's 100-column limit.** The next column added to that table has to fold into an existing JSON column, as `ch_ua_extra` already did, or the migration will fail.
- **`device_profiles.ua_baseline` and `ch_ua_baseline` still hold raw User-Agent strings.** They feed the matcher, so tokenising them needs the same treatment on both sides of the comparison. Deferred rather than done carelessly.

The VGSpartans Handbook

PART 4. SECURITY

AUDIT AND ANOMALY DETECTION

What gets recorded, and what the platform does when a pattern looks wrong.

Explanation . For admins, developers . Checked 2026-07-25 . Unfinished . /documentation/security/audit-and-anomaly

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the audit trail and the anomaly pipeline.

The VGSpartans Handbook

PART 4. SECURITY

CONTENT SAFETY

The automated checks every piece of content passes through.

Explanation . For admins, developers . Checked 2026-07-25 . Unfinished . /documentation/security/content-safety

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the classifiers, the cascade, and the fail-open behaviour.

The VGSpartans Handbook

PART 4. SECURITY

EMAIL SECURITY

How login mail is sent, and what stops it being forged or lost.

Explanation . For admins, developers . Checked 2026-07-25 . Unfinished . /documentation/security/email-security

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the transport chain and the sender authentication.

The VGSpartans Handbook

PART 4. SECURITY

PRIVACY AND REGULATIONS

What the platform collects, why, and the rights it honours voluntarily.

Explanation . For admins, developers . Checked 2026-07-25 . Unfinished . /documentation/security/privacy-and-regulations

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the data inventory and the regional provisions.

The VGSpartans Handbook

PART 4. SECURITY

REPORTING A VULNERABILITY

How to report a security problem, and what happens after you do.

How-to . For admins, developers . Checked 2026-07-25 . Unfinished . /documentation/security/reporting-a-vulnerability

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the disclosure route and the bounty programme.

The VGSpartans Handbook

PART 5

ARCHITECTURE

How a request travels, how the workers split, and why each big decision went the way it did.

PART 5. ARCHITECTURE

ARCHITECTURE

One picture of how VGSpartans fits together, and an analogy for the shape of it.

Explanation . For developers . Checked 2026-07-25 . Unfinished . /documentation/architecture

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the whole-system diagram and the vocabulary the rest of the part uses.

The VGSpartans Handbook

PART 5. ARCHITECTURE

LIFE OF A REQUEST

Everything that happens between a click and a page appearing.

Explanation . For developers . Checked 2026-07-25 . Unfinished . /documentation/architecture/life-of-a-request

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the full request path, hop by hop.

The VGSpartans Handbook

PART 5. ARCHITECTURE

THE GATEWAY

The single edge worker that decides which platform answers a request.

Explanation . For developers . Checked 2026-07-25 . Unfinished . /documentation/architecture/the-gateway

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover host routing, the registry, and the edge gates.

The VGSpartans Handbook

PART 5. ARCHITECTURE

THE MULTI-WORKER SPLIT

Why VGSpartans runs as nine separate programs instead of one, how they talk to each other, and where identity lives.

Explanation . For developers . Checked 2026-07-25 . /documentation/architecture/the-multi-worker-split

IN PLAIN TERMS

VGSpartans is not one program. It is nine, each running a different part of the platform, with one small program at the front deciding which of the other eight should answer each visitor. That front program is the gateway. Think of a building with one receptionist and eight departments: the receptionist never does any of the work, they just know which door every visitor needs.

THE WORKERS

Every one of these is a separately deployed Cloudflare Worker, with its own folder under `platforms/`, its own configuration, and its own bindings.

Worker	Folder	What it owns
`vg-gateway`	`platforms/gateway`	All production routes. Decides which worker answers a request and forwards it. Renders almost nothing.
`vgcore`	`platforms/core`	The apex site, the account system, the admin and developer consoles, and the identity database.
`vgsites`	`platforms/vgsites`	The personal-site dashboard and the published personal sites.
`vgclubs`	`platforms/vgclubs`	The club officer consoles, the club site builder, and the public club sites.
`vgwiki`	`platforms/vgwiki`	VGWiki.
`vgwrites`	`platforms/vgwrites`	VGWrites.
`vgsynergy`	`platforms/vgsynergy`	VGSynergy.
`vgagora`	`platforms/vgagora`	VGAgora.
`vgnews`	`platforms/vgnews`	VGNews.

The gateway holds a service binding to each of the other eight, listed in `platforms/gateway/wrangler.jsonc`:

The VGSpartans Handbook

platforms/gateway/wrangler.jsonc

```
"services": [
  { "binding": "CORE", "service": "vgcore" },
  { "binding": "VGSITES", "service": "vgsites" },
  { "binding": "VGCLUBS", "service": "vgclubs" },
  { "binding": "VGWIKI", "service": "vgwiki" },
  { "binding": "VGWRITES", "service": "vgwrites" },
  { "binding": "VGSYNERGY", "service": "vgsynergy" },
  { "binding": "VGAGORA", "service": "vgagora" },
  { "binding": "VGNEWS", "service": "vgnews" }
]
```

WHY IT IS SPLIT AT ALL

> **WHY IT'S THIS WAY**
>
> A Cloudflare Worker has a hard size ceiling: 3 MB gzipped on the free plan, 10 MB on paid. When the whole product was one Astro app compiled into one Worker, every platform folded into it added its routes and its dependencies to the same bundle. A monolith that keeps absorbing platforms eventually stops deploying, and the failure arrives as a deploy error on the day you least want one.
>
> Splitting also makes each platform portable. A folder plus its own configuration can move to another repository or another Cloudflare account later with little rework.

WHY A GATEWAY RATHER THAN ROUTES PER WORKER

The obvious shortcut is to give each worker its own Cloudflare route patterns and skip the gateway. It breaks immediately.

`admin.vgspartans.org` and a club subdomain like `nhs.vgspartans.org` both match the same route glob, `*.vgspartans.org/*`. Cloudflare matches routes on URL pattern alone. It has no way to express "a known console subdomain, as opposed to a club that might exist", which is exactly the distinction the host router draws. Encoding that in route globs means reimplementing the router in a form that is harder to test and has to be edited every time a console is added.

A gateway keeps that decision in one place, written in code, unit tested. Worker to worker dispatch over a service binding runs in process on Cloudflare's side: no extra network hop, no added latency, and the callee's code does not count against the gateway's bundle.

THE SEAM THIS WAS BUILT ON

Two things already existed, which is what made the split tractable rather than a rewrite.

1. **Routing was already a pure decision.** A request's `Host` header is classified into a logical platform by the host router in `packages/kernel/src/hosts.ts`. It has no side effects, so it could be moved in front of everything without changing what it decides.

The VGSpartans Handbook

2. **Data was already partitioned.** Each bounded context had its own database and its own storage bucket before there was more than one worker. The database split was the blueprint for the worker split.

So most of the work was turning "rewrite the path and render it here" into "forward this to the worker that owns it". The routing logic barely changed. What changed is what executes the decision.

IDENTITY LIVES IN EXACTLY ONE PLACE

This is the part that matters most, and the part most likely to be broken by a well-meaning change.

`vgcore` owns the authentication library and the identity database. No other worker binds an identity database for writing, and no other worker bundles the auth library. Instead, every other worker reaches identity through a `WorkerEntrypoint` called `CoreRPC`, defined in `platforms/core/src/lib/rpc.ts`, over a `CORE` service binding.

That entrypoint is the whole surface. It covers session verification (`verifySession`, `verifyPanel`, `guardConsole`), the mutation and step-up path (`guardMutation`, `stepUpIssue`, `stepUpVerify`), device binding (`dpopRegister`, `deviceRevoke`), profile reads and writes, the moderation handoffs (`moderateText`, `moderateImage`, `moderateVideo`), content reports and support tickets.

> **WHY IT'S THIS WAY**
>
> The call is session-anchored on purpose. A subplatform passes the inbound cookie and gets back a decision; it never holds a credential of its own and can only ever act as the person whose cookie it forwarded. That is what keeps a bug in one small platform from becoming a way to act as anybody. The heaviest dependency on the platform then also lives in exactly one worker, and there is one source of truth for who you are.

The console guard is the same code for every worker: one guard chain in `packages/services`, called through `guardConsole`, so a subplatform console cannot drift from what core would have decided.

THE SHARED PACKAGES

Source-only packages, imported by the workers, so behaviour cannot fork between them.

Package	What it holds
`@vg/kernel`	Hosts and routing, security headers, the edge router, request validation, user-content helpers
`@vg/services`	Identity, step-up, devices, audit, rate limiting, fingerprinting, the Tor gate, storage
`@vg/db`	The database schema, the Drizzle client, and club content serving
`@vg/ui`	The design system, the layouts, the page chrome, and the console React kit
`@vg/editor`	The shared word editor several platforms embed

The VGSpartans Handbook

None of them has a build step. Their `exports` maps point straight at TypeScript, and each consumer's bundler compiles them. That resolves cleanly across Astro and Vite, the vitest workers pool, and `astro check`.

ONE REPOSITORY, MANY SCHOOLS

The longer-term goal this shape serves: run the same platform for another school, adding or omitting platforms per school, without editing the gateway or core.

A school is one configuration object under `schools/`, typed as `SchoolConfig`. It lists the school's domains, the platforms it has enabled, the hosts each one claims, and its branding. The gateway reads that registry to dispatch, so:

- **Omitting a platform for a school** is leaving it out of the config and not deploying its worker. Its hosts then fall through to `vgcore`.
- **Adding one** is a new `platforms/<id>/` worker, a registry entry, and a deploy.

The gateway's own code never changes for either.

Some files are generated from that registry rather than hand-maintained, and CI fails if they drift. The check is `node scripts/gen-school.mjs --check`, wired into the typecheck job in `.github/workflows/ci.yml`.

WHERE SHARED MACHINERY LIVES

- `LoginEmailWorkflow` and its binding live on `vgcore`.
- Each worker's `wrangler.jsonc` declares only the bindings its own routes touch.
- Each `migrations-*` directory is owned by whichever worker binds that database.

THINGS TO WATCH

- The gateway is a single point every request passes through. It is kept deliberately thin for that reason: it renders almost nothing, so there is little in it to break.
- The auth library's trusted origins and the domain-wide cookie have to keep covering every console origin across every worker. That list is centralised in `@vg/kernel` so it cannot fork.
- Deploying the clubs worker remotely without warning advisors first is against the club launch policy.

The VGSpartans Handbook

PART 5. ARCHITECTURE

THE MONOREPO

How the workspace is laid out and what belongs where.

Explanation . For developers . Checked 2026-07-25 . Unfinished . /documentation/architecture/the-monorepo

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover platforms/, packages/, schools/, scripts/ and the rules about each.

The VGSpartans Handbook

PART 5. ARCHITECTURE

DATA

Every database, bucket and cache the platform binds, what lives in each, and how they are kept in step.

Explanation . For developers . Checked 2026-07-25 . /documentation/architecture/data

IN PLAIN TERMS

The platform does not have one database. It has a dozen, each holding one kind of thing, plus a set of file stores and a couple of caches. Splitting them that way means a mistake in one part cannot corrupt another, and a platform can be added or removed without touching anyone else's data. This page is the map of what lives where.

THE THREE KINDS OF STORE

Cloudflare gives the platform three storage primitives, and each is used for one thing:

- **D1** is SQL. Anything the platform queries, joins or counts.
- **R2** is object storage. Files: uploaded images, published site content, database backups.
- **KV** is a distributed cache. Values that are read constantly and can be rebuilt if lost.

Every one of them reaches code as a binding, declared in that worker's `wrangler.jsonc` and read through `import { env } from "cloudflare:workers"`.

DATABASES

These are the bindings on `vgcore`, which owns the migrations and the provisioning for the whole school. Other workers bind the subset they need.

The VGSpartans Handbook

Binding	Database	What it holds
`USER_DB`	`vgspartans-user`	Identity: accounts, sessions, the auth library's own tables
`CORE_DB`	`vgspartans-core`	Platform-wide records: configuration, moderation, support, bounty
`CLUBS_DB`	`vgspartans-clubs`	Clubs, their officers, and their site content
`VGSITES_DB`	`vgspartans-vgsites`	Personal sites
`VGWRITES_DB`	`vgspartans-vgwrites`	VGWrites
`VGWIKI_DB`	`vgspartans-vgwiki`	VGWiki
`VGSYNERGY_DB`	`vgspartans-vgsynergy`	VGSynergy
`VGAGORA_DB`	`vgspartans-vgagora`	VGAgora
`VGNEWS_DB`	`vgspartans-vgnews`	VGNews
`AUDIT_DB`	`vgspartans-audit`	The audit trail's base database
`AUDIT_DB_1`, `AUDIT_DB_2`, `AUDIT_DB_3`	`vgspartans-audit-1`, `-2`, `-3`	Audit shards

WHY IT'S THIS WAY

Identity is in its own database, separate from everything else, and only `vgcore` writes to it. That is the same boundary the multi-worker split draws: one place that decides who you are, so a bug in a small platform can never become a way to become somebody else. Every other platform gets its own database for the same reason at a smaller scale, and the audit trail is sharded because it grows faster than anything else and its rows are never joined against application data.

The VGSpartans Handbook

BUCKETS

Binding	Bucket	What it holds
`STORAGE_PEOPLE`	`vgspartans-people`	Personal-site files and avatars
`STORAGE_CLUBS`	`vgspartans-clubs`	Club assets, and video masters under a `video/` prefix
`STORAGE_WIKI`	`vgspartans-vgwiki`	VGWiki uploads
`STORAGE_WRITES`	`vgspartans-vgwrites`	VGWrites uploads
`STORAGE_SYNERGY`	`vgspartans-vgsynergy`	VGSynergy uploads
`STORAGE_AGORA`	`vgspartans-vgagora`	VGAgora uploads
`STORAGE_NEWS`	`vgspartans-vgnews`	VGNews hero images and video masters
`STORAGE_EVIDENCE`	`vgspartans-evidence`	Preserved copies of content that drew moderation attention
`D1_BACKUP`	`vgspartans-d1-backup`	Database snapshots

`STORAGE_EVIDENCE` is deliberately its own bucket. Nothing public serves from it, its lifetime is the moderation record's, and mixing it into a content bucket would put removed material back inside a store that has public read paths.

CACHES

Binding	What it holds
`CACHE`	The `platform_config` cache and computed tiles
`SESSION`	Session storage, and short-lived one-time values such as claimed proof-of-work challenges

WORKFLOWS AND SCHEDULES

`vgcore` also binds four Cloudflare Workflows, which are the long-running jobs that must survive a request ending:

Binding	Workflow	What it does
`LOGIN_EMAIL_WF`	`vgspartans-login-email`	Sends a login code through the transport chain
`TEXT_MOD_WF`	`vgspartans-mod-text`	Runs the text moderation cascade
`IMAGE_MOD_WF`	`vgspartans-mod-image`	Runs the image moderation cascade
`VIDEO_MOD_WF`	`vgspartans-mod-video`	Runs the video moderation cascade

Two cron triggers run on `vgcore`: one daily at 07 UTC and one hourly.

The VGSpartans Handbook

HOW THE SHAPE OF A DATABASE CHANGES

Schema changes ship as numbered SQL files under a `migrations-*` directory, one directory per database, generated from the Drizzle schema in `packages/db` by `pnpm db:generate`. They are applied by `wrangler d1 migrations apply`, which records what it has run, so only new files execute.

> **DANGER**
>
> A migration file that has been applied anywhere is finished. Wrangler tracks applied migrations by FILENAME, so editing a shipped migration means the change never runs on any stack that already applied it, while a fresh stack gets a different schema. To change something a migration created, add a new migration.

The full procedure is in **Migrations**.

KEEPING LOCAL AND PRODUCTION THE SAME

Local development and CI apply the same migration files and the same seed through the same commands, so they stay in parity by construction. That parity is then checked two ways.

- `pnpm test:unit` runs Vitest inside the real edge runtime, using `@cloudflare/vitest-pool-workers`. It applies the real migrations and the real seed to a local simulator and asserts the endpoints against them.
- `pnpm db:verify` diffs the applied-migration set and the per-table row counts between local and remote, and exits non-zero on drift. It needs a seeded remote and a Cloudflare token.

PROVISIONING

Creating the resources is idempotent. `pnpm infra:provision` runs `platforms/core/scripts/cf-provision.mjs`, which lists the existing databases, buckets and namespaces, creates only what is missing, and writes the resolved ids back into `wrangler.jsonc`. Re-running it is a no-op.

The production deploy job runs the whole chain on every push:

```
infra:provision -> db:migrate:remote -> db:seed:remote -> db:verify -> wrangler deploy
```

First run creates everything. Later runs apply only new migrations and redeploy.

The API token it needs

Create a token in the Cloudflare dashboard under My Profile, API Tokens, Create Token, Custom, with these account-scoped permissions:

- Workers Scripts, Edit
- Workers KV Storage, Edit
- D1, Edit

The VGSpartans Handbook

- Workers R2 Storage, Edit
- Account Settings, Read

Add it to GitHub as the repository secrets `CLOUDFLARE_API_TOKEN` and `CLOUDFLARE_ACCOUNT_ID`. That is all CI needs; it provisions everything else.

LOCAL DEVELOPMENT

```
pnpm db:reset:local   # clear local state, migrate, and seed D1, R2 and KV
pnpm dev              # start the dev stack against those local bindings
```

The Cloudflare adapter runs the local simulator itself, so `import { env } from "cloudflare:workers"` resolves to local storage persisted under `.wrangler/state`. Authentication lives only in `vgcore`, and the subplatform dev servers reach it over the `CORE` service binding, so `vgcore` has to be running alongside them when exercising a gated console. `pnpm dev` boots it by default for that reason.

The VGSpartans Handbook

PART 5. ARCHITECTURE

AUTH INTERNALS

How identity actually works underneath the sign-in screen.

Explanation . For developers . Checked 2026-07-25 . Unfinished . /documentation/architecture/auth-internals

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the auth library, the adapters, and the cross-worker guard.

The VGSpartans Handbook

PART 5. ARCHITECTURE

RENDERING

What is built ahead of time, what is rendered per request, and why.

Explanation . For developers . Checked 2026-07-25 . Unfinished . /documentation/architecture/rendering

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the prerender split and the island model.

The VGSpartans Handbook

PART 5. ARCHITECTURE

THE DESIGN SYSTEM

The tokens, the stylesheets, and the rules that keep the platform looking like one thing.

Explanation . For developers . Checked 2026-07-25 . Unfinished . /documentation/architecture/the-design-system

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the token set, the sheet order, and the guardrails.

The VGSpartans Handbook

PART 5. ARCHITECTURE

EMAIL

How the platform sends mail and what happens when a provider fails.

Explanation . For developers . Checked 2026-07-25 . Unfinished . /documentation/architecture/email

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the transport chain and the templates.

The VGSpartans Handbook

PART 5. ARCHITECTURE

AI USAGE

Every place VGSpartans calls a model, what each call costs, and why the help bubble is proxied through our own worker.

Explanation . For developers, admins . Checked 2026-07-25 . /documentation/architecture/ai-usage

IN PLAIN TERMS

Two parts of VGSpartans use machine-learning models. One checks uploaded text, images and video for things that break the rules. The other is the little question bubble on the public pages, which reads what you type and points you at the right page. Both run on Cloudflare's models. The moderation one is by far the more expensive, and neither ever sees your account.

WHERE MODELS ARE CALLED

Where	What it does	Page
Content moderation	Classifies text, images and video against the rules	Moderation
The help bubble	Answers a visitor's question with links, or with prose	This page

WHAT IT COSTS

Measured off the Workers AI dashboard during setup, over one day of testing:

Model	Neurons	What it is
`@cf/meta/llama-guard-3-8b`	15320	Content moderation
`@cf/meta/llama-3.3-70b-instruct-fp8-fast`	406.62	Help bubble generation
`@cf/baai/bge-reranker-base`	74.67	Help bubble reranking
`@cf/qwen/qwen3-embedding-0.6b`	36.83	Help bubble embeddings
`@cf/meta/llama-3.1-8b-instruct-fast`	4.23	Help bubble query rewrite

The free allowance is 10000 neurons per day, at $0.011 per 1000 beyond it.

The VGSpartans Handbook

> ⚠ **WARNING**
>
> Moderation is about 97 percent of the platform's AI spend, and on that one day of testing it exceeded the entire free daily allowance by itself. When reading the Workers AI dashboard, use the PER-MODEL view: the "Text Generation" card combines models, so moderation's usage disappears into the same number as the help bubble's and the split becomes invisible.

THE HELP BUBBLE

A floating launcher on the marketing pages where a visitor can ask a question in their own words and get pointed at the right page. It is built on a Cloudflare AI Search instance, reached only through our own worker.

Piece	Where
Policy, search, reply formatting, chat proxy	`packages/services/src/ai-search.ts`
Config probe	`platforms/core/src/pages/api/v1/ai-search/index.ts`
Answering route	`platforms/core/src/pages/api/v1/ai-search/chat/completions.ts`
Widget mount, theme sync, shadow-DOM bridge	`packages/ui/src/public/HelpBubble.astro`
Styling, custom properties only	`packages/ui/src/styles/spartan/ai-chat.css`
Rate bucket	`packages/services/src/ratelimit.ts`

Turning it on

1. Enable the public endpoint on the instance: Cloudflare dashboard, AI Search, your instance, Settings, Public Endpoint. Copy the URL.
2. Set `AI_SEARCH_URL` to that URL. Locally that is `.dev.vars`. In production a plain variable is fine; it is not a credential.
3. In the admin console, Settings tab, "Help bubble" panel, turn on "Ask bubble on public pages".

Both halves are required. With no `AI_SEARCH_URL` the policy reads as off whatever the console says, so a deploy without an instance never renders a launcher that would fail.

The Settings panel carries the three knobs worth a control: the switch, the mode, and sources per answer. `platform.ai_search_max_chars` and `platform.ai_search_max_turns` have no dedicated control and are edited from the developer console's "Flags and config" raw editor, the same way `platform.altcha_max_number` is.

While you are in the dashboard, restrict the endpoint's CORS rules to your own origins and leave its rate limit in place. Neither is what protects us, but there is no reason to leave them wide.

The two modes

`platform.ai_search_mode` decides how a question gets answered. The browser cannot tell the difference: it posts the same body to the same route and reads the same event stream either way. Only the server changes.

The VGSpartans Handbook

`retrieval`**, the default.** Ask AI Search for matching chunks, remove duplicates by URL, and format the top few into a short reply of links. No generation model runs. The only cost is embedding the question, a fraction of a neuron.

`chat`**.** Hand the turn to AI Search's own completions endpoint and stream the answer through. Real prose. Cheap, but not free: roughly $10 to $13 a month at 100 conversations a day.

Switching is a console edit and a page refresh. Nothing needs rebuilding or redeploying, and no code in `packages/ui` knows which mode is live.

> **WHY IT'S THIS WAY**
>
> Retrieval is the shipped default because this platform is free and dev-funded, not because chat mode is expensive. The measured numbers above put all of AI Search at about 522 neurons for a day of testing, well inside the free allowance.

Three things worth knowing before switching to `chat`:

- **Retrieval size is the cost.** Prompt size is the number of results times the chunk size. `platform.ai_search_max_results`, default 3, is passed straight through. The instance's match threshold is set in the dashboard.
- **Model choice barely matters.** Input runs $0.270 to $0.293 per million tokens across the candidate models. This workload is almost entirely input tokens, so switching models saves single-digit percentages. Volume is the lever.
- **Query rewriting is a second model call** that also receives the transcript. The server enables it only from the second user turn, and the client is explicitly told not to send its own configuration.

Reranking is worth keeping on: 74.67 neurons for 264 thousand input tokens, and it is what lets the result count stay low without hurting answer quality.

Why the traffic goes through our worker

The widget can talk to the public endpoint directly. It does not, for four reasons.

1. **Content Security Policy.** The app policy is `script-src 'self'` and `connect-src 'self'` (`packages/kernel/src/security-headers.ts`). Pointing the browser at the AI Search host would mean widening both directives on every public page, permanently, to a host whose contents we do not control. The widget is bundled from npm and the API is same-origin, so neither changes.
2. **Rate limiting.** The public endpoint's own limit is instance-wide, so one abuser drains it for every visitor at once. Ours keys the signed client-id cookie and never the IP, because the school puts hundreds of students behind a few egress addresses.
3. **Request-body control.** The widget's API lets the page name a rewrite model and a rewrite prompt. Anything a browser can send, a hostile client can forge: that is a cost lever and a prompt-injection lever aimed at our account. `sanitizeChatBody()` is an allowlist that discards the client's copy and rebuilds the body server-side, and refuses `role: "system"` messages outright. This is what makes chat mode safe to enable later.
4. **The instance id never reaches page source.**

The VGSpartans Handbook

`aiSearchEndpoint()` pins `AI_SEARCH_URL` to HTTPS and to a `search.ai.cloudflare.com` host. That matters: this module fetches a URL from configuration on a public unauthenticated route, and a forwarder that will dial any host is a server-side request forgery gadget. A bad value degrades to "the bubble is off".

The guard chain

The completions route is a mutation, so it goes through the CSRF check: an `/api/v1` POST has to carry an `Origin` or `Referer` matching this origin. Browsers send `Origin` on a same-origin POST as a matter of course, so the widget passes and a cross-site script cannot spend our budget through a visitor's browser.

The probe GET is deliberately on the generic `api` bucket rather than the chat bucket. That GET is what mints the client-id cookie for a first-time visitor, and an unverified client keys the shared anonymous bucket, so charging it to the chat budget would let a handful of arriving visitors drain the anonymous allowance before anyone asked anything.

Where it appears

`PublicLayout` takes an opt-in `help` prop, default false. It is set on the core marketing pages and deliberately not on `/sign-in`, `/sign-in/email`, `/mfa`, the settings screens, `/report/submit`, `/console-check` or a person's profile page. Those screens exist to get one thing done, and a floating panel there is both a distraction and a place to mis-type a question that gets sent off-platform.

It is core-only in practice, since the answering route lives on `vgcore`. The subplatform 404 pages that share `PublicLayout` must not pass `help`.

Client behaviour

The rendered markup is an empty div. Everything happens client-side, in this order:

1. Probe the config endpoint. A probe that fails for any reason resolves to "off".
2. Only if enabled, load the widget when the browser is idle.

So a disabled bubble costs a visitor one small GET rather than about 76 KB of JavaScript for a button that would only fail. The widget is a separate lazy chunk; the eager part of the component is about 4 KB.

`requestIdleCallback` is unavailable in Safari and iOS Safari, so the `setTimeout` fallback is the real path on iPhones rather than a formality.

Styling

The widget paints inside a shadow root, so ordinary selectors cannot reach it. All styling goes through the roughly seventy CSS custom properties it reads, declared on the host in `ai-chat.css`, which is why that file is almost entirely variables.

One contrast trap: `--search-snippet-primary-color` is both the launcher background and the colour of links inside a reply, and in retrieval mode a reply is mostly links. Brand gold on paper fails WCAG AA as text, so the property is set to `--gold-text` and the launcher's glyph colour flips per theme to stay contrasty against it.

The VGSpartans Handbook

Two things the component hardcodes in its own shadow stylesheet cannot be reached by custom properties: the panel's size, and its open and close transform. Both are bridged out through an adopted stylesheet in `HelpBubble.astro` that carries no design values, only references to `--vg-help-*` properties declared in `ai-chat.css`. That keeps the sizing and the reduced-motion handling with the rest of the design system.

`transition:persist` keeps the panel across client-side navigation. Astro moves the persisted node with `Element.moveBefore()` where it exists, which preserves the custom element. Safari has no `moveBefore` yet, so there the node is re-appended, the element's connected callback re-runs, and the panel starts fresh. Graceful, not broken.

The hardening path

Swap the outbound `fetch()` for the `ai_search` worker binding and turn the public endpoint off in the dashboard entirely, removing the last unauthenticated surface. It needs the instance name, not its id, at deploy time, and it has no local simulator, so the bubble would stop working under `pnpm dev`. That is why it is not the shipped default.

Privacy

Whatever a visitor types is sent to Cloudflare AI Search and, in chat mode, to a Workers AI model.

> ⚠ WARNING
>
> These pages are public and the audience includes minors. The privacy policy has to say so before this is switched on for real traffic.

The VGSpartans Handbook

PART 5. ARCHITECTURE

DESIGN DECISIONS

Short records of the decisions that shaped the platform, and why each went the way it did.

Explanation . For developers . Checked 2026-08-02 . /documentation/architecture/decisions

IN PLAIN TERMS

Some choices on this platform look wrong until you know what the alternative cost. This part records those: what was decided, what it was weighed against, and what would have to change for the answer to change. The code shows what was done. These pages are the only place that says why the other option was not.

WHAT IS RECORDED HERE

Decision	In one line
Why the app CSP still allows inline script	Removing it would cost the persistent header, footer and console shell

WHAT BELONGS HERE

A decision earns a record when all three are true:

- Someone reading the code would reasonably think it is a mistake.
- The obvious alternative was considered and rejected for a specific reason.
- That reason is not written down anywhere else.

Most choices need none of this. A record that only restates what the code already says is worse than nothing, because it is one more thing to keep true.

WHY IT'S THIS WAY

These are their own pages rather than code comments because they cross too many files to live in any one of them. The CSP record touches the security headers, a layout, twelve components and the lint config; a comment in any single one of those would be invisible from the other eleven.

Each record states the conditions under which it should be reopened. When one of those is met, the honest move is to rewrite the record, not to quietly do the other thing and leave the page saying otherwise.

The VGSpartans Handbook

PART 5. ARCHITECTURE

WHY THE APP CSP STILL ALLOWS INLINE SCRIPT

What it would cost to remove 'unsafe-inline' from the app Content-Security-Policy, why that price is not worth paying yet, and what is in place instead.

Explanation . For developers . Checked 2026-08-02 . /documentation/architecture/decisions/csp-unsafe-inline

IN PLAIN TERMS

The app's Content-Security-Policy allows inline script. That is a real weakness and it is written down here rather than left as a silent gap. Removing it would mean giving up the persistent header, footer and console shell on every navigation, for a benefit the rest of the policy already largely provides. The decision is to keep it, with compensating controls, and revisit when Astro can do both.

THE GAP

`packages/kernel/src/security-headers.ts` builds the app policy, and its `script-src` carries `'unsafe-inline'`. The policy is enforcing, not report-only. Inline script is the classic cross-site-scripting sink: if user content ever reached the page as markup, `'unsafe-inline'` is what would let it execute.

Everything else in that policy is tight. `object-src 'none'`, `base-uri 'self'`, `form-action 'self'`, `frame-ancestors 'none'` in production, and an explicit, short list of external script hosts. The published student sites are a separate matter entirely: they run on their own registrable domain with no shared cookies, which is why that policy is deliberately looser and why it is safe to be.

WHAT REMOVING IT WOULD TAKE

Astro ships CSP support. It was stabilised in Astro 6.0.0, this platform runs 7.1.3, and it works by emitting **hashes** for the scripts Astro itself injects. Turning it on would resolve the gap in one config block.

It is not usable here, for a reason the Astro documentation states outright: view transitions using `<ClientRouter />` are not supported.

That sounds like a small dependency and is not. Counting what actually uses it:

What	Count	Note
`<ClientRouter />` mounts	1	`packages/ui/src/layouts/BaseLayout.astro`
Files hooking `astro:page-load`	41	Would mostly simplify, not port
`transition:persist` directives	12	**No native equivalent exists**

The single mount is misleading in the encouraging direction, and the 41 lifecycle hooks are misleading in the discouraging one. Under native MPA view transitions every navigation is a real document load, so

The VGSpartans Handbook

most of those handlers would get simpler or disappear rather than needing a rewrite.

The twelve `transition:persist` directives are the actual blocker. Every one of them is on persistent chrome:

- `Header.astro`, `Footer.astro`, `SubNav.astro`, `DocsHeader.astro`
- `HelpBubble.astro`
- `ConsoleShell.astro`, `BaseLayout.astro`

`transition:persist` keeps a DOM node **alive across a navigation**. The native View Transition API has no equivalent and structurally cannot have one: a cross-document transition animates between the outgoing and incoming page states through the `::view-transition-old()` and `::view-transition-new()` pseudo-elements, which are snapshots. Nothing is carried from one document into the next. Migrating would mean the header, footer, sub-navigation, console shell and help bubble are destroyed and rebuilt on every single navigation. The help bubble would close mid-conversation, and anything mid-interaction in the chrome would reset.

There is a second, independent reason to wait. MDN still lists the `@view-transition` at-rule that drives cross-document transitions as **limited availability, explicitly not Baseline**, because it does not work in some of the most widely used browsers. Trading a working navigation model for one that a share of this school's students could not use, in order to close a CSP directive, is the wrong way round.

> WHY IT'S THIS WAY
>
> So the trade is not "spend a week porting 36 files". It is "give up persistent chrome platform-wide, permanently, to close one CSP directive". Astro's CSP implementation and this platform's navigation model are mutually exclusive by design, not by oversight, and no amount of migration effort reconciles them.

THE DECISION

Keep `'unsafe-inline'`. Record it here as a known, accepted deviation rather than leaving a checklist item quietly open.

Revisit when either side moves: if Astro supports CSP alongside `<ClientRouter />`, or if the native View Transition API grows a node-persistence primitive. Both are plausible; neither is available now.

WHAT STANDS IN FOR IT

Nothing here is a substitute for a nonce or a hash. They are the reasons the residual risk is small enough to accept.

- **The app origin renders no user-supplied markup.** User content is escaped, and the one place arbitrary HTML runs is the separate user-content domain, which shares no cookies with the app and is covered by its own policy.
- **The rest of the policy is intact.** `object-src 'none'` and `base-uri 'self'` close the two most common ways an injected tag escalates, and `form-action 'self'` stops a planted form exfiltrating

The VGSpartans Handbook

to another origin.

- **`no-unsanitized/property` and `no-unsanitized/method` run at error level** in `eslint.config.mjs` over every `.ts` and `.tsx`, so a future `innerHTML` fed anything user-controlled fails the build. That rule was turned on specifically because this CSP gap exists.
- **Framing is denied outright** in production, so the page cannot be used as a clickjacking surface.

> **DANGER**
>
> The one change that would make this decision wrong is rendering user-supplied HTML on an app-domain page. If that is ever proposed, this record has to be reopened first: with `'unsafe-inline'` in place, a stored-XSS bug on the app origin is directly exploitable rather than mitigated.

The VGSpartans Handbook

PART 6

OPERATIONS

Deploys, migrations, backups, monitoring, and a runbook for every symptom.

PART 6. OPERATIONS

OPERATIONS

Running VGSpartans: deploying it, changing its databases, backing it up, watching it, and fixing it when it breaks.

Explanation . For admins, developers . Checked 2026-07-25 . /documentation/operations

IN PLAIN TERMS

This part is for whoever is responsible for VGSpartans still working tomorrow. It covers how a change reaches the live site, how the databases are changed safely, what is backed up and how to put it back, and what to do when something is on fire. If something is broken right now, skip everything else and go to the incident runbooks.

IF SOMETHING IS BROKEN RIGHT NOW

Go to **the incident runbooks**. It is a table of symptoms. Find yours, open its runbook, and follow the steps in order. Every runbook has the same six headings, and step 4 is always the fix.

Do not start by reading this page.

THE SHAPE OF IT

VGSpartans runs entirely on Cloudflare. There are no servers to log into, no operating system to patch, and nothing that can be fixed by restarting a machine. What that buys is that most classes of outage simply cannot happen here. What it costs is that when something does break, the fix is almost always a configuration change or a deploy, not an intervention on a running process.

The practical consequences, worth internalising before an incident:

- **There is no "restart the server".** A worker is redeployed, not restarted.
- **A rollback is a deploy of older code**, and it does not undo a database migration. That asymmetry is the single most important thing on this page. See **Rolling back**.
- **Most switches are environment variables or database rows**, not code. Turning a protection off, or on, is usually a repository variable and a deploy.
- **Almost every destructive operation is idempotent and rerunnable.** Provisioning creates only what is missing, migrations only apply files that have not run, cleanup only deletes names the manifest

The VGSpartans Handbook

does not claim.

WHAT IS IN THIS PART

Page	What it covers
Deploys	How code becomes the live site, the preview stack, and rolling back
Provisioning	Creating the databases, buckets and namespaces a stack needs
Migrations	Changing a database's shape, and the rule that must never be broken
Backups and restore	What is backed up, how often, and how to put it back
Secrets management	Every secret's name and how to rotate it. Never its value
Monitoring	What is watched and where the signals appear
What it costs	Every bill, and what more funding would buy
Incident runbooks	A symptom index, and one runbook per symptom

WHO CAN DO WHAT

Task	Needs
Read the incident runbooks	Nothing. They are public
Work the moderation queue, manage people	An admin console account
Change a platform setting	An admin account, and for some keys a developer one
Deploy, provision, migrate, restore	The repository, a Cloudflare API token, and developer console access

> ⚠ WARNING
>
> The highest-privilege operations need a Cloudflare API token that can create and delete resources. There is exactly one of those, it lives in the repository's Actions secrets, and it should never be copied onto a personal machine that is not already trusted with production.

THE RULES THAT ARE NOT NEGOTIABLE

These are stated once here and repeated in the pages that need them.

1. **Never edit a migration that has already been applied anywhere.** Add a new one. The reasoning is in Migrations.
2. **Never push directly to `main`.** A push to `main` deploys. The route is the branch ladder in Development workflow.

The VGSpartans Handbook

3. **Take a snapshot before anything destructive.** The reset workflow does this for you; a manual operation does not.
4. **Capture evidence before you fix.** Every runbook's second section is "Before you touch anything" for this reason: a fix usually destroys the evidence of what went wrong, and an incident you cannot explain afterwards will happen again.

The VGSpartans Handbook

PART 6. OPERATIONS

INCIDENT RUNBOOKS

A table of symptoms. Find the one you are seeing, open its runbook, and follow the steps in order.

Reference . For admins, developers . Checked 2026-07-25 . /documentation/operations/incidents

IN PLAIN TERMS

This is the index of things that go wrong and what to do about each one. Find the row that matches what you are seeing, open that runbook, and work through it from the top. You do not need to understand the platform to follow one. Every runbook names the exact screen, button, command or file for every step.

FIND YOUR SYMPTOM

What you are seeing	Runbook
People cannot sign in. No code arrives, or the code is refused	**Users cannot sign in**
An admin or developer console refuses someone who should have access	**Locked out of a console**
Login codes or notifications are not reaching anyone	**Emails are not arriving**
One page or route errors while the rest of the site is fine	**A page returns an error**
The whole platform is unreachable or very slow	**The site is slow or down**
Content is being blocked wrongly, or nothing is being checked at all	**Moderation is misbehaving**
The preview deploy failed, or the preview site does not answer	**The preview stack is broken**
A screen is showing something that does not match reality	**The data looks wrong**
An account, a session, or the platform may have been compromised	**A suspected security incident**

If nothing matches, start with **The site is slow or down**, which begins by working out how much is broken.

HOW TO USE A RUNBOOK

Every one has the same six headings, in the same order, always:

1. **Symptom.** Check this matches what you are actually seeing before going further. Following the wrong runbook wastes the first ten minutes, which are the ones that matter.

The VGSpartans Handbook

2. **Before you touch anything.** What to capture or check first. Skipping this is how an incident becomes one you cannot explain afterwards, which means it happens again.
3. **Likely causes**, ordered by how often they are actually the cause. Work down the list; do not start with the interesting theory.
4. **Fix, step by step.** This is always section 4. If you are in a hurry and you have already matched the symptom, this is the section you want.
5. **Verify it worked.** Prove the fix landed rather than hoping. An incident is not over because the error stopped appearing in the one place you were looking.
6. **If this did not fix it.** Who to escalate to, and what to bring them.

BEFORE YOU START ANYTHING

> ⚠ WARNING
>
> Three things, every time, no matter which runbook.
>
> **Write down when it started.** Not roughly. The exact time you first saw it, and the exact time of the last thing that was deployed or changed. Almost every incident here is caused by the most recent change, and the two timestamps together usually identify it in under a minute.
>
> **Do not make two changes at once.** If you change two things and it recovers, you have learned nothing, and you now have an untested change live as well.
>
> **Capture before you fix.** The fix usually destroys the evidence.

WHAT YOU MAY NEED ACCESS TO

Thing	Where
The admin console	`admin.<appDomain>`
The developer console	`developer.<appDomain>`, behind Cloudflare Access
Worker logs and traces	The Cloudflare dashboard, Workers, the worker, Logs
Deploy history and CI logs	The repository's Actions tab
Error reports	Sentry, if a DSN is configured. See **Monitoring**

If you cannot reach the developer console, that is itself a runbook: **Locked out of a console**.

ESCALATION

There is one developer. The escalation path is the address in `SECURITY.md`, and what to bring is written at the end of each runbook.

For anything that looks like a compromise rather than a fault, go to **A suspected security incident** first. It is the one runbook whose first instruction is to preserve rather than to fix.

The VGSpartans Handbook

PART 6. OPERATIONS

USERS CANNOT SIGN IN

Sign-in is failing for one person or for everyone. Work out which, then which of the four gates in front of it is refusing.

Runbook . For admins, developers . Checked 2026-07-27 . /documentation/operations/incidents/users-cannot-sign-in

---

IN PLAIN TERMS

Somebody cannot get in. There are only a handful of reasons that can happen, and they are easy to tell apart once you know whether it is one person or everyone. This runbook works down them in the order they actually occur, starting with the boring ones.

SYMPTOM

One or more of:

- The sign-in form accepts an email address but no code ever arrives.
- The code arrives but is refused as wrong or expired.
- The form refuses the address outright, or says to try again later.
- Sign-in appears to succeed and then bounces straight back to the sign-in page.

If the last one is happening to an ADMIN on a console subdomain, stop here and go to **Locked out of a console** instead. That is a different failure with a different fix.

BEFORE YOU TOUCH ANYTHING

1. **Establish the blast radius.** Try it yourself, on a school-domain address you control. One person failing and everyone failing have almost no causes in common, and getting this wrong sends you down the wrong half of the list.
2. **Write down the exact address that is failing**, including its domain. Most single-person cases turn out to be an address that was never eligible.
3. **Check the last deploy time** in the repository's Actions tab, against when this started.
4. **Do not clear the person's throttle or revoke sessions yet.** Both destroy the evidence of what refused them.

LIKELY CAUSES, MOST COMMON FIRST

1. **The address is not eligible.** It is not a school address, and it is not on the allowlist. This is by far the most common single-person case.
2. **The send throttle has locked them out.** They pressed resend too many times, or tried several addresses.

The VGSpartans Handbook

3. **The mail did arrive** and is in a spam folder or a filtered mailbox.
4. **A mail transport is failing.** If nobody is receiving codes, this is the cause.
5. **A bounty enrolment or account status closed**, which withholds the code entirely and by design.
6. **Turnstile or the proof-of-work gate is misconfigured**, so the form itself is refusing every submission.
7. **The authentication secret changed**, invalidating every existing session at once.
8. **The sign-in lockdown is on**, which refuses everyone except a short allowlist. Rare, but it is the one cause on this list that somebody switched on deliberately, so check it early when it is everyone.

FIX, STEP BY STEP

Step 1. Decide: one person, or everyone

Try to sign in yourself with a known-good `@cguhsd.org` address.

- **You get a code and it works.** It is one person. Go to step 2.
- **You get no code either.** It is everyone. Skip to step 5.

Step 2. Check the address is eligible

The platform withholds the login code entirely from an address that is not allowed to sign in, and the response looks identical to a successful one. That is deliberate anti-enumeration: it stops an outsider using the sign-in form to discover who has an account.

Eligibility is decided in `packages/services/src/identity.ts`. An address is eligible if any of these hold:

- it is on the school's domain, `@cguhsd.org`
- it is listed in `MASTER_ADMIN_EMAILS` or `DEVELOPER_EMAIL`
- it is on `LOGIN_EMAIL_WHITELIST`, the repository variable that lets specific non-school addresses in
- it holds an open bug bounty enrolment

If the address is a personal one that should be allowed: add it to the `LOGIN_EMAIL_WHITELIST` repository variable (Settings, Secrets and variables, Actions, Variables), then redeploy. It is comma-separated.

If the address is a school one and it is still being refused: go to step 3.

Step 3. Check whether they are throttled

The send throttle in `platforms/core/src/lib/otp-throttle.ts` is deliberately tight, and its numbers are worth knowing so you can tell a lockout from a fault:

- 30 seconds between sends of one code
- 3 sends to one address, then a 5 minute cool-off
- 3 different addresses tried, then a 2 hour pause on all sending

The VGSpartans Handbook

The client-side half persists in the browser's local storage, so a reload or a new tab does not shake it off, and there is a matching per-inbox backstop on the server.

The fix is to wait. Tell them the number: five minutes, or two hours if they tried several addresses. A person who has been clicking resend for ten minutes usually assumes the site is broken, and saying "it will work at 14" ends the incident.

Do not raise the limits to unblock one person. They exist because a school puts hundreds of students behind a handful of network addresses, which is exactly why the limits are per-inbox and not per-address.

Step 4. Check the mail actually went

Ask them to check spam, and to check they are looking at the right mailbox.

If they are certain nothing arrived, this is now a mail problem for one person, which is usually a receiving-side filter. Go to **Emails are not arriving** and read the "one recipient" branch.

Step 5. Check the sign-in lockdown is not on

Before you go hunting for a fault, rule out the switch that is supposed to do this. The developer console has a **Sign-in lockdown** control that refuses every sign-in and every new account, school addresses included, and admits only the addresses named in `DEVELOPER_EMAIL`, `MASTER_ADMIN_EMAILS` and `LOGIN_EMAIL_WHITELIST`. It also signs out everyone who is not on that list, so the symptom is total and immediate: nobody gets a code, and anybody who was already signed in is bounced to the sign-in page on their next click.

It is a maintenance and incident control, so somebody turned it on for a reason. Find out what that reason was before you turn it off.

How to check: open the developer console, Flags & config. The Sign-in lockdown panel says whether the door is open or shut, and lists exactly who is still admitted. The same state is visible in the `platform_config` table on the same tab, as `auth.lockdown`.

The fix, if it should not be on: set the toggle to off. It takes effect on the next sign-in attempt. Everyone who was signed out has to sign in again; that is expected and there is no way to give those sessions back.

> ⚠ **WARNING**
>
> If the allowlist is empty, turning this on locks out every account on the platform including your own, and the only way back is to set `DEVELOPER_EMAIL` and redeploy. The console warns you before you flip it. Read the warning.

If the lockdown is off, carry on to step 6.

Step 6. Nobody is receiving codes

This is a mail transport failure, and it has its own runbook: **Emails are not arriving**. Go there. Come back here only if mail is proven to be working.

The VGSpartans Handbook

Step 7. Check the form itself is not refusing everyone

If the form refuses submissions before any mail could be sent, the bot gate is misconfigured. There are two switches and both fail closed.

Check the repository variables:

- `ALTCHA_MODE` and `PUBLIC_ALTCHA` must be **both on or both off**. The production deploy has a gate that refuses a half-configured pair, so this can only be wrong if the variables were changed after the last deploy.
- `TURNSTILE_SITE_KEY` and `TURNSTILE_SECRET_KEY` must both be set in production. Turnstile verification fails closed when the secret is missing, so an unset secret refuses every submission.

The full behaviour is in **Bot protection**.

The fix: set both halves of whichever pair is half-configured, then redeploy. Turning the proof-of-work gate off means clearing BOTH `ALTCHA_MODE` and `PUBLIC_ALTCHA`, not one.

Step 8. Check whether the authentication secret changed

If everyone was signed out at the same moment and nobody can sign back in, `BETTER_AUTH_SECRET` may have been rotated or lost.

Rotating it is survivable on its own: everyone signs in again. Losing it, or setting it to an empty value, is not, because several other things derive from it, including the proof-of-work signing key.

The fix: restore the correct value with `wrangler secret put BETTER_AUTH_SECRET` on `vgcore`, and set the same value on every subplatform worker. They all have to match. If they do not, the cookies that `vgcore` mints will not verify anywhere else.

Step 9. Check for a closed bounty enrolment

If the person is a security researcher rather than a student, their access is time-boxed and closes itself. A closed enrolment withholds the login code entirely, which looks exactly like this incident.

The fix: renew it in the developer console, Bug bounty, Manage. Note that a revoked enrolment is terminal and has to be re-provisioned rather than renewed. See **The bug bounty console**.

VERIFY IT WORKED

Do all three. Any one of them alone can pass while sign-in is still broken.

1. **Sign in yourself**, in a private window, on a school address, from the sign-in page. You should receive a code and reach the hub.
2. **Have the original reporter try**, on their own device. Their device, their network, and their throttle state are all different from yours.
3. **Check the Worker logs** in the Cloudflare dashboard for the minute you signed in, and confirm there are no refusals recorded for your attempt.

The VGSpartans Handbook

IF THIS DID NOT FIX IT

Escalate to the developer at the address in `SECURITY.md`. Bring:

- **Whether it is one person or everyone**, and how you established that.
- **The exact email address**, and whether it is a school address.
- **The exact time** the first failure was seen, and the time of the last deploy.
- **What the person sees**, word for word, including any code or message on screen.
- **Which of the steps above you ran**, and what each one showed. “I checked eligibility and the address is on the school domain” is worth ten minutes of someone else’s time.

Do not include the login code itself, and do not ask the person for it. A valid code in a chat log is a working credential.

The VGSpartans Handbook

PART 6. OPERATIONS

LOCKED OUT OF A CONSOLE

An admin or developer console refuses someone who should have access.
Four independent gates can do that, and they fail in different ways.

Runbook . For admins, developers . Checked 2026-07-26 . /documentation/operations/incidents/console-locked-out

IN PLAIN TERMS

Somebody who should be able to open a console cannot. There are four separate locks in front of these screens, deliberately, and each one refuses differently. This runbook tells them apart by what the person actually sees, which is faster than guessing.

SYMPTOM

One or more of:

- A console subdomain shows a Cloudflare sign-in page that never lets them through, or refuses them outright before the site loads.
- They sign in successfully, then land back on the hub with a “denied” notice.
- The console loads but every action fails, and the page eventually signs them out.
- A browser check page refuses to let them continue.

If sign-in itself is failing, before any console is involved, that is **Users cannot sign in**.

BEFORE YOU TOUCH ANYTHING

1. **Get the exact screen.** A screenshot, or the text word for word. The four gates are told apart by what they say, and this is the whole diagnosis.
2. **Get the exact URL**, including the subdomain. `admin.` and `developer.` are different consoles with different locks.
3. **Ask which browser, and whether it is up to date.** One of the gates is a browser check.
4. **Do not grant a role or widen an allowlist yet.** Three of the four causes are not permission problems, and widening access to fix a device problem leaves a permanent hole behind.

LIKELY CAUSES, MOST COMMON FIRST

1. **The browser attestation gate** refused the browser. Most common on the developer and admin consoles after a browser change.
2. **The device fingerprint gate** has no valid device proof for the session, so every console API is refused.
3. **The person genuinely does not hold the panel.** Their role or allowlist entry is missing.

The VGSpartans Handbook

4. **Cloudflare Access** is refusing at the edge, before the request reaches the platform at all. Developer console only.
5. **A bounty enrolment closed**, which shuts both the console and sign-in.

FIX, STEP BY STEP

Step 1. Work out which gate refused, from what they saw

What they see	Which gate	Go to
A Cloudflare-branded login or "you do not have access", before any VGSpartans page	Cloudflare Access	Step 5
A VGSpartans page asking them to use a supported browser	Browser attestation	Step 2
They reach the hub with a "denied" notice naming a console	Panel permission	Step 4
The console loads, then actions fail and they are signed out	Device fingerprint	Step 3

Step 2. The browser attestation gate

The highest-privilege consoles are gated on a browser check served at `/console-check`, implemented in `packages/services/src/browser-attest.ts`. It is a positive allowlist of browsers that are known to enforce the protections the console relies on, not a blocklist of bad ones.

Its behaviour is controlled by one variable, `ADMIN_ATTEST_MODE`, which mirrors the other rollout switches:

- unset, which the code treats as **off**
- `shadow`, which logs what it would have refused and lets everyone through
- `enforce`, which refuses

If one person is affected: they are on a browser the allowlist does not cover. The fix is for them to use a supported one. Do not widen the allowlist for a single person's convenience: the list is the whole protection.

If everyone is affected right after a browser release: the allowlist may need its version floor moved. That is a code change in `browser-attest.ts` and a deploy, and it should be reviewed rather than rushed, because lowering the floor weakens the gate for everyone.

> ⚠ **WARNING**
>
> Chrome moved to a two-week release cadence from 2026-09-08. That halves the time between releases, so a version floor that was comfortable becomes tight. If this gate starts refusing people every fortnight, the cadence is why, and the answer is to widen the accepted range rather than to lower the floor.

The VGSpartans Handbook

To unblock one person while you work out the right fix: add their address to `ADMIN_ATTEST_BREAKGLASS`, a comma-separated allowlist of accounts that bypass the gate.

To unblock everyone immediately: arm the global kill-switch. One key-value write, no deploy, effective at once:

```
wrangler kv key put --binding=CACHE attest:breakglass:all 1
```

Disarm it with `wrangler kv key delete --binding=CACHE attest:breakglass:all` once the real fix has landed. It logs loudly on every use, which is the only reminder anybody gets that a protection is currently off.

> ⚠ **WARNING**
>
> Do not reach for `ADMIN_ATTEST_MODE=shadow` as the incident lever. The production deploy refuses to run unless that variable is exactly `enforce`, so setting it to `shadow` fails the deploy rather than unblocking anybody. A deliberate shadow rollout means relaxing that check in `.github/workflows/ci.yml`, which is a reviewed change, not an under-pressure one. The break-glass levers above are what this situation is for. The full picture is in **Admin browser attestation**.

Step 3. The device fingerprint gate

Console APIs require a valid device proof on the session. When there is not one, the API answers **428** and records a strike. It deliberately does NOT sign the session out.

That distinction matters, and it is why this gate looks different from the others: the person stays signed in, but every action fails. The client is meant to render the 428 as a message asking them to re-verify the device.

> ⛔ **DANGER**
>
> If people are being signed out in a loop rather than shown a 428, that is the failure this design exists to prevent, and it was a real production outage once. The gate must answer 428 and record a strike, never revoke the session. If you see a sign-out loop, the gate is misbehaving and this is a code problem, not a configuration one. Escalate rather than working around it by turning protections off.

The fix for one person: have them sign out fully and sign in again on the device they intend to use. That mints a fresh device proof.

If everyone is affected, check the two mode variables that gate this layer. Both are repository variables, and the production deploy refuses to run unless both are exactly `enforce`:

- `DPOP_MODE`
- `DEVICE_ANCHOR_MODE`

An unset value is treated as off, which silently disables the protection rather than breaking it, so this cannot be the cause of a lockout on its own. If they were changed to something other than `enforce`,

The VGSpartans Handbook

the deploy would have failed.

The other requirement is that `BETTER_AUTH_SECRET` is **identical** on `vgcore` and on every subplatform worker. The cookies `vgcore` mints are verified in the subplatform that ingests the device fingerprint, so a mismatch makes that verification throw, and every console API fails. If a secret sync ran against only some workers, this is the cause.

Step 4. The panel permission

Reaching the hub with a “denied” notice means the person signed in successfully and simply does not hold that console.

Who holds what is decided by `panelOk` in `packages/services/src/identity.ts`:

- **The admin console** admits an admin or a developer. A developer outranks an admin and therefore holds it too.
- **The developer console** admits a developer, and admits a master admin as deliberate break-glass for the door.

An account becomes an admin through `MASTER_ADMIN_EMAILS`, or through the `developer` or `it_admin` profile role. `DEVELOPER_EMAIL` is the developer allowlist.

The fix: add the address to the right allowlist (a repository secret, then a deploy) or seat the role from the admin console.

> ⚠ WARNING
>
> Break-glass is a door, not a role. A master admin reaching the developer console does not make them a developer, and anything that mints or manages an administrator separately requires a real developer and refuses self-targeting. Do not “fix” a permission problem by granting someone the developer role because the door let them in.

Step 5. Cloudflare Access

Only the developer console sits behind Cloudflare Access, and it refuses at the edge before any request reaches the platform. That is why the refusal looks Cloudflare-branded rather than like a VGSpartans page.

Access is configured in the Cloudflare dashboard, not in the repository. Check, under Zero Trust, Access, Applications, the application for the developer subdomain:

- The person’s email is in the Include list of the Allow policy.
- The Allow policy is ABOVE the default-deny Block policy.
- The session duration has not expired for them.

The full setup is in **The developer console**.

The VGSpartans Handbook

> ⚠ **WARNING**
>
> Getting into the developer console needs BOTH Access and the panel permission. Fixing only one leaves them still locked out, and it is easy to conclude the fix did not work. Check both before declaring it broken.

Step 6. A closed bounty enrolment

If the person is a security researcher, their access is time-boxed and closes itself. A closed enrolment shuts the console AND withholds their login code, so they will usually report the sign-in symptom rather than this one. Renew it in the developer console, Bug bounty, Manage.

VERIFY IT WORKED

1. **Have the affected person try again**, on the same device and browser, in a fresh private window. Your device is not the one that was refused.
2. **Confirm they reach the console and can complete one real action**, not just load the page. The fingerprint gate lets the page load and fails the actions.
3. **Confirm nothing else was widened.** If you changed an allowlist or a mode variable, check that it is the narrowest change that works, and that any temporary loosening (a `shadow` mode) has a note against it to be reverted.
4. **Check the audit log** in the console for the grant, if you seated a role. Every privileged change is recorded.

IF THIS DID NOT FIX IT

Escalate to the developer at the address in `SECURITY.md`. Bring:

- **A screenshot of the exact refusal**, and the full URL.
- **Which of the four gates you concluded it was**, and why.
- **The browser and version**, and whether another browser works.
- **Whether it is one person or everyone.**
- **Any variable you changed**, and whether it is still changed.

If you armed either attestation break-glass lever to unblock people, say so explicitly and prominently. That is a protection currently switched off, and it must not be forgotten once the incident closes.

The VGSpartans Handbook

PART 6. OPERATIONS

EMAILS ARE NOT ARRIVING

Login codes or notifications are not reaching people. Work out whether it is one mailbox or the whole chain, then which transport is failing.

Runbook . For admins, developers . Checked 2026-07-25 . /documentation/operations/incidents/emails-not-arriving

IN PLAIN TERMS

The platform sends email through four different providers, one after another, so that one of them failing is invisible. That design also means a failure can hide for weeks until the last one standing goes down too. This runbook works out whether the problem is one mailbox or all of them, and which provider is actually refusing.

SYMPTOM

One or more of:

- Somebody asks for a login code and none arrives.
- Nobody is receiving login codes.
- A notification that should have been sent was not.
- The live-provider check in CI failed, blocking a deploy.

Sign-in failing for other reasons is a different runbook: **Users cannot sign in**.

BEFORE YOU TOUCH ANYTHING

1. **Note the exact time** of a send that should have happened, and the recipient address.
2. **Check whether it is one recipient or all of them.** Ask for a code on an address you control, on a different mail provider from the reporter's if you can.
3. **Do not rotate a provider key yet.** A rotation is the fix for exactly one of the causes below, and doing it first destroys the evidence that would tell you which.
4. **Check the last deploy time.** A deploy that dropped a mail secret is the most common whole-chain cause.

LIKELY CAUSES, MOST COMMON FIRST

1. **A receiving-side filter.** The mail was sent and is in spam, or a school filter dropped it. This is most single-recipient cases.
2. **The address was never eligible**, so no mail was ever sent. The platform withholds the code silently by design.
3. **A provider key was rotated at the vendor and not here**, or was dropped from the Worker's secrets by a deploy.

The VGSpartans Handbook

4. **A provider is throttling or is down.** The chain absorbs this, so it shows up as slowness rather than failure, unless several are down.
5. **The sending domain's authentication broke.** A DNS change to the mail records makes every recipient's provider reject the mail.
6. **The send quota was exhausted.** ZeptoMail is prepaid credits, not a subscription.

FIX, STEP BY STEP

Step 1. Establish the blast radius

Ask for a login code on an address you control.

- **It arrives.** It is one recipient. Go to step 2.
- **It does not.** It is the chain. Skip to step 4.

Step 2. One recipient: check they were eligible at all

If the address is not allowed to sign in, no mail is ever sent, and the response looks identical to a successful one. That is deliberate: it stops the sign-in form being used to discover who has an account.

Check the address against the eligibility rules in step 2 of **Users cannot sign in**. If it is not eligible, that is the whole explanation and the fix is there, not here.

Step 3. One recipient: check the receiving side

- Have them check spam and any filtered folders.
- Have them check they are looking at the right mailbox, and that any forwarding rule is not eating it.
- If they are on the school's own mail system, a district-level filter can drop mail from a new sending domain. That is not fixable from this platform; it needs whoever administers the district's mail.

If the address is a personal one on a major provider and nothing arrives, continue to step 4: it may be a domain reputation problem that is only visible on some receivers.

Step 4. Find out which transport is failing

Login mail goes out through a chain, in this order, defined in `packages/services/src/email.ts`:

1. **ZeptoMail**, the production primary
2. **AWS SES**, through an API gateway
3. **Cloudflare**
4. **Resend**

The chain moves to the next provider on a transport failure, so a single dead provider is invisible in normal operation. Two things tell you which one is actually being used:

- **The Worker logs** in the Cloudflare dashboard, under the `vgcore` worker. Each send records which provider carried it.
- **The mail send log** in the database, which records the same thing durably.

The VGSpartans Handbook

Run the live provider check. This is the tool built for exactly this question. It sends real mail through every transport in the chain individually, so a provider that is down, unpaid, or holding a rotated key fails visibly instead of hiding behind the next one:

```
pnpm --filter @vg/core test:live
```

It needs the provider secrets in the environment and the `TEST_EMAIL_RECIPIENTS` variable, which is the allowlist of mailboxes it may send to. Every address in it has to be deliverable by all four transports.

> **WARNING**
>
> This sends real email and spends real quota. It is the same job CI runs before every production and preview deploy, which is why a provider outage blocks a deploy rather than shipping into one.

Step 5. Fix the failing provider

A rotated or missing key. Set it again on `vgcore`:

```
wrangler secret put ZEPTO_API_TOKEN
```

The full list of mail secrets and what each one is for is in **Secrets management**. The names are `ZEPTO_API_TOKEN`, `ZEPTO_API_URL`, `ZEPTO_MAIL_FROM`, `SES_GATEWAY_URL`, `SES_GATEWAY_AUTH_TOKEN`, `MAIL_FROM`, `RESEND_API_KEY`, `RESEND_MAIL_FROM` and `CF_MAIL_FROM`.

> **DANGER**
>
> The secret sync in the deploy skips a BLANK value rather than writing it. That is deliberate, so a missing repository secret does not wipe a working one. The consequence is that clearing a repository secret does not clear it on the worker: the old value keeps being used until it is explicitly overwritten. If you rotated a key at the vendor and mail still fails, the worker may still be holding the old one.

Exhausted credits. ZeptoMail is prepaid. Check the balance in the ZeptoMail console and top it up. The budget page puts the yearly figure at 120000 emails for $30; see **What it costs**.

Broken sending-domain authentication. Each provider verifies its own sending domain, and they are deliberately different subdomains so one provider's reputation problem cannot take the others down. If a DNS change removed or altered those records, restore them from the provider's own dashboard, which shows exactly which records it expects.

Step 6. If the chain is healthy but mail is still not delivered

At this point the platform has handed the message to a provider that accepted it, and the failure is between that provider and the recipient. Check the provider's own delivery log for the message: it will say whether it was delivered, deferred, bounced or rejected, and a rejection reason from the receiving side is the answer.

The VGSpartans Handbook

VERIFY IT WORKED

1. **Run the live provider check again** and confirm every transport passes, not just the primary. A green primary with a dead backup is the state this whole design exists to avoid.
2. **Request a real login code** on an address you control and confirm it arrives.
3. **Have the original reporter confirm**, on their own mailbox. Their receiving side is the one that was failing.
4. **Check the Worker logs** for the minute of your send and confirm which provider carried it. If it fell through to a backup, the primary is still broken even though mail is arriving.

IF THIS DID NOT FIX IT

Escalate to the developer at the address in `SECURITY.md`. Bring:

- **Whether it is one recipient or all**, and how you established that.
- **The output of the live provider check**, in full. It names the failing provider and its error.
- **The recipient address and the exact send time**, so the provider's own log can be searched.
- **Whether any provider key was rotated recently**, and where.
- **The provider's delivery log entry** for the message, if you got that far.

Never paste a provider API key, or a login code, into an escalation. Both are live credentials.

The VGSpartans Handbook

PART 6. OPERATIONS

A PAGE RETURNS AN ERROR

One page or route is failing while the rest of the site works.

Runbook . For admins, developers . Checked 2026-07-25 . Unfinished . /documentation/operations/incidents/a-page-500s

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the single-route failure runbook.

The VGSpartans Handbook

PART 6. OPERATIONS

THE SITE IS SLOW OR DOWN

The whole platform is unreachable or crawling.

Runbook . For admins, developers . Checked 2026-07-25 . Unfinished . /documentation/operations/incidents/site-slow-or-down

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the whole-site outage runbook.

The VGSpartans Handbook

PART 6. OPERATIONS

MODERATION IS MISBEHAVING

Content is being flagged wrongly, or not at all.

Runbook . For admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/operations/incidents/moderation-misbehaving

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the moderation pipeline runbook.

The VGSpartans Handbook

PART 6. OPERATIONS

THE PREVIEW STACK IS BROKEN

The preview deploy failed, or the preview site does not answer.

Runbook . For admins, developers . Checked 2026-07-25 . Unfinished . /documentation/operations/incidents/preview-stack-broken

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the preview stack runbook.

The VGSpartans Handbook

PART 6. OPERATIONS

THE DATA LOOKS WRONG

A screen is showing something that does not match reality.

Runbook . For admins, developers . Checked 2026-07-25 . Unfinished . /documentation/operations/incidents/data-looks-wrong

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the data integrity runbook.

The VGSpartans Handbook

PART 6. OPERATIONS

A SUSPECTED SECURITY INCIDENT

Something suggests an account, a session, or the platform has been compromised.

Runbook . For admins, developers . Checked 2026-07-25 . Unfinished .
/documentation/operations/incidents/suspected-security-incident

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the security incident runbook.

The VGSpartans Handbook

PART 6. OPERATIONS

DEPLOYS

How code becomes the live site, what runs before it is allowed to, and what a deploy can and cannot undo.

Explanation . For developers . Checked 2026-07-25 . /documentation/operations/deploys

IN PLAIN TERMS

Nobody deploys VGSpartans by hand. Code reaching the production branch is what deploys it, automatically, after a long list of checks have passed. This page is the shape of that; the detail is in the three pages under it.

THE MODEL IN FOUR SENTENCES

A push to `main` deploys production. A push to `preview` deploys a complete second copy of the platform on its own domains with its own data. Every other branch deploys nothing. Both deploy branches run the full check suite first, and neither can deploy if any of it fails.

WHAT HAS TO PASS FIRST

Both deploy jobs depend on all five check jobs plus the live provider gate:

Job	What it proves
Playwright tests	The end-to-end flows still work, against a freshly provisioned local stack
Build	Every platform in the workspace builds, not just the one you changed
Typecheck	Every app and the gateway typecheck, and the generated registry mirrors are not stale
Lint	Server-side async safety, no embedded styles in pages, no duplicated CSS, handbook references resolve
Data layer	The real migrations and seed apply to a local database and the endpoints behave
Live providers	Every email transport and every content classifier answers on the real wire

The VGSpartans Handbook

WHY IT'S THIS WAY

The live provider gate exists because both of those chains are designed to hide exactly the failure it looks for. Login mail falls through four providers, and moderation catches a failed classifier and fires its backup, so a rotted backup credential is invisible until the night it is the last one standing. This job makes every provider prove itself individually, primaries and backups alike, before anything ships.

It is push-only and never runs on a pull request: it needs repository secrets, it spends real money and real mail quota, and a vendor outage should block a deploy rather than a code review.

THE PAGES UNDER THIS ONE

- **How a push deploys**, the production job step by step.
- **The preview stack**, the isolated second copy and how it is torn down.
- **Rolling back**, and the asymmetry between code and data that makes it harder than it sounds.
- **The egress guard**, the outbound firewall a credentialed job runs behind, and how to add a host to it.
- **Moving deploy secrets into environments**, a change that has not been made, and what it would involve.

THE THING TO INTERNALISE BEFORE YOU NEED IT

DANGER

A deploy moves code. It does not move data.

Deploying an older version of the code does not undo a migration that ran, does not un-delete a row, and does not restore a bucket. If a release changed the shape of the database, the previous release will meet a database it does not recognise.

That is why every migration on this platform has to be additive and has to leave the previous release able to run against it. Read **Rolling back** before you are in a position to need it, not during.

ORDER MATTERS

The deploy publishes workers in a fixed order, and it is not arbitrary:

1. `vgcore` **first.** Every other worker holds a service binding to it. A gateway deployed against a core that does not exist yet fails.
2. **The subplatform workers next.**
3. **The gateway last.** It binds all of them, so every one has to exist before it is published.

The same rule applies when adding a platform: publish the new worker before the gateway that binds it.

The VGSpartans Handbook

PART 6. OPERATIONS

HOW A PUSH DEPLOYS

Every step between a merge into the production branch and the site changing, in the order they run and why that order matters.

Explanation . For developers . Checked 2026-07-25 . /documentation/operations/deploys/how-a-push-deploys

IN PLAIN TERMS

A merge into the production branch is the deploy. Nobody runs a command. This page is what happens after that merge, in order, so that when a deploy fails you know which step it failed at and what that step was trying to do.

The whole thing is one job in `.github/workflows/ci.yml`. It runs only on a push to `main`, never on a pull request, and only after every check job and the live provider gate have passed.

THE GATES IT WILL NOT START WITHOUT

Before any of the steps below, the job refuses to run unless four things are explicitly configured. Each one fails the deploy CLOSED, on purpose, because each protects something that would otherwise regress by omission rather than by decision.

Gate	What it requires	Why it fails closed
Turnstile	`TURNSTILE_SITE_KEY` and `TURNSTILE_SECRET_KEY` both set	Verification fails closed without the secret, so deploying without the pair either breaks every form post or silently drops bot protection
ALTCHA consistency	`ALTCHA_MODE` and `PUBLIC_ALTCHA` both on, or both off	Half-on refuses every form post; half-on the other way burns client CPU on a proof nobody checks
Device binding	`DPOP_MODE` and `DEVICE_ANCHOR_MODE` both exactly `enforce`	An unset value is treated as off, which silently disables the stolen-cookie defence
Browser attestation	`ADMIN_ATTEST_MODE` set	The same: unset means off, and this gate once shipped silently disabled for exactly that reason

The VGSpartans Handbook

> **WHY IT'S THIS WAY**
>
> These read as bureaucracy until you notice the pattern: every one of them guards a protection whose “off” state is indistinguishable from working software. Turnstile off looks like a site with no captcha. Device binding off looks like a site that signs people in. The only moment anyone could notice is a deploy, so the deploy is where it is checked.
>
> To run a deliberate shadow rollout, relax the specific check, which is one reviewed place. Do not unset the variable.

WHAT THE JOB CAN REACH, AND WHAT HOLDS THE KEYS

Two things constrain the deploy before its first real step runs.

The network is fenced. The job brings up **the egress guard** before it installs a single dependency. Outbound traffic is default-deny, and the allowlist is the handful of hosts a deploy legitimately needs: Cloudflare, the npm registry, and GitHub itself.

The credentials are scoped to the steps that use them. The Cloudflare API token is set on the individual steps that call the Cloudflare API, not on the job. The steps that run other people’s code, `pnpm install` and all nine `pnpm build` steps, run with no Cloudflare token in their environment at all. Publishing a worker and building one are separate steps for this reason.

> **WHY IT'S THIS WAY**
>
> A deploy runs thousands of packages nobody on this project has read, in the same VM as the keys to the whole Cloudflare account. Cloudflare has no OIDC federation for API tokens, so the token cannot be made short-lived and the blast radius has to be reduced some other way.
>
> The two controls fail differently on purpose. Scoping is what holds if the fence has a gap, because a key that is not in the environment cannot be read regardless of what the network allows. The fence is what holds if a credential leaks into a place nobody expected, because a key that cannot be sent anywhere is not much use. Neither is sufficient alone.

THE STEPS

1. Provision

`node scripts/cf-provision.mjs` lists the existing databases, buckets and namespaces, creates only what is missing, and writes the resolved ids back into the configuration. Re-running is a no-op, so this is safe on every push and is what makes a first deploy work with no manual setup.

2. Migrate

`node scripts/db-migrate.mjs --remote` applies every unapplied migration to every database in the manifest, in order. Incremental: files that already ran do not run again.

The VGSpartans Handbook

3. Seed

`node scripts/db-seed.mjs --remote`, plus the file and cache seeds. Idempotent.

4. Verify

`node scripts/db-verify.mjs` diffs the applied-migration set and the per-table row counts and fails the deploy on drift.

5. Clean up

`cf-cleanup.mjs --scope orphans` reclaims renamed or outdated resources, and `db-prune-tables.mjs` drops tables no migration creates. Both are constrained: cleanup only deletes names the manifest does not claim, and the prune only drops tables nothing creates. Live data in a current table is never touched.

6. Build

`pnpm build`, which also writes the search index over the built output.

7. Publish the workers, in order

This order is not arbitrary and a deploy will fail if it is changed:

1. `vgcore`, the backend every other worker's service binding points at.
2. **The subplatform workers**, each built and deployed through its own generated configuration.
3. **The gateway, last**, because it binds all of them and a binding to a worker that does not exist yet fails.

The generated configuration is what gets deployed, not the source one: the source `main` entry is a development-only virtual entry that wrangler cannot bundle on its own.

8. Sync the secrets

Each worker gets the secrets it actually needs, and only those. `vgcore` gets the authentication material and the whole mail chain; the subplatforms get the shared authentication secret, the domains, and the detail-privacy pair.

> ⚠ **WARNING**
>
> The secret sync SKIPS a blank value rather than writing it, with a per-key warning. That is deliberate, so a missing repository secret cannot wipe a working one. The consequence is the one that surprises people: clearing a repository secret does not clear it on the worker. The old value stays in use until something explicitly overwrites it.
>
> There is one exception, and it is there because the general rule was wrong for it: the ALTCHA mode is written as an explicit `off` when the variable is cleared, so turning that gate off actually disables it instead of stranding a stale `enforce`.

9. Tear down a merged preview stack

After a successful production deploy, if `preview` is fully merged into `main`, every `-preview` resource is deleted. If `preview` has unmerged commits, the stack is left alone.

The VGSpartans Handbook

10. Publish the handbook PDF

Non-blocking, after the deploy. It prints the handbook and uploads it. A failure here cannot fail or roll back the deploy; the worst case is that `/documentation/pdf` serves the previous release's document until the next one.

WHEN A STEP FAILS

Step that failed	What it usually means	Where to look
A gate, before step 1	A repository variable or secret is unset or half-set	The job's error message names the exact variable
Provision	The Cloudflare token lacks a permission, or a name collides	The wrangler output
Migrate	A migration cannot apply. On preview this is caught by the local rehearsal first	**Migrations**
Verify	Local and remote disagree on the applied set or row counts	`pnpm db:verify` output
Publish	A worker was published out of order, or a binding target does not exist	The deploy log for the first worker that failed
Secret sync	A repository secret is blank, which is skipped with a warning rather than failing	The step's warnings, which are easy to scroll past

If production is now in a bad state rather than merely un-deployed, go to **Rolling back**.

RERUNNING IS SAFE

The whole chain is idempotent. Provisioning creates only what is missing, migrations are incremental, cleanup only deletes non-manifest names, and the table prune only drops tables nothing creates. A rerun after a partial failure picks up where it stopped rather than starting a second, conflicting deploy.

The VGSpartans Handbook

PART 6. OPERATIONS

THE PREVIEW STACK

A complete second copy of the platform, on its own domains with its own data and its own credentials, and how it tears itself down.

Explanation . For developers . Checked 2026-07-25 . /documentation/operations/deploys/the-preview-stack

---

IN PLAIN TERMS

Pushing to the preview branch builds an entire second VGSpartans: its own workers, its own databases, its own files, on a different domain. It exists so a release can be tried on something that behaves exactly like the real site without being the real site. Nothing in it can touch production data, and it deletes itself once its changes have shipped.

WHAT GETS BUILT

A push to `preview` deploys the whole stack under `-preview` names: every worker, plus suffixed databases, caches and buckets, migrated and seeded independently.

It serves on **dedicated domains**, `vgspartanstest.org` and `vgusercontenttest.org`, with the gateway as the entry point. That matters more than it looks: a separate registrable domain means the preview stack exercises production's real host routing, cookie scoping and content-security policy, on a domain that is genuinely foreign to production, rather than approximating them on a development host.

HOW THE ISOLATION WORKS

`platforms/core/scripts/cf-preview-target.mjs` rewrites `wrangler.jsonc` **in that checkout only** before anything is built: the worker names get the suffix, the database, bucket and workflow names get the suffix, the production routes are removed. The rewrite is never committed.

From there the job mirrors the production deploy against the suffixed resources.

Three separations are load-bearing:

Data. Preview has its own databases and buckets. Production's are not bound into it, and the naming is what makes that structural rather than a matter of care: the preview target only ever sees names with a `-preview` segment.

Credentials. The preview stack reads its own `PREVIEW_*` repository secrets, never production's. Its authentication secret is different, so a stolen preview session cannot be replayed against production. Its authentication URL is the test domain, so cookies are minted for it. Its admin and developer allowlists resolve to test accounts.

The VGSpartans Handbook

> ⚠ **WARNING**
>
> If a `PREVIEW_*` secret is unset it is SKIPPED, never filled in from the production secret. The preview worker then fails authentication closed, which is the safe direction, and it will look like preview is broken. That is the intended behaviour, not a bug: the alternative is a preview stack quietly running on production credentials.

Access. The preview consoles serve on real domains, so they need their own Cloudflare Access applications on the test zone, exactly like production. Do not assume a preview console is protected because it is “not production”.

THE ONE THING PREVIEW SHARES WITH PRODUCTION

Mail transports.

A deployed preview worker runs in production mode, so the development path that stashes a login code locally never fires, and it really sends mail. There is no preview-only sender, so it reuses the production mail credentials and sends from the real domains.

Preview keeps its own database, so its mail quota and its send log never touch production’s.

THE MIGRATION REHEARSAL

The preview job does something the production job does not, and it is the most useful safety property in the pipeline. Before touching any remote database, it:

1. resets the local state
2. applies every migration from scratch
3. verifies the result

A migration that cannot apply cleanly to an empty database fails there, on a runner, rather than half-applying to a real one.

TEARDOWN

Once everything on `preview` has landed on `main`, the stack has served its purpose and every `-preview` resource is deleted: workers, databases, caches, buckets, workflows and queues.

That runs only after a successful production deploy, and only when `preview` is fully merged. If `preview` still has unmerged commits, the stack is left alone. If the branch does not exist, there is nothing to do.

Production resources are out of reach of that teardown by construction, for the same reason the data is isolated: the preview target only ever sees `-preview` names.

The VGSpartans Handbook

WHEN PREVIEW IS BROKEN

It has its own runbook: **The preview stack is broken**.

The most common cause is worth knowing here, because the symptom is so specific: if the apex works but every subdomain returns 404, the gateway is missing its domain configuration. It host-routes from the configured domains, so without them it falls back to the hardcoded production default, every preview host classifies as foreign, and it dispatches everything to core, which then rejects the hosts it does not recognise. Production hides this failure mode entirely, because there the default happens to be correct.

The VGSpartans Handbook

PART 6. OPERATIONS

ROLLING BACK

Getting the live site back to a version that worked, and the asymmetry between code and data that makes it harder than it sounds.

Runbook . For developers . Checked 2026-07-25 . /documentation/operations/deploys/rolling-back

IN PLAIN TERMS

Something shipped and broke production. Rolling back sounds like pressing undo, and for the code it nearly is. For the database it is not undo at all: a migration that ran has run, and putting the old code back means the old code meets the new database. This runbook is about doing that safely, and about the cases where rolling forward is genuinely the better answer.

SYMPTOM

A release is live and is causing harm: errors on pages that worked, a broken sign-in, a console that refuses everyone, or behaviour that is wrong in a way you cannot leave up while you investigate.

BEFORE YOU TOUCH ANYTHING

1. **Identify the release.** Find the deploy in the repository's Actions tab and note its commit. Note the commit of the one before it too. You need both.
2. **Establish whether the release included a migration.** Look at the diff for any new file under a `migrations-*` directory. This single fact decides which half of this runbook applies, and getting it wrong is how a rollback makes things worse.
3. **Capture the failure.** Screenshots, the error text, the worker logs for the affected minute. A rollback erases the evidence of what the release did.
4. **Take a snapshot** if any data may have been written incorrectly: `pnpm infra:snapshot`. It is minutes of work and it is the difference between a recoverable mistake and a permanent one.

LIKELY CAUSES, MOST COMMON FIRST

1. **A code defect** in the release. Nothing structural, just wrong behaviour.
2. **A configuration change** that shipped with it: a variable, a secret, a platform setting.
3. **A migration** that changed something the application no longer agrees with.
4. **A dependency** that changed behaviour under the same version range.

FIX, STEP BY STEP

Step 1. Decide between rolling back and rolling forward

This is the actual decision, and it is worth thirty seconds.

The VGSpartans Handbook

Situation	Do this
No migration in the release, and the fix is not obvious	Roll back. Steps 2 and 3
No migration, and the fix is one obvious line	Roll forward. Step 4
The release contained a migration	Roll forward if at all possible. Step 4, then read step 5 before considering a rollback
Data is actively being corrupted right now	Step 6 first, then decide

> **DANGER**
>
> A migration that has run has run. Rolling the code back does NOT roll the schema back, because there are no down-migrations on this platform. The previous release will meet a database whose shape it does not know about.
>
> Whether that is survivable depends entirely on whether the migration was additive. A migration that only ADDED tables or columns is usually survivable: the old code ignores what it does not know about. A migration that renamed, dropped, or changed the meaning of an existing column is not, and rolling back to code that reads the old shape will fail, or worse, will read the new column as though it meant the old thing.

Step 2. Roll back by re-pushing the previous commit

The deploy is driven by what is on `main`, so a rollback is a commit that makes `main` be the previous code again. Use a revert, not a force push:

```
git checkout main
git pull
git revert --no-edit <the-bad-commit>
git push
```

A revert is a new commit, so the history stays honest and the deploy is triggered the normal way. A force push rewrites production history, breaks everyone's clone, and can leave the deployed version and the branch disagreeing.

> **WARNING**
>
> `main` is normally protected against direct pushes, and that protection is correct. A revert during an incident is one of the few legitimate exceptions, and it is the same route a `hotfix/*` branch takes: cut from `main`, pull request straight back into `main`, still passing CI.
>
> If CI is what is broken, and you genuinely cannot get a revert through it, that is an escalation, not a reason to bypass the checks.

Step 3. Wait for the deploy, and watch it

The revert runs the full check suite again before it deploys. That is deliberate: an incident is the worst possible time to ship a second untested change. Watch the Actions run rather than assuming.

The VGSpartans Handbook

Step 4. Rolling forward instead

If the fix is small and obvious, or if the release contained a migration, a fix forward is usually safer:

1. Cut `hotfix/<short-description>` from `main`.
2. Make the smallest change that stops the harm. Not the correct long-term fix, the smallest one.
3. Open a pull request straight into `main`. It still has to pass CI.
4. After it merges and deploys, **merge `main` down into `preview`, `develop` and `experimental`**.

> **DANGER**
>
> Step 4 is the one people skip, and skipping it silently reverts the hotfix at the next promotion. `preview` still holds the broken code, and `preview` is what becomes `main`. The fix disappears and nobody connects the two events.

Step 5. If you must roll back past a migration

Only when rolling forward is genuinely not possible.

1. **Read the migration.** Determine whether it is additive. If it only adds tables or columns, the previous code will almost certainly tolerate it, and you can roll the code back and leave the schema alone.
2. **If it is not additive**, the schema has to be brought back too, and the only mechanism for that on this platform is a restore from a snapshot taken before the migration ran. That loses every write since the snapshot. See **Backups and restore**.
3. **Never hand-edit the migration file** to make it look like it did not happen. Applied migrations are tracked by filename; editing one makes stacks disagree silently. See **Migrations**.

Step 6. Stopping active harm before you decide

If data is being written wrongly right now, stopping the writing matters more than choosing the elegant fix. Options, roughly in order of preference:

- **Turn off the feature** through its platform setting, if it has one. This is a database row and takes effect immediately with no deploy.
- **Set the relevant mode variable to `off` or `observe`** and deploy. Several protections and pipelines have exactly this switch.
- **Revert**, as in step 2.

VERIFY IT WORKED

1. **Confirm the deployed version.** Check the Actions run completed and the commit it deployed is the one you expected. A revert that failed CI has changed nothing.
2. **Reproduce the original failure** and confirm it no longer happens, on the real site, not locally.
3. **Check the worker logs** for the minutes after the deploy and confirm the error is gone rather than merely rarer.
4. **Run `pnpm db:verify`** if any migration was involved, to confirm local and remote agree on the applied set.

The VGSpartans Handbook

5. **Check the branches.** If you hotfixed, confirm `main` has been merged down into `preview`, `develop` and `experimental`. Do it now, not later.

IF THIS DID NOT FIX IT

Escalate to the developer at the address in `SECURITY.md`. Bring:

- **The two commits**: the bad release and what you rolled back to.
- **Whether the release contained a migration**, and the migration file if so.
- **What you have already done**, precisely. A revert that is already deployed changes what the next person should try.
- **The original failure**, captured before you rolled back.
- **Whether you turned any protection off** to stop the harm, and which. That must not be forgotten when the incident closes.

The VGSpartans Handbook

PART 6. OPERATIONS

THE EGRESS GUARD

The default-deny outbound firewall every job that holds a credential runs behind, why it exists, and how to add a host to one when a deploy needs a new endpoint.

Explanation . For developers . Checked 2026-08-01 . /documentation/operations/deploys/the-egress-guard

IN PLAIN TERMS

Every deploy runs thousands of packages nobody here has read, on the same machine as the keys to the whole Cloudflare account. The egress guard is the thing that stops those packages from being able to send a key anywhere.

WHAT IT DOES

Before a credentialed job installs a single dependency, it brings up a default-deny outbound firewall. Traffic to an allowlisted host is permitted. Everything else is logged, and in `block` mode dropped.

The implementation is `.github/actions/egress-guard`, a composite action.

WHY IT IS NOT A LIST OF IP ADDRESSES

`api.cloudflare.com`, `registry.npmjs.org` and `blob.core.windows.net` all sit behind CDNs. Their addresses rotate, and new ones appear partway through a job, so an allowlist resolved once at startup is wrong within minutes.

Instead, dnsmasq becomes the only resolver on the machine, and every address it resolves for an allowlisted name is written straight into the nftables set the filter matches on. Rotation maintains itself. A name that was never allowlisted never lands in the set, so a connection to it is a connection to an address nothing accepted.

WHY IT'S THIS WAY

A connection made directly to an IP address, skipping DNS entirely, never populates the set either, so it meets the same final rule. That is why the chain is default-deny rather than a blocklist: the interesting case is always the one nobody thought to enumerate.

THE TWO MODES

Mode	What happens to an unallowlisted connection
`audit`	Logged, and allowed through
`block`	Logged, and dropped

The VGSpartans Handbook

Either way, the companion `.github/actions/egress-guard-report` action writes what was logged into the job summary, as a list of address and port pairs with a count. It never fails the job.

ADDING A HOST

When a job legitimately needs a new endpoint, add it to that job's `allow:` list in the workflow. Matching is by domain suffix, so `cloudflare.com` also covers `api.cloudflare.com`, and a leading `*.` is accepted and ignored.

```
- uses: ./.github/actions/egress-guard
  with:
    mode: audit
    allow: |
      cloudflare.com
      r2.cloudflarestorage.com
```

The base allowlist is inside the action and always applies: GitHub's own runner endpoints, plus the npm registry. Those are not optional. The runner agent shares the machine's network with the job, so a fence that blocks them takes down log streaming and artifact upload rather than anything an attacker was doing.

MOVING A JOB FROM AUDIT TO BLOCK

Do not skip this. Going straight to `block` fails a deploy on an endpoint nobody predicted, and a half-deployed stack is worse than an unfenced one.

1. Leave the job on `mode: audit` and let it run for real.
2. Read the drop report in the job summary. Resolve each address it lists.
3. Add the legitimate ones to that job's `allow:`.
4. Change `mode` to `block`, and watch the next run of that job specifically.

The report lists addresses rather than names, because what the filter sees is what a connection dialled, not what it was called.

> **WHY IT'S THIS WAY**
>
> There is one host the audit report will always show and the allowlist cannot name: the SES backup transport posts to a URL carried in a secret, so it cannot be written in a workflow file. Resolve it from the report and add it before moving the live provider job to block.

WHAT IT DOES NOT DO

It is a network control, not a sandbox. It does not stop a malicious package from reading a credential, corrupting a build, or writing to the repository checkout. It stops it from sending what it found to an address this project never approved.

The VGSpartans Handbook

The other half of that pairing is credential scoping, described in **How a push deploys**: the steps that execute third-party code do not have the Cloudflare token in their environment in the first place.

The VGSpartans Handbook

PART 6. OPERATIONS

MOVING DEPLOY SECRETS INTO ENVIRONMENTS

A checklist for putting the production and preview deploy jobs behind GitHub environments, what it buys, and the plan requirement that decides whether it is available.

How-to . For developers . Checked 2026-08-01 . /documentation/operations/deploys/github-environments

IN PLAIN TERMS

Right now every deploy secret is a repository secret, readable by any job in any workflow in this repository. GitHub environments would narrow that to the two jobs that deploy. This page is what that change involves. It has not been made.

WARNING

Check this first. Environments on a **private** repository require GitHub Pro or GitHub Team. On the Free plan the option does not exist, and none of the steps below can be carried out. Everything else in this handbook works on any plan; this page is the exception.

WHAT IT BUYS

Today	With environments
Any job in any workflow can read `CLOUDFLARE_API_TOKEN`	Only a job that names the environment can
A new workflow gets access to production secrets by existing	A new workflow gets nothing until it is deliberately pointed at an environment
Nothing records that a deploy happened, beyond the run log	A deployment history per environment, with what deployed and when
A push to the deploy branch deploys	Optionally, a push to the deploy branch waits for a named human

The first row is the real one. The rest are consequences.

THE CHECKLIST

1. Create two environments in **Settings, Environments**: `production` and `preview`.
2. On each, set a **deployment branch policy** so only the matching branch can use it. `production` gets `main`, `preview` gets `preview`. Without this a branch could name the environment and read its secrets.
3. Move the secrets. Production values go to the `production` environment, the `PREVIEW_*` values to `preview`. The mail transport secrets are the exception and belong in both, because the preview stack deliberately shares production's senders. The list of names is in **Secrets management**.

The VGSpartans Handbook

4. Add `environment: production` to the `deploy` job in `.github/workflows/ci.yml`, and `environment: preview` to `deploy-preview`.
5. Delete the now-duplicated repository secrets, one at a time, watching a deploy after each. A secret that is still read from somewhere else fails loudly at that point rather than quietly later.

TWO THINGS WORTH DECIDING AT THE SAME TIME

Required reviewers. An environment can require a named person to approve before the job runs. On a one-person project this mostly adds latency to your own deploys. It is more useful on the two destructive workflows, `Reset data` and `Restore from reset snapshot`, where the pause is the point.

The live provider job. It holds the mail and moderation credentials but is not a deployment. Either give it its own environment or leave it on repository secrets, and if it stays on repository secrets, then step 5 above cannot remove the mail secrets from the repository scope.

> **WHY IT'S THIS WAY**
>
> Step 2 is the one that is easy to skip and is the whole point. An environment without a deployment branch policy is a naming convention, not a boundary: any branch can declare `environment: production` in a workflow file and read the secrets. The branch policy is what turns it into a control.

The VGSpartans Handbook

PART 6. OPERATIONS

PROVISIONING

Creating the databases, buckets and namespaces a stack needs.

How-to . For developers . Checked 2026-07-25 . Unfinished . /documentation/operations/provisioning

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the provisioning script and what it creates.

The VGSpartans Handbook

PART 6. OPERATIONS

MIGRATIONS

Changing the shape of a database safely, the one rule that must never be broken, and how a migration is rehearsed before it touches production.

How-to . For developers . Checked 2026-07-25 . /documentation/operations/migrations

IN PLAIN TERMS

A migration is a numbered file of database instructions that runs exactly once, ever. They are how tables get added and changed. The whole system depends on one rule: once a migration has run anywhere, it is finished and can never be edited. Changing a shipped migration is the fastest way to give two copies of the platform two different databases without anyone noticing.

THE RULE

DANGER

Never edit a migration file that has been applied anywhere. To change something a migration created, write a NEW migration.

Wrangler tracks which migrations it has applied BY FILENAME. Editing a shipped file means the change never runs on any stack that already applied it, while a stack built from scratch gets the edited version. The two then have different schemas, both believe they are up to date, and the difference surfaces later as a query failing on one stack and working on the other.

This has caused a real outage on this platform. The fix is always ALTER TABLE ADD in a new file, never an edit in place.

WHERE THEY LIVE

One directory per database, under `platforms/core`, and `vgcore` owns all of them even for databases another worker binds:

The VGSpartans Handbook

Directory	Database
`platforms/core/migrations-user`	Identity
`platforms/core/migrations-core`	Platform-wide records
`platforms/core/migrations-clubs`	Clubs
`platforms/core/migrations-vgsites`	Personal sites
`platforms/core/migrations-vgwiki`	VGWiki
`platforms/core/migrations-vgwrites`	VGWrites
`platforms/core/migrations-vgsynergy`	VGSynergy
`platforms/core/migrations-vgagora`	VGAgora
`platforms/core/migrations-vgnews`	VGNews
`platforms/core/migrations-audit`	The audit base
`platforms/core/migrations-audit-shard`	The audit shards

The list of databases and their migration directories comes from `platforms/core/scripts/lib/manifest.mjs`, which is the single source of truth. Adding a database means adding it there; the migrate, verify, provision and cleanup scripts all derive their work from it.

WRITING ONE

Schemas are defined in Drizzle, in `packages/db/src/schema/`, and the SQL is generated from them:

```
pnpm db:generate
```

That runs `drizzle-kit generate` once per database and writes the new numbered `.sql` file into the right directory.

Read what it generated before you commit it. The generator diffs your schema against the last snapshot, and when a snapshot is missing or stale it can decide the safest thing to do is drop and recreate a table. That is data loss, expressed as a routine-looking file.

> ⚠ **WARNING**
>
> This has happened here. Some core migrations shipped without snapshots, so a later `db:generate` diffed against a much older snapshot and emitted a destructive recreate of the reports and moderation tables. The fix was to hand-write that migration and keep its snapshot as a re-baseline. If a generated migration contains a DROP you did not ask for, do not commit it.

APPLYING ONE

Locally, against the local simulator:

The VGSpartans Handbook

```
pnpm db:migrate:local
```

Against the real databases:

```
pnpm db:migrate:remote
```

The script is `platforms/core/scripts/db-migrate.mjs`. It walks every database in the manifest in order. Two details are deliberate:

- **A local run is non-interactive.** It hands wrangler a non-TTY stdin so it skips the per-database "apply migrations?" prompt, which would otherwise ask once per database on every fresh start.
- **A remote run keeps its confirmation.** An accidental production migration should never be auto-approved.

Migrations are incremental. Only files that have not been applied run, so rerunning is safe and is what CI does on every deploy.

HOW PRODUCTION GETS THEM

The deploy job runs the chain on every push to `main`:

```
infra:provision -> db:migrate:remote -> db:seed:remote -> db:verify -> wrangler deploy
```

The preview deploy does something the production one does not, and it is the most useful safety property in this whole pipeline:

It rehearses the migration on a wiped local database first. Before touching any remote database, the preview job resets the local state, applies every migration from scratch, and verifies the result. A migration that cannot apply cleanly to an empty database fails there, on a runner, instead of half-applying to a real one.

```
node scripts/db-reset-local.mjs
node scripts/db-migrate.mjs --local --target preview
node scripts/db-verify.mjs --local-only --target preview
```

VERIFYING

```
pnpm db:verify
```

`platforms/core/scripts/db-verify.mjs` diffs the applied-migration set and the per-table row counts between local and remote, and exits non-zero on drift. It needs a seeded remote and a Cloudflare token.

The VGSpartans Handbook

REMOVING A TABLE

Tables that no migration creates are dropped by `platforms/core/scripts/db-prune-tables.mjs`, which the deploy runs. That is how a table stops existing: you delete the migration's creation of it in a NEW migration, and the prune reclaims what is left over.

It only ever drops tables no migration creates, so live data in a current table is never touched by it.

THINGS THAT WILL BITE YOU

Symptom	Likely cause	Fix	How to confirm
A migration applied on one stack and not another, both reporting success	A shipped file was edited	Write a new migration expressing the difference; never re-edit	`pnpm db:verify` reports drift between local and remote
A generated migration drops and recreates a table you did not change	A missing or stale snapshot in that directory	Hand-write the migration and keep its snapshot as the re-baseline	Read the generated SQL before committing; look for DROP
The deploy fails at the local rehearsal step	The migration cannot apply to an empty database	Fix the migration; it would have half-applied to production	The preview job's "Verify migrations apply cleanly" step output
A query fails in production and works locally	The remote is missing a migration	Run `pnpm db:migrate:remote`	`pnpm db:verify` shows a different applied set

WHAT A MIGRATION CANNOT DO

- **It cannot be rolled back.** There is no down-migration in this system. A rollback of code does not undo a schema change, which is why **Rolling back** insists that every migration be additive and backwards-compatible with the previous release.
- **It cannot move data between databases.** There are no cross-database joins, because the database engine has none. Anything spanning two databases is stitched in application code.

The VGSpartans Handbook

PART 6. OPERATIONS

BACKUPS AND RESTORE

What is copied, how often, where it goes, what is deliberately not in a backup, and how to put a stack back.

How-to . For developers . Checked 2026-08-01 . /documentation/operations/backups-and-restore

IN PLAIN TERMS

Three separate things back this platform up: a nightly copy of every database, a nightly copy of every file store, and a full snapshot taken immediately before anything deliberately destructive. This page is what each one covers, and the things none of them can capture, which matter most on the day you need them.

THE THREE MECHANISMS

What	When	Where it goes	Driven by
Every database, dumped	Nightly, 07 UTC	The `vgspartans-d1-backup` bucket	The worker's own cron, in `platforms/core/src/services/backup.ts`
Every file bucket, copied	Nightly, 10 UTC (03 MST)	The `vgspartans-r2-backup` bucket, dated	A GitHub Action, `.github/workflows/r2-backup.yml`
The whole stack, as one snapshot	Before a reset, or on demand	The `vgspartans-snapshots` bucket	`platforms/core/scripts/cf-backup-snapshot.mjs`

WHY IT'S THIS WAY

The file backup is a GitHub Action rather than part of the worker's cron, and that is not an inconsistency. A worker invocation caps out near a thousand subrequests, and copying a bucket from inside one means a get and a put per object, so a large bucket simply cannot be copied in a single run. The Action uses rclone, which performs a true server-side copy: Cloudflare moves the objects internally, nothing is downloaded to the runner, and there is no object-count ceiling.

The file backup prunes anything older than 45 days. The snapshot bucket deliberately has no automatic prune at all, because a reset snapshot silently aged out by an unrelated retention policy is a trap.

WHAT A SNAPSHOT CONTAINS

`cf-backup-snapshot.mjs` captures everything that can be captured for one stack:

- **Databases**, one SQL dump each, including schema, data, and the migration table, so a restored database carries the exact migration state it had.

The VGSpartans Handbook

- **Caches**, one JSON file per namespace, with every key, its value, its metadata and its absolute expiration.
- **File buckets**, every user-content bucket, copied server-side into the snapshot bucket.

The layout inside the bucket is keyed by target and snapshot id, and a `manifest.json` is written LAST, on purpose. That file is the completion marker: the restore script refuses any snapshot that lacks one, so a run that died halfway can never be mistaken for a complete backup and half-restored over a live stack.

WHAT NO BACKUP CONTAINS

> **DANGER**
>
> Three things are absent from every backup on this platform, and each one will stop a restore dead if you have not planned for it.
>
> **Worker secrets and variables.** They are write-only by design; no API can read them back. A restored stack has its data and none of its credentials. Re-set them with `wrangler secret put`, from wherever you keep them.
>
> **In-flight workflow instances.** Login emails and moderation scans that were mid-run are gone. They are transient by nature.
>
> **The worker code itself.** That lives in git and is redeployed, not restored.

The practical consequence: a restore is a data operation, and it has to be followed by a deploy and a secret sync before the stack works. Plan for all three, in that order.

HOW YOU FIND OUT A BACKUP FAILED

A backup that quietly stops working is worse than no backup, because it still looks like one. Three independent things watch for that, and they are deliberately independent: each one covers a case the others cannot see.

Mechanism	Catches	Still works when
Mail to `DEVELOPER_EMAIL`	A run that failed for one or more databases	Sentry is not configured
Sentry cron monitor	A run that failed, and a run that never ran	The mail chain is down
The nightly Action's final step	Nothing has been written for two days	The worker or its cron is dead

The mail names which databases failed and quotes the error, because the useful question when one arrives is which database, and whether it is the same one as last night.

The VGSpartans Handbook

> **WHY IT'S THIS WAY**
>
> The third check lives in `.github/workflows/r2-backup.yml` rather than in the worker, and that is the whole point of it. A cron that never fires cannot report its own absence. Cloudflare's cron triggers have no retry and no failure alerting of their own, and they have been observed to stop firing silently, so a check that runs inside the worker goes quiet in exactly the situation it exists to detect. The Action already runs nightly and already holds R2 credentials, so it can assert from outside that something recent landed.

The window is 48 hours, not 24, because the D1 backup runs at 00 MST and the Action at 03 MST; a single night's window would false-alarm on ordinary schedule wobble. Two nights missed is unambiguous.

> **DANGER**
>
> None of this retries. When one of these fires, the databases named have no snapshot from that night and will not get one on their own. Fix the cause, then run the backup again rather than waiting for the next scheduled attempt.

TAKING A SNAPSHOT

```
pnpm infra:snapshot                    # production
node platforms/core/scripts/cf-backup-snapshot.mjs --target preview --reason manual
node platforms/core/scripts/cf-backup-snapshot.mjs --target preview --dry
```

`--dry` reports what it would capture without writing anything, which is worth running once on an unfamiliar stack before the real one.

Take one before anything destructive. The reset workflow does it for you; a manual operation does not.

RESTORING

```
pnpm infra:restore
```

`platforms/core/scripts/cf-restore-snapshot.mjs` replays a snapshot back over a stack. It refuses any snapshot without a completion manifest.

There is also a workflow for the whole cycle, `.github/workflows/restore-from-reset.yml`, paired with `.github/workflows/reset-data.yml`.

The VGSpartans Handbook

> **WARNING**
>
> A restore overwrites live data with older data. Everything written between the snapshot and the restore is lost, and there is no merge. Before running one, be certain that losing that window is better than the state you are in, and take a snapshot of the CURRENT state first so the decision is reversible.

THE MONTHLY DRILL

A backup nobody has restored is a hope rather than a backup. A truncated dump, a manifest listing a file the bucket does not hold, a schema the restore cannot replay: none of those show up until the day you need them. So `.github/workflows/restore-drill.yml` exercises the path on the first of every month, and it can be dispatched by hand from the Actions tab.

It runs as two jobs that prove two different things, and neither one moves production data.

Job	What it does	What that proves
`production-snapshot`	Dry-runs the newest production snapshot	It is complete and every file it claims exists
`preview-cycle`	Snapshots preview, restores that snapshot back, verifies the schema	The machinery actually works end to end

> **WHY IT'S THIS WAY**
>
> The full cycle runs on preview rather than against a production snapshot on purpose. Restoring production data somewhere to prove it restores would put real students' data, who are minors, onto a test stack. So the drill proves the machinery on preview, and proves the production snapshot's completeness without reading its contents. Do not "improve" this by restoring production data into preview.

A dry run only checks that files are present. It never replays a dump or rebuilds a virtual table, which is why the preview half has to do a real restore for the drill to be worth running at all.

> **DANGER**
>
> The drill wipes preview. Normally that nets out to nothing, because it restores the same snapshot it just took. The real risk is a restore that fails part way: the restore wipes before it writes, so preview is left empty. The drill keeps its snapshot whenever anything fails and names it in the job summary, so recovery is re-running the restore against that id. On success the snapshot is deleted, because the snapshots bucket has no automatic prune and twelve copies of preview a year would accumulate in it.

AFTER A RESTORE, IN THIS ORDER

1. **Re-set the worker secrets.** Nothing works without them and they are not in the snapshot.

The VGSpartans Handbook

2. **Deploy.** The worker code is not in the snapshot either.
3. **Verify.** `pnpm db:verify` confirms the applied-migration set and the row counts line up.
4. **Sign in.** An end-to-end sign-in exercises identity, the mail chain and the session store in one action, which is the fastest proof the stack is really back.

THE CREDENTIALS THE FILE BACKUP NEEDS

The nightly file backup authenticates differently from everything else, and this catches people out. R2's S3-compatible API, which rclone speaks, uses an AWS-style access key and secret pair. That is a different credential type from the Cloudflare bearer token wrangler uses, so it cannot be reused:

- `CLOUDFLARE_ACCOUNT_ID`, already set
- `R2_ACCESS_KEY_ID`
- `R2_SECRET_ACCESS_KEY`

Create the pair in the dashboard under R2, Manage R2 API Tokens, with Object Read and Write. Scope it to the source buckets, the backup bucket, and `vgspartans-d1-backup`. That last one only needs to be listable: the Action never writes there, it just checks that the worker's database backup produced something in the last two days.

> ⚠ **WARNING**
>
> Adding a new platform's bucket to the backup's source list means re-scoping or re-issuing that token. If you do not, the copy fails with a permission error on the new bucket only, and every other bucket keeps backing up successfully, so the failure is easy to miss.

THINGS THAT WILL BITE YOU

Symptom	Likely cause	Fix	How to confirm
A restore refuses to run	The snapshot has no completion manifest, so it is incomplete	Use a different snapshot; the incomplete one is not trustworthy	List the snapshot's prefix and look for `manifest.json`
A restored stack cannot sign anyone in	The secrets were not re-set, because they are not in the snapshot	`wrangler secret put` for each, then redeploy	The worker logs show authentication failing immediately
The nightly file backup fails on one bucket	The S3 token is not scoped to a newly added bucket	Re-scope or re-issue the token	The Action's log names the bucket and returns 403
A backup exists but is 45 days old and the one you want is gone	The file backup prunes at 45 days	Nothing, after the fact. Take a snapshot before anything risky	The dated prefixes in the backup bucket

The VGSpartans Handbook

PART 6. OPERATIONS

SECRETS MANAGEMENT

The name of every secret the platform reads, where each one is set, and how to rotate it. Names and procedures only, never values.

How-to . For developers . Checked 2026-07-25 . /documentation/operations/secrets-management

---

IN PLAIN TERMS

The platform reads about twenty secret values: keys for the email providers, the key that signs sessions, and so on. None of them are in the code, none are in this handbook, and none can be read back out once set. This page is the list of what they are called and what to do when one has to change.

DANGER

No secret VALUE appears anywhere in this handbook, in the repository, or in any backup, and none ever should. Not a real one, and not a realistic-looking placeholder either: a fake key that looks real gets copied into a config file by someone in a hurry and then debugged for an hour.

Every value below is set with `wrangler secret put <NAME>`, which prompts for it, or as a GitHub Actions secret. Both are write-only. There is no command that prints one back.

WHERE A SECRET LIVES

There are three places, and which one a thing belongs in is decided by whether it is sensitive and whether it is needed at build time.

Kind	Where	Example
A real credential	GitHub Actions **Secrets**, synced to the worker by the deploy	`BETTER_AUTH_SECRET`, `ZEPTO_API_TOKEN`
A non-sensitive switch or list	GitHub Actions **Variables**, plain text and readable in logs	`DPOP_MODE`, `LOGIN_EMAIL_WHITELIST`
A value the browser needs	A `PUBLIC_*` build variable	`PUBLIC_TURNSTILE_SITE_KEY`

WHY IT'S THIS WAY

Mode switches are Variables rather than Secrets on purpose. They carry nothing sensitive, and being readable is the point: during an incident you need to be able to see whether a protection is on without having the ability to read credentials. A switch hidden in a secret store is a switch nobody can check.

Locally, the same values go in a `.dev.vars` file next to the worker, which is gitignored and so is not in the repository. Copy `platforms/core/.dev.vars.example` to create it, or run `scripts/setup-env.sh`,

The VGSpartans Handbook

which does the same thing. Every var in it is optional: with none set, dev uses a fixed insecure auth secret, localhost URLs, and prints login codes to the dev-server console rather than mailing them.

Anything read at build time instead goes in the shell environment or `.env`, because `.dev.vars` is a runtime file and a build-time variable will silently not be there.

WHERE A SECRET IS REACHABLE FROM

Being set is not the same as being reachable. In the deploy workflows, a credential is attached to the individual steps that use it, never to the job. A step that does not need a secret does not have it in its environment.

That distinction matters most for the Cloudflare API token, because the steps it is kept away from are the ones that run other people's code:

Step	Cloudflare token
`pnpm install`	No
`pnpm build`, for all nine apps	No
`astro check`, Playwright, the test suites	No
`wrangler deploy`, `wrangler secret bulk`	Yes
The provision, migrate, verify and cleanup scripts	Yes

Building a worker and publishing it are two separate steps for exactly this reason. Alongside it, **the egress guard** fences what a job can reach on the network at all.

WHY IT'S THIS WAY

Cloudflare has no OIDC federation for API tokens, so this token cannot be short-lived the way a cloud role can. When you cannot shorten a credential's life, the remaining lever is to shrink the number of places it exists at all.

There is a second reason to prefer step-level `env:` over a job-level one, unrelated to third-party code: it is self-documenting. A step's env block is a list of what that step is allowed to touch, so a reviewer can see the answer without tracing the whole job.

The VGSpartans Handbook

THE SECRETS, BY WHAT THEY DO

Identity

Name	What it is for
`BETTER_AUTH_SECRET`	Signs sessions and cookies. Several other keys derive from it
`BETTER_AUTH_URL`	The origin cookies are minted for
`MASTER_ADMIN_EMAILS`	The admin allowlist
`DEVELOPER_EMAIL`	The developer allowlist, and the entire scope of detail privacy

> ⚠ **WARNING**
>
> `BETTER_AUTH_SECRET` has to be IDENTICAL on `vgcore` and on every subplatform worker. The cookies core mints are verified in whichever worker ingests the device fingerprint, so a mismatch makes that verification throw and every console API fails. A partial sync is one of the harder failures to diagnose because most of the site keeps working.

Email

`ZEPTO_API_TOKEN`, `ZEPTO_API_URL`, `ZEPTO_MAIL_FROM`, `SES_GATEWAY_URL`, `SES_GATEWAY_AUTH_TOKEN`, `MAIL_FROM`, `RESEND_API_KEY`, `RESEND_MAIL_FROM`, `CF_MAIL_FROM`.

ZeptoMail is the primary; the rest are the fallback chain. `ZEPTO_API_URL` is only needed for a non-US data centre.

Bot protection

`TURNSTILE_SITE_KEY` and `TURNSTILE_SECRET_KEY`, plus the build-time `PUBLIC_TURNSTILE_SITE_KEY`. The proof-of-work gate needs no secret at all: its signing key is derived from `BETTER_AUTH_SECRET` under a fixed label.

Privacy

`DETAIL_PEPPER` is the key behind the one-way tokens that replace raw addresses and devices. It is **required**: the production deploy refuses to run without it.

Content moderation

`OPENAI_API_KEY`, and the parked `AZURE_CS_ENDPOINT`, `AZURE_CS_KEY`, `AWS_REKOGNITION_ENDPOINT`, `AWS_REKOGNITION_KEY`. Every classifier no-ops or fails safe when its secret is absent.

Infrastructure

`CLOUDFLARE_API_TOKEN` and `CLOUDFLARE_ACCOUNT_ID` for the deploy; `R2_ACCESS_KEY_ID` and `R2_SECRET_ACCESS_KEY` for the nightly file backup, which speaks the S3 API and therefore needs a different credential type.

The VGSpartans Handbook

The preview set

Preview reads its own: `PREVIEW_BETTER_AUTH_SECRET`, `PREVIEW_BETTER_AUTH_URL`, `PREVIEW_MASTER_ADMIN_EMAILS`, `PREVIEW_DEVELOPER_EMAIL`, `PREVIEW_DETAIL_PEPPER`, `PREVIEW_TURNSTILE_SITE_KEY`, `PREVIEW_TURNSTILE_SECRET_KEY`.

An unset `PREVIEW_*` secret is skipped, never filled in from production's. Preview then fails authentication closed, which is the safe direction.

ROTATING ONE

1. **Generate the new value** at the vendor, or with `openssl rand -hex 32` for one of ours.
2. **Update the GitHub Actions secret.**
3. **Deploy**, which syncs it to the workers. Or set it directly with `wrangler secret put <NAME>` if you cannot wait for a deploy.
4. **Verify** the thing it protects still works. For a mail key, that is `pnpm --filter @vg/core test:live`.
5. **Revoke the old value at the vendor**, once the new one is proven. Not before.

> ⚠ **WARNING**
>
> The deploy's secret sync SKIPS a blank value rather than writing it, so that a missing repository secret cannot wipe a working one. That means clearing a repository secret does NOT clear it on the worker: the old value stays in use. If you rotated at the vendor and the old key still appears to work, this is why.

ROTATIONS WITH CONSEQUENCES

`DETAIL_PEPPER` re-keys every future privacy token. Rows written before the rotation stop correlating with rows written after, permanently. Bump `KEY_VERSION` in `packages/services/src/detail-privacy.ts` in the same change so a reader can tell which era a token belongs to instead of drawing a wrong conclusion from a mismatch. This is why it is a separate secret from the authentication one: those two must not share a rotation schedule.

`BETTER_AUTH_SECRET` signs everyone out. That is routine and low-consequence on its own, but it has to be set on every worker in the same change.

Any mail key should be rotated one provider at a time, with the live provider check run in between. Rotating several at once and then finding mail broken tells you nothing about which one.

IF A SECRET IS EXPOSED

Treat a leaked `DETAIL_PEPPER` as equivalent to disclosing the raw addresses it protects: the input spaces are small enough to brute-force once the key is known.

Otherwise: rotate immediately, deploy, verify, revoke the old value, and then go to **A suspected security incident** for what to preserve and who to tell.

The VGSpartans Handbook

PART 6. OPERATIONS

MONITORING

What is watched, where the signals appear, what a healthy platform looks like, and the gaps that are still open.

Explanation . For developers . Checked 2026-08-02 . /documentation/operations/monitoring

IN PLAIN TERMS

This page is what watches the platform, and what does not. The second half matters as much as the first: a monitoring page that lists only what exists reads as coverage, and the gaps are where incidents get long.

WHAT IS WATCHED NOW

Signal	Where it comes from	Where it lands
Uncaught errors	`withVgSentry` on every Worker entry	Sentry, if `SENTRY_DSN` is set
Workflow step failures	`instrumentVgWorkflow` on the moderation workflows	Sentry
Requests, logs, traces	Cloudflare's native observability, on at 100%	The Cloudflare dashboard
Backup failure	The nightly cron mails `DEVELOPER_EMAIL`	Your inbox
Backup that never ran	A Sentry cron monitor, and the nightly Action	Sentry, and a failed Action
Restore actually working	The monthly drill	A failed Action

The backup signals are covered in full on **Backups and restore**; they are listed here so this page is a complete index rather than a partial one.

WHY IT'S THIS WAY

Sentry ships DARK. With no `SENTRY_DSN` the SDK initialises disabled and nothing leaves the Worker, which means half this table is inert on a stack where that secret was never set. That is deliberate, so that landing the code could not change production behaviour on its own, but it does mean "we have Sentry" and "Sentry is reporting" are two different claims. Check the secret before believing the table.

THE HEALTH ENDPOINTS

`/api/health` and `/api/v1/health` probe D1, KV and R2 and return 503 if any layer errors, rather than reporting a bare process-is-up. They were built for an external monitor to poll.

The VGSpartans Handbook

Nothing polls them.

WHAT IS NOT WATCHED

Three gaps, stated plainly because each one has a real consequence.

No external uptime monitor. If the platform goes down, the first report comes from a person. Every signal in the table above is generated *by the platform*, so a total outage silences the monitoring along with everything else.

No synthetic for the sign-in journey. Sign-in crosses identity, the mail chain, the OTP gate and the session store. Any one of those can break while every page still returns 200, and nothing would notice until somebody could not get in.

No SLOs and no error budget. There is no number that says whether the platform was good enough last month, so "is it healthy" is currently a judgement rather than a measurement.

> **DANGER**
>
> Do not close the uptime gap with a scheduled GitHub Action. Two reasons, and the second is the disqualifying one.
>
> **Cost.** Polling every fifteen minutes is about 2,880 runs a month. Even at twenty seconds a run that is roughly 960 minutes, close to half the 2,000 minutes the Free plan includes for private repositories, spent on work `curl` does in two seconds.
>
> **It is not a reliable heartbeat.** GitHub documents that the `schedule` event "can be delayed during periods of high loads of GitHub Actions workflow runs," and that when load is high enough "some queued jobs may be dropped." A heartbeat that the platform silently skips is worse than none: a dropped run is indistinguishable from a run that found nothing wrong, and a delayed one reports an outage long after it started. The floor is five minutes between runs in any case.
>
> An external uptime service polls at a real interval, alerts properly, and is free at this scale. The Actions runner is the right place for the backup dead-man check, which runs once a night and needs repository credentials; it is the wrong place for a heartbeat.

The sign-in synthetic needs one more thing before it can exist: a dedicated test account whose one-time codes go somewhere a robot can read. Running it against a real account would mail a real person every few minutes, so this is blocked on that account rather than on the monitoring itself.

The VGSpartans Handbook

PART 6. OPERATIONS

WHAT IT COSTS

Every bill behind VGSpartans for a year, how each figure was worked out, and what more funding would buy.

Reference . For everyone, admins, developers . Checked 2026-07-25 . /documentation/operations/what-it-costs

---

IN PLAIN TERMS

Running VGSpartans costs about $28 a month. Right now the developer pays all of it personally. The platform is free for everyone on campus and stays that way; the plan is for member clubs to split the cost evenly, with contributions optional, so a club can sign up and use everything without paying a cent. This page exists so anyone deciding whether to chip in can see exactly where the money goes.

All figures are per year, in US dollars. Prices were confirmed against vendor pricing pages on 2026-07-24.

The VGSpartans Handbook

WHAT IT COSTS TO RUN

	Line	What it buys	Per year
1	Cloudflare Registrar (domains)	Four `.org` domains at cost: the live pair plus the preview pair used to test deploys safely	$51
2	Cloudflare Workers Paid	The servers. Also covers all database and cache usage below	$60
3	Cloudflare R2	File storage: profile photos, club assets, wiki images, video staging, the evidence locker	$25
4	Cloudflare D1	The databases. Fits inside the Workers Paid plan at our size	$0
5	Cloudflare KV	Sessions and caching. Also inside the Workers Paid plan	$0
6	Cloudflare Workers AI	Content moderation on text, images and video audio. The free daily allowance covers us	$15
7	Cloudflare Media Transformations	Pulls frames and audio out of uploaded video so it can be checked before anyone sees it	$30
8	Bunny Stream	Hosting and playback for video that has passed moderation	$25
9	GitHub Pro	Private repository, CI, and the tooling the codebase is built with	$48
10	ZeptoMail (Zoho)	Sign-in codes, security alerts and notification email. 120000 emails	$30
11	AWS (SES and Rekognition)	Backup email transport, and a second opinion on borderline images	$55
12	Azure AI Content Safety and OpenAI moderation	Third and fourth opinion on borderline content. Both run on free tiers	$0
	Total		**$339**

That is about $28 a month to run a platform for the whole campus.

The VGSpartans Handbook

The floor, and the part that moves

Four of those lines are fixed no matter how busy the site gets.

Fixed	Per year
Domains	$51
Workers Paid	$60
GitHub Pro	$48
ZeptoMail	$30
Hard floor	**$189**

The remaining $150 scales with how much the platform actually gets used. If usage is light, the real bill lands closer to the floor. The estimates below are deliberately set above expected usage so a busy month does not become a surprise.

HOW EACH ESTIMATE WAS WORKED OUT

Domains, $51. Cloudflare Registrar sells at cost with no markup. Four `.org` domains: `vgspartans.org` and `vgusercontent.org` for the live site, plus `vgspartanstest.org` and `vgusercontenttest.org` so changes get tested on real domains before students see them.

Workers Paid, $60. $5 a month, the minimum charge. This one plan covers the workers, all database usage and all cache usage.

Databases and cache, $0. The Workers Paid plan includes 25 billion database rows read, 50 million rows written and 5 GB of database storage per month, plus 10 million cache reads and 1 GB of cache storage. A high school does not come close. Worth knowing which one runs out first: cache reads, at roughly 100000 page views a day. We are far below that.

R2, $25. First 10 GB free, then $0.015 per GB per month, and no charge to serve files out. Storage grows as students upload, so this line climbs over time. The video archive is what makes it grow, and the 90 day retention window on video keeps the climb slow.

Workers AI, $15. Every post, comment, image and video is screened before it goes live. The free allowance is 10000 neurons a day, roughly 200 to 300 moderated items daily at no cost. Beyond that it is $0.011 per 1000 neurons. Realistically this is $0 most months. The $15 is headroom for a busy stretch, like a club posting a full event photo set.

VGGallery changes that arithmetic and it is worth stating plainly: posting a full event photo set is not a busy stretch there, it is the daily job. It is also the only surface that checks two images per item, because a photo's thumbnail is served at its own URL and so has to be checked in its own right, which puts a gallery photo at roughly double the image cost of a post elsewhere. A normal week still sits inside the free allowance. Homecoming week will not.

Four knobs hold the line, all of them in the Admin console under Settings, and none of them needs a deploy: the per person daily caps (60 campus photos, 10 arts pieces), the per album cap (300), and the stored photo size. A fifth saving needs no setting at all: a photo is fingerprinted on the way in, so re-

The VGSpartans Handbook

uploading the same file reuses the verdict already reached on it instead of paying for a second opinion. When you check what this is actually costing, read the per model view in the Workers AI dashboard rather than the Text Generation card, which combines models and will not tell you which one is spending the money.

Media Transformations, $30. Uploaded video is held in private storage and checked before publication. Pulling out still frames and the audio track is what this pays for. The first 5000 operations a month are free. Audio extraction bills per second of audio, so a three minute clip is about 200 operations, which puts roughly 25 videos a month inside the free tier. $30 covers around 50 to 100 videos a month.

Bunny Stream, $25. $1 a month minimum, $0.01 per GB stored, about $0.01 per GB delivered, and transcoding is free. Video moved here from Cloudflare Stream, which would have cost roughly $54 a year in storage alone at the same volume before counting playback. Same job, less than half the price.

GitHub Pro, $48. $4 a month.

ZeptoMail, $30. Credits, not a subscription: $2.50 buys 10000 emails, valid six months. $30 is 120000 emails a year. Sign-in codes are the bulk of it, since the platform uses emailed one-time codes rather than passwords alone.

AWS, $55. Two things. SES is the backup email path that takes over if ZeptoMail fails, at $0.10 per 1000 emails, so a few dollars at most. Rekognition is the second opinion on images the first checks are unsure about, at $1.00 per 1000 images, running an estimated $3 to $6 a month.

Azure and OpenAI, $0. Azure AI Content Safety runs on the free tier, 5000 text records and 5000 images a month, and is only called on genuinely ambiguous content. OpenAI's moderation endpoint is free and, per their terms, does not train on what is sent. Both are backstops behind the checks above.

OPTIONAL, AND WHAT EACH ONE WOULD FIX

These are not needed to keep the platform up. Each one solves a real problem.

	Line	What it would fix	Per year
1	Greencloud VDS (development)	A proper machine to build and test on. The single biggest quality-of-life gain on the list	$960
2	Cloudflare Pro on `vgspartans.org`	Web application firewall, bot protection and image optimization on the main site	$240
3	Extended video capacity	Longer retention than the 90 day window, and room for more uploads without watching the meter	$120
	Total if all three were funded		**$1320**

The VGSpartans Handbook

Cloudflare Pro is $240 billed annually or $25 a month billed monthly, so the annual option saves $60.

WHERE THE MONEY WOULD GO FIRST

If contributions came in gradually, this is the order that helps most.

1. **Cover the $339 base.** This is the whole bill today. Split across even a handful of clubs it is a rounding error for each of them.
2. **Cloudflare Pro, $240.** The firewall and bot protection matter most as the site gets more traffic and more attention.
3. **The development machine, $960.** The biggest single line, and the one that most affects how fast the platform improves.

Fully funded, everything above comes to $1659 a year.

NOTES

> ⚠ WARNING
>
> Prices were verified on 2026-07-24 against the vendors' own pricing pages. Cloud pricing moves. Re-check before committing to a number in front of anyone.

- ZeptoMail has a pricing change taking effect for new sign-ups from 2026-07-01. The existing account is on current pricing, but confirm before adding credits.
- Usage-based lines are estimates. The fixed $189 is not.
- The per-service moderation cost breakdown is in **Moderation**. That table still lists Cloudflare Stream and needs updating for the move to Bunny.
- The AI spend split is in **AI usage**, which also explains why the dashboard's combined view hides where it goes.

The VGSpartans Handbook

PART 7

CONTRIBUTING

The workflow, the standards, the tests, and how to write for this handbook.

PART 7. CONTRIBUTING

CONTRIBUTING

How work gets into VGSpartans: the branches, the pull requests, the sign-off, and the checks that have to pass.

Explanation . For developers . Checked 2026-07-25 . /documentation/contributing

---

IN PLAIN TERMS

VGSpartans is open source, and changes get in through a fixed route: you make a branch, you open a pull request, a set of automatic checks runs, and the change climbs through four branches before it reaches the live site. This part of the handbook explains that route and the standards a change is held to along the way.

You do not have to write code to contribute. Reporting a bug, suggesting a change, and fixing a wrong sentence in this handbook all count, and the last one is a pull request like any other.

THE SHORT VERSION

1. Cut a branch from `experimental`, named `type/short-description` where the prefix is one of `feature/`, `fix/`, `chore/` or `docs/`.
2. Keep one branch to one purpose.
3. Open a pull request back into `experimental`.
4. Sign off every commit with `git commit -s`.
5. Reviewed work moves up through `develop`, then `preview`, then `main`.

The full rules are in `CONTRIBUTING.md` and `docs/REPO-RULES.md`, and this part of the handbook expands on them.

THE BRANCH LADDER

The repository has four permanent branches. Code gets more stable as it climbs.

The VGSpartans Handbook

Branch	What it is	Who merges into it
`experimental`	The sandbox. Allowed to be messy and allowed to break.	Working branches, by pull request. Direct pushes are tolerated here.
`develop`	Stable integration. Should always build and pass tests.	`experimental`, by pull request, with green checks.
`preview`	The release candidate. Gets final QA against a near-production build.	`develop`, by pull request, with green checks.
`main`	Production. A push here deploys to Cloudflare.	`preview` only, by pull request, with green checks and a person confirming preview was checked.

Working branches are short-lived. Branch from `experimental`, merge back into `experimental`, delete the branch.

> ⚠ **WARNING**
>
> A push to `main` deploys the live site. It never takes a direct push, and nothing merges into it except `preview` or a `hotfix/*` branch cut from `main` itself. After a hotfix lands, merge `main` back down into `preview`, `develop` and `experimental` so they do not drift.

SIGN YOUR COMMITS

Every commit needs a `Signed-off-by` line. That is the Developer Certificate of Origin: a statement that you wrote the change and have the right to contribute it. A GitHub check enforces it on every pull request, so an unsigned commit blocks the merge.

Git writes the line for you:

```
git commit -s -m "Fix the header overflow on narrow screens"
```

The name and email on the line have to match the ones on the commit.

WHAT THE CHECKS RUN

Continuous integration runs on every pull request, and all of it has to pass before a change can climb. The jobs are defined in `.github/workflows/ci.yml`:

- **Playwright tests**, the end-to-end suite.
- **Build**, which builds every platform in the workspace, not just the one you changed.
- **Typecheck**, which runs `astro check` over each app plus the gateway.
- **Lint**, which is typed ESLint over the server-side TypeScript, plus two guardrails: no embedded `<style>` blocks in Astro pages, and no duplicated CSS rules across the shared stylesheets.
- **Data layer**, the unit and integration tests that apply the real migrations to a local database.

Run the tests yourself before you push:

The VGSpartans Handbook

```
pnpm test
```

REPORTING A SECURITY PROBLEM

Do not open a normal issue or pull request for a security bug. `SECURITY.md` explains how to report it privately, and Part 4 of this handbook covers the disclosure programme.

THE LICENCE

The project is under the GNU AGPL v3. By contributing, you agree your change is released under the same licence.

The VGSpartans Handbook

PART 7. CONTRIBUTING

DEVELOPMENT WORKFLOW

The four permanent branches, the route a change takes through them, and the rules that govern each step.

Explanation . For developers . Checked 2026-07-25 . /documentation/contributing/development-workflow

---

IN PLAIN TERMS

Code does not go straight to the live site. It climbs a ladder of four branches, getting more checked at every rung, and only the top rung is the real site. This page is that ladder and the rules for each step. Think of it as a series of increasingly serious rehearsals before opening night.

THE FOUR PERMANENT BRANCHES

The repository has four branches that always exist. Code flows upward through them and gets more stable at each step. Below them are short-lived working branches you create and delete as you go.

1. **`main` is production.** A push to `main` auto-deploys to Cloudflare, so treat it with care. It stays untouched unless the code has passed every check and been reviewed on `preview` first. Nothing merges here except `preview`, or a `hotfix/*` branch.
2. **`preview` is the final stop before `main`.** It is the release candidate: what you expect to ship next. Use it for final QA against a near-production build. When it holds up, you promote it to `main`.
3. **`develop` is the stable integration branch.** Finished work lands here once it is reviewed and green. `develop` should always build and pass tests, even if a feature is not yet ready to release.
4. **`experimental` is the sandbox.** Spikes, rough drafts, and in-progress work integrate here first. It is allowed to be messy and it can break.

The ladder:

```
feature/* , fix/* , chore/* , docs/*
        |  (PR)
        v
   experimental   sandbox, may break
        |  (PR, green CI)
        v
     develop      stable integration
        |  (PR, green CI)
        v
     preview      release candidate, final QA
        |  (PR, green CI + manual sign-off)
        v
      main        production, auto-deploys to Cloudflare
```

The VGSpartans Handbook

SHORT-LIVED BRANCHES

All day-to-day work happens on a short-lived branch cut from `experimental`. Name it `type/short-description` in kebab-case. The prefix says what the branch is for:

- `feature/` a new capability, for example `feature/user-login`
- `fix/` a bug fix, for example `fix/header-overflow`
- `chore/` tooling, dependencies, or config, for example `chore/bump-deps`
- `docs/` documentation only, for example `docs/repo-rules`

The rules for these branches:

- Branch from `experimental` and open a pull request back into `experimental`.
- Keep one branch to one purpose. If the scope grows, cut a new branch.
- Delete the branch after it merges. They are disposable.

HOTFIXES

When something is broken in production and cannot wait for the full ladder:

1. Cut `hotfix/short-description` from `main`.
2. Open a pull request straight back into `main`. It still has to pass CI.
3. After it merges and deploys, merge `main` down into `preview`, `develop` and `experimental` so they do not drift.

> ⚠ **WARNING**
>
> Step 3 is the one people skip. If you leave it out, the next promotion up the ladder silently reverts the hotfix, because `preview` still holds the broken code and it is `preview` that becomes `main`.

PROMOTION RULES

- Every move up the ladder is a pull request, never a direct push.
- CI has to be green before a merge into `develop`, `preview` or `main`. That is the Playwright tests, the build, the typecheck, the lint job, the data-layer tests and the workflow audit.
- A merge into `main` also needs a person to confirm `preview` was checked.
- `experimental` is the exception. Direct pushes are fine there for quick iteration.

CHANGING A WORKFLOW FILE

The files under `.github/` are audited on every pull request, because they are the code that holds every production credential this platform has. Three tools run, and any of them failing blocks the merge:

The VGSpartans Handbook

Tool	What it checks
`yamllint`	The YAML itself, against `.yamllint`
`actionlint`	That it is a valid workflow, plus shellcheck over every `run:` block
`zizmor`	That it is safe: injection, over-broad permissions, unpinned actions, credentials

Run all three before you push:

```
yamllint .github/workflows/*.yml
actionlint
zizmor --collect all .github
```

Two rules the audit enforces that are easy to trip over:

- **Never put a `${{ }}` expression inside a `run:` block.** Pass the value through `env:` and reference it as a quoted shell variable. An expression is pasted into the script before bash parses it, so a value containing a semicolon becomes a command.
- **Pin every action to a full commit SHA**, with the version as a trailing comment. Dependabot updates both.

If a finding is a genuine false positive, suppress that one finding with a `# zizmor: ignore[rule-name]` comment inside the span it points at, and put the reason next to it. Do not relax the audit for everything to silence one line.

SIGN-OFF

Every commit needs a `Signed-off-by` line, which is the Developer Certificate of Origin. A GitHub check enforces it on every pull request, so an unsigned commit blocks the merge. Git writes it for you with `git commit -s`, and the name and email on the line have to match the ones on the commit.

THE BOOTSTRAP EXCEPTION

Early in the project, before the backend and frontend were established, direct pushes to `main` were allowed. Once real features existed and the deploy was live, that exception ended, and the rules above apply in full.

The VGSpartans Handbook

PART 7. CONTRIBUTING

CODE STANDARDS

The conventions this codebase holds itself to, and the checks that enforce them.

Reference . For developers . Checked 2026-07-25 . Unfinished . /documentation/contributing/code-standards

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the language, layout and naming conventions, and the lint rules behind them.

The VGSpartans Handbook

PART 7. CONTRIBUTING

DESIGN STANDARDS

The rules for anything that renders: where styles live and what never goes in a page.

Reference . For developers . Checked 2026-07-25 . Unfinished . /documentation/contributing/design-standards

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the stylesheet rules, the token rules, and the responsive rules.

The VGSpartans Handbook

PART 7. CONTRIBUTING

ACCESSIBILITY

The standard the platform holds to, what the automated checks actually prove, and the gaps we know about.

Reference . For developers . Checked 2026-08-04 . /documentation/contributing/accessibility

---

VGSpartans targets **WCAG 2.2, Levels A and AA**, on every public page.

2.2 rather than 2.1 because it is the current W3C Recommendation: it keeps the 2.0 and 2.1 criteria, adds nine, and removes one (SC 4.1.1 Parsing, declared obsolete because modern browsers make it moot). That removal is why "meets 2.2" is not automatically "meets 2.1" in the strict sense — so the test suite tags **both**, `wcag21a`/`wcag21aa` alongside `wcag22aa`, and the question does not arise in practice.

Not 3.0, which is a working draft with no stable conformance model and is not a compliance target for anyone yet.

This page is the source for that claim. Before it existed the test suite asserted a standard the handbook had never actually written down, which is the kind of gap that looks like a commitment and is really just a comment.

WHAT IS ENFORCED AUTOMATICALLY

`platforms/core/tests/accessibility.spec.ts` runs **axe-core** (https://github.com/dequelabs/axe-core) against four public routes on every CI run, tagged `wcag2a`, `wcag2aa`, `wcag21a`, `wcag21aa`, `wcag22aa`:

Route	What it represents
`/`	the apex marketing page (PublicLayout)
`/sign-in`	the account-creation entry point
`/policies`	the policy index (docs shell)
`/documentation`	the handbook (docs shell + disclosure sidebar)

One page per *layout*, not per URL. Covering thirty pages that share one shell would cost thirty page loads in CI and prove the same thing once.

What a green run does not mean

Automated tooling catches roughly a third of real WCAG failures. It finds contrast, missing accessible names, bad roles, broken landmark structure and unlabelled form fields. It cannot tell you whether the focus order makes sense, whether a live region says anything useful, or whether a screen reader can complete a task.

A worked example from this codebase: the public pages had no skip link for months while CI stayed green the whole time. axe's `bypass` rule passes if a page has a skip link **or** a heading **or** a landmark, and every page had a `<main>`. The rule was right — **Understanding 2.4.1** (https://www.w3.org/WAI/WCAG22/U

The VGSpartans Handbook

nderstanding/bypass-blocks.html) does list ARIA landmarks as a sufficient technique — but a landmark does nothing for a sighted keyboard user who is not running a screen reader, which is exactly who a skip link is for. Green meant "no machine-checkable failure", not "accessible".

WHAT IS BUILT IN

- **Bypass links.** `.skip-link` in `packages/ui/src/styles/spartan/base.css`, rendered by `PublicLayout`, `ConsoleShell` and `ClubLayout`. Hidden until focused, which **WebAIM** (https://webaim.org/techniques/skipnav/) endorses; the target carries `tabindex="-1"` so focus genuinely moves rather than the page merely scrolling. The docs shell passes `skipTarget="docs-main"` because it renders its own header and sidebar inside `<main>`, so a jump to `#main` would bypass nothing.
- **Focus.** One global `:focus-visible` ring, `--gold-deep`, chosen to clear the 3 non-text contrast rule on cream. Brand gold at 2.3 would not.
- **Motion.** A blanket kill under `prefers-reduced-motion`, plus per-component blocks and a separate rule for the view-transition pseudo-elements, which the universal selector never matches.
- **Transparency and contrast.** The glass material flattens to plain opaque surfaces under `prefers-reduced-transparency`, `prefers-contrast: more`, `forced-colors: active` and print.
- **Preferences.** Motion, transparency and text size are also settable on the site itself, for people the media queries cannot reach. The control sits beside the light/dark toggle in every bar the platform draws — the marketing header, the console shell (both the Astro shell and its React twin), the docs header, club sites, VGGallery and VGNews — and inline on `/accessibility`. See below.
- **Widgets.** The FAQ is a real ARIA APG tablist with roving `tabindex` and arrow keys, and it degrades to plain jump links with no JavaScript. The accessibility panel is a disclosure over native radio groups rather than a custom widget.

Why there is a site-level preference layer at all

The media queries are the default and do the work. The preferences only override them, because two groups cannot reach the OS setting a query reads:

- `prefers-reduced-transparency` does not ship in Safari or iOS Safari at all (**WebKit 175497** (https://webkit.org/b/175497), open) and is behind a pref in Firefox. On the platforms where "Reduce Transparency" is the setting people actually use, the query never matches.
- A student on a school-managed Chromebook frequently cannot change OS accessibility settings.

Each preference has three states, not two: absent means "follow the system" and writes nothing, so the OS setting stays live. Only an explicit pick stamps an attribute on `<html>`. Transparency deliberately has no force-on state — an OS "reduce transparency" is a deliberate accessibility choice, and a control to override it would exist only to undo that.

The implementation lives in the pre-paint bootstrap (`packages/ui/src/layouts/ThemeScript.astro`) rather than a bundled module, because a text-size change applied after first paint is a visible reflow. `packages/ui/src/a11y.ts` is a typed delegate over it and holds no logic, mirroring `packages/ui/src/theme.ts`.

This layer is temporary by design. The standards-track answer is **the Web Preferences API** (https://developer.mozilla.org/en-US/docs/Web/API/PreferenceManager), which lets a site override a preference at the *browser*

The VGSpartans Handbook

level so the existing `@media` blocks simply start matching, with no bespoke CSS. It is currently Chrome/Edge only, behind `#enable-experimental-web-platform-features`, unimplemented in Firefox and Safari, and secure-context only. When it ships broadly, this whole layer and its CSS can be deleted.

KNOWN GAPS

Listed because an accessibility page that claims no gaps is not credible.

- **Coverage.** Sixteen-odd public routes are unscanned, including `/platforms`, `/clubs`, `/the-developer`, the per-platform marketing pages, `/report`, `/bug-bounty`, `/404`, and every public subplatform reading page.
- **The help bubble is never scanned.** It mounts a third-party Cloudflare web component into a shadow root on `requestIdleCallback`, long after axe runs on `page.goto`, and only when AI search is enabled. Its internals are not ours to fix and are currently unmeasured.
- **No keyboard or focus-order tests.** The docs shell has a focus trap in its mobile navigation and nothing guards it. Focus order, live-region usefulness and screen-reader task completion are all manual-only today.
- **Heading order.** The shared footer jumps from `h2` to `h5`. axe's `heading-order` rule is tagged `best-practice`, not WCAG, so it is not in the enforced tag set.
- **The ambient wash leaks past "reduce transparency".** Dark bands rebind `--glass-wash-a` on a descendant, which outranks the root-level flattening for that subtree. The panes themselves do go fully opaque; only the decorative background gradient survives. This affects the OS media query and the site preference identically.
- **Text-size steps are modest.** They are a comfort feature, not the mechanism for SC 1.4.4, which asks for 200% and is met by browser zoom. Nothing should cite the panel as satisfying it.

CHANGING SOMETHING

Run the suite before you push:

```
pnpm exec playwright test accessibility --project=chromium
```

If you add a public route with a layout no existing test covers, add it to `PUBLIC_PAGES` in the spec. If you add a glass token, add it to **both** blocks in `packages/ui/src/styles/spartan/theme.css` — the media query and the `[data-glass="off"]` rule are kept in step by hand. If you add a preference group, it has to be added to the allowlist in `packages/ui/src/layouts/ThemeScript.astro`, to `packages/ui/src/public/A11yMenu.astro`, and to its React twin `packages/ui/src/console/ui/A11yMenu.tsx`.

The VGSpartans Handbook

PART 7. CONTRIBUTING

LANGUAGES AND TRANSLATION

How the site is published in more than one language, where the copy lives, and which names must never be translated.

Reference . For developers . Checked 2026-08-04 . /documentation/contributing/translation

---

The public site is published in **English and Spanish**. English is the default and keeps every URL it has ever had; Spanish is served under `/es/`.

Spanish, specifically, because the school is in Casa Grande, Arizona and a large share of the families this platform is for read Spanish at home. The variety is **Latin American**, not peninsular: `reportar` rather than `denunciar`, `computadora` rather than `ordenador`, and `tú` throughout — a school site talking to a student is not writing a formal letter.

WHERE A LANGUAGE LIVES

In the URL, not in a cookie. `astro build` bakes the copy into one HTML file per page, so there is no request-time hook that could swap the words for a returning visitor. That is also the right shape regardless: a language that lives only in a cookie has no address, so it cannot be linked, bookmarked, shared, crawled or cited.

`packages/kernel/src/i18n.ts` is the source of truth for the locale set and owns the four functions every localised link is built from — `splitLocale`, `localePath`, `localeAlternates` and `negotiateLocale`. They are unit-tested in `platforms/core/test/i18n.test.ts`, because a bug in any of them is a wrong `<link rel="alternate">` or a redirect loop, neither of which shows up in a page that renders fine.

`Astro.currentLocale` is how the language reaches a component. It is derived from the URL by Astro's own i18n routing (configured in `platforms/core/astro.config.mjs`), so it works on a prerendered page and nothing has to be threaded through the component tree. It is `string | undefined` — undefined in a Worker with no i18n block, which is every subplatform, since only vgcore publishes languages.

WHERE THE COPY LIVES

Copy	Lives in
Shared chrome: nav, footer, alpha strip, skip link, the accessibility panel	`packages/ui/src/i18n.ts`
A page's own copy	the app that owns the page
Policies	`platforms/core/src/content/policies/<locale>/**`
The handbook	`docs/handbook/<locale>/**`

`en` is the source of truth in the dictionary and every other locale is typed as `Record<keyof typeof en, string>`, so a **missing Spanish key is a build error** rather than a silent English word in the middle

The VGSpartans Handbook

of a Spanish sentence. Do not add a runtime fallback: a fallback hides exactly the bug you need to see.

For content, the locale is a directory *inside* the collection, so the id carries it and the existing loader, schema and route need no new fields.

NAMES THAT ARE NEVER TRANSLATED

`Spartans` is not a word, it is the school's team name. A Spanish page that says "Espartanos" has invented a second name for one thing, and the reader can no longer match what they are reading to the sign on the building, the URL in their address bar, or the app they were told to open. The same is true of every subplatform name (each is also a subdomain someone is told to type), of the school's and the district's legal names, and of exact identifiers like `AGPL-3.0`.

WCAG points the same way: **SC 3.1.2 Language of Parts** (https://www.w3.org/WAI/WCAG22/Understanding/language-of-parts.html) exempts "proper names, technical terms, words of indeterminate language, and words or phrases that have become part of the vernacular of the immediately surrounding text", so these names sit unmarked inside Spanish prose and that is correct rather than an omission.

The list is `PROTECTED_TERMS` in `packages/kernel/src/i18n.ts`, and it is **enforced**: `pnpm check:translations` fails the build on a translated file that renders one of them as a known-bad spelling, and on one that drops a name its English source carries. The script imports the glossary rather than copying it. Add a term the moment you add a feature with a name.

DRAFTING A TRANSLATION

`pnpm translate <file>` produces a Spanish twin of an English Markdown document through DeepL and writes it to the `es/` directory beside it. Translating is **never** automatic: no build step calls DeepL, CI has no key, and what the tool writes is a normal source file that lands in a normal pull request. Only its `--self-test`, which touches no network, is a CI gate.

It exists because the corpus is ~70,000 words. Hand-writing all of it is weeks; machine-translating all of it and shipping it unread is worse than not offering Spanish at all. So it produces a **draft**, and reviewing is still a person's job.

Three things make it safe enough to use:

- **The glossary is generated from `PROTECTED_TERMS`**, each term mapped to itself, which is how you pin a term through DeepL. So the names survive at the API, and `check-translations.mjs` then verifies they survived — two independent layers on the one constraint that cannot slip.
- **Markdown structure never reaches the translator.** Fenced code is skipped whole; a line's structural prefix (`#`, `-`, `1.`, `>`, indentation) is split off and re-attached; inline code, link and image targets, HTML tags and bare URLs become `<x/>` placeholders listed in `ignore_tags`. Link *text* stays in the sentence, because it is prose. In front matter only `title` and `description` are translated — `updated`, `order` and `platform` are data.
- **`--self-test` proves the round trip is lossless**, running decompose-then-recompose with an identity translation and requiring byte-identical output. It needs no key and no network, runs over all 172 source documents in about a second, and is a CI gate. This is the property everything else rests

The VGSpartans Handbook

on: if the plumbing is not lossless, every difference the translator introduces is indistinguishable from a bug in the script. It caught a real one on all thirteen policy files the first time it ran — the raw-HTML pattern was matching the script's own placeholders and nesting them.

```
export DEEPL_API_KEY=...                # host is derived from the key; a free key ends ":fx"
pnpm translate --self-test              # no key needed
pnpm translate <file> --dry-run         # segment and request counts, calls nothing
pnpm translate <file>                   # writes es/<file>; refuses to overwrite without --force
```

The legal documents get read before they ship. A mistranslated retention period in the privacy policy is a false statement to a family, and no checker can see that. The Terms and the Privacy Policy are hand-written and reviewed rather than drafted by the tool.

Two things about DeepL that could not be confirmed from its own documentation and are worth checking before relying on them: whether the **Free** tier includes glossaries, and its monthly character cap. The pricing and support pages did not state either.

THE LANGUAGE CONTROL

It is the first group in the accessibility panel (`packages/ui/src/public/AllyMenu.astro`), above motion, transparency and text size, because it is the one control there that a reader may be unable to read the rest of the panel without.

Its options are **plain links**, not radios: each language is a different URL, so switching is a navigation. That means the control works with no JavaScript at all; the script only records the pick. Each option is labelled in its own language ("Español", never "Spanish") and carries `lang` and `hreflang`, which is what **the W3C's guidance** (https://www.w3.org/International/questions/qa-navigation-select) asks for.

The group renders only when the page declares alternates. A page that is published in one language offers no language control, because offering "Español" on a page with no Spanish version is a link to a 404 dressed up as a feature.

BEING SENT TO YOUR OWN LANGUAGE

On a **first visit only**, the pre-paint bootstrap (`packages/ui/src/layouts/ThemeScript.astro`) reads `navigator.languages`, and if it matches a language we publish and the current page declares a twin, it replaces the URL with that twin. It records the choice at the same moment, so it never fires again.

That bound is deliberate. Redirecting on every load would break the ordinary case of someone deliberately sharing an English link with a Spanish-configured reader: they would be bounced off the page they were sent, every time, with no way to stay. The W3C is explicit that automatic content negotiation "should not be considered a replacement for providing links on a page" — the control in the panel is that link, and this is only the first guess.

Three details that are load-bearing:

- It reads the alternates off the `hreflang` set in the head rather than gluing `/es` onto the current path, so a page with no twin is inert instead of linking to a 404.

The VGSpartans Handbook

- It uses `location.replace`, so Back does not return to the un-asked-for language and bounce forward again.
- It carries the query and fragment across, because those are the reader's position on the page and not part of its language.

ADDING A LANGUAGE

1. Add it to `LOCALES` in `packages/kernel/src/i18n.ts` and to `i18n.locales` in `platforms/core/astro.config.mjs`.
2. Add its block to the dictionary in `packages/ui/src/i18n.ts`. The type will tell you every key you still owe.
3. Add its directory to the content collections.
4. Add the mistranslations you expect to `PROTECTED_TERMS`.

KNOWN GAPS

- **Only the shared chrome and the routing are in place.** The marketing pages, the policies and the handbook are still English-only; their Spanish files have not been written, so no page passes `localized` yet and the language control does not appear.
- **The consoles are not translated and are not planned to be.** They are staff and admin tooling. The React accessibility panel deliberately has no language group.
- **Pagefind indexes one language today.** It is language-aware and keys off `<html lang>`, which is now correct, so a Spanish page will index into its own bundle when one exists.
- **`hreflang` is emitted only where a page opts in** with `localized`. That is the safe default — Google discards an incomplete or one-way set — but it does mean a page that gains a translation and forgets the prop is silently monolingual to a crawler.

The VGSpartans Handbook

PART 7. CONTRIBUTING

TESTING

The four test suites, what each one covers, and how to run them.

How-to . For developers . Checked 2026-07-25 . Unfinished . /documentation/contributing/testing

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the end-to-end, unit, data-layer and live-provider suites.

The VGSpartans Handbook

PART 7. CONTRIBUTING

ADDING A SUBPLATFORM

Everything a new platform needs, from its worker to its routing.

How-to . For developers . Checked 2026-07-25 . Unfinished . /documentation/contributing/adding-a-subplatform

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the full checklist for standing up a new platform.

The VGSpartans Handbook

PART 7. CONTRIBUTING

WRITING DOCUMENTATION

The style guide for this handbook: where pages live, the front-matter, the page anatomy, and the rule against writing anything you have not verified.

How-to . For developers, admins . Checked 2026-07-25 . /documentation/contributing/writing-documentation

IN PLAIN TERMS

This page is the handbook's rules for itself. If you are adding or fixing a page, read it first. The short version: put the file in the right folder, fill in the front-matter, open with a plain-terms box, and never write a fact about the platform that you have not just read in the code.

WHERE PAGES LIVE

Every page is one markdown file under `docs/handbook`. The folder structure IS the sidebar, so where you put the file decides where it appears. There is no separate index to update.

File	Page id	URL
`docs/handbook/index.md`	`index`	`/documentation`
`docs/handbook/operations/index.md`	`operations`	`/documentation/operations`
`docs/handbook/operations/backups-and-restore.md`	`operations/backups-and-restore`	`/documentation/operations/backups-and-restore`
`docs/handbook/operations/incidents/index.md`	`operations/incidents`	`/documentation/operations/incidents`

A folder with an `index.md` is both a page and a section: it is clickable in the sidebar and it opens to show its children.

The nine top-level parts are fixed. Adding a new one means adding a folder AND a line in the `PARTS` list, in `platforms/core/src/lib/handbook.ts`. A top-level folder that is not in that list fails the build rather than quietly appearing at the bottom of the sidebar.

FRONT-MATTER

Every page starts with a front-matter block. The build validates it, so a missing or misspelled field fails the build rather than rendering a broken page.

The VGSpartans Handbook

docs/handbook/security/sessions-and-devices.md

```yaml
---
title: Sessions and devices
description: One sentence, written for search results and link previews.
type: explanation
audience: [admin, developer]
order: 40
updated: 2026-07-25
---
```

Field	Required	What it does
`title`	yes	The page heading, its sidebar entry, and its PDF chapter title
`description`	yes	One sentence. Shown in search results and link previews
`type`	yes	One of `tutorial`, `howto`, `runbook`, `explanation`, `reference`
`audience`	yes	Any of `everyone`, `admin`, `developer`. At least one
`order`	no	Sort key within its folder. Defaults to 100
`updated`	yes	`YYYY-MM-DD`. The date the content was last checked, not the date the file was touched
`stub`	no	`true` on a page that is deliberately not written yet

Pick `type` honestly. A runbook that is really an explanation is worse than either, because someone will open it mid-incident expecting steps.

> ⚠ **WARNING**
>
> Quote the `description` if it contains a colon followed by a space. YAML reads `description: Running it: the basics` as a nested mapping and fails the build with a parse error rather than a useful message. `description: "Running it: the basics"` is fine. The date needs no quotes either way; both spellings are accepted.

PAGE ANATOMY

In order, every page has:

1. **A plain-terms callout**, first thing after the front-matter. Two to four sentences a non-technical reader understands. Use an analogy only if it genuinely helps; a forced one costs more than it earns.
2. **The technical content**, with examples.

The VGSpartans Handbook

3. **Rationale in a `why` callout** where a decision needs defending. If it runs longer than a paragraph or two, it belongs in its own page under `architecture/decisions/` and the callout links there.
4. **Troubleshooting as a table**, if the page has any: symptom, likely cause, fix, how to confirm.

The breadcrumb, the badges, the edit link and the previous/next pair are drawn by the layout. Do not write them into the markdown.

CALLOUTS

Callouts are container directives. Open with `:::name`, close with `:::`.

```
:::plain
Two to four sentences a non-technical reader understands.
:::

:::why
The reasoning behind a decision, so the next person does not "tidy" it away.
:::
```

The available names are `plain`, `why`, `note`, `tip`, `important`, `warning` and `danger`. A name outside that list is left as literal text, so a typo is visible rather than silent.

CODE BLOCKS

A code block that quotes the repository names the file it came from, using `title=` on the fence:

````
```ts title="platforms/core/src/lib/handbook.ts"
export const HANDBOOK_DIR = "docs/handbook";
```
````

That caption is not decoration. It is what turns a snippet from a claim into a citation the reader can go and check.

LINKING

- **To another handbook page**, use its URL: `/documentation/operations/backups-and-restore`. The link checker fails the build on a link to a page that does not exist.
- **To a file in the repository**, put the path in backticks: `` `platforms/core/src/lib/auth.ts` ``. The checker fails the build if the path is not a real file. Do not write repository paths as hyperlinks; they would break the moment the site is read offline or as a PDF.
- **To an external page**, use an ordinary markdown link.

THE RULE ABOUT FACTS

This is the one that matters.

The VGSpartans Handbook

> ⚠ **WARNING**
>
> Never write a statement about how the platform behaves from memory or from inference. Before you write it, open the file that implements it and read it. Then name that file in the page.

That applies to all of it: config values, environment variable names, route paths, table names, script names, button labels, and the order steps happen in. Copy them from the source. Do not type them from recall, and do not reason about what the code "must" do.

If you cannot find the code that proves a claim, you have two honest options and one dishonest one. The honest ones are: leave the claim out, or write it as `TODO(verify): what you could not confirm and why`. The dishonest one is writing a sentence that sounds right. During an incident someone will follow it.

Walkthroughs of a screen get the same treatment. Describe the controls that exist in the page source under `platforms/*/src/pages/`, not the controls a console of that kind would usually have.

STUBS

A page that is not written yet says so, plainly, and points at whatever the real source of truth is today:

```
:::warning
This page is not written yet. The source of truth today is `docs/BUDGET.md`.
:::
```

Set `stub: true` in the front-matter so the page carries a visible notice and the completeness check can count it. Never pad a stub with generic prose to make it look finished. An obviously empty page sends the reader somewhere useful; a plausible empty page wastes their time and then fails them.

VOICE

Follow the repository's style, which is the same one `CONTRIBUTING.md` asks for:

- Plain and direct. Cut anything that does not need to be there.
- Sentence-case headings.
- Second person for instructions. "Open the console", not "the console should be opened".
- No marketing language, no "simply", no "just".
- **No em dashes.** Use a comma, a full stop, or brackets.
- Say the scary part out loud. If a step deletes data, the sentence before it says so.

BEFORE YOU OPEN THE PULL REQUEST

```
pnpm build
```

The VGSpartans Handbook

The build validates the front-matter, fails on a top-level folder that is not a known part, and runs the link and citation checker over every page.

The VGSpartans Handbook

PART 7. CONTRIBUTING

SECURITY POLICY

How to report a security problem in VGSpartans privately, what is in scope, and what happens after you send it.

Explanation . For everyone, developers . Checked 2026-07-25 . /documentation/contributing/security-policy

---

IN PLAIN TERMS

If you find a way to break into VGSpartans or see something you should not, do not post it anywhere public. Report it privately through GitHub's security page or by email. You will get a reply, you will get credit if you want it, and you will not get into trouble for looking, as long as you followed the rules below.

The canonical policy is `SECURITY.md` at the root of the repository. This page is the same policy in the handbook's shape, and if the two ever disagree, `SECURITY.md` wins.

DO NOT REPORT IT IN PUBLIC

That includes GitHub issues, pull requests, and anywhere else people can read them. Describing the problem before it is fixed hands it to whoever reads it next.

HOW TO REPORT

Through GitHub, privately:

1. Open the **Security tab** (https://github.com/vgspartans/platform/security) of the repository.
2. Click "Report a vulnerability", or go straight to `https://github.com/vgspartans/platform/security/advisories/new`.
3. Describe what you found.

That creates a private report only you and the developer can see.

If you would rather use email, the addresses are in `SECURITY.md`.

WHAT TO INCLUDE

The more of this you can give, the faster it gets fixed:

- what the problem is, and what someone could do with it
- the steps to reproduce it: a link, a screenshot, or the exact requests
- the page or file it affects
- a fix, if you happen to have one

The VGSpartans Handbook

WHAT HAPPENS NEXT

Receipt is confirmed within about a week. After that you are kept updated while it is worked on, and the timing of any public discussion is agreed with you. If you want credit, you are named when the fix is published.

WHICH VERSION GETS FIXED

There are no tagged releases, so the only version patched is the current code on `main`. Older commits and other branches are not maintained.

SCOPE

In scope

- the VGSpartans application and the code in this repository
- common web problems: broken logins or permissions, cross-site scripting, injection, exposed private data, CSRF, insecure configuration

Out of scope

- the official Vista Grande or district websites, which this project does not run and cannot change
- problems in services this is built on, such as GitHub or Cloudflare. Report those to them
- anything needing physical access to a device, or tricking the maintainer into clicking something
- denial-of-service testing, raw scanner output with nothing exploitable behind it, and missing-header reports with no working attack

RULES FOR TESTING

While you look into a problem:

- only sign in to accounts you own or have permission to test
- leave other people's data alone. If you reach someone's personal information, stop and say so
- do not take the site down or slow it for other people
- give a reasonable chance to fix the problem before making it public

Follow these rules and act in good faith, and no action will be taken against you for your research.

REWARDS

There is no money. The project is free and run by one student. Credit on publication is offered, with your permission.

The VGSpartans Handbook

THE STANDING PROGRAMME

You do not need to join anything to report something. There is also an invite-only programme with its own console, covered in **The bug bounty console**, carrying your scope, the rules, any tasks, and a private line to the maintainer.

Three things worth knowing before asking for an invite:

- **Use a personal email, not a school one.** Bug bounty access is its own account, kept separate from any school account on purpose.
- **Access is time-boxed**, never more than two months at a time, and it closes itself. Renewals are not automatic.
- **Two-factor is required.** An authenticator app, a passkey, or a security key.

NOT A SECURITY PROBLEM?

- To report abusive or inappropriate content on the site, use the report page at `/report`.
- For an ordinary bug, use the bug page at `/bug-bounty`, or open a GitHub issue.

The VGSpartans Handbook

PART 8

REFERENCE

The lists: variables, config keys, routes, schemas, codes, tokens and scripts.

PART 8. REFERENCE

REFERENCE

The lookup tables: variables, keys, routes, schemas, codes and tokens.

Reference . For admins, developers . Checked 2026-07-25 . Unfinished . /documentation/reference

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover an index of the reference pages and which are generated.

The VGSpartans Handbook

PART 8. REFERENCE

ENVIRONMENT VARIABLES

Every environment variable the platform reads, and what it does.

Reference . For admins, developers . Checked 2026-07-25 . Unfinished . /documentation/reference/environment-variables

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the variable names, their owners, and their effects.

The VGSpartans Handbook

PART 8. REFERENCE

PLATFORM CONFIG KEYS

Every tunable setting stored in the database, with its default.

Reference . For admins, developers . Checked 2026-07-25 . Unfinished . /documentation/reference/platform-config-keys

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the config keys and their defaults.

The VGSpartans Handbook

PART 8. REFERENCE

FEATURE FLAGS AND MODES

The switches that turn parts of the platform on, off, or into observe-only.

Reference . For admins, developers . Checked 2026-07-25 . Unfinished . /documentation/reference/feature-flags-and-modes

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover every mode switch and what each value means.

The VGSpartans Handbook

PART 8. REFERENCE

API ROUTES

Every endpoint the platform serves, and what it takes.

Reference . For admins, developers . Checked 2026-07-25 . Unfinished . /documentation/reference/api-routes

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the route table.

The VGSpartans Handbook

PART 8. REFERENCE

DATABASE SCHEMAS

Every database, every table, and what each one holds.

Reference . For admins, developers . Checked 2026-07-25 . Unfinished . /documentation/reference/database-schemas

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the schema listing per database.

The VGSpartans Handbook

PART 8. REFERENCE

EVENTS AND AUDIT CODES

Every action written to the audit trail, what it means, and how sensitive it is.

Reference . For admins, developers . Checked 2026-08-02 . /documentation/reference/events-and-audit-codes

IN PLAIN TERMS

The platform keeps a permanent record of everything anyone does to it. Each entry names who did it, what they did, what they did it to, and where. This page lists every kind of entry that record can contain, in plain words.

Every row the admin and developer consoles show comes from this table, so the wording here is the wording on screen. The **code** column is what is actually stored in `audit_log.action`; it is what to grep for, and it never changes when the wording does.

WHAT IS RECORDED, AND WHO CAN READ IT

Every entry carries the time in UTC, who acted, what they did, what they did it to, and where they were on the network. Addresses are stored as a keyed token for accounts inside the detail-protection scope, so the same person can still be followed across entries without the address itself being readable.

Sign-in codes, passwords, recovery codes and session tokens are never written to an entry. Where an entry concerns a second factor it names the **kind** of factor used, never the secret.

Entries about sign-in attempts record the address that was typed, including addresses with no account here. That is what makes a run of attempts against many addresses visible as one event rather than as unrelated noise.

The trail is readable only from the admin and developer consoles. The **standard event** column, where present, gives the OWASP Logging Vocabulary name for the same event, so entries can be matched up with other systems' logs.

CLASSES

Every entry carries one of four classes. A class is the console's shorthand for how much attention the entry deserves, not a permission.

Class	Means
Routine	An ordinary action by someone doing their own work.
Privileged	Someone acting on another person's account, or changing how the platform behaves.
Sensitive	Destructive, or security-relevant. Worth reading even when nothing looks wrong.
Info	Recorded for completeness. Nothing changed.

The VGSpartans Handbook

A class shown against a specific entry can differ from the one listed below: the class stored on the entry wins when the code that wrote it recorded something more specific about that particular event. The table gives the verb's usual class.

The VGSpartans Handbook

SIGNING IN AND SESSIONS

Code	What it means	Usual class	Standard event
`attest.failed`	**Browser check failed** A privileged session was turned away because its browser did not pass the console's check.	Sensitive	`authz_fail`
`attest.passed`	**Browser check passed** An admin or developer session proved it was running in a browser the console accepts.	Routine	
`auth.account.created`	**Account created** A first sign-in minted the account and its profile. This is the only way an account comes into existence here.	Privileged	`user_created`
`auth.login.blocked`	**Sign-in refused** A sign-in was stopped at the last step, after the code was accepted. The target says why: the platform was shut to sign-ins, the account is suspended or deleted, or the address never came through sign-up.	Sensitive	`authn_login_fail`
`auth.login.success`	**Signed in** Someone completed a sign-in and got a session. The device check that runs alongside it is recorded separately.	Routine	`authn_login_success`
`auth.logout`	**Signed out** Someone ended their own session.	Routine	`session_logout`
`auth.otp.failed`	**Sign-in code rejected** Someone entered a code that was wrong, expired, or already used. A run of these against one address is what a brute-force attempt looks like.	Sensitive	`authn_login_fail`
`auth.otp.issued`	**Sign-in code sent** A single-use code was emailed to an address that asked for one.	Info	`authn_token_created`
`auth.otp.refused`	**Sign-in code withheld** A code was asked for and not sent. The target says why. The person is told nothing	Info	`authn_login_fail`

The VGSpartans Handbook

Code	What it means	Usual class	Standard event
	either way, so this row is the only record that they asked.		
`auth.sessions.revoked`	**Sessions dropped** Every open session on an account was ended by the platform rather than by the person. The target says what triggered it.	Sensitive	`session_expired`

The VGSpartans Handbook

SECOND FACTOR AND STEP-UP

Code	What it means	Usual class	Standard event
`device.confirmed`	**Confirmed a device** Someone vouched for the machine they are on, so it counts as known on that account from now on.	Privileged	
`device.revoked`	**Removed a device** A device was taken off the account and its sessions dropped. Used when a machine is lost or was never theirs.	Sensitive	`authn_token_revoked`
`mfa.factor.added`	**Added a second factor** Someone enrolled a new way of proving who they are. The target names which kind.	Privileged	
`mfa.factor.removed`	**Removed a second factor** Someone took one of their own factors off the account. The target names which kind. This is a real reduction in that account's protection.	Sensitive	
`mfa.failed`	**Second factor rejected** Someone failed a second-factor check. The target names which kind was tried.	Sensitive	`authn_login_fail`
`mfa.locked`	**Second factor locked** Too many failed attempts in a row, so the account cannot try again for a while. This protects the account and also blocks its owner, so a run of these is worth reading.	Sensitive	`authn_login_lock`
`mfa.password.changed`	**Changed a password** Someone set or replaced their own password.	Privileged	`authn_password_change`
`mfa.recovery.regenerated`	**Reissued recovery codes** A fresh set of one-time recovery codes was generated, which invalidates every code the person held before.	Privileged	
`mfa.verified`	**Confirmed a second factor** Someone proved a second factor. The target names which kind was used.	Routine	`authn_login_success`

The VGSpartans Handbook

Code	What it means	Usual class	Standard event
`stepup.failed`	**Failed to re-confirm identity** A step-up check was not passed, so the privileged action did not run. The target names the action that asked.	Sensitive	
`stepup.issued`	**Asked to re-confirm identity** A privileged action demanded a fresh identity check before it would run. The target names the action that asked.	Info	
`stepup.passed`	**Re-confirmed identity** A step-up check was passed, which clears the way for privileged actions for the next few hours. The target names the action that asked.	Routine	

The VGSpartans Handbook

ACCOUNTS

Code	What it means	Usual class
`deleteUser`	**Deleted an account** The profile was tombstoned and all access revoked.	Sensitive
`grantAdmin`	**Granted admin access** Promoted an account into an administrator role. Only the developer console can do this.	Sensitive
`reactivateUser`	**Reactivated an account** A suspended account can sign in again.	Privileged
`resetMfa`	**Cleared someone's second factor** Removed every second factor and recovery code. They enrol again on their next sign-in, so this is a real reduction in that account's protection until they do.	Sensitive
`securityReset`	**Forced a re-login** Dropped every session on the account and cleared its sign-in code lockout. Used when someone is locked out, or when a session may not be theirs.	Privileged
`setUserProfile`	**Edited someone's profile** An admin changed the display name or scope shown on another person's account.	Privileged
`setUserRole`	**Changed someone's role** Changes what the person can reach across the platform, and resets their sessions so the new role takes effect at once.	Privileged
`suspendUser`	**Suspended an account** The person can no longer sign in, and every session they had open was dropped immediately.	Sensitive

PLATFORM SETTINGS

Code	What it means	Usual class
`setConfig`	**Changed a platform setting** Wrote a value in the platform configuration. The target names which setting and what it became.	Privileged
`setSeatCap`	**Changed a club seat limit** Adjusted how many club seats a role may hold.	Privileged

The VGSpartans Handbook

MODERATION

Code	What it means	Usual class
`cancelPurge`	**Cancelled a permanent delete** Stopped the 72-hour clock, so the removed item's evidence copy is kept.	Privileged
`markContentReport`	**Triaged a report** Marked a member's report on a piece of content as seen, resolved, or dismissed.	Privileged
`resolveFlag`	**Decided a flagged item** Closed something in the moderation queue, either keeping it up or taking it down.	Privileged
`schedulePurge`	**Armed a permanent delete** Started the 72-hour clock after which a removed item and its evidence copy are destroyed for good. Cancellable until it runs.	Sensitive
`takedown`	**Took content down** Removed a piece of content. An exact copy was captured to the evidence locker first, so the decision stays reviewable.	Sensitive
`takedownAlbum`	**Took an album down** Pulled a published album off the site. An exact copy was captured first.	Sensitive
`takedownArticle`	**Took an article down** Pulled a published article off the site. An exact copy was captured first.	Sensitive
`takedownItem`	**Took a gallery item down** Removed a photo or artwork. An exact copy was captured first.	Sensitive
`takedownReported`	**Took down reported content** Removed content after a member reported it, and closed the report.	Sensitive

The VGSpartans Handbook

INVESTIGATION

Code	What it means	Usual class
`forensic.download`	**Downloaded a forensic bundle** Streamed a previously built archive of one person's history.	Sensitive
`forensic.export`	**Built a forensic bundle** Assembled one person's complete device, finding and action history into a downloadable archive.	Sensitive
`investigate.anomalies`	**Read someone's findings** Listed what the security detectors have flagged on one person's account.	Privileged
`investigate.backtestFingerprint`	**Tested a device-match cutoff** Replayed stored device readings against a candidate cutoff to see what it would have reclassified. Reads only, nothing is applied.	Privileged
`investigate.backtestThresholds`	**Tested a detector limit** Replayed stored history against a candidate limit to see what it would have changed. Reads only, nothing is applied.	Privileged
`investigate.concurrent`	**Read someone's session spread** Listed the distinct networks one person's account was active from recently.	Privileged
`investigate.crossUser`	**Searched for a shared device** Looked for other accounts using the same browser fingerprint. This reads across accounts, not just the one under investigation.	Sensitive
`investigate.detectorHealth`	**Read detector accuracy** Read how often each detector is dismissed as wrong. A platform-wide number, not about one person.	Privileged
`investigate.deviceInventory`	**Read someone's devices** Listed every device on record for one person.	Privileged
`investigate.fingerprintCutoffs`	**Read the device-match cutoffs** Read the scores at which a device counts as recognised, or as one we should ask about.	Privileged
`investigate.resolveUser`	**Looked up an account** Turned an email address into the account id the device records are keyed on. The first step of an investigation.	Privileged
`investigate.reviewAnomaly`	**Closed a finding** Recorded a verdict on a security finding, which stops it	Privileged

The VGSpartans Handbook

Code	What it means	Usual class
	applying pressure to that account's sign-ins.	
`investigate.revokeSuppression`	**Un-muted a cause** A silenced finding cause can fire again.	Privileged
`investigate.setFingerprintCutoffs`	**Changed the device-match cutoffs** Retuned how closely a browser must match a known device before the person is trusted, for everyone. A loose value here means an unfamiliar device is treated as a familiar one.	Sensitive
`investigate.setThresholds`	**Changed a detector limit** Retuned when a security detector fires, for everyone. A loose value here quietly weakens a check.	Sensitive
`investigate.signalDiff`	**Compared a device reading** Put one device reading side by side with the account's known device to see what changed.	Privileged
`investigate.snapshots`	**Read someone's device checks** Listed the recent device readings taken on one person's account.	Privileged
`investigate.suppressions`	**Read the muted causes** Listed the finding causes a reviewer has silenced, and for whom.	Privileged
`investigate.thresholds`	**Read the detector limits** Read the tuning values the security detectors fire on.	Privileged
`investigate.timeline`	**Read someone's timeline** Pulled one person's actions, device checks and findings onto a single time axis.	Privileged

The VGSpartans Handbook

SUPPORT TICKETS

Code	What it means	Usual class
`ticket.claim`	**Claimed a ticket** Took ownership of a support ticket so two people do not answer it at once.	Routine
`ticket.close`	**Closed a ticket** Marked a support ticket as finished.	Routine
`ticket.release`	**Released a ticket** Gave a support ticket back to the unclaimed queue.	Routine
`ticket.reopen`	**Reopened a ticket** Put a closed support ticket back into the queue.	Routine
`ticket.reply`	**Replied to a ticket** Added a message to a support ticket thread.	Routine
`ticket.thread`	**Opened a ticket thread** Read the full message history of a support ticket.	Info

BUG BOUNTY

Code	What it means	Usual class
`bounty.post`	**Posted a bounty notice** Published a message to the researcher programme.	Routine
`bounty.postDelete`	**Deleted a bounty notice** Removed a message from the researcher programme.	Routine
`bounty.provision`	**Enrolled a researcher** Granted a time-boxed bug bounty enrollment. There is no self-service path to this.	Privileged
`bounty.reinstate`	**Reinstated a researcher** Lifted a pause on a bug bounty enrollment.	Privileged
`bounty.renew`	**Renewed an enrollment** Extended a researcher's time-boxed access.	Privileged
`bounty.revoke`	**Revoked an enrollment** Ended a bug bounty enrollment. Expiry closes the whole account, not just the programme access.	Sensitive
`bounty.scope`	**Changed the bounty scope** Adjusted what a researcher is authorised to test.	Privileged
`bounty.suspend`	**Suspended a researcher** Paused a bug bounty enrollment.	Sensitive

The VGSpartans Handbook

CLUBS: GOVERNANCE

Code	What it means	Usual class
`appointMember`	**Seated a member** Gave someone a seat and its permissions in a club or newsroom.	Privileged
`approve`	**Approved a club request** An advisor signed off on something a club asked for.	Routine
`createClub`	**Created a club** Registered a new club and its site.	Privileged
`removeMember`	**Removed a member** Took away someone's seat and its permissions.	Privileged
`setClubAdvisor`	**Assigned a club advisor** Named the accountable adult for a club.	Privileged
`setClubInfo`	**Edited club details** Changed a club's name, description or contact details.	Routine
`setClubStatus`	**Changed a club's status** Moved a club between draft, live, paused or quarantined.	Privileged
`setGovernance`	**Changed club governance** Adjusted how a club's roles and approvals work.	Privileged
`setMemberRole`	**Changed a member's seat** Adjusted what a seated member can do.	Privileged
`setPublicationAdviser`	**Assigned a publication adviser** Named the accountable adult for the newsroom or the gallery.	Privileged
`setSiteLive`	**Changed a club site's visibility** Published or unpublished a club's public site.	Privileged

The VGSpartans Handbook

CLUBS: FILES

Code	What it means	Usual class
`checkFile`	**Checked a file** Ran the content checks over an uploaded file.	Info
`createFolder`	**Created a folder** Added a folder to a club's file store.	Routine
`deleteAsset`	**Deleted an asset** Removed an image or file from a club site.	Sensitive
`deleteFile`	**Deleted a file** Removed a file from a club file store or a personal site. The bytes go with it.	Sensitive
`deleteFolder`	**Deleted a folder** Removed a folder and what was in it.	Sensitive
`moveFile`	**Moved a file** Moved a file somewhere else in the file store.	Routine
`moveFolder`	**Moved a folder** Moved a folder somewhere else in the file store.	Routine
`renameFile`	**Renamed a file** Changed a file's name.	Routine
`renameFolder`	**Renamed a folder** Changed a folder's name.	Routine
`saveFile`	**Saved a file** Wrote a change to a page or file on a site.	Routine
`setDefaultAccess`	**Changed the default access** Changed what visibility newly added files start with.	Privileged
`setFileAccess`	**Changed file access** Changed who can see a file.	Privileged
`setFolderAccess`	**Changed folder access** Changed who can see a folder's contents.	Privileged
`upload`	**Uploaded a file** Added a file to a club's file store.	Routine
`uploadAsset`	**Uploaded an asset** Added an image or file to a club site.	Routine
`uploadFile`	**Uploaded a file** Added a file to a personal or staff site.	Routine

The VGSpartans Handbook

CLUBS: SITE CONTENT

Code	What it means	Usual class
`createContent`	**Created site content** Added a new block of content to a club site.	Routine
`deleteContent`	**Deleted site content** Removed a block of content from a club site.	Sensitive
`deleteVersion`	**Deleted a version** Removed a saved version of a site.	Sensitive
`discardLayoutDraft`	**Discarded a layout draft** Threw away an unsaved page layout.	Routine
`markSignupSeen`	**Marked a sign-up seen** Acknowledged someone's request to join a club.	Routine
`pinVersion`	**Pinned a version** Marked a saved version so it is kept.	Routine
`publishContent`	**Published site content** Made a block of content publicly visible.	Routine
`publishLayout`	**Published a layout** Made a page layout live.	Routine
`publishSite`	**Published a site** Made a personal or staff site publicly visible.	Routine
`restoreVersion`	**Restored a version** Rolled a site back to an earlier saved version.	Privileged
`saveLayoutDraft`	**Saved a layout draft** Stored a work-in-progress page layout.	Routine
`saveVersion`	**Saved a version** Stored a restorable snapshot of a site.	Routine
`setContentOrder`	**Reordered site content** Changed the order content appears in.	Routine
`setProfile`	**Edited a club's public profile** Changed what a club shows about itself.	Routine
`setTaxonomy`	**Changed a site's categories** Adjusted the categories content is filed under.	Routine
`setTheme`	**Changed a site's theme** Changed how a club site looks.	Routine
`submitContent`	**Submitted content for review** Sent a draft to whoever signs it off.	Routine
`submitLayout`	**Submitted a layout** Sent a page layout for sign-off.	Routine
`togglePage`	**Turned a page on or off** Showed or hid a page on a club site.	Routine

The VGSpartans Handbook

Code	What it means	Usual class
`unpublishContent`	**Unpublished site content** Took a block of content out of public view.	Routine
`unpublishLayout`	**Unpublished a layout** Took a page layout out of public view.	Routine
`unpublishSite`	**Unpublished a site** Took a personal or staff site out of public view.	Routine
`updateContent`	**Edited site content** Changed a block of content on a club site.	Routine
`updateProfile`	**Edited a site profile** Changed the details shown on a personal or staff site.	Routine

CLUBS: SECRETARY

Code	What it means	Usual class
`addMeeting`	**Added a meeting** Put a meeting on a club's record.	Routine
`addRosterMember`	**Added someone to the roster** Recorded a person as part of a club.	Routine
`deleteMeeting`	**Deleted a meeting** Removed a meeting from a club's record.	Sensitive
`fileMinutes`	**Filed the minutes** Made a meeting's minutes the club's official record.	Routine
`removeRosterMember`	**Removed someone from the roster** Took a person off a club's roster.	Privileged
`saveMinutes`	**Saved draft minutes** Stored a work-in-progress record of a meeting.	Routine
`setAttendance`	**Recorded attendance** Marked who was at a meeting.	Routine
`updateMeeting`	**Edited a meeting** Changed a meeting's details.	Routine
`updateRosterMember`	**Edited a roster entry** Changed what the roster records about someone.	Routine

The VGSpartans Handbook

CLUBS: TREASURER

Code	What it means	Usual class
`addLedgerEntry`	**Added a ledger entry** Recorded money in or out of a club's books.	Routine
`addReimbursement`	**Filed a reimbursement** Recorded a request to pay someone back.	Routine
`deleteBudgetLine`	**Deleted a budget line** Removed a budget line from a club's plan.	Sensitive
`deleteLedgerEntry`	**Deleted a ledger entry** Removed a line from a club's books.	Sensitive
`setBudgetLine`	**Set a budget line** Changed what a club has budgeted for something.	Routine
`setReimbStatus`	**Decided a reimbursement** Approved, paid or rejected a reimbursement request.	Privileged

The VGSpartans Handbook

NEWSROOM

Code	What it means	Usual class
`acceptPitch`	**Accepted a pitch** Turned someone's story idea into an assignment.	Routine
`approveArticle`	**Approved an article** An editor signed an article off as ready.	Routine
`createArticle`	**Started an article** Created a new article draft.	Routine
`createAssignment`	**Created an assignment** Put a story on the assignment board for a reporter to take.	Routine
`createSection`	**Created a section** Added a new section to the newsroom.	Routine
`createShow`	**Created a broadcast series** Added a new broadcast series.	Routine
`declinePitch`	**Declined a pitch** Turned down someone's story idea.	Routine
`deleteAssignment`	**Deleted an assignment** Took a story off the assignment board.	Routine
`deleteCorrection`	**Deleted a correction** Removed a correction before it was posted.	Privileged
`deletePitch`	**Deleted a pitch** Removed a story idea from the pitch list.	Routine
`deletePlate`	**Removed an article photo** Took a photo out of an article.	Routine
`discardArticle`	**Discarded a draft** A writer threw away their own unpublished draft.	Routine
`fileCorrection`	**Filed a correction** Raised a correction against a published article.	Routine
`kickbackArticle`	**Sent an article back** An editor returned an article to its writer for changes.	Routine
`news-hero`	**Set an article's lead image** Chose the image that heads an article.	Routine
`news-media`	**Uploaded newsroom media** Added a photo or file to an article.	Routine
`news-pitch`	**Pitched a story** Someone suggested a story idea to the newsroom.	Routine
`postCorrection`	**Posted a correction** Published a correction on the article it belongs to.	Routine
`publishArticle`	**Published an article** Put an article on the public site.	Routine

The VGSpartans Handbook

Code	What it means	Usual class
`reorderPlates`	**Reordered article photos** Changed the order photos appear in an article.	Routine
`reorderSections`	**Reordered the sections** Changed the order sections appear in, which changes the site's navigation.	Routine
`reorderShows`	**Reordered the broadcast series** Changed the order broadcast series appear in.	Routine
`setFeatured`	**Changed what is featured** Chose which story leads the front page.	Routine
`setFrameCaption`	**Captioned a photo (photo desk)** The photo desk wrote or changed a caption on someone else's piece.	Routine
`setNewsSettings`	**Changed newsroom settings** Adjusted how the newsroom runs.	Privileged
`setPlateCaption`	**Captioned a photo** Wrote or changed the caption on an article photo.	Routine
`setRequireCaptions`	**Changed the caption requirement** Turned on or off whether photos must be captioned before an article publishes.	Privileged
`setRequireReview`	**Changed the review requirement** Turned on or off whether articles need an editor before they publish.	Privileged
`setVisibility`	**Changed an article's visibility** Adjusted who can see an article.	Privileged
`startAssignment`	**Took an assignment** A reporter picked a story off the assignment board and started drafting.	Routine
`submitArticle`	**Submitted an article** Sent an article to an editor for review.	Routine
`unpublishArticle`	**Unpublished an article** Took an article off the public site.	Privileged
`updateArticle`	**Edited an article** Changed an article's text or details.	Routine
`updateAssignment`	**Edited an assignment** Changed an assignment's brief.	Routine
`updateCorrection`	**Edited a correction** Changed the wording of a correction.	Routine
`updateSection`	**Edited a section** Changed a newsroom section's details.	Routine
`updateShow`	**Edited a broadcast series** Changed a broadcast series' details.	Routine

The VGSpartans Handbook

Code	What it means	Usual class
`withdrawCorrection`	**Withdrew a correction** Took a published correction back down.	Privileged

The VGSpartans Handbook

GALLERY

Code	What it means	Usual class
`appointCrew`	**Seated a gallery crew member** Gave someone a seat and its permissions in the gallery.	Privileged
`approveAlbum`	**Approved an album** Signed an album off as ready.	Routine
`createAlbum`	**Created an album** Started a new gallery album.	Routine
`gallery-upload`	**Uploaded to the gallery** Added a photo to an album or the arts wall.	Routine
`kickbackAlbum`	**Sent an album back** Returned an album to its maker for changes.	Routine
`kickbackPiece`	**Sent artwork back** Returned a piece to its maker for changes.	Routine
`publishAlbum`	**Published an album** Put an album on the public site.	Routine
`publishPiece`	**Published artwork** Put a piece on the public arts wall.	Routine
`removeCrew`	**Removed a crew member** Took away someone's gallery seat.	Privileged
`reorderPhotos`	**Reordered an album** Changed the order photos appear in an album.	Routine
`setAlbumAudience`	**Changed who can see an album** Adjusted an album's audience. This is one of the two visibility contracts the gallery runs on, so it decides whether the photos are public.	Privileged
`setAlbumCover`	**Set an album cover** Chose the photo that represents an album.	Routine
`setAlbumFeatured`	**Featured an album** Chose which album leads the gallery.	Routine
`setArtsReviewRequired`	**Changed the arts review requirement** Turned on or off whether artwork needs sign-off before it appears.	Privileged
`setCrewRole`	**Changed a crew seat** Adjusted what a seated gallery crew member can do.	Privileged
`setGallerySettings`	**Changed gallery settings** Adjusted how the gallery runs.	Privileged
`setItemAudience`	**Changed who can see an item** Adjusted a single photo or artwork's audience.	Privileged

The VGSpartans Handbook

Code	What it means	Usual class
`submitAlbum`	**Submitted an album** Sent an album for sign-off.	Routine
`submitPiece`	**Submitted artwork** Sent a piece to the arts wall.	Routine
`unpublishAlbum`	**Unpublished an album** Took an album off the public site.	Privileged
`unpublishPiece`	**Unpublished artwork** Took a piece off the public arts wall.	Privileged
`updateAlbum`	**Edited an album** Changed an album's title or details.	Routine
`updateItem`	**Edited a gallery item** Changed a photo or artwork's details.	Routine
`withdrawPiece`	**Withdrew artwork** Took a piece back off the arts wall.	Routine

The VGSpartans Handbook

THE OPEN SUBPLATFORMS

Code	What it means	Usual class
`agora.comment`	**Commented on an idea** Added a comment to a VGAgora idea.	Routine
`agora.comment.delete`	**Deleted a comment** Removed a comment from a VGAgora idea.	Routine
`agora.idea`	**Posted an idea** Someone put an idea on VGAgora.	Routine
`agora.idea.delete`	**Deleted an idea** Removed an idea from VGAgora.	Routine
`agora.idea.pin`	**Pinned or unpinned an idea** Moved an idea to or from the top of the board.	Routine
`agora.idea.status`	**Changed an idea's status** Moved an idea between open, planned, done and the rest.	Routine
`agora.idea.update`	**Edited an idea** Changed the text of an idea on VGAgora.	Routine
`news-video:abort`	**Abandoned a video upload** A newsroom video upload was cancelled part-way.	Routine
`news-video:begin`	**Started a video upload** Opened a chunked upload for a newsroom video.	Routine
`news-video:finish`	**Finished a video upload** Completed a newsroom video upload and handed it to the content checks.	Routine
`news-video:part`	**Uploaded a video chunk** Sent one piece of a newsroom video. A single upload emits many of these.	Info
`synergy.accept`	**Accepted an answer** Marked a reply as the one that solved the question.	Routine
`synergy.answer`	**Answered a post** Replied to a VGSynergy post.	Routine
`synergy.answer.delete`	**Deleted an answer** Removed a reply from a VGSynergy post.	Routine
`synergy.circle`	**Proposed a circle** Asked for a new VGSynergy circle to be created.	Routine
`synergy.circle.approve`	**Backed a circle** Added an approval to a proposed circle. Ten approvals, or one admin, opens it.	Routine
`synergy.post`	**Posted in a circle** Someone posted to a VGSynergy circle.	Routine
`synergy.post.delete`	**Deleted a post** Removed a post from a VGSynergy circle.	Routine

The VGSpartans Handbook

Code	What it means	Usual class
`synergy.post.edit`	**Edited a post** Changed the text of a VGSynergy post.	Routine
`video:abort`	**Abandoned a video upload** A club video upload was cancelled part-way.	Routine
`video:begin`	**Started a video upload** Opened a chunked upload for a club video.	Routine
`video:finish`	**Finished a video upload** Completed a club video upload and handed it to the content checks.	Routine
`video:part`	**Uploaded a video chunk** Sent one piece of a club video. A single upload emits many of these.	Info
`wiki.delete`	**Deleted a wiki page** Removed a page from VGWiki.	Sensitive
`wiki.edit`	**Edited a wiki page** Saved a change to a VGWiki page.	Routine
`wiki.media.upload`	**Uploaded to the wiki** Added an image or file to VGWiki.	Routine
`wiki.move`	**Moved a wiki page** Changed a VGWiki page's title or location.	Routine
`wiki.protect`	**Protected a wiki page** Restricted who may edit a VGWiki page.	Privileged
`wiki.restore`	**Restored a wiki page** Rolled a VGWiki page back to an earlier revision.	Privileged
`wiki.talk`	**Posted on a talk page** Added a message to a VGWiki page's discussion.	Routine
`writes.chapter`	**Wrote a chapter** Added or changed a chapter of a VGWrites work.	Routine
`writes.comment`	**Commented on a work** Left a comment on a piece of writing.	Routine
`writes.contest_create`	**Created a contest** Opened a writing contest.	Routine
`writes.contest_entry`	**Entered a contest** Submitted a piece of writing to a contest.	Routine
`writes.contest_status`	**Changed a contest's status** Opened, closed or judged a writing contest.	Routine
`writes.create`	**Started a work** Created a new piece of writing on VGWrites.	Routine
`writes.delete_chapter`	**Deleted a chapter** Removed a chapter from a VGWrites work.	Routine

The VGSpartans Handbook

Code	What it means	Usual class
`writes.delete_comment`	**Deleted a comment** Removed a comment from a piece of writing.	Routine
`writes.delete_work`	**Deleted a work** Removed a piece of writing from VGWrites.	Routine
`writes.publish`	**Published a work** Made a piece of writing readable by others.	Routine
`writes.tag_add`	**Added a tag** Tagged a piece of writing.	Routine
`writes.tag_remove`	**Removed a tag** Took a tag off a piece of writing.	Routine
`writes.update`	**Edited a work** Changed a VGWrites work's details.	Routine

PER-USER MODERATION

Code	What it means	Usual class
`moderate.record`	**Opened someone's moderation record** Read one person's full moderation history: what they posted, what was flagged, what others reported, and their trust score.	Privileged
`moderate.restrict`	**Restricted someone from posting** Blocked this account from posting anywhere on the platform until the date given. They can still read and manage their own account.	Sensitive
`moderate.setTrust`	**Adjusted someone's trust score** Changed the reputation score that decides how strictly this person's future posts are checked. A reason is required.	Sensitive
`moderate.takedownItem`	**Took down someone's content** Removed one piece of a person's content from their moderation record. An exact copy was captured first.	Sensitive
`moderate.unrestrict`	**Lifted a posting restriction** This account can post again.	Privileged

WHERE AN ENTRY HAPPENED

The **Where** column reads `Platform` for an action that is not tied to one place, the subplatform's name (`VGNews`, `VGGallery`) for one that is, and the club's own name for anything on a club site. It is derived from `audit_log.club_id`, which carries both real club ids and subplatform tags.

The VGSpartans Handbook

WHAT IS NOT HERE

An action with no wording yet shows in the console as its de-slugged code with `(unworded)` beside it, rather than as a guess. If you see one, the fix is to add it to `packages/services/src/audit-humanize.ts`; the test suite fails when a verb that can reach the audit log has no entry, so this should not happen in practice.

The VGSpartans Handbook

PART 8. REFERENCE

EMAIL TEMPLATES

Every message the platform sends, and when it sends it.

Reference . For admins, developers . Checked 2026-07-25 . Unfinished . /documentation/reference/email-templates

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the template list and their triggers.

The VGSpartans Handbook

PART 8. REFERENCE

ERROR MESSAGES

Errors a person can see, what causes each, and what to do.

Reference . For admins, developers . Checked 2026-07-25 . Unfinished . /documentation/reference/error-messages

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the user-facing error table.

The VGSpartans Handbook

PART 8. REFERENCE

COMMANDS AND SCRIPTS

Every command in the workspace and what it does.

Reference . For admins, developers . Checked 2026-07-25 . Unfinished . /documentation/reference/cli-and-scripts

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the script table.

The VGSpartans Handbook

PART 8. REFERENCE

DESIGN TOKENS

Every colour, size and font variable in the design system.

Reference . For admins, developers . Checked 2026-07-25 . Unfinished . /documentation/reference/design-tokens

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the token table.

The VGSpartans Handbook

PART 8. REFERENCE

DOMAINS AND ROUTING

Every hostname the platform answers on, and which worker answers it.

Reference . For admins, developers . Checked 2026-07-25 . Unfinished . /documentation/reference/domains-and-routing

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the host table.

The VGSpartans Handbook

PART 8. REFERENCE

GLOSSARY

Every term and acronym used across VGSpartans, in one alphabetical list.

Reference . For admins, developers . Checked 2026-07-25 . Unfinished . /documentation/reference/glossary

---

NOT WRITTEN YET

This page has not been written. What follows points at whatever the source of truth is today.

There is no written source of truth for this yet. The code is the only one.

When this page is written it will cover the full alphabetical glossary.