ARCHITECTURE OF GRACE · SETUP COMPANION

Set up *your own Sheet*

A Google Sheet in an account you control, with its own sync script — from a computer, an iPad, or an iPhone. Roughly half an hour, all of it inside your own Google account. Nothing is purchased, nothing is installed, and no data leaves your control at any point.

Who this is for. In a district, usually IT. In a single school, a homeschool, or a private practice, it is simply whoever owns the Google account — the steps are the same either way, once per Sheet. The touch-specific turns are marked along the way.

What this actually is. A Google Sheet in an account you control, plus an Apps Script Web App bound to it that accepts one row per completed student reflection. That is the entire backend. There is no vendor server, no database, and no account for anyone to create.

What lands in the Sheet. One row per reflection: a short student code the student types (not a name), grade, class label, the seasonal window, item responses and three domain scores. No free-text writing, no journal entries — those never leave the student's own device.

Where everything lives

- **The dashboard:** architectureofgrace.org → **Open the Educator Dashboard**.
- **The connect card:** in the dashboard, choose **Set up** ("Hand it out, export, connect") → scroll to **Syncing & distribution** → the **Connect your school's Sheet** card. The code download and two step-by-step guides sit right on this card.
- **Links and QR codes:** the same Set up area → **Distribute** → pick the instrument (Home check-in, Daily check-in, Exit slip, Reflection).

How many Sheets

One Sheet holds every row that reaches it, and the passcode that reads it reads all of them. There is no per-teacher filter in the app. So the number of Sheets is a decision about who may read whose students — not a technical one — and it decides how many times you run the steps below.

If one person runs this — a homeschool, a tutor, a clinician, a single classroom — the question does not arise. One Sheet, both values, kept to yourself. Everything below is about what happens when several adults share one.

For a school with several teachers, one Sheet is right — with the keys split. Run the setup once. Give the AoG lead (or the counselor who reads results) *both* values. Give every other teacher the **write key only**: their classroom links feed the same Sheet, their own dashboard still shows everything their own devices collected, and none of them can read another teacher's students back.

Run it again, on a separate Sheet, when two groups must not see each other's students at all — a related-services caseload, a therapeutic program, two buildings that simply keep their data apart. Teachers can also do this themselves: the dashboard carries the same six steps under *How to create your Apps Script Web App*.

> **The thing not to do:** handing `ADMIN_PULL_KEY` to every teacher and assuming the app narrows what each of them sees. It does not. Every holder of that passcode can pull every row in that Sheet, including students who are not theirs.

Before you start: invent the two keys

Each is a password you make up: 15+ characters, only letters, numbers and dashes, different from each other, one pair per Sheet. Write them down, then copy-paste them everywhere they go — one retyped character fails silently.

`BACKEND_AUTH_KEY`	The **write key**. Lets a device add a row, and only add. It travels inside the classroom links your AoG lead hands out, which is how a student Chromebook or a family phone posts without ever visiting a dashboard. It is not published on the website. Rotate it whenever you like.
`ADMIN_PULL_KEY`	The **pull key**. Lets a teacher dashboard read rows back. It never leaves the staff computer it is typed into, and never rides in a link. Give it only to staff who should see results.

Setup

STEP 1

Create the Sheet

Create a blank Sheet — name it something like *AoG Screener Responses*. Put it in the account that should still own it in three years — an organisation's rather than one person's, wherever you have that choice. **Whoever owns this file owns the data.** Blank is right: the script builds every tab and header row itself, and its tab names must never be changed afterwards.

> **On an iPad or iPhone:** work in **a browser, not the Google Sheets app** — the menu you need in step 2 only exists in the desktop Sheets editor at docs.google.com. Any browser works; an iPad usually gets the desktop editor automatically, and on a phone ask for it first — Safari: **aA** button → *Request Desktop Website*; Chrome: ⋯ menu → *Request Desktop Site*. Then **stay in that one browser for the whole setup**: the connection you save in step 5 lives per-browser, and links generated anywhere else won't carry it.

STEP 2

Open the bound script

In that Sheet: **Extensions ▸ Apps Script**. A project opens named "Untitled project" with a few lines of sample code — that is normal, and the attachment to this Sheet is the whole trick. A project started at script.google.com on its own is attached to nothing and can never work; if you make one by accident, delete it.

Delete the sample `myFunction` stub and paste in the whole of `AoG-Screener-Sync-Code.gs` (the **Download Code.gs** link on the connect card — it opens in its own tab as plain text). Save.

> **Copying the code on a touchscreen:** tap once anywhere in the code text, then tap-and-hold → *Select All* → *Copy*. If Select All grabs only a paragraph, tap-and-hold the selection and choose Select All a second time. In Apps Script: tap into the code area, Select All, paste over *everything*, save.

STEP 3

Add the two script properties

Project Settings (gear) ▸ Script properties ▸ Edit script properties. Add both keys from your card — property names exactly `BACKEND_AUTH_KEY` and `ADMIN_PULL_KEY` — then Save.

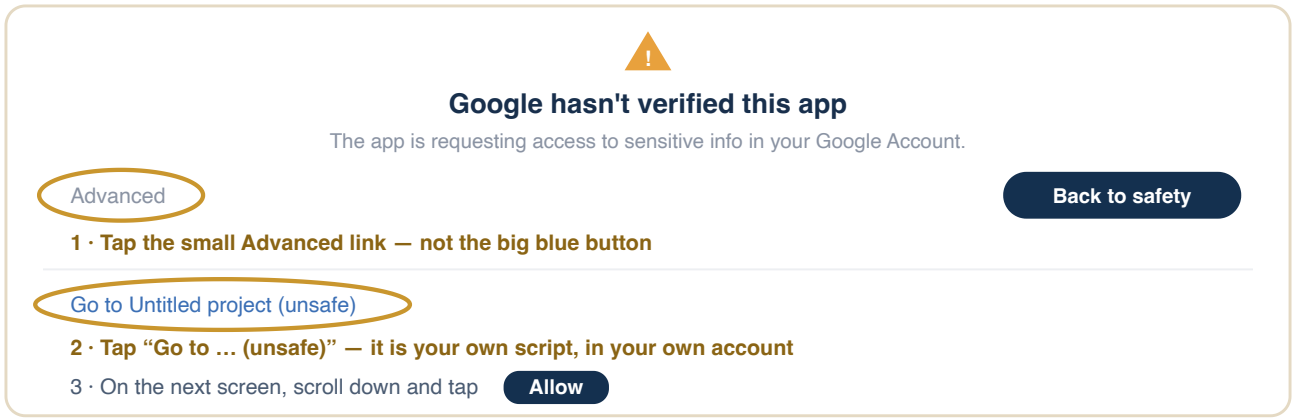
> **The one thing that will bite you:** Apps Script's *Save script properties* replaces the entire property set with whatever rows are on screen — and the table renders empty until you click *Edit script properties*. Add both keys in the same edit, save, then re-open and confirm both are still listed. Saving one on its own silently deletes the other, and submissions start failing with `Unauthorized`.

STEP 4

Deploy it as a Web App

Tap the blue **Deploy** button (top right) → **New deployment**. Then the click everyone misses: the deployment types hide behind a **tiny gear icon next to "Select type."** Tap the gear, choose **Web app**, set *Execute as: Me* and *Who has access: Anyone*, then Deploy.

The warning page is fine, and it appears once. The first deploy asks you to *Authorize access*, pick your account, and then shows a page that looks alarming: *"Google hasn't verified this app."* That is Google being cautious about a script it has not reviewed — *your* script, that you just pasted, in your own account. The way through:



Anyone is required because a student Chromebook posts without a Google sign-in. It does not make the Sheet public: the script rejects any request that does not carry `BACKEND_AUTH_KEY`, and it will not return data to anything but `ADMIN_PULL_KEY`. The deployment URL is the door; the key is the lock.

> **Know your truth window:** in the Apps Script left rail, the **Executions** page with *Show in real time* on shows every arriving submission as a `doPost` within seconds. If no doPost appears, the message went somewhere else — no exceptions. You will use this in step 6.

STEP 5

Connect the dashboard and test

Copy the Web app URL — it ends in `/exec`. On the teacher dashboard, in the same browser: **Set up ▸ Syncing & distribution ▸ Connect your school's Sheet**, paste the URL, put `ADMIN_PULL_KEY` in *Passcode* and `BACKEND_AUTH_KEY` in *Write key*, then Save and Test. Wait for green with no amber sentence. A test row appears in the Sheet; **Pull from sheet** reads it back. Delete the test row when you are satisfied.

> **The two boxes are easy to swap,** especially on a small screen. If Test says *writing works, but that passcode was refused for reading*, the two values are sitting in each other's boxes — swap them and test again.

STEP 6

Make a link and prove it round-trip

Go to **Distribute** (same Set up area) and generate a link there — never type one by hand. A real link is **long**: the Sheet's address rides inside it. A short, clean-looking link carries no destination and will sync to whatever the site's default is instead of your Sheet.

Test it in a **private window** — iPad/iPhone Safari: tabs button → *Private* → new tab; computer: *File* → *New Private/Incognito Window*. (A browser that already holds a saved connection sends to *its* Sheet and ignores every link, so a private window is the only honest test.) Submit once, watch **Executions** for the `doPost`, and look for the instrument's own tab in the Sheet — home check-ins create **HomeCheckins**, daily check-ins **DailyCheckins**, reflections land in **Responses**.

Four rules from the first field test

A link freezes its destination at birth. Whatever URL and write key the connect card held when a link was generated is baked into it forever. Change anything — the URL, a key — and every existing link is stale. Regenerate and re-send them.

A connected browser ignores links. Any browser with its own saved connection sends to its Sheet, no matter whose link it opened. That protects a teacher's own computer — and it means a link can never be tested from a connected browser.

"Sent to the school team" is not proof. A destination broken in just the wrong way (a sign-in-gated or dead deployment) can still produce a Sent message. The Executions page is the proof; the message is the hope.

Only a script born inside the Sheet can write to it. A standalone script.google.com project is attached to nothing, and it will haunt your testing until you delete it.

If nothing shows up

Rows never arrive in the Sheet

Almost always a key mismatch. The write key in the dashboard must equal the Script property exactly — check length first, then re-paste both.

Pull says Unauthorized

The Passcode box holds a typo'd pull key. Re-copy it from Script properties → Edit.

Pull works but you see nothing

Wrong panel. Pull lives on Set up; rows appear under **Student view**, and adult/home records under **Family & adults**.

The screen said Sent, but no doPost appears in Executions

The link aims at the wrong deployment. Match the connect card's URL against **Deploy ▸ Manage deployments**, Save, Test green, then generate a fresh link. Old links can't be repaired, only replaced.

It works in one browser but not another

The failing browser has its own saved connection overriding the link. Disconnect it there (on the connect card), or test in a private window.

Then hand two things to your AoG lead

To whoever runs this day to day — and to nobody outside your school, practice or family:

- The `/exec` Web app URL
- The value of `BACKEND_AUTH_KEY` — the write key only

If that is you, this is one step and not a handover. Both values go in once, on the computer that reads results, at **Set up ▸ Syncing & distribution ▸ Connect your school's Sheet**. From then on the classroom links and QR codes they hand out carry the destination with them, so a student Chromebook or a family phone that has never seen this dashboard still posts into *your* Sheet. Keep `ADMIN_PULL_KEY` narrower still — only the people who should read results back ever need it.

> **What that means for the write key:** because it rides inside the link, treat a classroom link the way you would treat the key itself — hand it out through your own channels, not from a public page. The key can only *append* a row; it cannot read one, change one, or delete one — but anyone holding it can append rows that look real. Rotate it by changing `BACKEND_AUTH_KEY` in Script properties and re-entering it in the dashboard; links carrying the old key stop working the moment you do.

Questions you are likely to have

Where does the data live?

Only in your Sheet, and in the browser of the device that produced it. Architecture of Grace runs as a single static web page — it has no server that could receive a student response even in principle.

Is it identifiable?

Rows carry a short code the student enters; the field is labelled "a code you assign — not their name," and the app challenges anything that looks like a full name. It is pseudonymous, not anonymous — whoever holds the roster mapping can re-identify a student, which is exactly what makes it useful to a teacher and exactly why the Sheet belongs to you and not to a vendor.

Does anything reach the people who wrote the software?

No. Neither key is sent anywhere outside your own account, and neither is stored on the website. Architecture of Grace is a static page with no backend of its own — there is nothing on it that could receive a key. Both values live in your Apps Script properties, in the browser of the computer that entered them, and inside the links that computer generates.

What if we want it switched off?

Undeploy the Web App, or rotate `BACKEND_AUTH_KEY`. Writes stop immediately, and reflections simply stay on each device as they do by default.

Does this need a purchase, a contract, or a DPA review?

There is nothing to buy and no vendor to contract with. Whether it needs a data review is your call, not ours — a district has a process for this, a private practice has its own obligations, and a family has neither. The request is only that the answer be written down before student information is collected.

Architecture of Grace is independent work by its author. This document describes a setup inside your own Google account; it is not the publication of any school or district and carries no endorsement from one. Files referenced: `AoG-Screener-Sync-Code.gs` · `aog-sync-config.js` · the dashboard at architectureofgrace.org