ARCHITECTURE OF GRACE

IT Deployment & Data Guide

Universal Check-In — Hosting, Optional Google Sheet Sync, and Public Voices Wall

CRITICAL LEGAL & SECURITY NOTICE — READ BEFORE DEPLOYMENT Before enabling sync or distributing this tool to students, the district must complete its own privacy and legal review and execute a Student Data Privacy Agreement (DPA) with Architecture of Grace. Enabling sync without a signed DPA may violate district policy, FERPA, COPPA, or state student data privacy laws. The passcode described in this guide is visible in the page source and is a convenience control only — not strong security. This document does not constitute legal advice.

What This Document Covers

The Architecture of Grace Universal Check-In is a single, self-contained HTML file (index.html). It requires no server, database, or build step. By default, every response is stored privately in the viewer's own browser and nothing leaves the device. A school or district can optionally enable sync so completed check-ins are sent to a Google Sheet the district owns and controls. The page also includes an optional public comment wall ("Voices"). This guide covers all three scenarios.

1. Data & Privacy Model

Local-First by Default

Responses save to the browser's localStorage on the device used. With no Sheet configured, nothing is transmitted anywhere. The Provider (Architecture of Grace) operates no backend and has no access to on-device data.

Optional Sync to a District-Owned Sheet

When configured, a completed check-in is POSTed to the district's own Google Apps Script, which appends a row to a Google Sheet the district controls. No third-party server sits in this path. Architecture of Grace runs no backend for live school deployments.

Pilot note. In the current pilot, sync and the Voices wall may point to a Google Sheet on Architecture of Grace's own (personal Google) account used for testing. Before any student use, change the destination to the district's own Workspace account so the data lives under district control.

Optional Public Comment Wall ("Voices")

When VOICES_URL is configured, the page shows a public "Voices" wall independently of check-in sync. Posts go to a separate "Comments" tab in the same Sheet and are shown publicly. See Section 6 for important governance considerations.

Use an institutional Google account. The Sheet should live on a school or district Google Workspace account, not a personal Gmail. Student wellbeing data belongs to the institution under its own access controls and retention policy.

De-identification is supported. The check-in is designed to be taken with initials, a seat number, or a roster ID rather than full names.

2. Hosting the Check-In File

The check-in is one static HTML file, so it can be hosted anywhere that serves static files. Viewers' saved results stay on their own devices and survive redeploys.

Option A — Netlify (Recommended for Simplicity)

1. Rename the file to index.html if it is not already named that.

2. Go to app.netlify.com, sign in (or create a free account on an institutional email).

3. Open the Deploys tab for the site and drag the file (or the whole site folder) onto the deploy drop zone.

4. Wait for the status to read "Published," then open the site link and hard-refresh.

5. To update later, drag the new index.html onto the same site — the link stays the same and saved results are unaffected.

Option B — District-Hosted or Other Static Hosts

The file also runs on Cloudflare Pages, GitHub Pages, or any district web server or intranet share that serves HTML. There is no special runtime to install. If the Downloads library is used, host those files alongside the page (e.g., a /files folder) or on Google Drive / Dropbox set to "anyone with link."

3. Optional: Collect Results in One Google Sheet

SECURITY & GOVERNANCE WARNING — READ CAREFULLY The Apps Script uses a PASSCODE for basic access control. You MUST change the default value "sky" to a strong, unique passphrase before deploying. Because this passcode is embedded in the client-side HTML file, it is visible to anyone who views the page source. It is a convenience gate, not strong security. Restrict who holds the published HTML file and generated class links. Configure the receiving Google Sheet's own sharing permissions appropriately. For higher-security needs, use a hosted version behind district SSO or a custom server-side solution. Schools must complete privacy/legal review and execute a DPA before enabling sync.

This step is only needed if a school wants completed check-ins to gather centrally rather than staying on each device. The file ships local-only by design. Turning sync on is a deliberate governance decision — do it only after the district's data-privacy agreement is signed and its privacy review is complete.

Step 1 — Create the Sheet on an Institutional Account

On the school/district Google Workspace account, create a new Google Sheet. Do not use a personal Gmail account for student wellbeing data.

Step 2 — Open Apps Script and Paste the Receiver

In that Sheet: Extensions → Apps Script. Delete any placeholder code.

Paste the entire Code.gs from the Appendix of this document, then Save.

REQUIRED: Set the passcode line (const PASSCODE) to a strong, unique value of your own — not the demo value "sky" — and make it match the screener's SCHOOL_SYNC_KEY exactly.

Step 3 — Deploy as a Web App

Click Deploy → New deployment, choose Web app.

Set Execute as: Me and Who has access: Anyone.

Click Deploy and approve the permission prompt.

Copy the Web App URL it returns (it ends in /exec).

Important: If you change the script later, use Deploy → Manage deployments → Edit (pencil) → Version: “New version” → Deploy. That keeps the same /exec URL. Choosing “New deployment” instead creates a brand-new URL that must be pasted back into the HTML file.

4. Wire the File to the Sheet

Open index.html in any text editor and set these values near the top of the script section, then re-host the file. These are the only edits needed.

Setting	Value to Set	Purpose
SCHOOL_SYNC_URL	.../exec web-app URL	Where completed check-ins are sent
SCHOOL_SYNC_KEY	your-strong-passcode	Must match PASSCODE in Code.gs
ACCESS_CODE	educator	Unlocks the results dashboard

To turn sync ON: Paste your /exec URL into SCHOOL_SYNC_URL and set SCHOOL_SYNC_KEY to a strong passcode that matches Code.gs. To keep check-ins local-only, leave SCHOOL_SYNC_URL empty. The public Voices wall is controlled separately by VOICES_URL (set it to your /exec URL to show the wall, or leave it empty to hide the wall). The two features are independent.

5. How Sync Behaves (for Support)

Write: A completed check-in POSTs the record plus the passcode to your /exec URL; the script appends one row to the records tab.

Local copy is the source of truth: Records are saved on-device first and re-sent if a send fails, so nothing is lost if the network drops.

Read-back: In the results dashboard, “Pull from sheet” requests rows back (passcode-gated) and merges them in, marked “from sheet,” without overwriting local records.

Bare-URL device stays private: Sync turns on only when a destination is configured or the link carries managed parameters.

Privacy note on read-back. The dashboard’s read-back has two paths. The preferred one sends the sync passcode in the request body (private). A fallback path sends it as a URL parameter (?key=…), which can appear in server logs and browser history. For an internal sync key this is low-risk, but it is disclosed so IT can decide: prefer the body path, and use the URL-parameter fallback only for one-time setup testing, not for ongoing reads.

6. Scaling to a District (500+ Buildings): Central Directory & Identity

A single Google Sheet is ideal for one school. Across a large district — hundreds of buildings — maintaining a separate Sheet, script, and passcode for every building becomes the data-management bottleneck. Two provided files turn that into a hub-and-spoke model that district IT stands up once: AoG-District-Directory-Sync.gs (the central directory and record router) and aog-identity.js (the student

identity layer for the check-in page). This section is for district IT; a single-school pilot can ignore it and use Sections 3–5 as written.

The shift — one hub, not 500 sheets

Instead of wiring each building to its own Sheet (point-to-point, which gets unmanageable at scale), the district deploys ONE Apps Script Web App backed by ONE Sheet. A Directory tab becomes the system of record for which schools exist; it is populated automatically from the district's authoritative source — an SIS export, OneRoster, Clever, or ClassLink — never by hand, building by building. Each completed check-in is routed to a per-school results tab that is created automatically on first write. IT manages one deployment, not five hundred. The record schema, the passcode-gated write and read-back contract, and the on-device-first privacy model from Sections 1 and 5 are all unchanged.

Step 1 — Deploy the directory once

1. On an institutional Google Workspace account (not a personal Gmail), create ONE Google Sheet — the district hub and system of record.

2. Extensions → Apps Script. Delete any placeholder code and paste the entire AoG-District-Directory-Sync.gs provided alongside this guide. It is a superset of the single-school Code.gs in the Appendix and stays backward-compatible with existing per-school deployments.

3. Project Settings (gear) → Script properties. Add DISTRICT_ID (e.g. d204) and ADMIN_PASSCODE (a strong district-IT secret that guards roster import and district rollups). Optionally add AUTO_PROVISION = true to accept a first write from a not-yet-listed school (still admin-gated).

4. Deploy → New deployment → Web app. Set Execute as: Me and Who has access: Anyone, then Deploy and authorize. Copy the /exec URL — this single URL serves every building in the district.

Step 2 — Import the school roster from your SIS

This one import replaces hundreds of manual building setups. POST your school list to the /exec URL with the admin passcode. The script auto-detects and normalizes native rows, OneRoster orgs, or Clever schools, and auto-issues each school its own passcode in the Directory tab.

```
{ "action":"roster", "adminPasscode":"...", "orgs":[ {"sourcedId":"a01","name":"North"}, ... ] }
```

Re-running the import keeps each school's existing passcode. Add "deactivateMissing": true to retire (not delete) schools that drop out of the feed — a SCIM-style deprovision: the row is kept and marked inactive. Then hand each building its line from the Directory: its schoolId, its passcode, and the one shared /exec URL. In the AoG dashboard (Section 4) staff paste those three values — no new Sheet, script, or deployment per building.

Step 3 — Connect students (identity)

To attach a roster of 550+ students without building traditional user accounts, the platform offers two paths, both handled by aog-identity.js. Option B (tokens) is the default and preserves the no-account, no-email privacy posture described in Section 1. Option A (Google SSO) is opt-in and off unless a district turns it on.

Option B — Roster-link tokens (default, privacy-first)

1. Enroll a school's students by POSTing a CSV (or array) of anonymized codes with that school's passcode. The script mints one opaque token per student and is idempotent — re-running keeps every token stable.

```
{ "action":"enrollRoster", "schoolId":"a01", "passcode":"…",
"csv":"studentId,classId,grade\nStudent_A01,RM12,3\n…" }
```

2. Build each student's link by appending ?t=<token> to your hosted page URL. Print these as personalized QR strips or share them in Google Classroom. Export a school's tokens any time with { "action":"tokens", "schoolId":"…", "passcode":"…" }.

3. When a student opens their link, aog-identity.js resolves the token to their anonymized code and fills it in for them automatically — no login and no email. The raw code never appears in the URL; only the opaque token does. A take-home device carrying just the token can still sync its single row.

Option A — Google Workspace SSO (opt-in)

If a district prefers single sign-on and accepts collecting verified student emails server-side, enable SSO. In Script properties set SSO_ENABLED = true and, strongly recommended, HOSTED_DOMAIN to your students' Google domain (e.g. students.d204.org) to restrict who can sign in. Provide your Google OAuth client ID to the page via an AOG_SSO_CONFIG value. The Student ID code box is then replaced by a "Sign in with your School Google Account" button; the Google credential is verified server-side and matched to a roster spot.

Privacy note. Even with SSO on, only the anonymized code is stored in results and synced — the email is used solely to look up the student and is never written to a results tab or returned to a student device. Enabling SSO does store verified student emails in the Roster tab; treat that as a policy decision and reflect it in your Data Privacy Statement and DPA before turning it on. Leaving SSO off keeps the tool strictly email-free.

What this unlocks

With identity attached, a student's history follows them across devices: a child who switches from a school Chromebook to a tablet at home keeps their check-in trajectory and saved "What Helps Me" calming tools, securely. The dashboard can track individual cohort trajectories across multiple years automatically — surfacing growth or burnout signals across a whole building without a teacher pulling records one by one. The district rollup, { "action":"rollup", "key":"<admin passcode>" }, aggregates every school's de-identified rows for the District / Admin view.

Endpoint reference (district directory)

Action	Who calls it	What it does
roster	District IT	Import and normalize schools from the SIS; auto-issue each school a passcode
enrollRoster	School / IT	Mint stable link tokens from a roster of anonymized student codes
tokens	School / IT	Export a school's tokens for printing personalized QR strips
resolve	Student device	Exchange a link token for the anonymized code (carries no PII)
resolveEmail	Student device	Verify a Google sign-in and match it to a roster spot (SSO only)
pull	Dashboard	Read back one school's rows, passcode-gated
rollup	District / Admin	Aggregate every school's de-identified rows
(write)	Any device	Append one check-in, routed to its school's tab

Script properties reference

Script property	Set to	Purpose
DISTRICT_ID	e.g. d204	Stamped on every synced record
ADMIN_PASSCODE	a strong secret	Guards roster import and district rollup
AUTO_PROVISION	true (optional)	Accept a first write from a new school (admin-gated)
SSO_ENABLED	true (optional)	Turns on Option A (Google SSO); off by default
HOSTED_DOMAIN	e.g. students.d204.org	Restricts SSO sign-in to your Google domain

7. The Public “Voices” Comment Wall — Governance Considerations

The page includes an optional public comment wall (“Voices”) for testimonials. It is active whenever VOICES_URL is configured, independently of check-in sync. Posts go to a separate “Comments” tab in the same Sheet and are displayed publicly on the page and in a short strip on the welcome screen.

Public by design. Any visitor — including students, if the page is open to them — can post a short comment (optional first name, a role, and a message). No account is required.

Built-in filtering. The page and the Apps Script both reject any post containing an email address, phone number, or link, and block common profanity. Length is capped at 300 characters.

Moderation. The wall is open — posts appear immediately. To remove one, open the Comments tab and either delete its row or put TRUE in the hidden column. The operator of the Sheet is responsible for monitoring the wall while it is live.

Isolation. The comment endpoints are public (no passcode) but can only touch the Comments tab. Student records in Sheet1 stay passcode-gated and are never exposed by the comment path. *The LEA should decide deliberately whether to keep the wall enabled, assign someone to monitor it, and treat it as public communication rather than protected Student Data.*

Appendix — Apps Script Receiver (Code.gs)

Before you paste this: Change const PASSCODE = "sky"; to your own strong, unique passphrase before deploying, and make it match SCHOOL_SYNC_KEY in index.html exactly. The passcode is a convenience gate, not strong security.

Version note: This code is current as of June 2026. Before any large-scale deployment, check https://architectureofgrace.org/ or contact Theopus77@yahoo.com to confirm you have the latest version.

```
/**
 * Architecture of Grace — Check-In sync + public comment wall (Google Apps Script)
 * ---------------------------------------------------------------------------
 * This web app does two separate things, kept deliberately apart:
 *
 *   1. STUDENT RECORDS (private)  — passcode-locked. The screener sends each
 *      completed check-in here; the dashboard reads them back. Tab: "Sheet1".
 *
 *   2. PUBLIC COMMENT WALL (public) — NO passcode. Anyone can post a short
 *      comment and read the wall. Tab: "Comments". The comment code can ONLY
 *      ever touch the Comments tab — it never reads or returns student records.
 *
 * Safety built into the public wall (server-side, can't be bypassed by the page):
 *   - rejects any message/name containing an email, phone number, or link
 *   - rejects basic profanity/slurs
 *   - caps length, trims, stores a "hidden" flag
```

```
 *   - you moderate by editing the Comments tab: set hidden = TRUE (or delete the
 *     row) and the comment vanishes from the wall.
 *
 * UPDATING (keep the SAME link):
 *   Extensions -> Apps Script -> paste this whole file over the old one -> Save.
 *   Then Deploy -> Manage deployments -> (pencil/Edit) -> Version: "New version"
 *   -> Deploy. That keeps your existing .../exec URL, so nothing needs rewiring.
 */

// ⬇ MUST match SCHOOL_SYNC_KEY inside index.html
const PASSCODE = "sky";

const SHEET_NAME    = "Sheet1";      // student records (private) — your first tab
const COMMENTS_NAME = "Comments";    // public wall

const HEADERS = [
  "timestamp", "studentId", "context", "grade", "window",
  "districtId", "schoolId", "classId", "language", "mode",
  "domainA", "domainB", "domainC", "composite",
  "normA", "normB", "normC", "normComposite",
  "tier", "trustedAdultFlag",
  "reflection1", "reflection2", "reflection3", "closingWord",
  "raw", "intensities"
];
const COMMENT_HEADERS = ["timestamp", "name", "role", "message", "hidden"];

const COMMENT_MAX = 300;     // characters
const NAME_MAX    = 40;
const ALLOWED_ROLES = ["Teacher", "Parent", "Student", "Counselor", "Administrator", "Other"];

// ---- helpers ---------------------------------------------------------------

function _json(obj) {
  return ContentService.createTextOutput(JSON.stringify(obj))
    .setMimeType(ContentService.MimeType.JSON);
}

function _sheet() {
  const ss = SpreadsheetApp.getActiveSpreadsheet();
  let sh = ss.getSheetByName(SHEET_NAME);
  if (!sh) sh = ss.insertSheet(SHEET_NAME);
  if (sh.getLastRow() === 0) { sh.appendRow(HEADERS); sh.setFrozenRows(1); }
  return sh;
}

function _commentsSheet() {
  const ss = SpreadsheetApp.getActiveSpreadsheet();
  let sh = ss.getSheetByName(COMMENTS_NAME);
  if (!sh) sh = ss.insertSheet(COMMENTS_NAME);
  if (sh.getLastRow() === 0) { sh.appendRow(COMMENT_HEADERS); sh.setFrozenRows(1); }
  return sh;
}

function _recordToRow(rec) {
  const refl = Array.isArray(rec.reflections) ? rec.reflections : ["", "", ""];
  const get = function (key) {
    switch (key) {
      case "reflection1": return refl[0] != null ? refl[0] : "";
      case "reflection2": return refl[1] != null ? refl[1] : "";
      case "reflection3": return refl[2] != null ? refl[2] : "";
      case "raw":         return JSON.stringify(rec.raw || []);
      case "intensities": return JSON.stringify(rec.intensities || []);
      default:            return rec[key] != null ? rec[key] : "";
    }
  };
  return HEADERS.map(get);
}

function _allRecords() {
  const sh = _sheet();
  const last = sh.getLastRow();
  if (last < 2) return [];
  const width = HEADERS.length;
```

```
  const values = sh.getRange(2, 1, last - 1, width).getValues();
  return values.map(function (row) {
    const o = {};
    for (let i = 0; i < width; i++) o[HEADERS[i]] = row[i];
    return o;
  });
}

// ---- public comment wall (isolated; never touches records) ----------

// Returns "" if clean, or a short reason string if the text must be blocked.
function _blockReason(text) {
  const t = String(text || "");
  if (/[\w.+-]+@[\w-]+\.[\w.-]+/.test(t)) return "no_email";
  if (/(https?:\/\/|www\.)/i.test(t)) return "no_link";
  if (/[\w-]+\.(com|net|org|io|co|edu|gov|us|me|app|xyz|info|biz|tv|ly)\b/i.test(t)) return "no_link";
  // 7+ digits (allowing spaces/dashes/parens) looks like a phone number
  if (/(?:\d[\s().+-]?){7,}/.test(t)) return "no_phone";
  const bad =
["fuck","shit","bitch","cunt","asshole","nigger","nigga","faggot","fag","retard","whore","slut","dick","pussy","bastard","damn
you","kys"];
  const low = " " + t.toLowerCase().replace(/[^a-z\s]/g, " ") + " ";
  for (let i = 0; i < bad.length; i++) { if (low.indexOf(" " + bad[i] + " ") >= 0) return "blocked"; }
  return "";
}

function _commentAdd(body) {
  let msg  = String(body.message || "").replace(/\s+/g, " ").trim();
  let name = String(body.name || "").replace(/[\r\n]+/g, " ").trim();
  let role = String(body.role || "Other").trim();
  if (!msg) return _json({ error: "empty" });
  if (msg.length > COMMENT_MAX) msg = msg.slice(0, COMMENT_MAX);
  if (name.length > NAME_MAX) name = name.slice(0, NAME_MAX);
  if (ALLOWED_ROLES.indexOf(role) < 0) role = "Other";
  const reason = _blockReason(msg) || _blockReason(name);
  if (reason) return _json({ error: reason });
  _commentsSheet().appendRow([new Date().toISOString(), name, role, msg, false]);
  return _json({ ok: true });
}

function _commentsList() {
  const sh = _commentsSheet();
  const last = sh.getLastRow();
  if (last < 2) return [];
  const values = sh.getRange(2, 1, last - 1, COMMENT_HEADERS.length).getValues();
  const out = [];
  for (let i = 0; i < values.length; i++) {
    const r = values[i];
    const hidden = (r[4] === true || String(r[4]).toUpperCase() === "TRUE");
    if (hidden) continue;
    out.push({ timestamp: r[0], name: r[1], role: r[2], message: r[3] });
  }
  out.reverse();              // newest first
  return out.slice(0, 100);   // cap
}

// ---- web app entry points -------------------------------------------------

function doPost(e) {
  try {
    const body = (e && e.postData && e.postData.contents) ? JSON.parse(e.postData.contents) : {};

    // PUBLIC comment endpoints — no passcode, Comments tab only.
    if (body.action === "comment_add")   return _commentAdd(body);
    if (body.action === "comments_list") return _json({ comments: _commentsList() });

    // Everything below concerns student records and REQUIRES the passcode.
    if (String(body.passcode) !== String(PASSCODE)) return _json({ error: "bad passcode" });

    if (body.action === "pull") return _json({ records: _allRecords() });

    const sh = _sheet();
    const key = String(body.timestamp) + "|" + String(body.studentId);
```

```
    const last = sh.getLastRow();
    let foundRow = 0;
    if (last >= 2) {
      const keys = sh.getRange(2, 1, last - 1, 2).getValues();
      for (let i = 0; i < keys.length; i++) {
        if (String(keys[i][0]) + "|" + String(keys[i][1]) === key) { foundRow = i + 2; break; }
      }
    }
    const row = _recordToRow(body);
    if (foundRow) sh.getRange(foundRow, 1, 1, HEADERS.length).setValues([row]);
    else sh.appendRow(row);
    return _json({ ok: true });
  } catch (err) {
    return _json({ error: String(err) });
  }
}

function doGet(e) {
  const p = (e && e.parameter) ? e.parameter : {};
  // PUBLIC: read the comment wall — ...exec?comments=1
  if (p.comments) return _json({ comments: _commentsList() });
  // PRIVATE: read records — ...exec?key=PASSCODE
  if (String(p.key) !== String(PASSCODE)) return _json({ error: "bad passcode" });
  return _json({ records: _allRecords() });
}
```

Last updated: June 2026. Questions: Theopus77@yahoo.com