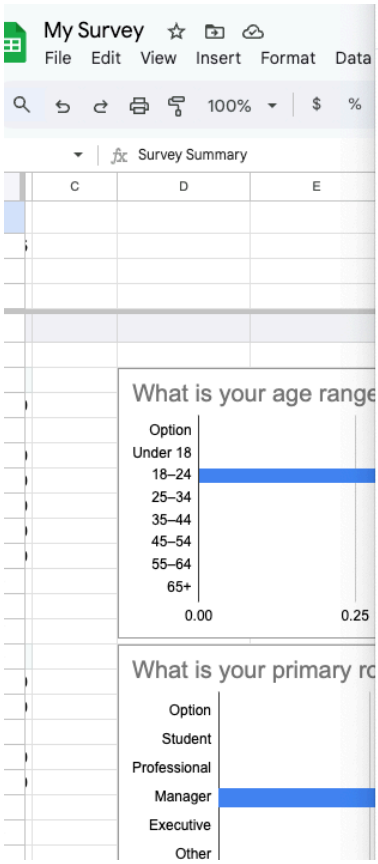
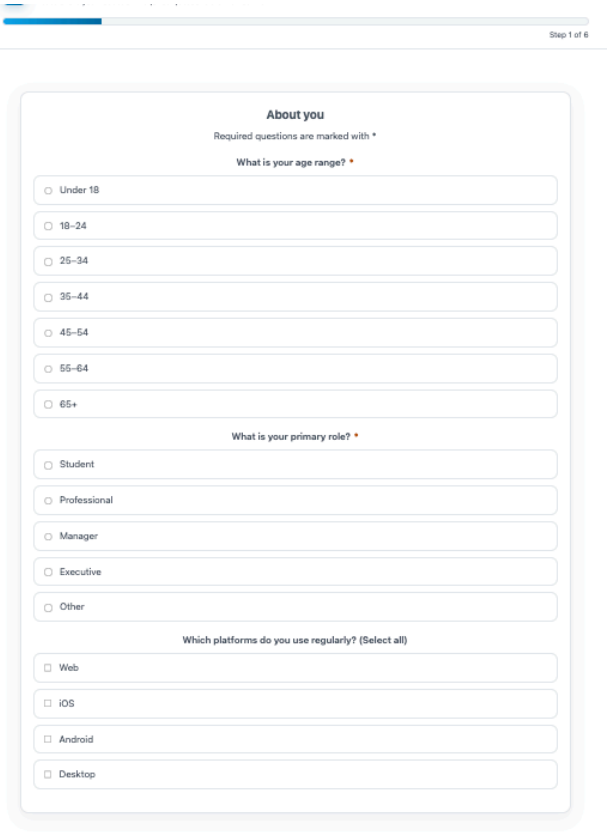
Google Apps Script Dynamic Sheets Data Survey





Introduction

This walkthrough shows you how to build a **data-driven survey web app** on top of **Google Sheets + Google Apps Script**—from a blank spreadsheet to a polished, multi-page form with a review step, clean submissions, and an optional one-click summary dashboard with charts.

Unlike Google Forms, this approach gives you full control over the look, behavior, and data model. The survey is driven by a simple `Questions` sheet (no code changes needed to add, remove, or reorder questions), and every submission is stored as **one readable row** in a `Responses` sheet (perfect for analysis, exports, and dashboards).

Who this is for

- People who want a **custom survey UI** without running servers.
- Teams who prefer **Sheet-first** workflows and easy data access.

- Beginners who are comfortable copying/pasting code and following clear steps.

What you'll build

- A **multi-page** survey with a progress bar and **Review & Submit** step.
- A **wide-format** `Responses` table (one submission per row, readable columns).
- (Optional) A **Summary** sheet with charts and a **Verbatims** tab for open-ended answers.

Why this pattern works

- **Data-driven**: Change questions in Sheets, not in code.
- **Maintainable**: Question "codes" become column headers—easy to read, filter, pivot, and chart.
- **Robust**: Includes locking and retry logic to handle concurrent submissions.
- **Extensible**: Add validation, conditional logic, custom styling, or connect to other tools later.

What you need

- A Google account with access to **Google Sheets** and **Apps Script**.
- A new (or existing) Google Sheet where you'll add two tabs: `Questions` and `Responses`.
- Willingness to copy/paste four small files: `Index.html`, `Styles.html`, `Script.html`, and `Code.gs` (plus an optional `Summary.gs`).

How the guide is organized

1. **Set up your Sheet** (create tabs and headers; optional seeder to add 10 example questions).
2. **Add the Apps Script files** (copy–paste and save).
3. **Deploy the Web App** (choose "Execute as: Me" and "Anyone with the link" or your domain).
4. **Customize your survey** (edit the `Questions` tab—no code changes required).
5. **Collect responses** (each submission is a new row in `Responses`).
6. **Summarize & chart** (run the optional Summary builder to create charts and verbatims).

Accessibility & UX considerations

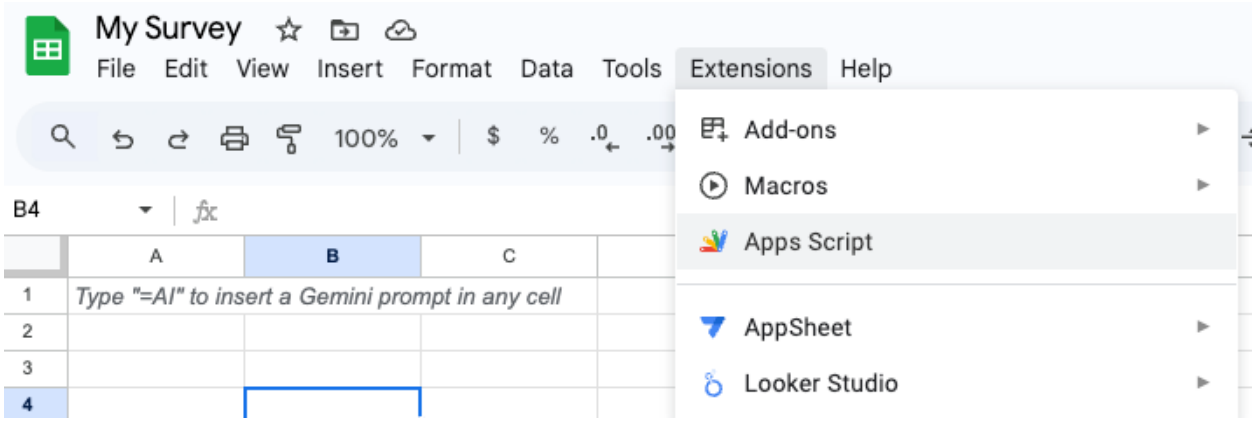
- The default stylesheet uses a **light, high-contrast theme** (dark text on white) and visible focus rings.
- Textareas are **full width** and auto-grow for long answers.
- The **Review** page lets respondents confirm before sending; after submission, the app shows a **clear, read-only summary** of what they submitted.

Privacy tips (recommended)

- Avoid collecting sensitive data unless absolutely necessary.
- If you include an email field, label it clearly as **optional**.
- Consider restricting the web app to your domain if responses are internal.

Ready? Start by creating the two tabs—`Questions` and `Responses`—then paste in the code files that follow. From there, you can customize the 10 starter questions or replace them with your own.

Create a new sheet and open up the Apps Script Code Editor



What you'll build

- A Google Sheet with two tabs:
  Questions (drives the form) and **Responses** (one row per submission).
- An Apps Script **Web App** that reads `Questions`, shows a **multi-page survey**, then writes a single row into `Responses`.
- A **Summary** menu that creates charts + tables and a **Verbatims** sheet for open-ended answers.

Sheet structure

Create a Google Spreadsheet with two tabs:

Tab: `Questions` (row 1 are headers)

code	section	order	type	text	required	options	scale_min	scale_max

What these mean

- **code** – short ID used as a column name in `Responses` (e.g., `Q1`, `Q2`).
- **section** – page grouping (e.g., "About you", "Experience").
- **order** – question order within its section (1, 2, 3…).
- **type** – one of: `short`, `long`, `single`, `multi`, `likert`.
- **text** – the question text shown to users.

- **required** – `TRUE` or `FALSE`.
- **options** – for `single`/`multi`: choices separated by `|` (e.g., `Yes|No`).
- **scale_min/scale_max** – for `likert` scales (e.g., `1`..`5` or `0`..`10`).

Tab: Responses

Leave it empty—code will build headers.
It will become: `timestamp | submission_id | Q1 | Q2 | ...`
Every submission appends **one row** (multi-selects joined by `;`).

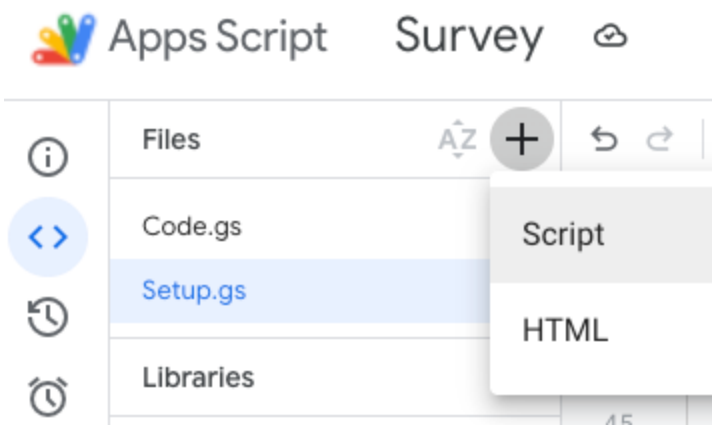
Install the code (5 files + manifest)

Open your Spreadsheet → **Extensions → Apps Script**.
Create these files (exact names) and paste the code.

STEP 1) `appsscript.json` (Project Settings → "Show manifest file" → paste)

```
{
  "timeZone": "America/Toronto",
  "exceptionLogging": "STACKDRIVER",
  "runtimeVersion": "V8"
}
```

Create a new file



What this Setup script does

STEP 2) `Setup.gs` (creates tabs + seeds 10 standard questions)

Run **`seedStandardSurvey()`** once. It creates `Questions`, `Responses`, and fills 10 generic questions for you to test.

- Creates (or reuses) two sheets: **`Questions`** and **`Responses`**.
- Seeds **10 standard, mixed-type questions** into `Questions` so you can test the app immediately.
- Builds the `Responses` header so each future submission becomes **one row**:
  `timestamp | submission_id | Q1 | Q2 | …`
- Leaves you ready to customize the `Questions` sheet (no code changes needed).

You run **`seedStandardSurvey()`** **once** to initialize things. You can run it again later to rebuild headers if you add/remove question codes (just note it will overwrite the contents of both tabs).

Function-by-function explanation

`seedStandardSurvey()`

```
function seedStandardSurvey() {
  var ss = SpreadsheetApp.getActive();
```

- **Gets the current spreadsheet** (the one your Apps Script is attached to).
- You should run this while the *target* Google Sheet is open.

```
 var qSh = ensureSheetWithHeaders_(ss, 'Questions',

['code','section','order','type','text','required','options','scal
e_min','scale_max']);
  var rSh = ensureSheetWithHeaders_(ss, 'Responses',
['timestamp','submission_id']);
```

- **Creates or reuses** the `Questions` and `Responses` sheets.
- Also **writes the header row** for each sheet.
- Headers in `Questions` tell the web app how to render the UI:
  - `code` → the column name in `Responses` (e.g., `Q1`)
  - `section` → page grouping
  - `order` → question ordering **within the section**
  - `type` → `short | long | single | multi | likert`
  - `text` → the question label the user sees
  - `required` → `TRUE` or `FALSE`
  - `options` → for `single/multi`, a pipe `|` separated list (e.g., `Yes|No`)
  - `scale_min/scale_max` → for `likert` (e.g., `1` and `5`)

```
 // Clear Questions below the header
```

```
  clearBelowHeader_(qSh);
```

- **Removes any old question rows** (keeps the header).
- This keeps the seeding step deterministic and clean.

```
 // 10 simple, mixed-type questions (feel free to edit later)
  var rows = [
    ['Q1','About you',1,'single','What is your age
range?',true,'Under 18|18–24|25–34|35–44|45–54|55–64|65+','', ''],
    ['Q2','About you',2,'single','What is your primary
role?',true,'Student|Professional|Manager|Executive|Other','',
''],
    ['Q3','About you',3,'multi','Which platforms do you use
regularly? (Select all)',false,'Web|iOS|Android|Desktop','', ''],

    ['Q4','Experience',1,'likert','Rate your overall
satisfaction',true,'',1,5],
    ['Q5','Experience',2,'likert','How easy was it to complete
tasks?',false,'',1,5],

    ['Q6','Feedback',1,'short','One thing we did well
was…',false,'','', ''],
    ['Q7','Feedback',2,'long','If we could improve one thing, it
would be…',false,'','', ''],

    ['Q8','Preferences',1,'single','Would you recommend us to a
friend?',true,'Yes|No|Not sure','', ''],
    ['Q9','Preferences',2,'multi','What features interest you
most? (Select all)',false,'Speed|Analytics|Integrations|Mobile
access|Security','', ''],
    ['Q10','Contact',1,'short','(Optional) Your email if you want
follow-up',false,'','', '']
  ];
```

- This is a **starter survey**: 10 questions across 4 sections.
- You can safely **edit this array** to fit your needs (or delete it entirely and paste your own rows into the `Questions` sheet later).
- Note the mix of types: `single`, `multi`, `likert`, `short`, `long`.

```
 if (rows.length)
qSh.getRange(2,1,rows.length,rows[0].length).setValues(rows);
```

- Writes the whole 2D array to the `Questions` sheet in **one call** (fast and atomic).
- Starts at row 2, column 1 (since row 1 is the header).

```
 // Build Responses header: timestamp | submission_id | codes…
```

```
  buildResponsesHeaderFromQuestions_();
  SpreadsheetApp.getUi().alert('Standard survey seeded. You can
customize Questions now.');
}
```

- **Generates the `Responses` header** from the question **codes** in `Questions`.
- The order is **by section**, then **by order** within the section (so exported rows read logically).
- Pops a UI alert when done.

Helper: `ensureSheetWithHeaders_(ss, name, headers)`

```
function ensureSheetWithHeaders_(ss, name, headers) {
  var sh = ss.getSheetByName(name) || ss.insertSheet(name);
  if (sh.getLastRow() === 0) {
    sh.getRange(1,1,1,headers.length).setValues([headers]);
    sh.setFrozenRows(1);
  } else {
    sh.getRange(1,1,1,headers.length).setValues([headers]);
    sh.setFrozenRows(1);
  }
  return sh;
}
```

- **Finds or creates** a sheet by name.
- **Writes the header row** and freezes it (always).
- Returns the sheet so the caller can keep working with it.

Why always write headers?
It guarantees your header row stays correct even if someone manually changed it.

Helper: `clearBelowHeader_(sh)`

```
function clearBelowHeader_(sh) {
  var lr = sh.getLastRow(), lc = sh.getLastColumn();
  if (lr > 1) sh.getRange(2,1,lr-1,lc).clearContent();
}
```

- Deletes **all content below row 1** (the header), across all columns.
- This resets the `Questions` sheet before seeding.

If you already have questions and don't want them removed, **don't call this** (or comment it out).

Helper: `buildResponsesHeaderFromQuestions_()`

```
function buildResponsesHeaderFromQuestions_() {
  var ss = SpreadsheetApp.getActive();
  var qSh = ss.getSheetByName('Questions');
  var rSh = ss.getSheetByName('Responses');

  var q = qSh.getDataRange().getValues();
  if (q.length < 2) return;
```

- Reads the entire `Questions` sheet.
- If there are no question rows, exits gracefully.

```
 // Collect codes, sorted by section->order
  var body = q.slice(1).map(function(r){
    return { code:r[0], section:r[1], order:Number(r[2]||0) };
  }).filter(function(x){ return x.code; });

  body.sort(function(a,b){
    return a.section === b.section ? a.order - b.order :
String(a.section).localeCompare(String(b.section));
  });
```

- Collects **only what we need** to build headers:
  - `code` (becomes a column in `Responses`)
  - `section` + `order` (sorting)
- Ignores any blank rows.
- Sorts to keep `Responses` columns in a **human-friendly order**.

```
 var headers = ['timestamp','submission_id'];
  for (var i=0;i<body.length;i++) headers.push(body[i].code);

  rSh.clearContents();
  rSh.getRange(1,1,1,headers.length).setValues([headers]);
  rSh.setFrozenRows(1);
```

```
}
```

- Constructs the `Responses` header: **timestamp**, **submission_id**, then each question **code**.
- Clears any previous `Responses` data (since the header may have changed).
- Writes and freezes the header row.

Why wide format?
Each submission is a **single row** that's easy to read, export, and chart.
Multi-select answers are joined with `;` (done later by the submit function in `Code.gs`).

Common tweaks you might want

- **Don't wipe existing `Questions`**: comment out `clearBelowHeader_(qSh)` and manage rows manually.
- **Change seed questions**: edit the `rows` array or set it to `[]` and maintain questions only in the sheet.
- **Rebuild only `Responses` header** after editing questions: run `buildResponsesHeaderFromQuestions_()` manually (or include a small menu item to trigger it).
- **Enforce max 10 questions**: you can add a guard before writing `rows` to limit length, but the web app itself doesn't enforce a maximum—only your content does.

Safety/maintenance tips

- Keep **codes unique** (`Q1`, `Q2`, …). These become column names in `Responses`.
- If you **change** a code after collecting data, that creates a **new column**; the old column remains (good for audit trails).
- Avoid very long codes—they're used as column headers and can make sheets unwieldy.
- If you delete a question, re-run `buildResponsesHeaderFromQuestions_()` to rebuild the header (it will remove that code from new submissions, but your old data remains in older columns unless you manually clean it up).

TL;DR (what to run)

1. Open your Sheet → **Extensions → Apps Script**.
2. Paste this **Setup** file.
3. Click **Run → seedStandardSurvey()** (authorize).
4. You now have:
    - `Questions` with 10 demo questions,
    - `Responses` with a proper header.
5. Next, deploy the web app and start collecting responses.

```
/**************************************************
 * Setup: Create tabs and seed 10 standard questions
 * Run seedStandardSurvey() once (then customize Questions).
 **************************************************/
function seedStandardSurvey() {
  var ss = SpreadsheetApp.getActive();

  var qSh = ensureSheetWithHeaders_(ss, 'Questions',

['code','section','order','type','text','required','options','scale_mi
n','scale_max']);
  var rSh = ensureSheetWithHeaders_(ss, 'Responses',
['timestamp','submission_id']);

  // Clear Questions below the header
  clearBelowHeader_(qSh);

  // 10 simple, mixed-type questions (feel free to edit later)
  var rows = [
    ['Q1','About you',1,'single','What is your age range?',true,'Under
18|18–24|25–34|35–44|45–54|55–64|65+','', ''],
    ['Q2','About you',2,'single','What is your primary
role?',true,'Student|Professional|Manager|Executive|Other','', ''],
    ['Q3','About you',3,'multi','Which platforms do you use regularly?
(Select all)',false,'Web|iOS|Android|Desktop','', ''],

    ['Q4','Experience',1,'likert','Rate your overall
satisfaction',true,'',1,5],
    ['Q5','Experience',2,'likert','How easy was it to complete
tasks?',false,'',1,5],
```

```
    ['Q6','Feedback',1,'short','One thing we did well
was…',false,'','', ''],
    ['Q7','Feedback',2,'long','If we could improve one thing, it would
be…',false,'','', ''],

    ['Q8','Preferences',1,'single','Would you recommend us to a
friend?',true,'Yes|No|Not sure','', ''],
    ['Q9','Preferences',2,'multi','What features interest you most?
(Select all)',false,'Speed|Analytics|Integrations|Mobile
access|Security','', ''],
    ['Q10','Contact',1,'short','(Optional) Your email if you want
follow-up',false,'','', '']
  ];

  if (rows.length)
qSh.getRange(2,1,rows.length,rows[0].length).setValues(rows);

  // Build Responses header: timestamp | submission_id | codes…
  buildResponsesHeaderFromQuestions_();
  SpreadsheetApp.getUi().alert('Standard survey seeded. You can
customize Questions now.');
}

/************ helpers (Setup) ************/
function ensureSheetWithHeaders_(ss, name, headers) {
  var sh = ss.getSheetByName(name) || ss.insertSheet(name);
  if (sh.getLastRow() === 0) {
    sh.getRange(1,1,1,headers.length).setValues([headers]);
    sh.setFrozenRows(1);
  } else {
    sh.getRange(1,1,1,headers.length).setValues([headers]);
    sh.setFrozenRows(1);
  }
  return sh;
}
function clearBelowHeader_(sh) {
  var lr = sh.getLastRow(), lc = sh.getLastColumn();
  if (lr > 1) sh.getRange(2,1,lr-1,lc).clearContent();
}
function buildResponsesHeaderFromQuestions_() {
  var ss = SpreadsheetApp.getActive();
  var qSh = ss.getSheetByName('Questions');
  var rSh = ss.getSheetByName('Responses');

  var q = qSh.getDataRange().getValues();
  if (q.length < 2) return;

  // Collect codes, sorted by section->order
```

```
  var body = q.slice(1).map(function(r){
    return { code:r[0], section:r[1], order:Number(r[2]||0) };
  }).filter(function(x){ return x.code; });

  body.sort(function(a,b){
    return a.section === b.section ? a.order - b.order :
String(a.section).localeCompare(String(b.section));
  });

  var headers = ['timestamp','submission_id'];
  for (var i=0;i<body.length;i++) headers.push(body[i].code);

  rSh.clearContents();
  rSh.getRange(1,1,1,headers.length).setValues([headers]);
  rSh.setFrozenRows(1);
}
```

What Code.gs script does

This file powers the **Web App** side of your survey:

- **`doGet()`** serves the HTML app.
- **`loadQuestions()`** reads the `Questions` sheet and sends a clean JSON payload to the browser.
- **`submitRow()`** writes one submission (one row) into the `Responses` sheet, safely (with a lock + retry).
- **`withRetry_()`** is a small helper that retries operations if Google's backend is momentarily busy.

Design choices that make this robust and beginner-friendly:

- It returns **JSON strings** (not raw objects) to the client to avoid the "structured-clone / deserialize" errors Apps Script can hit.
- It only reads the **used range** of the `Questions` sheet (no huge blank tails).
- It appends responses in **wide format** (one submission = one row; readable columns).
- It uses a **document lock + exponential backoff** to survive concurrent submissions.

Walkthrough by function

Project/Sheet binding

```
var SPREADSHEET_ID = ''; // set if this project is NOT bound to the
Sheet
```

- If your script is **container-bound** (opened from the Sheet via Extensions → Apps Script), leave this blank.
- If your script is **standalone**, paste your Sheet's ID here (the long string in the Sheet URL).
  This ensures the script opens the correct spreadsheet.

```
function getSS_() {
  if (SPREADSHEET_ID && SPREADSHEET_ID.trim()) return
SpreadsheetApp.openById(SPREADSHEET_ID.trim());
  var ss = SpreadsheetApp.getActive();
  if (!ss) throw new Error('Spreadsheet not found. Set
SPREADSHEET_ID.');
  return ss;
}
```

- Helper that returns the **Spreadsheet** object.
- Uses the explicit ID when provided; otherwise falls back to the **active** (container-bound) spreadsheet.
- Throws a helpful error if neither is available.

Tip: If you move to a new Sheet later, updating `SPREADSHEET_ID` is the only change you need.

Serving the web app

```
function doGet() {
  return HtmlService.createTemplateFromFile('Index')
    .evaluate()
    .setTitle('Survey')
    .setXFrameOptionsMode(HtmlService.XFrameOptionsMode.ALLOWALL);
}
```

- Required entry point for **Web Apps** in Apps Script.
- Loads your `Index.html` file (which itself includes `Styles.html` and `Script.html`).
- Sets the browser tab title and allows embedding iframes (handy if you ever embed the

app elsewhere).

Deployment checklist: Deploy → Web app → *Execute as: Me; Who has access: Anyone with the link* (or your domain).

Loading questions for the UI

```
function loadQuestions() {
  try {
    var ss = getSS_();
    var qSh = ss.getSheetByName('Questions');
    if (!qSh) throw new Error('Questions sheet not found.');
```

- Ensures the `Questions` sheet exists; if not, it returns an error to the client.

```
   var lastRow = qSh.getLastRow(), lastCol = qSh.getLastColumn();
    if (lastRow < 2) return JSON.stringify({ ok:false, error:'No
questions in sheet.' });
```

- If there's only a header row (or empty sheet), we report "no questions" cleanly.

```
   var headers = qSh.getRange(1,1,1,lastCol).getValues()[0];
    var ix = {}; for (var i=0;i<headers.length;i++) ix[headers[i]] =
i;
    var data = qSh.getRange(2,1,lastRow-1,lastCol).getValues();
```

- Reads **only the used rows** (from row 2 down).
- Builds an index `ix` (header name → column number) so we can read by header label, not by hard-coded positions.

```
   var items = [];
    data.forEach(function(row){
      var code = String(row[ix.code]||'').trim();
      if (!code) return;
      items.push({
        code: code,
        section: String(row[ix.section]||'').trim(),
        order: Number(row[ix.order]||0),
        type: String(row[ix.type]||'').toLowerCase(),
        text: String(row[ix.text]||'').trim(),
        required: String(row[ix.required]||'').toLowerCase()==='true',
        options:
String(row[ix.options]||'').split('|').map(function(s){return
s.trim();}).filter(Boolean),
```

```
        scale_min: Number(row[ix.scale_min]||0),
        scale_max: Number(row[ix.scale_max]||0)
      });
    });
```

- Converts each question row into a plain JS object for the client:
  - Parses booleans and numbers.
  - Splits the `options` field on `|` into an array.

```
   items.sort(function(a,b){
      return a.section===b.section ? a.order - b.order :
String(a.section).localeCompare(String(b.section));
    });
```

- Sorts questions by **section**, then by **order** (so the UI pages are predictable and readable).

```
   withRetry_(function(){
      var rSh = ss.getSheetByName('Responses') ||
ss.insertSheet('Responses');
      if (rSh.getLastColumn() < 2) {
        rSh.clearContents();

rSh.getRange(1,1,1,2).setValues([['timestamp','submission_id']]);
        rSh.setFrozenRows(1);
      }
    });
```

- Double-checks that a `Responses` sheet exists and has at least the two base columns (`timestamp`, `submission_id`). Creates/fixes it if not.
- Wrapped in `withRetry_()` (explained later) in case the sheet is temporarily busy.

```
   return JSON.stringify({ ok:true, questions:items });
  } catch (e) {
    return JSON.stringify({ ok:false, error: e.message || String(e)
});
  }
}
```

- **Returns a JSON string** (not a raw object). Returning strings avoids Apps Script's occasional "deserialize" errors when passing data back to the browser.

Customization: You can add fields in your `Questions` sheet; the client only uses the known ones (extra columns will be ignored safely).

Writing a submission (one row)

```
function submitRow(payload) {
  try {
    if (!payload || !payload.answers) return JSON.stringify({
ok:false, error:'Invalid payload' });

    var ss = getSS_();
    var rSh = ss.getSheetByName('Responses');
    if (!rSh) throw new Error('Responses sheet not found.');
```

- `payload.answers` is sent by the browser as `{ code: value, ... }`.
- We ensure the `Responses` sheet exists (it will, thanks to `loadQuestions()`).

```
   var lastCol = Math.max(2, rSh.getLastColumn());
    var headers = rSh.getRange(1,1,1,lastCol).getValues()[0];
    var map = {}; for (var i=0;i<headers.length;i++) map[headers[i]] =
i;
```

- Reads the current headers of `Responses` to know which column each code maps to.

```
   var row = new Array(headers.length);
    row[map.timestamp] = new Date();
    var sid = Utilities.getUuid();
    row[map.submission_id] = sid;
```

- Prepares a new row array and stamps it with the **current time** and a unique **submission ID**.

```
   var answers = payload.answers;
    Object.keys(answers).forEach(function(code){
      if (!map.hasOwnProperty(code)) {
        // Add new column if Questions changed after Responses header
built
        var newCol = rSh.getLastColumn()+1;
        rSh.getRange(1,newCol).setValue(code);
        headers.push(code);
        map[code] = headers.length-1;
        row.push('');
      }
      var v = answers[code];
      if (Array.isArray(v)) v = v.join('; ');
```

```
      row[map[code]] = v;
    });
```

- For each answer:
    - If a new `code` appears that isn't in the header yet, it **adds a new column** on the fly.
      (This can happen if you add a question and forget to rebuild the header—nice safety net.)
    - For multi-select answers (arrays), joins values with `;` so they're readable in one cell.

```
  withRetry_(function(){
     var lock = LockService.getDocumentLock();
     lock.waitLock(8000);
     try {
       var lr = rSh.getLastRow();
       rSh.getRange(lr+1,1,1,headers.length).setValues([row]);
       SpreadsheetApp.flush();
     } finally {
       lock.releaseLock();
     }
   });

   return JSON.stringify({ ok:true, submissionId: sid });
```

- Uses a **document lock** to prevent two users appending at exactly the same time from stepping on each other.
- Writes the row in **one `setValues` call** (more reliable/faster than `appendRow` under load).
- Returns the `submissionId` to the browser so you can show a confirmation.

```
 } catch (e) {
    return JSON.stringify({ ok:false, error: e.message || String(e)
});
  }
}
```

- Errors are sent back as `{ok:false, error:'...'}`; the client shows a helpful message.

Customization idea: If you want to enforce "known codes only," remove the block that auto-adds new columns and instead error out when `!map[code]`.

Retry helper for transient issues

```
function withRetry_(fn) {
  var attempts = 5, lastErr;
  for (var i=0;i<attempts;i++) {
    try { return fn(); }
    catch(e) {
      var msg = String(e.message||e);
      if (msg.indexOf('FAILED_PRECONDITION')>-1 ||
msg.indexOf('Internal error')>-1 || msg.indexOf('Rate Limit')>-1) {
        lastErr = e;
        Utilities.sleep(200*(i+1)*(i+1)); // 200ms, 800ms, 1800ms, ...
        continue;
      }
      throw e;
    }
  }
  throw lastErr || new Error('Unknown storage error after retries.');
}
```

- Retries a function **up to 5 times** if the error looks transient:
  - `FAILED_PRECONDITION` (storage hiccups),
  - backend "Internal error",
  - "Rate Limit".
- Uses **exponential backoff** (increasing waits) to be polite to Google's backend.

This makes your app resilient when several people submit at the same time or Sheets is momentarily busy.

How the client uses these endpoints

- **`Script.html`** calls:
  - `google.script.run.loadQuestions()` on load → receives `{ok:true, questions:[...]}` → builds the UI.
  - On submit: `google.script.run.submitRow({answers})` → receives `{ok:true, submissionId}` → shows a read-only summary.

Because both methods return **JSON strings**, the client does:

```
const raw = await google.script.run...
const data = (typeof raw === 'string') ? JSON.parse(raw) : raw;
```

This avoids the structured-clone issue Apps Script sometimes hits when sending complex objects.

Common customizations

- **Require login (internal surveys):** Deploy the web app to your domain only. In `submitRow`, you can read the user's email with `Session.getActiveUser().getEmail()` (works for domain-restricted scripts executed as you).
- **Validate answers server-side:** Inspect `payload.answers` before writing; return `{ok:false, error:'...'}`
- **Add metadata columns:** Extend `Responses` header (e.g., `user_email`, `user_agent`) and set them in `submitRow`.

Troubleshooting tips

- **Blank page / fails to load:** Ensure `Questions` exists and has at least one row; redeploy the web app after changes; set `SPREADSHEET_ID` if standalone.
- **"deserialize / dropping postMessage" errors:** You're already protected by returning **JSON strings**; keep that pattern.
- **"FAILED_PRECONDITION":** The code already uses lock + retry; if it persists, check that the web app executes as **Me** and that "Me" has edit access to the Sheet.

That's the full tour. With this server, your front end stays simple and your data stays clean: one submission, one row—easy to analyze, chart, and export.

STEP 3) `Code.gs` (server: loads questions, appends submissions)

If your Apps Script is **standalone** (not attached to the Sheet), set `SPREADSHEET_ID`.

```
/**************************************************
 * Survey Web App (wide format: one submission = one row)
 **************************************************/
var SPREADSHEET_ID = ''; // set if this project is NOT bound to the
```

```
Sheet

function getSS_() {
  if (SPREADSHEET_ID && SPREADSHEET_ID.trim()) return
SpreadsheetApp.openById(SPREADSHEET_ID.trim());
  var ss = SpreadsheetApp.getActive();
  if (!ss) throw new Error('Spreadsheet not found. Set
SPREADSHEET_ID.');
  return ss;
}

function doGet() {
  return HtmlService.createTemplateFromFile('Index')
    .evaluate()
    .setTitle('Survey')
    .setXFrameOptionsMode(HtmlService.XFrameOptionsMode.ALLOWALL);
}

/** Return questions as a JSON string (avoids structured-clone issues)
*/
function loadQuestions() {
  try {
    var ss = getSS_();
    var qSh = ss.getSheetByName('Questions');
    if (!qSh) throw new Error('Questions sheet not found.');

    var lastRow = qSh.getLastRow(), lastCol = qSh.getLastColumn();
    if (lastRow < 2) return JSON.stringify({ ok:false, error:'No
questions in sheet.' });

    var headers = qSh.getRange(1,1,1,lastCol).getValues()[0];
    var ix = {}; for (var i=0;i<headers.length;i++) ix[headers[i]] =
i;
    var data = qSh.getRange(2,1,lastRow-1,lastCol).getValues();

    var items = [];
    data.forEach(function(row){
      var code = String(row[ix.code]||'').trim();
      if (!code) return;
      items.push({
        code: code,
        section: String(row[ix.section]||'').trim(),
        order: Number(row[ix.order]||0),
        type: String(row[ix.type]||'').toLowerCase(),
        text: String(row[ix.text]||'').trim(),
        required: String(row[ix.required]||'').toLowerCase()==='true',
        options:
String(row[ix.options]||'').split('|').map(function(s){return
```

```
s.trim();}).filter(Boolean),
        scale_min: Number(row[ix.scale_min]||0),
        scale_max: Number(row[ix.scale_max]||0)
      });
    });

    items.sort(function(a,b){
      return a.section===b.section ? a.order - b.order :
String(a.section).localeCompare(String(b.section));
    });

    // Ensure Responses has at least timestamp/submission_id
    withRetry_(function(){
      var rSh = ss.getSheetByName('Responses') ||
ss.insertSheet('Responses');
      if (rSh.getLastColumn() < 2) {
        rSh.clearContents();

rSh.getRange(1,1,1,2).setValues([['timestamp','submission_id']]);
        rSh.setFrozenRows(1);
      }
    });

    return JSON.stringify({ ok:true, questions:items });
  } catch (e) {
    return JSON.stringify({ ok:false, error: e.message || String(e)
});
  }
}

/** Append one row; add columns if new codes appear; lock + retry for
safety */
function submitRow(payload) {
  try {
    if (!payload || !payload.answers) return JSON.stringify({
ok:false, error:'Invalid payload' });

    var ss = getSS_();
    var rSh = ss.getSheetByName('Responses');
    if (!rSh) throw new Error('Responses sheet not found.');

    var lastCol = Math.max(2, rSh.getLastColumn());
    var headers = rSh.getRange(1,1,1,lastCol).getValues()[0];
    var map = {}; for (var i=0;i<headers.length;i++) map[headers[i]] =
i;

    var row = new Array(headers.length);
    row[map.timestamp] = new Date();
```

```
    var sid = Utilities.getUuid();
    row[map.submission_id] = sid;

    var answers = payload.answers;
    Object.keys(answers).forEach(function(code){
      if (!map.hasOwnProperty(code)) {
        // Add new column if Questions changed after Responses header
built
        var newCol = rSh.getLastColumn()+1;
        rSh.getRange(1,newCol).setValue(code);
        headers.push(code);
        map[code] = headers.length-1;
        row.push('');
      }
      var v = answers[code];
      if (Array.isArray(v)) v = v.join('; ');
      row[map[code]] = v;
    });

    withRetry_(function(){
      var lock = LockService.getDocumentLock();
      lock.waitLock(8000);
      try {
        var lr = rSh.getLastRow();
        rSh.getRange(lr+1,1,1,headers.length).setValues([row]);
        SpreadsheetApp.flush();
      } finally {
        lock.releaseLock();
      }
    });

    return JSON.stringify({ ok:true, submissionId: sid });
  } catch (e) {
    return JSON.stringify({ ok:false, error: e.message || String(e)
});
  }
}

/** Retry wrapper for transient storage/concurrency hiccups */
function withRetry_(fn) {
  var attempts = 5, lastErr;
  for (var i=0;i<attempts;i++) {
    try { return fn(); }
    catch(e) {
      var msg = String(e.message||e);
      if (msg.indexOf('FAILED_PRECONDITION')>-1 ||
msg.indexOf('Internal error')>-1 || msg.indexOf('Rate Limit')>-1) {
        lastErr = e;
```

```
        Utilities.sleep(200*(i+1)*(i+1)); // 200ms, 800ms, 1800ms, ...
        continue;
      }
      throw e;
    }
  }
  throw lastErr || new Error('Unknown storage error after retries.');
}
```

What Index.html Does

Purpose of `Index.html`

- Defines the **basic structure** of the survey page (header, main content, footer).
- Pulls in the **Styles.html** for design and **Script.html** for logic.
- Provides **containers** (`<div>`s, `<button>`s) that the JavaScript will fill and update dynamically.
- Keeps things **lightweight** — no hardcoded questions here. The questions come from the Sheet, loaded by the script.

Line-by-line explanation

```
<!DOCTYPE html>
<html>
<head>
  <base target="_top">
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width,initial-scale=1">
```

- Standard HTML header.
- `<base target="_top">` ensures links open in the full window, not inside Apps Script's frame.
- Meta tags set the character encoding (UTF-8) and make it mobile-friendly.

```
 <?!= HtmlService.createHtmlOutputFromFile('Styles').getContent(); ?>
  <title>Survey</title>
</head>
```

- This special Apps Script tag `<?!= ... ?>` includes the contents of `Styles.html`.
- That file holds all your CSS (colors, layout, typography).

- The `<title>` sets the browser tab label.

```
<body>
  <header class="app-header">
    <div class="brand">
      <div class="logo">S</div>
      <div>
        <h1>Survey</h1>
        <p>Please share your feedback. Required questions are marked
with *</p>
      </div>
    </div>
```

- The **page header** area.
- Shows a logo “S” (you could replace with your own branding), a title, and a short description.
- Styled by CSS in `Styles.html`.

```
   <div class="progress-wrap">
      <div id="progressBar" class="progress"><span
style="width:0%"></span></div>
      <div id="stepLabel" class="step-label">Step 1 of 1</div>
    </div>
  </header>
```

- A **progress bar** and label (`Step X of Y`) to give users feedback as they move through pages.
- The JavaScript (`Script.html`) updates `progressBar` and `stepLabel` as you click “Next” or “Back”.

```
 <main class="container">
    <div id="app" class="loading">Loading…</div>
  </main>
```

- The **main container** where the survey questions will appear.
- Initially shows “Loading…” text.
- When `Script.html` runs, it replaces this `<div id="app">` with dynamically rendered questions (based on your `Questions` sheet).

```
 <footer class="nav-footer" id="navFooter" hidden>
    <div class="nav-inner">
```

```
      <button class="btn" id="btnBack" disabled>Back</button>
      <div class="spacer"></div>
      <button class="btn" id="btnReview">Review</button>
      <button class="btn" id="btnNext">Next</button>
      <button class="btn primary" id="btnSubmit"
hidden>Submit</button>
    </div>
  </footer>
```

- The **navigation footer**.
- Contains the action buttons:
  - **Back** → goes to previous page (disabled on page 1).
  - **Next** → moves forward one page.
  - **Review** → jumps to the review page (where you can check all answers).
  - **Submit** → only visible on the review page; sends answers to the Sheet.

- It starts as `hidden`, and `Script.html` makes it visible once the survey is loaded.

```
 <?!= HtmlService.createHtmlOutputFromFile('Script').getContent(); ?>
</body>
</html>
```

- Includes the contents of `Script.html`.
- That file is your **JavaScript brain**:
  - Loads questions via `google.script.run.loadQuestions()`.
  - Builds pages dynamically.
  - Validates required fields.
  - Submits answers to `submitRow()`.
  - Updates progress bar and buttons.

Key takeaways

- `Index.html` is just a **skeleton**: header, container, footer.
- No survey logic or styling here—that's in `Script.html` and `Styles.html`.
- Think of it like a **frame**:
  - **`Styles.html`** paints it,
  - **`Script.html`** makes it interactive,
  - **`Code.gs`** connects it to your Sheet.

STEP 4) `Index.html` (shell that pulls in styles + script)

```
<!DOCTYPE html>
<html>
<head>
  <base target="_top">
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width,initial-scale=1">
  <?!= HtmlService.createHtmlOutputFromFile('Styles').getContent(); ?>
  <title>Survey</title>
</head>
<body>
  <header class="app-header">
    <div class="brand">
      <div class="logo">S</div>
      <div>
        <h1>Survey</h1>
        <p>Please share your feedback. Required questions are marked
with *</p>
      </div>
    </div>
    <div class="progress-wrap">
      <div id="progressBar" class="progress"><span
style="width:0%"></span></div>
      <div id="stepLabel" class="step-label">Step 1 of 1</div>
    </div>
  </header>

  <main class="container">
    <div id="app" class="loading">Loading…</div>
  </main>

  <footer class="nav-footer" id="navFooter" hidden>
    <div class="nav-inner">
      <button class="btn" id="btnBack" disabled>Back</button>
      <div class="spacer"></div>
      <button class="btn" id="btnReview">Review</button>
      <button class="btn" id="btnNext">Next</button>
      <button class="btn primary" id="btnSubmit"
hidden>Submit</button>
    </div>
  </footer>

  <?!= HtmlService.createHtmlOutputFromFile('Script').getContent(); ?>
</body>
```

```
</html>
```

What Styles does

Theme variables

```
:root{
  --bg:#ffffff; --text:#111827; --muted:#4b5563; --card:#ffffff;
--border:#e5e7eb;
  --accent:#0ea5e9; --accent-2:#0369a1; --ring:#0ea5e933;
--danger:#b91c1c;
  --radius:14px; --shadow:0 8px 24px rgba(17,24,39,.08);
}
```

- Defines a **color palette** and reusable design tokens:
  - `--bg` white background, `--text` dark gray text (accessible contrast).
  - `--accent` blue for highlights and buttons.
  - `--danger` red for errors.
  - `--radius`, `--shadow` = rounded corners and subtle shadows.

This makes it easy to update the look later — just change the variables.

Page basics

```
body{
  margin:0;
  background:var(--bg);
  color:var(--text);
  font:16px/1.6 system-ui, -apple-system, Segoe UI, Roboto, Helvetica,
Arial;
}
```

- White background, dark text, modern system font.
- Default font size = 16px, with readable line height.

Header and progress bar

```
.app-header{position:sticky;top:0;...}
.progress{height:10px;background:#f1f5f9;...}
.progress
span{background:linear-gradient(90deg,var(--accent),var(--accent-2));}
.step-label{font-size:13px;color:var(--muted);}
```

- Sticky header always visible at top.
- Progress bar shows survey progress with a blue gradient.
- Step label (`Step 1 of X`) styled in muted gray.

Sections and cards

```
.section-card{
  background:var(--card);
  border:1px solid var(--border);
  border-radius:var(--radius);
  box-shadow:var(--shadow);
  padding:20px;margin:16px 0;
}
```

- Each page (section) of the survey appears inside a **card** with a border, shadow, and spacing.
- Keeps the form organized and visually grouped.

Questions and inputs

```
.q .title{font-weight:600;}
.q .req{color:#b45309;}
input[type="text"], textarea{width:100%; border:1px solid
var(--border);}
textarea{min-height:220px;}
input:focus, textarea:focus{border-color:var(--accent); box-shadow:0 0
0 4px var(--ring);}
```

- Titles bold; required questions marked with an orange `*`.
- Inputs and textareas are full-width, rounded, and expand on focus.
- Textareas are tall enough for long answers.

Choice styles

```
.chip{display:inline-flex; padding:10px 12px; border:1px solid
var(--border);}
.chip input{accent-color:var(--accent-2);}
.segmented label{padding:8px 10px; border:1px solid var(--border);}
.segmented input:checked + span{outline:3px solid var(--ring);}
```

- **Chip style**: clean radio/checkbox look, clickable boxes.
- **Segmented style**: Likert scale (1–5) choices styled as button-like boxes.

Buttons and footer nav

```
.btn{padding:10px 14px; border-radius:10px; background:#fff;}
.btn.primary{background:var(--accent);color:#fff;}
```

```
.nav-footer{position:fixed;bottom:0;...}
```

- Buttons are simple, rounded, with hover effects.
- Primary button = blue background with white text.
- Navigation footer stays fixed at bottom with Back/Next/Submit buttons.

Utility classes

- `.error` → red text for validation errors.
- `.muted` → gray text for hints.
- `.success` → green background/outline for success messages.
- `.table` → clean styling for review/summary tables.

TL;DR

- **`Styles.html` is your skin**: sets colors, fonts, spacing, and makes the survey readable and modern.
- Uses **CSS variables** so you can quickly change the look (dark mode, branding, etc.).
- Organizes key parts: header, progress bar, question cards, inputs, buttons, footer.

STEP 5) `Styles.html` (accessible light theme)

```
<style>
  :root{
    --bg:#ffffff; --text:#111827; --muted:#4b5563; --card:#ffffff;
--border:#e5e7eb;
    --accent:#0ea5e9; --accent-2:#0369a1; --ring:#0ea5e933;
--danger:#b91c1c;
    --radius:14px; --shadow:0 8px 24px rgba(17,24,39,.08);
  }
  html,body{height:100%}
  body{margin:0;background:var(--bg);color:var(--text);font:16px/1.6
system-ui, -apple-system, Segoe UI, Roboto, Helvetica, Arial;}

.app-header{position:sticky;top:0;z-index:50;background:#ffffffe2;back
drop-filter:saturate(130%) blur(4px);border-bottom:1px solid
var(--border);}

.brand{display:flex;align-items:center;gap:14px;max-width:980px;margin
:0 auto;padding:14px 16px;}
```

```
.logo{width:40px;height:40px;border-radius:10px;background:linear-grad
ient(135deg,var(--accent),var(--accent-2));color:#fff;display:grid;pla
ce-items:center;font-weight:800;box-shadow:0 6px 16px
rgba(3,105,161,.25);}
  .brand h1{margin:0 0 2px;font-size:18px;}
  .brand p{margin:0;color:var(--muted);font-size:13px;}
  .progress-wrap{max-width:980px;margin:0 auto 12px;padding:0 16px;}

.progress{height:10px;background:#f1f5f9;border-radius:999px;overflow:
hidden;border:1px solid var(--border);}
  .progress
span{display:block;height:100%;background:linear-gradient(90deg,var(--
accent),var(--accent-2));transition:width .3s;}

.step-label{margin-top:6px;text-align:right;color:var(--muted);font-si
ze:13px;}
  .container{max-width:980px;margin:24px auto 120px;padding:0 16px;}
  .section-card{background:var(--card);border:1px solid
var(--border);border-radius:var(--radius);box-shadow:var(--shadow);pad
ding:20px;margin:16px 0;}
  .section-card h2{margin:0 0 8px;font-size:20px;}
  .section-desc{color:var(--muted);margin:0 0 12px;}
  .q{margin:18px 0;}
  .q .title{font-weight:600;display:block;margin-bottom:8px;}
  .q .req{color:#b45309;margin-left:6px;}
  .options{display:grid;gap:10px}
  .chip{display:inline-flex;align-items:center;gap:10px;padding:10px
12px;border:1px solid
var(--border);border-radius:12px;background:#fff;cursor:pointer;}
  .chip input{accent-color:var(--accent-2);}
  .segmented{display:flex;flex-wrap:wrap;gap:8px}
  .segmented label{user-select:none;cursor:pointer;padding:8px
10px;border:1px solid
var(--border);border-radius:10px;background:#fff;min-width:40px;text-a
lign:center;}
  .segmented input{display:none}
  .segmented input:checked + span{outline:3px solid
var(--ring);border-color:var(--accent-2);}
  input[type="text"]{width:100%;padding:12px
14px;border-radius:12px;border:1px solid
var(--border);background:#fff;color:var(--text);}
  textarea{width:100%;min-height:220px;resize:vertical;padding:12px
14px;border-radius:12px;border:1px solid
var(--border);background:#fff;color:var(--text);}
  input[type="text"]:focus,
textarea:focus{outline:none;border-color:var(--accent);box-shadow:0 0
0 4px var(--ring);}
  .error{color:var(--danger);font-size:14px;margin-top:6px;}
```

```
.nav-footer{position:fixed;left:0;right:0;bottom:0;z-index:60;backgrou
nd:#ffffff2;backdrop-filter:saturate(130%) blur(4px);border-top:1px
solid var(--border);}
  .nav-inner{max-width:980px;margin:0
auto;display:flex;gap:10px;padding:12px 16px;}
  .spacer{flex:1}
  .btn{appearance:none;border:1px solid
var(--border);border-radius:10px;padding:10px
14px;cursor:pointer;background:#fff;color:var(--text);}
  .btn:hover{border-color:#cbd5e1;background:#f8fafc}
  .btn:disabled{opacity:.55;cursor:not-allowed}

.btn.primary{background:var(--accent);border-color:var(--accent-2);col
or:#fff;}
  .btn.primary:hover{background:var(--accent-2);}
  .loading{padding:30px;text-align:center;color:var(--muted)}
  .muted{color:var(--muted)}
  .success{background:#ecfdf5;border:1px solid
#10b981;color:#065f46;padding:12px;border-radius:10px;margin-top:12px;
}
  .table{width:100%;border-collapse:collapse;margin-top:10px;}
  .table th,.table td{border:1px solid var(--border);padding:8px
10px;vertical-align:top;background:#fff;}
  .table th{background:#f8fafc;text-align:left;width:30%}
</style>
```

What Script file does

What this file is

It's the **front-end logic** (plain JavaScript) that:

1. loads questions from Sheets,
2. renders a **multi-page** survey (grouped by `section`),
3. shows a **Review** page,
4. submits answers, then shows a **read-only summary**.

Everything is built dynamically from the `Questions` sheet — no hardcoded questions in the HTML.

Top: grab DOM elements

```
var app = document.getElementById('app');
var btnBack = ...; var btnNext = ...; var btnSubmit = ...; var
```

```
btnReview = ...;
var progress = document.querySelector('#progressBar span');
var stepLabel = document.getElementById('stepLabel');
var navFooter = document.getElementById('navFooter');
```

We cache the key UI nodes once so updates are fast and tidy.

Tiny helpers

```
function h(tag, attrs, children) { ... }
```

- A mini “DOM factory” so we can do `h('div', {class:'q'}, 'Text')` instead of verbose `document.createElement` each time.

```
function slug(s){...}   // safe ids for inputs
function grow(ta){...}  // auto-resize textarea
function toText(v){...} // turn arrays => "A; B" for display
```

Load questions from server

```
const raw = await
google.script.run.withSuccessHandler(...).withFailureHandler(...).load
Questions();
const data = (typeof raw === 'string') ? JSON.parse(raw) : raw;
```

- Calls Apps Script’s `loadQuestions()` (server), which returns a **JSON string**.
- We parse it and either render the survey or show an error.

Build pages (sectioned) + add a final Review page

```
var pages = []; // { kind:'section', name, items:[] } ... {
kind:'review' }
```

- Groups questions by their `section` header.
- Sorts each section by `order`.
- Appends one extra page: `{kind:'review'}`.

State

```
var page = 0;
var answers = {}; // { code: value }
```

- `page` tracks which section we’re on.
- `answers` stores the current values keyed by `code` (e.g., `Q3: ["Web","iOS"]`).

`renderPage()`

- Clears `#app` and renders:
  - a **section page** (questions) or
  - the **Review** page (pre-submit).
- Updates the **footer buttons** (show/hide Next/Submit/Review) and **progress bar**.

`renderQ(q)` — render one question

- Builds the right UI based on `q.type`:
  - `short` → `<input type="text">`
  - `long` → `<textarea>` (auto-grows)
  - `single` → radios (options from `q.options`)
  - `multi` → checkboxes (options from `q.options`)
  - `likert` → segmented radio scale from `scale_min..scale_max`
- Each control updates `answers[q.code]` on change.
- Shows a small `.error` area under the field for required validation messages.
- Calls `hydrate(q)` to restore values if you navigate back.

Review page (pre-submit)

`function renderReviewPreSubmit(){...}`

- Creates a table per section with your current answers (blanks labeled “(blank)”).
- Adds **“Edit section”** buttons that jump you back to that section (sets `page = idx`).

Validation

```
function validateCurrentPage(){
  // For required questions: must have a value
  // For multi: must have at least one selection
}
```

- Runs when you press **Next** or **Review**.
- Highlights missing required fields immediately on that page.

Navigation handlers

- **Back**: `page--`
- **Next**: validate → if ok `page++`
- **Review**: validate → jump to final review page
  Progress bar and labels update each time.

Submit handler

```
google.script.run.submitRow({ answers })
```

- Sends all answers to the server (Apps Script writes one row to `Responses`).
- On success:
  - Hides the **progress** and **footer**,
  - Replaces the form with a **Thank you** card and a **read-only summary** (only answered questions, grouped by section).
  - Disables Enter-to-navigate after submit.

If anything fails, it shows `Submit failed: ...` in the page message area.

Keyboard shortcut

```
document.addEventListener('keydown', ...)
// Enter = Next or Submit (but not inside textareas)
```

A small usability boost; lets users advance quickly without mousing.

Final call

```
renderPage();
```

Boots the UI once questions are loaded.

Why this approach is solid

- **Data-driven**: UI comes from the sheet; no hardcoded questions.
- **Resilient**: JSON string round-trip avoids Apps Script's object serialization quirks.
- **Accessible UX**: clear error messages, focus styles (from CSS), readable light theme.
- **Clear end state**: after submit, users see exactly what they sent (no confusing edit buttons).

STEP 6) `Script.html` (multi-page UI, review step, clear "submitted" summary)

```
<script>
(async function () {
  var app       = document.getElementById('app');
  var btnBack   = document.getElementById('btnBack');
  var btnNext   = document.getElementById('btnNext');
  var btnSubmit = document.getElementById('btnSubmit');
  var btnReview = document.getElementById('btnReview');
  var progress  = document.querySelector('#progressBar span');
  var stepLabel = document.getElementById('stepLabel');
  var navFooter = document.getElementById('navFooter');
```

```
  function h(tag, attrs, children){
    if (!attrs) attrs = {};
    var el = document.createElement(tag);
    for (var k in attrs){
      var v = attrs[k];
      if (k === 'class') el.className = v;
      else if (k === 'for') el.htmlFor = v;
      else if (k.indexOf('on') === 0 && typeof v === 'function')
el.addEventListener(k.substring(2), v);
      else el.setAttribute(k, v);
    }
    if (children != null){
      var arr = Array.isArray(children) ? children : [children];
      for (var i=0;i<arr.length;i++){ var c = arr[i]; if (c==null)
continue;
        el.appendChild(typeof c === 'string' ?
document.createTextNode(c) : c);
      }
    }
    return el;
  }
  function slug(s){ return
String(s).toLowerCase().replace(/[^a-z0-9]+/g,'-'); }
  function grow(ta){ ta.style.height='auto';
ta.style.height=Math.max(220, ta.scrollHeight)+'px'; }
  function toText(v){ if (v==null || v==='') return ''; return
Array.isArray(v) ? v.join('; ') : String(v); }

  // Load questions (server returns JSON string or object)
  var data;
  try {
    var raw = await new Promise(function(resolve,reject){

google.script.run.withSuccessHandler(resolve).withFailureHandler(rejec
t).loadQuestions();
    });
    data = (typeof raw === 'string') ? JSON.parse(raw) : raw;
  } catch (e) {
    data = { ok:false, error: e && e.message ? e.message : String(e)
};
  }
  if (!data || data.ok !== true || !Array.isArray(data.questions)) {
    app.innerHTML = '<div class="section-card"><div
class="error">Failed to load: ' + (data && data.error ? data.error :
'Unknown') + '</div></div>';
    return;
  }
```

```
  // Build pages: one per section + final Review
  var questions = data.questions.slice();
  var pages = []; // {kind:'section', name, items:[]} ...
{kind:'review'}
  for (var i=0;i<questions.length;i++){
    var q = questions[i], sec = null;
    for (var j=0;j<pages.length;j++){
      if (pages[j].kind==='section' && pages[j].name===q.section){ sec
= pages[j]; break; }
    }
    if (!sec){ sec = { kind:'section', name:q.section, items:[] };
pages.push(sec); }
    sec.items.push(q);
  }
  for (var p=0;p<pages.length;p++) if (pages[p].items)
pages[p].items.sort(function(a,b){ return a.order - b.order; });
  pages.push({ kind:'review' });

  var page = 0;
  var answers = {}; // code -> value

  function renderPage(){
    app.innerHTML = '';
    var current = pages[page];

    if (current.kind === 'section'){
      var card = h('div', {class:'section-card'}, [
        h('h2', {}, current.name),
        h('p', {class:'section-desc'}, page===0 ? 'Required questions
are marked with *' : ''),
        (function(){
          var frag = document.createDocumentFragment();
          for (var k=0;k<current.items.length;k++)
frag.appendChild(renderQ(current.items[k]));
          return frag;
        })(),
        h('div', {id:'pageMsg'})
      ]);
      app.appendChild(card);
    } else {
      app.appendChild(renderReviewPreSubmit());
    }

    navFooter.hidden = false;
    btnBack.disabled = (page === 0);
    btnNext.hidden   = (page >= pages.length - 1);
    btnSubmit.hidden = (page !== pages.length - 1);
    btnReview.hidden = (page === pages.length - 1);
```

```
    var pct = Math.round(((page+1)/pages.length)*100);
    progress.style.width = pct + '%';
    stepLabel.textContent = 'Step ' + (page+1) + ' of ' +
pages.length;
  }

  function renderQ(q){
    var wrap = h('div', {class:'q', id:'q-'+q.code});
    wrap.appendChild(h('label', {class:'title', for:'inp-'+q.code}, [
      q.text, q.required ? h('span',{class:'req'},'*') : '',
      q.type==='likert' ? ' (scale '+q.scale_min+'–'+q.scale_max+')' :
''
    ]));

    var node;
    if (q.type==='short'){
      node = h('input', {type:'text', id:'inp-'+q.code,
oninput:function(e){ set(q.code, e.target.value); }});
    } else if (q.type==='long'){
      node = h('textarea', {id:'inp-'+q.code, rows:'10',
oninput:function(e){ grow(e.target); set(q.code, e.target.value); }});
      setTimeout(function(){ if (answers[q.code]){ node.value =
answers[q.code]; grow(node);} },0);
    } else if (q.type==='single'){
      node = h('div', {class:'options'});
      for (var i=0;i<q.options.length;i++){
        (function(opt){
          var id = 'opt-'+q.code+'-'+slug(opt);
          node.appendChild(h('label', {class:'chip', for:id}, [
            h('input', {type:'radio', id:id, name:'r-'+q.code,
value:opt, onchange:function(){ set(q.code,opt); }}),
            h('span', {}, opt)
          ]));
        })(q.options[i]);
      }
    } else if (q.type==='multi'){
      node = h('div', {class:'options'});
      for (var m=0;m<q.options.length;m++){
        (function(opt){
          var id = 'chk-'+q.code+'-'+slug(opt);
          node.appendChild(h('label', {class:'chip', for:id}, [
            h('input', {type:'checkbox', id:id, value:opt,
onchange:function(e){
              var prev = Array.isArray(answers[q.code]) ?
answers[q.code].slice() : [];
              var dict = {}; for (var x=0;x<prev.length;x++)
dict[prev[x]] = true;
```

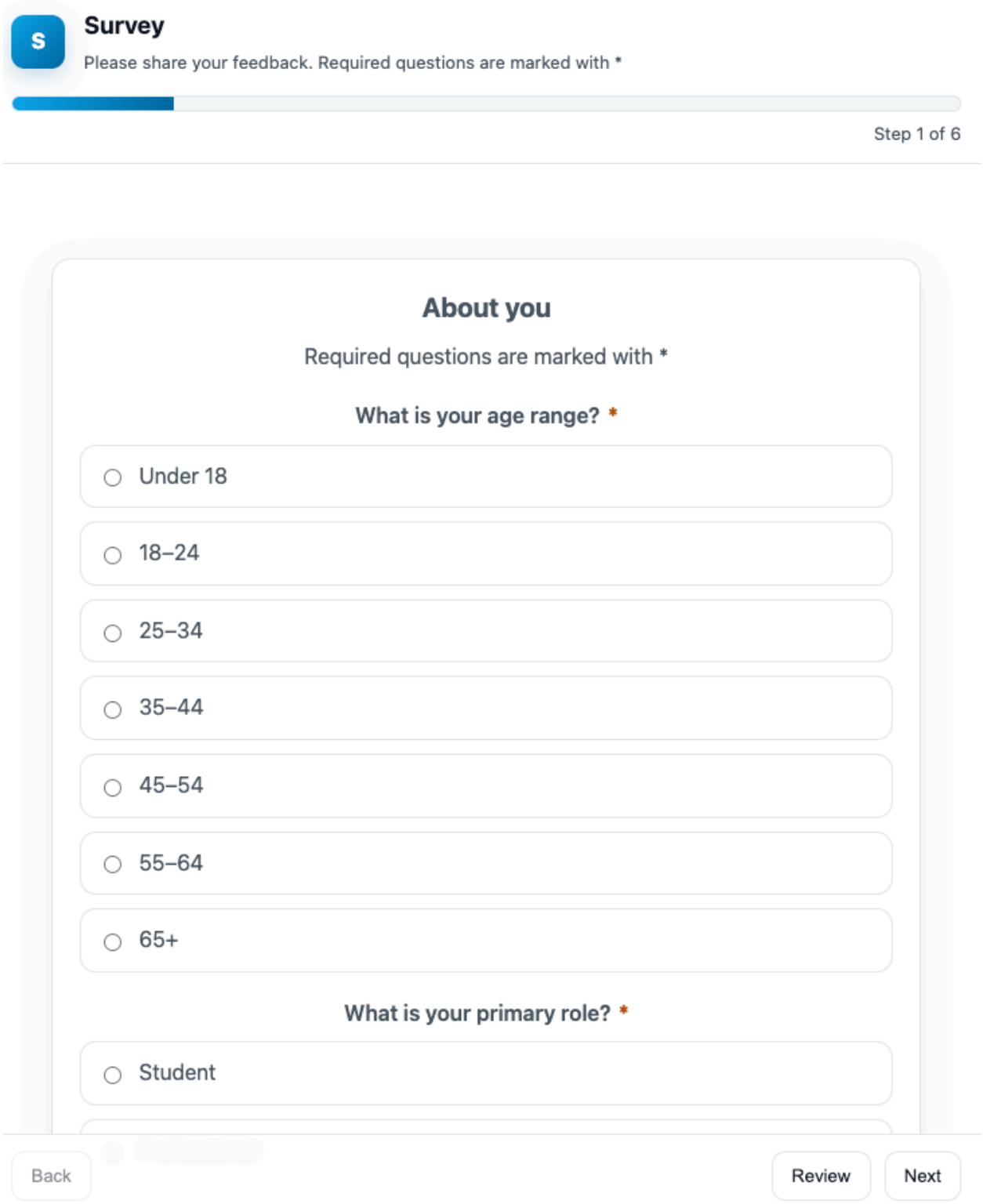
```
              if (e.target.checked) dict[opt]=true; else delete
dict[opt];
              var out=[]; for (var k in dict) if
(dict.hasOwnProperty(k)) out.push(k);
              set(q.code,out);
            }}),
            h('span', {}, opt)
          ]));
        })(q.options[m]);
      }
    } else if (q.type==='likert'){
      node = h('div', {class:'segmented'});
      for (var s=q.scale_min; s<=q.scale_max; s++){
        (function(val){
          var id = 'lk-'+q.code+'-'+val;
          node.appendChild(h('label', {}, [
            h('input', {type:'radio', name:'lk-'+q.code, id:id,
value:String(val), onchange:function(){ set(q.code, Number(val)); }}),
            h('span', {}, String(val))
          ]));
        })(s);
      }
    } else {
      node = h('div', {class:'muted'}, 'Unsupported: '+q.type);
    }

    wrap.appendChild(node);
    wrap.appendChild(h('div', {class:'error', id:'err-'+q.code}));
    setTimeout(function(){ hydrate(q); },0);
    return wrap;
  }

  function renderReviewPreSubmit(){
    var card = h('div', {class:'section-card'}, [
      h('h2', {}, 'Review'),
      h('p', {class:'section-desc'}, 'Please review your answers. Use
“Edit section” to jump back.')
    ]);
    for (var i=0;i<pages.length;i++){
      var pg = pages[i]; if (pg.kind!=='section') continue;

      var tbody = h('tbody');
      for (var j=0;j<pg.items.length;j++){
        var q = pg.items[j];
        var text = toText(answers[q.code]);
        tbody.appendChild(h('tr', {}, [
          h('td', {style:'width:30%; font-weight:600; border:1px solid
var(--border); padding:8px 10px; background:#f8fafc;'}, q.text),
```

```
          h('td', {style:'border:1px solid var(--border); padding:8px
10px; white-space:pre-wrap;'}, text ? text :
h('span',{class:'muted'},'(blank)'))
        ]));
      }
      var table = h('table', {class:'table'});
      table.appendChild(h('thead', {}, [ h('tr', {}, [ h('th',
{colspan:'2'}, pg.name) ]) ]));
      table.appendChild(tbody);

      var editBtn = h('button', {class:'btn', onclick:(function(idx){
        return function(){ page = idx; renderPage();
window.scrollTo({top:0, behavior:'smooth'}); };
      })(i)}, 'Edit section');

      card.appendChild(table);
      card.appendChild(h('div', {style:'margin:6px 0 16px'},
[editBtn]));
    }
    card.appendChild(h('div', {id:'pageMsg'}));
    return card;
  }

  function set(code,v){ answers[code]=v; var
el=document.getElementById('err-'+code); if (el) el.textContent=''; }
  function hydrate(q){
    var v=answers[q.code]; if (v==null) return;
    if (q.type==='short' || q.type==='long'){
      var el=document.getElementById('inp-'+q.code); if (el){
el.value=v; if (q.type==='long') grow(el); }
    } else if (q.type==='single'){
      var id='opt-'+q.code+'-'+slug(v); var
r=document.getElementById(id); if (r) r.checked=true;
    } else if (q.type==='multi' && Array.isArray(v)){
      for (var i=0;i<v.length;i++){ var
c=document.getElementById('chk-'+q.code+'-'+slug(v[i])); if (c)
c.checked=true; }
    } else if (q.type==='likert' && typeof v==='number'){
      var lk=document.getElementById('lk-'+q.code+'-'+v); if (lk)
lk.checked=true;
    }
  }
  function validateCurrentPage(){
    var pdef=pages[page]; if (pdef.kind!=='section') return {};
    var errs={};
    for (var i=0;i<pdef.items.length;i++){
      var q=pdef.items[i]; if (!q.required) continue;
      var v=answers[q.code];
```

```
      if (q.type==='multi'){ if (!Array.isArray(v) || v.length===0)
errs[q.code]='This question is required.'; }
      else if (v===undefined || v===null || v==='') errs[q.code]='This
question is required.';
    }
    for (var code in errs){ var
el=document.getElementById('err-'+code); if (el)
el.textContent=errs[code]; }
    return errs;
  }

  btnBack.addEventListener('click', function(){
    if (page>0){ page--; renderPage(); window.scrollTo({top:0,
behavior:'smooth'}); }
  });
  btnNext.addEventListener('click', function(){
    var errs=validateCurrentPage();
    if (Object.keys(errs).length){
      var first=Object.keys(errs)[0]; var
node=document.getElementById('q-'+first);
      if (node && node.scrollIntoView)
node.scrollIntoView({behavior:'smooth', block:'center'});
      return;
    }
    if (page < pages.length-1){ page++; renderPage();
window.scrollTo({top:0, behavior:'smooth'}); }
  });
  btnReview.addEventListener('click', function(){
    var errs=validateCurrentPage();
    if (Object.keys(errs).length){
      var first=Object.keys(errs)[0]; var
node=document.getElementById('q-'+first);
      if (node && node.scrollIntoView)
node.scrollIntoView({behavior:'smooth', block:'center'});
    } else {
      page = pages.length-1; renderPage(); window.scrollTo({top:0,
behavior:'smooth'});
    }
  });

  btnSubmit.addEventListener('click', async function(){
    var resBox=document.getElementById('pageMsg'); if (resBox){
resBox.className=''; resBox.textContent='Submitting…'; }
    try {
      var raw = await new Promise(function(resolve,reject){

google.script.run.withSuccessHandler(resolve).withFailureHandler(rejec
t).submitRow({ answers: answers });
```

```
      });
      var res = (typeof raw === 'string') ? JSON.parse(raw) : raw;
      if (!res || res.ok !== true) throw new Error(res && res.error ?
res.error : 'Unknown server response.');

      // Hide progress/footer and replace form with read-only summary
      var pw=document.querySelector('.progress-wrap'); if (pw)
pw.style.display='none';
      navFooter.hidden = true;

      var summary = h('div', {class:'section-card'}, [
        h('h2', {}, 'Thank you — your submission was received'),
        h('p', {class:'section-desc'}, 'Submission ID: ' +
res.submissionId)
      ]);
      for (var i=0;i<pages.length;i++){
        var pg=pages[i]; if (pg.kind!=='section') continue;
        var tbody=h('tbody'); var any=false;
        for (var j=0;j<pg.items.length;j++){
          var q=pg.items[j]; var val=toText(answers[q.code]); if
(!val) continue;
          any=true;
          tbody.appendChild(h('tr', {}, [
            h('td', {style:'width:30%;font-weight:600;border:1px solid
var(--border);padding:8px 10px;background:#f8fafc;'}, q.text),
            h('td', {style:'border:1px solid var(--border);padding:8px
10px;white-space:pre-wrap;'}, val)
          ]));
        }
        if (any){
          var table=h('table',{class:'table'});
          table.appendChild(h('thead', {}, [ h('tr', {}, [ h('th',
{colspan:'2'}, pg.name) ]) ]));
          table.appendChild(tbody);
          summary.appendChild(table);
        }
      }
      app.innerHTML=''; app.appendChild(summary);
window.scrollTo({top:0, behavior:'smooth'});
      document.onkeydown = null; // disable Enter nav after submit
    } catch(e) {
      if (resBox){ resBox.className='error';
resBox.textContent='Submit failed: '+(e && e.message ? e.message :
String(e)); }
    }
  });

  // Keyboard: Enter = Next/Submit (not in textarea)
```

```
  document.addEventListener('keydown', function(e){
    var tgt = e.target || e.srcElement;
    if (tgt && tgt.tagName === 'TEXTAREA') return;
    if (e.key === 'Enter'){
      e.preventDefault();
      if (page < pages.length - 1) btnNext.click();
      else btnSubmit.click();
    }
  });

  renderPage();
})();
</script>
```



How to create tables and a Summary of

Responses

- Adds a **menu**: **Survey → Build Summary & Charts**.
- Reads **Questions** and **Responses**.
- Produces two tabs:
    - **Summary**: per-question tables + charts (counts, Likert distributions, averages).
    - **Verbatims**: every non-empty **short/long** answer with submission metadata.

Top-level menu

```
function onOpen() {
  SpreadsheetApp.getUi()
    .createMenu('Survey')
    .addItem('Build Summary & Charts', 'buildSurveySummary')
    .addToUi();
}
```

- Runs when the sheet opens.
- Adds a custom menu so non-technical users can generate the dashboard with one click.

Main entry point

```
function buildSurveySummary() {
  var ss = SpreadsheetApp.getActive();
  var qSh = ss.getSheetByName('Questions');
  var rSh = ss.getSheetByName('Responses');
  ...
}
```

- Gets the active spreadsheet and both tabs.
- Bails with an alert if either is missing.

Prepare output sheets

```
var sSh = ensureSheet_(ss, 'Summary'); clearSheet_(sSh);
var vSh = ensureSheet_(ss, 'Verbatims'); clearSheet_(vSh);
```

- Creates **Summary** and **Verbatims** if they don't exist.
- **Clears** them (including old charts) to avoid duplicates.

Load input data

```
// Questions
var qHeaders = qSh.getRange(1,1,1,qlc).getValues()[0];
var qIdx = index_(qHeaders);
```

```
var qRows = qSh.getRange(2,1,Math.max(0, qlr-1),qlc).getValues();

// Responses
var rHeaders = rSh.getRange(1,1,1,rlc).getValues()[0];
var rIdx = index_(rHeaders);
var rRows = rSh.getRange(2,1,Math.max(0, rlr-1),rlc).getValues();
```

- Reads just the **used** cells for speed.
- Builds header → index lookups (`qIdx`, `rIdx`) so we can refer to columns by name.

Normalize question metadata

```
var questions = [];
qRows.forEach(function(r){
  // build {code, section, order, type, text, options, min, max}
});
questions.sort(...section then order...);
```

- Turns each question row into a structured object.
- Sorts by **section**, then **order** so the summary is human-friendly.

Summary header

```
sSh.getRange(1,1,3,2).setValues([
  ['Survey Summary',''],
  ['Generated at', new Date()],
  ['Total submissions', rRows.length]
]);
```

- Adds a small dashboard header at the top of **Summary**.
- Also sets column widths and freezes the header rows.

Verbatims header

```
vSh.getRange(1,1,1,5).setValues([['submission_id','timestamp','code','
question','answer']]);
```

- Prepares the **Verbatims** sheet to store open-text answers with metadata.

Per-question processing loop

For each question, the script writes a block into **Summary** (and sometimes into **Verbatims**):

1. **Insert a section title** when the section changes.
2. **Write the question title** (with its `code` for reference).
3. **Branch by type**:

A) `single` / `multi` (radio & checkbox)

```
// Tally counts by option; collect unexpected values as "(Other) ...".
var table = [['Option','Count'], ...];
// Write table
// Insert a bar chart next to it
```

- Counts how many responses chose each option.
- Unknown choices (e.g., options changed later) get grouped under “(Other) …”.
- Writes a **table** and a **bar chart**.

B) `likert` (e.g., 1–5 or 0–10)

```
// Build frequency for each score
// Compute average
// Write distribution table
// Insert a column chart with "(Avg: X.XX)"
// Write a small stats table (Responses, Average)
```

- Creates a score **distribution** (frequency for each number).
- Calculates an **average** and shows it in the chart title.
- (If you want NPS for 0–10 scales, you can extend here—see earlier variant.)

C) `short` / `long` (free text)

```
// Count non-empty answers
// Append each to Verbatims: submission_id, timestamp, code, question,
answer
// Write a tiny table with the non-empty count
```

- Pushes each text answer into **Verbatims** for reading/analysis.
- Adds a quick count summary to **Summary**.

Finishing touches

```
sSh.autoResizeColumns(1,3);
vSh.autoResizeColumns(1,5);
SpreadsheetApp.getUi().alert('Summary built. Open the "Summary" and
"Verbatims" sheets.');
```

- Auto-sizes columns for readability.
- Notifies the user when done.

Small helpers

```
function ensureSheet_(ss, name){ ... }   // get or create sheet
function clearSheet_(sh){ ... }          // remove charts + clear
content
function index_(headers){ ... }          // header name -> index map
function parseOptions_(s){ ... }         // "A|B|C" -> ["A","B","C"]
```

Why this pattern works

- **Sheet-driven**: No hardcoded question logic—new questions are automatically summarized.
- **Robust**: Unknown/changed options routed to **(Other)** so you don't lose counts.
- **Actionable**: Charts for choices, distributions for scales, and a dedicated **Verbatims** tab for qualitative review.

Customize ideas

- Add **NPS** metrics for 0–10 Likert questions (Promoters/Passives/Detractors).
- Compute **per-section averages** (e.g., average of all Likert questions in "Experience").
- Add conditional formatting to highlight low averages or top comments.

7) (Optional) `Summary.gs` — one-click charts + verbatims

Adds menu **Survey → Build Summary & Charts.** It makes a **Summary** sheet with bar/column charts and a **Verbatims** sheet with open-ended answers.

```
/**************************************************************
 * Summary & Charts (optional but recommended)
 **************************************************************/
function onOpen() {
  SpreadsheetApp.getUi()
    .createMenu('Survey')
    .addItem('Build Summary & Charts', 'buildSurveySummary')
    .addToUi();
}

function buildSurveySummary() {
  var ss = SpreadsheetApp.getActive();
  var qSh = ss.getSheetByName('Questions');
  var rSh = ss.getSheetByName('Responses');
  if (!qSh || !rSh) { SpreadsheetApp.getUi().alert('Questions or
```

```
Responses sheet not found.'); return; }

  var sSh = ensureSheet_(ss, 'Summary'); clearSheet_(sSh);
  var vSh = ensureSheet_(ss, 'Verbatims'); clearSheet_(vSh);

  // Questions
  var qlr = qSh.getLastRow(), qlc = qSh.getLastColumn();
  var qHeaders = qSh.getRange(1,1,1,qlc).getValues()[0];
  var qIdx = index_(qHeaders);
  var qRows = qSh.getRange(2,1,Math.max(0, qlr-1),qlc).getValues();

  // Responses
  var rlr = rSh.getLastRow(), rlc = rSh.getLastColumn();
  if (rlr < 2) { sSh.getRange(1,1).setValue('No responses yet.');
return; }
  var rHeaders = rSh.getRange(1,1,1,rlc).getValues()[0];
  var rIdx = index_(rHeaders);
  var rRows = rSh.getRange(2,1,Math.max(0, rlr-1),rlc).getValues();

  var questions = [], byCode = {};
  qRows.forEach(function(r){
    var code = String(r[qIdx.code]||'').trim(); if (!code) return;
    var meta = {
      code: code,
      section: String(r[qIdx.section]||'').trim(),
      order: Number(r[qIdx.order]||0),
      type: String(r[qIdx.type]||'').toLowerCase().trim(),
      text: String(r[qIdx.text]||'').trim(),
      options: parseOptions_(String(r[qIdx.options]||'')),
      min: Number(r[qIdx.scale_min]||0),
      max: Number(r[qIdx.scale_max]||0)
    };
    questions.push(meta); byCode[code]=meta;
  });
  questions.sort(function(a,b){ return a.section===b.section ? a.order
- b.order : String(a.section).localeCompare(String(b.section)); });

  // Header
  sSh.getRange(1,1,3,2).setValues([
    ['Survey Summary',''],
    ['Generated at', new Date()],
    ['Total submissions', rRows.length]
  ]);
  sSh.getRange(1,1,1,2).setFontWeight('bold').setFontSize(14);
  sSh.setFrozenRows(4);
  sSh.setColumnWidth(1, 380);
  sSh.setColumnWidth(2, 140);
  sSh.setColumnWidth(3, 140);
```

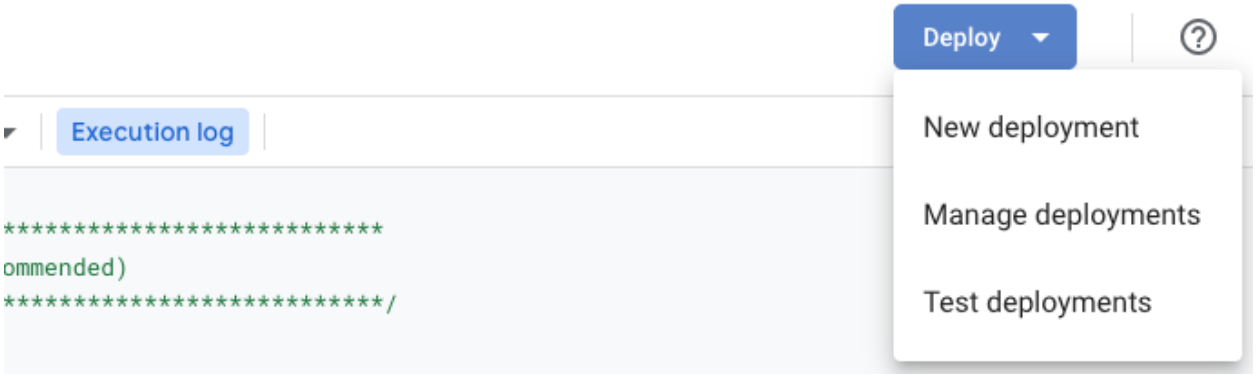
```
  for (var c=4;c<=12;c++) sSh.setColumnWidth(c, 110);

  // Verbatims header

vSh.getRange(1,1,1,5).setValues([['submission_id','timestamp','code','
question','answer']]).setFontWeight('bold');
  vSh.setFrozenRows(1);

  var row = 5, currentSection = null;

  questions.forEach(function(q){
    if (q.section !== currentSection) {

sSh.getRange(row,1).setValue(q.section).setFontWeight('bold').setFontS
ize(12);
      sSh.getRange(row,1,1,12).setBackground('#f3f4f6'); row++;
      currentSection = q.section;
    }

    sSh.getRange(row,1).setValue(q.text + '  [' + q.code +
']').setFontWeight('bold'); row++;

    if (!rIdx.hasOwnProperty(q.code)) {
sSh.getRange(row,1).setValue('No column in Responses for
'+q.code).setFontColor('#b91c1c'); row+=2; return; }
    var col = rIdx[q.code];

    var values = rRows.map(function(r){ return r[col];
}).filter(function(v){ return v!=='' && v!=null; });

    if (q.type==='single' || q.type==='multi') {
      var tally = {}; q.options.forEach(function(opt){ tally[opt]=0;
});
      var others = {};
      values.forEach(function(v){
        var parts = (q.type==='multi') ?
String(v).split(';').map(function(s){return
s.trim();}).filter(Boolean) : [String(v).trim()];
        parts.forEach(function(choice){
          if (tally.hasOwnProperty(choice)) tally[choice] += 1;
          else { if (!others[choice]) others[choice]=0;
others[choice]+=1; }
        });
      });
      var table = [['Option','Count']];
      q.options.forEach(function(opt){ table.push([opt,
tally[opt]||0]); });
      Object.keys(others).forEach(function(k){ table.push(['(Other)
```

```
'+k, others[k]]); });

      var h = table.length;
      sSh.getRange(row,1,h,2).setValues(table);

sSh.getRange(row,1,1,2).setFontWeight('bold').setBackground('#f8fafc')
;

      var chart = sSh.newChart().asBarChart()
        .setPosition(row,4,0,0)
        .addRange(sSh.getRange(row,1,h,2))
        .setOption('title', q.text)
        .setOption('legend',{position:'none'})
        .setOption('height', Math.max(220, 28*h))
        .build();
      sSh.insertChart(chart);
      row += h + 2;

    } else if (q.type==='likert') {
      var min=q.min||0, max=q.max||10, counts={}, total=0, sum=0;
      for (var s=min;s<=max;s++) counts[s]=0;
      values.forEach(function(v){
        var n=Number(v);
        if (!isNaN(n) && n>=min && n<=max){ counts[n]++; total++;
sum+=n; }
      });
      var avg = total ? sum/total : 0;
      var dist = [['Score','Count']]; for (var s2=min;s2<=max;s2++)
dist.push([s2, counts[s2]||0]);

      sSh.getRange(row,1,dist.length,2).setValues(dist);

sSh.getRange(row,1,1,2).setFontWeight('bold').setBackground('#f8fafc')
;

      var lchart = sSh.newChart().asColumnChart()
        .setPosition(row,4,0,0)
        .addRange(sSh.getRange(row,1,dist.length,2))
        .setOption('title', q.text + ' (Avg: ' + avg.toFixed(2) + ')')
        .setOption('legend',{position:'none'})
        .setOption('height', 260)
        .build();
      sSh.insertChart(lchart);

      var stats = [['Metric','Value'], ['Responses', total],
['Average', avg]];
      sSh.getRange(row,3,stats.length,2).setValues(stats);
```

```
sSh.getRange(row,3,1,2).setFontWeight('bold').setBackground('#f8fafc')
;

      row += Math.max(dist.length, stats.length) + 2;

    } else if (q.type==='short' || q.type==='long') {
      var count = 0, vOut=[];
      for (var ri=0; ri<rRows.length; ri++){
        var ans = rRows[ri][col];
        if (ans!=='' && ans!=null){
          count++;
          vOut.push([rRows[ri][rIdx.submission_id]||'',
rRows[ri][rIdx.timestamp]||'', q.code, q.text, ans]);
        }
      }
      if (vOut.length)
vSh.getRange(vSh.getLastRow()+1,1,vOut.length,5).setValues(vOut);
      var t = [['Responses (non-empty)','Count'], ['Total', count]];
      sSh.getRange(row,1,t.length,2).setValues(t);

sSh.getRange(row,1,1,2).setFontWeight('bold').setBackground('#f8fafc')
;
      row += t.length + 2;

    } else {
      sSh.getRange(row,1).setValue('Unsupported type: '+q.type); row
+= 2;
    }
  });

  sSh.autoResizeColumns(1,3); vSh.autoResizeColumns(1,5);
  SpreadsheetApp.getUi().alert('Summary built. Open the "Summary" and
"Verbatims" sheets.');
}

/*** small helpers ***/
function ensureSheet_(ss, name){ return ss.getSheetByName(name) ||
ss.insertSheet(name); }
function clearSheet_(sh){ sh.getCharts().forEach(function(c){
sh.removeChart(c); }); sh.clear(); }
function index_(headers){ var o={}; headers.forEach(function(h,i){
o[String(h)]=i; }); return o; }
function parseOptions_(s){ return
String(s||'').split('|').map(function(x){return
x.trim();}).filter(Boolean); }
```

Deploy the web app

1. In Apps Script → **Run** `seedStandardSurvey()` (authorize).
2. **Deploy → New deployment → Web app**
   - Execute as: **Me**
   - Who has access: **Anyone with the link** (or your domain)
3. Open the link → try the survey → check the `Responses` sheet.



How it works (concepts you can reuse)

1. **Data-driven form**
   - The form UI is **not hardcoded**. The `Questions` sheet defines a list of questions (code, type, text, options, required…).
   - `loadQuestions()` reads only non-blank rows and returns a **JSON string** to avoid browser clone/serialization issues.
2. **Multi-page flow**
   - Questions are grouped by `section`; each section renders as one page.
   - A final **Review** page summarizes answers and lets users return to any section to edit before submitting.
3. **Wide-format responses**
   - On submit, the app constructs **one row**: `timestamp`, `submission_id`, and each question `code` as a column.
   - Multi-select answers join with `;` — easy to read and pivot.
4. **Robust writes**
   - Uses a **document lock** + a **retry** wrapper to avoid "busy sheet" errors (`FAILED_PRECONDITION`) during concurrent submissions.
5. **Post-submit UX**
   - After a successful submit, the app **hides** the form and shows a **read-only**

summary of the answers provided.

6. **Summary & charts**
    - A menu action processes `Responses` and `Questions`, creating a `Summary` sheet with counts/averages and a `Verbatims` sheet for text answers.

Customize for your own survey

- **Change questions**: Edit rows in the `Questions` sheet (add/remove rows; keep the header names).

- **Types**:
    - `short` – single-line input
    - `long` – multi-line textarea
    - `single` – radio buttons; set **options** as `A|B|C`
    - `multi` – checkboxes; set **options** as `A|B|C`
    - `likert` – radio buttons in a range; set **scale_min/scale_max** (e.g., `1` and `5`)
- **Rebuild Responses headers**: If you add new codes, run the helper in `Setup.gs` → `buildResponsesHeaderFromQuestions_()` or just re-run `seedStandardSurvey()` (it will overwrite the header from your live `Questions`).

Troubleshooting

- **Nothing loads**:
    - Redeploy Web App (Manage deployments → **Edit** → **Deploy**).
    - Ensure `Questions` has the exact headers and at least one question.
    - If script is **standalone**, set `SPREADSHEET_ID` in `Code.gs`.
- **"dropping postMessage / deserialize error"**:
    - You're protected already—server returns a **JSON string**. Keep it that way.
- **"FAILED_PRECONDITION"**:
    - Handled by lock + retry. If it persists, ensure "Execute as: Me" and your account has edit access to the Sheet.