AI SUMMIT HANDS-ON LAB

Exercise 6: Build a Legislative Activity Tracker App

App Building • Public Data • Local-First • Human Labels • Verification

PREMIUM EXERCISE

You will need an AI subscription to do this exercise: ChatGPT (Codex), Claude Code, Gemini (Antigravity), or similar. Google does offer a free tier for their Antigravity coding agent—unknown whether you will be able to complete this exercise within its limits.

PARTICIPANT PACKET

In this exercise, you will ask an AI coding agent to build a small legislative-tracking app from official Maryland General Assembly data. The goal is not to learn programming. The goal is to see how far a clear specification, public data, and a capable coding agent can get you in one short build cycle.

The app will be a searchable, sortable, filterable bill table plus a lightweight notebook. You will assign your own internal position labels and notes; the app should not invent advocacy positions for you.

Suggested prompts are provided, but you are free to experiment once the core app works.

Workshop data rule

Use public legislative data only. Do not enter real client, case, employee, personnel, donor, strategy, or other confidential information into the coding tool or the prototype.

Recommended time 30 minutes

Best group size 2–4 participants per device

Participant outcome A working local browser app that loads official Maryland bill data, lets you find and filter bills, and saves your own position labels and notes on that device.

Start here

Today's constraint

Build the smallest useful version. No hosting. No accounts. No database. No API keys. No production deployment. Your app should run on just your laptop and be easy to show to the person sitting next to you.

What you need

- A laptop with a modern web browser and internet access.
- Access to an AI coding agent that can create and edit files. Examples include Codex, Claude Code, GitHub Copilot, and Google Antigravity.
- A way to give that agent an empty project folder. You do not need coding experience.

Do not spend the workshop installing a development stack

If your tool immediately asks you to install Node, React, Docker, a database, or a cloud service, steer it back to the prompt. The workshop target is a simple prototype that opens in a browser and runs entirely on your laptop (not the internet).

Scenario

Your organization follows legislation during the Maryland General Assembly session. Staff need one place to scan bills, find issues they care about, mark a bill as favorable/unfavorable/monitoring, and keep short internal notes. For today, you are building a personal local prototype — not a shared production system.

Core 30-minute work product

- A local bill table populated from the official 2026 Maryland General Assembly data.
- Search, sorting, and practical filters such as status, committee, subject, and your own position label.
- A bill detail view with official metadata, synopsis/history when available, an official MGA link, and an editable notes field.

What good work looks like

☐ The app opens without hosting or a complicated setup step.

☐ Official data and user-entered notes are visually distinct.

☐ Missing fields display as missing — the app does not invent metadata.

☐ Search/filter controls are obvious enough to use without instructions.

☐ The app does not assign political positions or recommendations on its own.

☐ A user can close/reopen or refresh the app and see whether their local labels/notes persist.

SOURCE A Official Maryland legislative data

OFFICIAL DATA SOURCE

Maryland General Assembly — Download Legislative Data

2026 Regular Session • JSON or CSV • MGA lists the 2026 download as updated hourly

Open the Maryland General Assembly site

The Maryland General Assembly provides downloadable legislative data in CSV and JSON formats. Its site describes the download as including bill number, sponsor, title, legislative status, synopsis, committee assignments, legislative history, hearing dates, and related fields. For this exercise, use the 2026 Regular Session data.

Why use the download instead of scraping bill pages?

It gives the coding agent one structured, official dataset to work from. That keeps the exercise focused on building the app rather than reverse-engineering a website.

Your prototype is a snapshot. For real session work, you would need a deliberate refresh process and a shared storage design.

Step 1 — Make an empty project folder

Create a new folder somewhere easy to find. Name it something like:

mga-legislative-tracker

Step 2 — Point your coding agent at that folder

Open the folder as a project/workspace, or start your coding agent from inside it. Give the agent permission to create files and, if your tool supports it, browse the web. Then paste the build prompt on the next pages.

If your agent cannot access the web

Open the MGA site above, use “Download Legislative Data,” download the 2026 Regular Session JSON, and place that file in your project folder. Then tell the agent: “Use the official MGA JSON file I just placed in this folder as the source data.”

The design trick that makes this workshop easy

Ask for one self-contained HTML file with the official bill data embedded during the build. That avoids package managers, local servers, databases, and browser cross-origin problems. When the agent is finished, the demo should be as simple as double-clicking index.html.

BUILD PROMPT Copy/paste this into your coding agent

Build prompt — Part 1 of 3

Build a polished, local-only legislative activity tracker for the Maryland General Assembly's 2026 Regular Session. Work entirely inside the current project folder.

This is a 30-minute workshop prototype for non-technical users. Optimize for reliability and simplicity, not architecture. Do NOT require cloud hosting, a database, an API key, Docker, Node/npm, Python, a local web server, or any other installation just to use the finished app.

DELIVERABLE / ARCHITECTURE
Create a self-contained index.html using vanilla HTML, CSS, and JavaScript. The finished app must work by double-clicking index.html in a modern browser. Do not use external libraries or CDNs.

During the build, retrieve the official 2026 Regular Session legislative-data JSON from the Maryland General Assembly's official website (mgaleg.maryland.gov), using its "Download Legislative Data" feature. Do not use third-party bill data. Inspect the actual schema; do not assume field names.

Embed the needed legislative data into index.html so the finished app does not need to fetch local files or call a remote service at runtime. If you cannot retrieve the official JSON yourself, stop and tell me exactly which official file to download and where to place it; once I provide it, continue without changing the rest of the design.

DATA HANDLING
Use the official dataset as the factual source. Preserve source values. Do not invent missing sponsors, committees, statuses, dates, subjects, summaries, or outcomes. Show a dash or "Not provided" when a field is absent. Use the dataset's own subject/category fields when available.

If a bill-detail URL is not supplied, create an official MGA link from the bill number/session only if you can verify the URL pattern. Do not generate, infer, or recommend a political position on any bill. Position labels are entirely user-entered.

BUILD PROMPT Continue — Part 2

Build prompt — Part 2 of 3

CORE UI
Create a modern, professional, responsive interface with a clear title, “2026 Regular Session,” a visible count of bills currently shown, and a small note that the official data is a snapshot. The main screen should be a dense but readable table suitable for legislative staff.

Include these table fields when present in the source data: Bill Number, Title, Sponsor, Committee, Legislative Status, Hearing Date, Subjects/Categories, My Position, and Notes indicator. It is fine to combine or omit a field if the official data clearly does not contain it.

INTERACTION REQUIREMENTS
1. Search across at least bill number, title, sponsor, subject/category, and synopsis when available.
2. Make sensible columns sortable.
3. Add filters for chamber/bill prefix, legislative status, committee, subject/category, and My Position. Include a one-click Reset Filters control.
4. Clicking a bill should open a detail panel or modal showing the most useful official information available: bill number/title, sponsor, status, committee, hearing information, subjects, synopsis, legislative history, and a link to the official MGA bill page.
5. In the detail view, let the user set My Position to: Not Reviewed, Favorable, Unfavorable, Monitor, or No Position.
6. Let the user add/edit free-text notes. Show in the table whether a bill has notes.
7. Save My Position and notes in browser localStorage, keyed by session + bill number. Official legislative data must remain read-only. If localStorage is unavailable, keep the app usable for the current session and show a small warning rather than failing.
8. Add a “Tracked only” filter that shows bills with a non-default position or notes.

BUILD PROMPT Finish — Part 3

Build prompt — Part 3 of 3

DESIGN / ACCESSIBILITY
Use a restrained professional visual design, not a flashy developer demo. Prioritize legibility. Use clear filter labels, good spacing, visible focus states, adequate contrast, keyboard-usable controls, and a sticky table header if practical. Use compact badges for status/position, but do not rely on color alone. Make it usable on a laptop-sized screen and reasonably responsive on smaller screens.

Include a small footer or info line stating that this is a local workshop prototype; official legislative status should be verified on the Maryland General Assembly site; and local notes stay on this browser/device unless exported.

QUALITY CHECK BEFORE YOU STOP
Open/test the app yourself if your environment allows browser testing. Confirm that search, sorting, filters, the detail view, position changes, notes, and persistence work. Check several randomly selected bills against the source JSON so field mapping is correct. Fix any obvious layout or console errors.

When finished, give me only:
A. the exact filename I should open;
B. one sentence explaining how to open it; and
C. any important limitation you could not solve in this local-only design.

Do not give me a tutorial on the code unless I ask.

If the agent asks you to choose a framework

Reply: “Use the simplest dependency-free version that opens by double-clicking index.html. No framework.”

Run it, then test it like a user

Step 3 — Open the finished app

When the coding agent finishes, find index.html in the project folder and double-click it. It should open in your default browser. If the agent produced a more complicated start procedure, ask it to simplify the project to the dependency-free, double-clickable version described in the prompt.

Five-minute acceptance test

☐ Search for a word that appears in a bill title and confirm the result set changes.

☐ Sort by Bill Number or Title, then apply at least two filters at once.

☐ Open a bill and confirm the detail view contains official metadata and an MGA link.

☐ Set one bill to Favorable, another to Unfavorable, and a third to Monitor. These are test labels, not recommendations.

☐ Add a short note to one bill, close/reopen or refresh the page, and see whether the label/note persists.

☐ Turn on “Tracked only” and confirm the expected bills remain visible.

Source check — verify the app, not just the code

Pick THREE bills from different parts of the table. For each, open the official MGA bill page or inspect the official source data and compare the fields below.

Bill	Title / Sponsor	Status	Committee / Hearing	Match?

Red flag

If the app presents an AI-created summary, position, “importance” score, or prediction as though it came from the official dataset, remove it. This exercise is about organizing public data and adding your own notes — not outsourcing advocacy judgments.

ITERATE: Make one useful improvement

If the core app works with time remaining, choose ONE improvement. Keep the iteration small enough that you can test it before the facilitator calls time.

Option A — Export your tracked list

Iteration prompt

Add an “Export tracked bills” button. Export only bills that have a position other than Not Reviewed or that have notes. Use a CSV download that includes bill number, title, official status, committee, subjects, My Position, and My Notes. Do not alter the source data or require a server. Test the export before you stop.

Option B — A mini dashboard

Iteration prompt

Add a compact summary strip above the table showing counts for all bills currently in the filtered view, plus counts by My Position. Clicking a position count should apply that position filter. Keep the table as the primary interface and do not infer anything about bills from these counts.

Option C — Make session refresh less painful

Iteration prompt

Without adding a server or build dependency, add a simple “About this snapshot” panel that records the session, source URL, and the date/time this embedded dataset was created. Then create a short REFRESH-INSTRUCTIONS.txt file explaining, in plain English, how a future user could ask a coding agent to rebuild the same app from a newer official MGA legislative-data download while preserving the same feature set.

Prototype limitation worth noticing

Browser localStorage is fine for a workshop demo, but it is not a shared case-management or collaboration system. Notes can be tied to one browser/profile/device and may disappear if browser storage is cleared. A real organizational tracker would need authentication, shared storage, permissions, backups, and a deliberate data-refresh process.

Core debrief — what did the AI actually do?

- Which parts of the result came from the official MGA dataset, and which parts came from your specification?
- What did the agent have to infer about the source schema or UI? Did any of those inferences need correction?
- Was the hardest part "coding," or describing exactly what a useful legislative workflow should do?
- Which acceptance test caught the most important problem?
- What would have to change before multiple staff could safely use this during a live session?
- What is one small internal tool in your own work that could be prototyped from public or non-confidential data the same way?

Use-it-Monday pattern

Public structured data → specific workflow → local prototype → human verification → one focused iteration

The coding agent can write a surprising amount of the implementation. Your job is to define the workflow, constrain the architecture, protect the data, and test whether the result actually matches the source and the work you need to do.

Your transfer idea

Name one recurring spreadsheet, public dataset, or manual tracking task from your own work that could become a small app:

Data/source:

What the user needs to do with it:

One feature the first prototype must have: