My backend part of the app, I am using streamlit to deploy the backend. I want to change that and upload with a custom link from google. Help me combine the frontend and backend part of the code. the backend part could change but it should have the key functions needed. . ├── \_\_pycache\_\_ │ └── test\_main.cpython-313-pytest-8.3.5.pyc ├── Notebooks │ ├── 01\_llm\_sanity.ipynb │ ├── calendar\_functions.ipynb │ ├── llm\_practice.ipynb │ ├── Open\_AI\_llm\_practive.ipynb │ └── unsloth\_llm\_practice.ipynb ├── project\_code │ ├── \_\_init\_\_.py │ ├── \_\_pycache\_\_ │ │ ├── \_\_init\_\_.cpython-313.pyc │ │ ├── auth.cpython-313.pyc │ │ ├── calendar\_methods.cpython-313.pyc │ │ ├── creating\_calendar.cpython-313.pyc │ │ ├── llm\_methods.cpython-313.pyc │ │ └── methods.cpython-313.pyc │ ├── auth.py │ ├── calendar\_methods.py │ ├── creating\_calendar.py │ ├── llm\_parse.py │ └── validation.py ├── project\_prompt.yaml ├── prompt.yaml ├── ReadMe.md ├── requirements.txt ├── run\_app.py ├── sandboxenv │ ├── bin │ │ ├── activate │ │ ├── activate.csh │ │ ├── activate.fish │ │ ├── Activate.ps1 │ │ ├── dateparser-download │ │ ├── debugpy │ │ ├── debugpy-adapter │ │ ├── distro │ │ ├── dotenv │ │ ├── f2py │ │ ├── flake8 │ │ ├── google-oauthlib-tool │ │ ├── httpx │ │ ├── ipython │ │ ├── ipython3 │ │ ├── jsonschema │ │ ├── jupyter │ │ ├── jupyter-kernel │ │ ├── jupyter-kernelspec │ │ ├── jupyter-migrate │ │ ├── jupyter-run │ │ ├── jupyter-troubleshoot │ │ ├── normalizer │ │ ├── numpy-config │ │ ├── openai │ │ ├── pip │ │ ├── pip3 │ │ ├── pip3.13 │ │ ├── py.test │ │ ├── pycodestyle │ │ ├── pyflakes │ │ ├── pygmentize │ │ ├── pyrsa-decrypt │ │ ├── pyrsa-encrypt │ │ ├── pyrsa-keygen │ │ ├── pyrsa-priv2pub │ │ ├── pyrsa-sign │ │ ├── pyrsa-verify │ │ ├── pytest │ │ ├── python -> python3.13 │ │ ├── python3 -> python3.13 │ │ ├── python3.13 -> /opt/homebrew/opt/python@3.13/bin/python3.13 │ │ ├── streamlit │ │ ├── streamlit.cmd │ │ ├── tqdm │ │ └── websockets │ ├── etc │ │ └── jupyter │ │ └── nbconfig │ │ └── notebook.d │ ├── include │ │ └── python3.13 │ ├── lib │ │ └── python3.13 │ │ └── site-packages │ │ ├── \_\_pycache\_\_ │ │ ├── \_pytest │ │ ├── \_yaml │ │ ├── altair │ │ ├── altair-5.5.0.dist-info │ │ ├── annotated\_types │ │ ├── annotated\_types-0.7.0.dist-info │ │ ├── anyio │ │ ├── anyio-4.10.0.dist-info │ │ ├── apiclient │ │ ├── appnope │ │ ├── appnope-0.1.4.dist-info │ │ ├── asttokens │ │ ├── asttokens-3.0.0.dist-info │ │ ├── attr │ │ ├── attrs │ │ ├── attrs-25.3.0.dist-info │ │ ├── blinker │ │ ├── blinker-1.9.0.dist-info │ │ ├── cachetools │ │ ├── cachetools-5.5.2.dist-info │ │ ├── certifi │ │ ├── certifi-2025.8.3.dist-info │ │ ├── charset\_normalizer │ │ ├── charset\_normalizer-3.4.3.dist-info │ │ ├── click │ │ ├── click-8.2.1.dist-info │ │ ├── comm │ │ ├── comm-0.2.3.dist-info │ │ ├── dateparser │ │ ├── dateparser\_cli │ │ ├── dateparser\_data │ │ ├── dateparser\_scripts │ │ ├── dateparser-1.2.2.dist-info │ │ ├── dateutil │ │ ├── debugpy │ │ ├── debugpy-1.8.17.dist-info │ │ ├── decorator-5.2.1.dist-info │ │ ├── decorator.py │ │ ├── decouple.py │ │ ├── distro │ │ ├── distro-1.9.0.dist-info │ │ ├── dotenv │ │ ├── executing │ │ ├── executing-2.2.1.dist-info │ │ ├── flake8 │ │ ├── flake8-7.3.0.dist-info │ │ ├── git │ │ ├── gitdb │ │ ├── gitdb-4.0.12.dist-info │ │ ├── gitpython-3.1.45.dist-info │ │ ├── google │ │ ├── google\_api\_core-2.25.1.dist-info │ │ ├── google\_api\_python\_client-2.179.0.dist-info │ │ ├── google\_auth\_httplib2-0.2.0.dist-info │ │ ├── google\_auth\_httplib2.py │ │ ├── google\_auth\_oauthlib │ │ ├── google\_auth\_oauthlib-1.2.2.dist-info │ │ ├── google\_auth-2.40.3.dist-info │ │ ├── google\_genai-1.39.1.dist-info │ │ ├── googleapiclient │ │ ├── googleapis\_common\_protos-1.70.0.dist-info │ │ ├── h11 │ │ ├── h11-0.16.0.dist-info │ │ ├── httpcore │ │ ├── httpcore-1.0.9.dist-info │ │ ├── httplib2 │ │ ├── httplib2-0.22.0.dist-info │ │ ├── httpx │ │ ├── httpx-0.28.1.dist-info │ │ ├── idna │ │ ├── idna-3.10.dist-info │ │ ├── iniconfig │ │ ├── iniconfig-2.1.0.dist-info │ │ ├── ipykernel │ │ ├── ipykernel\_launcher.py │ │ ├── ipykernel-6.30.1.dist-info │ │ ├── IPython │ │ ├── ipython\_pygments\_lexers-1.1.1.dist-info │ │ ├── ipython\_pygments\_lexers.py │ │ ├── ipython-9.5.0.dist-info │ │ ├── jedi │ │ ├── jedi-0.19.2.dist-info │ │ ├── jinja2 │ │ ├── jinja2-3.1.6.dist-info │ │ ├── jiter │ │ ├── jiter-0.10.0.dist-info │ │ ├── jsonschema │ │ ├── jsonschema\_specifications │ │ ├── jsonschema\_specifications-2025.4.1.dist-info │ │ ├── jsonschema-4.25.1.dist-info │ │ ├── jupyter\_client │ │ ├── jupyter\_client-8.6.3.dist-info │ │ ├── jupyter\_core │ │ ├── jupyter\_core-5.8.1.dist-info │ │ ├── jupyter.py │ │ ├── markupsafe │ │ ├── MarkupSafe-3.0.2.dist-info │ │ ├── matplotlib\_inline │ │ ├── matplotlib\_inline-0.1.7.dist-info │ │ ├── mccabe-0.7.0.dist-info │ │ ├── mccabe.py │ │ ├── narwhals │ │ ├── narwhals-2.1.2.dist-info │ │ ├── nest\_asyncio-1.6.0.dist-info │ │ ├── nest\_asyncio.py │ │ ├── numpy │ │ ├── numpy-2.3.2.dist-info │ │ ├── oauthlib │ │ ├── oauthlib-3.3.1.dist-info │ │ ├── openai │ │ ├── openai-1.103.0.dist-info │ │ ├── packaging │ │ ├── packaging-25.0.dist-info │ │ ├── pandas │ │ ├── pandas-2.3.1.dist-info │ │ ├── parso │ │ ├── parso-0.8.5.dist-info │ │ ├── pexpect │ │ ├── pexpect-4.9.0.dist-info │ │ ├── PIL │ │ ├── pillow-11.3.0.dist-info │ │ ├── pip │ │ ├── pip-25.0.1.dist-info │ │ ├── platformdirs │ │ ├── platformdirs-4.4.0.dist-info │ │ ├── pluggy │ │ ├── pluggy-1.6.0.dist-info │ │ ├── prompt\_toolkit │ │ ├── prompt\_toolkit-3.0.52.dist-info │ │ ├── proto │ │ ├── proto\_plus-1.26.1.dist-info │ │ ├── protobuf-6.32.0.dist-info │ │ ├── psutil │ │ ├── psutil-7.1.0.dist-info │ │ ├── ptyprocess │ │ ├── ptyprocess-0.7.0.dist-info │ │ ├── pure\_eval │ │ ├── pure\_eval-0.2.3.dist-info │ │ ├── py.py │ │ ├── pyarrow │ │ ├── pyarrow-21.0.0.dist-info │ │ ├── pyasn1 │ │ ├── pyasn1\_modules │ │ ├── pyasn1\_modules-0.4.2.dist-info │ │ ├── pyasn1-0.6.1.dist-info │ │ ├── pycodestyle-2.14.0.dist-info │ │ ├── pycodestyle.py │ │ ├── pydantic │ │ ├── pydantic\_core │ │ ├── pydantic\_core-2.33.2.dist-info │ │ ├── pydantic-2.11.7.dist-info │ │ ├── pydeck │ │ ├── pydeck-0.9.1.dist-info │ │ ├── pyflakes │ │ ├── pyflakes-3.4.0.dist-info │ │ ├── pygments │ │ ├── pygments-2.19.2.dist-info │ │ ├── pyparsing │ │ ├── pyparsing-3.2.3.dist-info │ │ ├── pytest │ │ ├── pytest-8.4.1.dist-info │ │ ├── python\_dateutil-2.9.0.post0.dist-info │ │ ├── python\_decouple-3.8.dist-info │ │ ├── python\_dotenv-1.1.1.dist-info │ │ ├── pytz │ │ ├── pytz-2025.2.dist-info │ │ ├── PyYAML-6.0.2.dist-info │ │ ├── pyzmq-27.1.0.dist-info │ │ ├── referencing │ │ ├── referencing-0.36.2.dist-info │ │ ├── regex │ │ ├── regex-2025.9.1.dist-info │ │ ├── requests │ │ ├── requests\_oauthlib │ │ ├── requests\_oauthlib-2.0.0.dist-info │ │ ├── requests-2.32.5.dist-info │ │ ├── rpds │ │ ├── rpds\_py-0.27.0.dist-info │ │ ├── rsa │ │ ├── rsa-4.9.1.dist-info │ │ ├── six-1.17.0.dist-info │ │ ├── six.py │ │ ├── smmap │ │ ├── smmap-5.0.2.dist-info │ │ ├── sniffio │ │ ├── sniffio-1.3.1.dist-info │ │ ├── st\_aggrid │ │ ├── stack\_data │ │ ├── stack\_data-0.6.3.dist-info │ │ ├── streamlit │ │ ├── streamlit\_aggrid-1.1.8.post1.dist-info │ │ ├── streamlit\_autorefresh │ │ ├── streamlit\_autorefresh-1.0.1.dist-info │ │ ├── streamlit-1.48.1.dist-info │ │ ├── tenacity │ │ ├── tenacity-9.1.2.dist-info │ │ ├── toml │ │ ├── toml-0.10.2.dist-info │ │ ├── tornado │ │ ├── tornado-6.5.2.dist-info │ │ ├── tqdm │ │ ├── tqdm-4.67.1.dist-info │ │ ├── traitlets │ │ ├── traitlets-5.14.3.dist-info │ │ ├── typing\_extensions-4.14.1.dist-info │ │ ├── typing\_extensions.py │ │ ├── typing\_inspection │ │ ├── typing\_inspection-0.4.1.dist-info │ │ ├── tzdata │ │ ├── tzdata-2025.2.dist-info │ │ ├── tzlocal │ │ ├── tzlocal-5.3.1.dist-info │ │ ├── uritemplate │ │ ├── uritemplate-4.2.0.dist-info │ │ ├── urllib3 │ │ ├── urllib3-2.5.0.dist-info │ │ ├── wcwidth │ │ ├── wcwidth-0.2.13.dist-info │ │ ├── websockets │ │ ├── websockets-15.0.1.dist-info │ │ ├── yaml │ │ └── zmq │ ├── pyvenv.cfg │ └── share │ ├── jupyter │ │ ├── kernels │ │ │ └── python3 │ │ └── nbextensions │ │ └── pydeck │ └── man │ └── man1 │ └── ipython.1 ├── streamlit\_app │ ├── \_\_init\_\_.py │ ├── \_\_pycache\_\_ │ │ ├── \_\_init\_\_.cpython-313.pyc │ │ ├── calendar\_utils.cpython-313.pyc │ │ ├── main.cpython-313.pyc │ │ └── ui.cpython-313.pyc │ ├── main.py │ └── ui.py ├── test\_main.py ├── Tests │ ├── \_\_init\_\_.py │ ├── \_\_pycache\_\_ │ │ ├── \_\_init\_\_.cpython-313.pyc │ │ ├── hash\_test.cpython-313-pytest-8.3.5.pyc │ │ ├── local\_cache\_helpers.cpython-313-pytest-8.3.5.pyc │ │ ├── test\_calendar\_api.cpython-313-pytest-8.3.5.pyc │ │ ├── test\_calendar\_functions.cpython-313-pytest-8.3.5.pyc │ │ ├── test\_creating\_calendar.cpython-313-pytest-8.3.5.pyc │ │ └── test\_parsers.cpython-313-pytest-8.3.5.pyc │ ├── hash\_test.py │ ├── local\_cache\_helpers.py │ ├── test\_calendar\_api.py │ ├── test\_calendar\_functions.py │ ├── test\_creating\_calendar.py │ └── test\_events\_local.py └── UserData ├── calendar\_id.txt └── user\_input\_format.md 240 directories, 107 files THis above is the tree structure for my calendar app. I want to upload this front end part: . ├── README.md ├── apphosting.yaml ├── components.json ├── docs │   └── blueprint.md ├── next.config.ts ├── package-lock.json ├── package.json ├── postcss.config.mjs ├── public ├── src │   ├── ai │   │   ├── dev.ts │   │   ├── flows │   │   │   ├── chat-to-events.ts │   │   │   └── suggest-event-changes.ts │   │   └── genkit.ts │   ├── app │   │   ├── bulk-import │   │   │   └── page.tsx │   │   ├── chat-parser │   │   │   └── page.tsx │   │   ├── event-builder │   │   │   └── page.tsx │   │   ├── favicon.ico │   │   ├── globals.css │   │   ├── help │   │   │   └── page.tsx │   │   ├── layout.tsx │   │   ├── manage-events │   │   │   └── page.tsx │   │   ├── page.tsx │   │   └── settings │   │   └── page.tsx │   ├── components │   │   ├── auth │   │   │   └── auth-button.tsx │   │   ├── home │   │   │   ├── event-popover.tsx │   │   │   ├── preview-calendar.tsx │   │   │   ├── signed-in-view.tsx │   │   │   └── signed-out-view.tsx │   │   ├── layout │   │   │   ├── app-chrome.tsx │   │   │   ├── app-header.tsx │   │   │   └── app-sidebar.tsx │   │   └── ui │   │   ├── accordion.tsx │   │   ├── alert-dialog.tsx │   │   ├── alert.tsx │   │   ├── avatar.tsx │   │   ├── badge.tsx │   │   ├── button.tsx │   │   ├── calendar.tsx │   │   ├── card.tsx │   │   ├── carousel.tsx │   │   ├── chart.tsx │   │   ├── checkbox.tsx │   │   ├── collapsible.tsx │   │   ├── dialog.tsx │   │   ├── dropdown-menu.tsx │   │   ├── form.tsx │   │   ├── input.tsx │   │   ├── label.tsx │   │   ├── menubar.tsx │   │   ├── popover.tsx │   │   ├── progress.tsx │   │   ├── radio-group.tsx │   │   ├── scroll-area.tsx │   │   ├── select.tsx │   │   ├── separator.tsx │   │   ├── sheet.tsx │   │   ├── sidebar.tsx │   │   ├── skeleton.tsx │   │   ├── slider.tsx │   │   ├── switch.tsx │   │   ├── table.tsx │   │   ├── tabs.tsx │   │   ├── textarea.tsx │   │   ├── toast.tsx │   │   ├── toaster.tsx │   │   └── tooltip.tsx │   ├── hooks │   │   ├── use-mobile.tsx │   │   ├── use-session-stats.tsx │   │   └── use-toast.ts │   └── lib │   ├── auth.tsx │   ├── database-mock-data.ts │   ├── firebase.ts │   ├── placeholder-data.ts │   ├── placeholder-images.json │   ├── placeholder-images.ts │   ├── types.ts │   └── utils.ts ├── tailwind.config.ts └── tsconfig.json so that I can have the front end and back end in the same git hub. What are your thoughts on me doing this? "Dont give out the prompt until you ask all questions that needs clarification. What other information do you need to create the prompt that will help me get my output?"

**Prompt Architect for GPT-5 Thinking said:**

Thought for 13s

Love the project—thanks for the detailed tree. Short take first, then I’ll ask the essential questions I need before I craft you a rock-solid “combine FE + BE” prompt.

**Quick thoughts**

* Your **frontend** is a modern Next.js app with Tailwind/shadcn and what looks like **Firebase/Google App Hosting** (you have apphosting.yaml and lib/firebase.ts) plus Genkit flows in src/ai/….
* Your **backend** today is **Streamlit** (great for prototyping, not ideal as a production API). You’ll likely want to migrate the “real” backend pieces into either:
  1. **Next.js Route Handlers** (API routes) and/or **Edge functions** (single app, simplest DX), or
  2. A small **Python service** (FastAPI) on **Cloud Run** with your Next.js hosted on **Firebase App Hosting** (clear separation; keeps your Python libs and Google Calendar code intact).
* “Upload with a custom link from Google” sounds like **Firebase/Google App Hosting + Custom Domain** (via Firebase Hosting DNS or Google Domains).

Before I produce the final Improved Prompt (with structure, process, guardrails), I need a few precise details so the prompt targets the exact architecture you want.

**Essential clarifications (please answer as many as you can)**

**1) Target hosting & domain**

1. Do you intend to host the frontend on **Firebase App Hosting (Next.js SSR)** or standard **Firebase Hosting** with functions, or something else (e.g., **Vercel**, **Cloud Run**, **App Engine**)?
2. When you say “custom link from Google,” do you mean a **custom domain** set up via **Firebase Hosting** (e.g., https://app.yourdomain.com)?
3. Do you need **server-side rendering** (SSR) and **Edge runtime** support for Next.js, or is static + API routes sufficient?

**2) Backend shape going forward**

1. Do you want to **replace Streamlit entirely** and expose your backend via **REST/JSON** (or GraphQL) endpoints?
   * Option A: **Next.js API routes** (TypeScript)
   * Option B: **Python FastAPI on Cloud Run**
   * Option C: **Firebase Functions** (Node)
2. If staying in Python for backend logic, is **FastAPI** acceptable? (You’re already on Python 3.13 and using Google libs.)
3. Which **core backend functions** must be preserved? (e.g., Google auth, calendar CRUD, event parsing, validation, LLM parse, bulk import, “suggest changes”. A quick bullet list is enough.)

**3) Google integrations**

1. How do users authenticate today—**Google OAuth** (user consent) or **Service Account impersonation**?
2. Where do you want to store **Google OAuth tokens** (if any)? (Firestore, server session, encrypted cookies?)
3. Do you already have a **Google Cloud project** + **OAuth client IDs** set up? If yes, will the frontend and backend share the same GCP project?

**4) Data & config**

1. Where is data stored now? Any DB beyond Google Calendar (e.g., **Firestore**, **Postgres**, files in UserData/)?
2. How should we manage **secrets** and config? (Firebase env, Google Secret Manager, .env files?)
3. Any **multi-tenant** needs (per-user calendars) or just single user?

**5) LLM & AI flows**

1. Your frontend has src/ai/genkit.ts and flows—are you using **Firebase Genkit** or another provider?
2. Does the **backend** currently call OpenAI/Google LLMs, or is that shifting to the **frontend/Edge**?
3. Any **latency or cost constraints** that push logic to Edge vs server?

**6) Monorepo + toolchain**

1. Are you comfortable with a **monorepo** like:

/apps/frontend (Next.js)

/apps/backend (FastAPI or Node functions)

/packages/shared (types, DTOs, validators)

Or do you prefer to fold the backend into **Next.js API routes** only?  
17. Node package manager preference (**npm**, **pnpm**, **yarn**)?  
18. Python env preference (**uv/poetry/pip + venv**)? You currently have a venv (sandboxenv/), keep or rebuild?

**7) Testing & CI/CD**

1. You have pytest tests—should CI run both **frontend tests** (Vitest/Playwright?) and **backend pytest**?
2. Preferred CI: **GitHub Actions**? Want **preview deployments** on PRs?

**8) Frontend UX expectations**

1. Should the app be **auth-gated** (require Google sign-in) before showing calendars?
2. Any specific pages in the new FE that must call backend endpoints (e.g., manage-events, bulk-import)? Map page → API desired if you can.

**9) Migration constraints**

1. Any **must-keep file paths** or module names from the Python code (e.g., keep project\_code/calendar\_methods.py logic)?
2. Are you okay **refactoring** Streamlit-specific code into pure library modules (decoupling UI from logic)?

**10) Domain & DNS**

1. Do you already have a **domain**? If yes, where is DNS managed (**Google Domains**, Cloudflare, GoDaddy)?
2. Do you want the backend under a subdomain (e.g., api.yourdomain.com) and the Next.js app at app.yourdomain.com or both under one host with /api routes?

If you give me those answers, I’ll generate a tailored **Improved Prompt** that:

* Specifies the **final architecture** (Next.js API vs Cloud Run FastAPI),
* Lays out the **monorepo structure**,
* Includes **deployment steps** (Firebase/Google App Hosting + custom domain),
* Adds **security & secrets** handling,
* Provides **shared types/interfaces** (OpenAPI → TS),
* And bakes in **CI/CD** and **tests**.

Once you reply, I’ll deliver:

1. **Strengths & Weaknesses** of your current approach,
2. A full **XML Sandwich + Perfection Loop** prompt you can run with GPT to produce code changes and scaffolding,
3. The **Rationale**,
4. Only the **critical missing details** (if any), and
5. **Micro add-ons** (guardrails like “Risk pass,” “Sanity check,” “BLUF first,” etc.).

**You said:**

ProjectOverview.md

File

I dont want the firebase app now. I want to migrate from firebase hosting to my custom url google iap web site domain type shit. For the database I want to use the google cloud data base with postgreSQL, 1.) I want to host the web app with google's custom domain, no firebase. 2.) no firebase hosting. Probably google IAP and google cloud base hosting? 3.) I hav eno idea what this is, again, this is my first time doing the web app coding. I only code the backend functions to build and train models. 4.) yes preferably, I would like to do that. but i wouldnt delete it. I would just keep it there but will not use it. 5.) I hav eno idea what Fast API is. but i think yea. 6.) I will get to that later. I have a lot of core back end functions and I want to use codex to help me do this. 7.) users shoud long in with thier google account, my streamlit app has that option .Where the user logs in and the app works. If not it shouldnt work becuase the app basically connects to your google calendar. 8.) Google OAuth tokens should be stored encrypted, every tokens should be stored encrypted. I am not sure how other web application developers do it but I was thinking of storing it in the google cloud database as a separate file where only I can look at that. 9.) I want to start from scratch. Help me redo this part so that front end and backend share the same Google Cloud Project. Sharing the same GCP makes more sense 10.) data is not currently being stored. The tokens and keys are being stored locally. If i want to use stream lit I am using streamlit's temp stoarge which is not that good. So I need to connect to google database to store data like user's logins so that i can bring user's usage data and things like that. 11.) I dont know, but probably google secret manager? What do other web applications do to manage their secrets? 12.) Not sure what this means but each user should have their own databse, user's should not get their user information mixed up with other users. How do other web application manages their user's information and their data for their account? 13.) src/ai/.genkit.ts ? idk what this is but I will probably not want to use firebase since I want to migrate to google cloud. I have a lot of firebase code in the path because I used firebase to design the front end version of the web application 14.) Not open ai, I am still working on the development of the LLMs, but using gemenai for now. Will change later 15.) Cost should be low as possible, i wanted to use the free credits that google gives when starting a new project? 16.) no firs ttime looking at that. I need more simpler explanations to help me understand these terminologies or front end and backend things 17.) idc, whatever the other start ups use to build their web application 18.) I could rebuild it, Ill use codex to clean this up sot hat requirements .txt has only the libraries I use and need. 19.) I dont know much about testing. So this will be your choice. Whatever is needed use it. I trust you. I was planning to not do any testing but figured that might not be the best move. 20.) I have no clue what that is 21.) yes, require google sign in before showing calendars 22.) Well, I want to first make sure that I can easily update from new front end changes? 23.) I want to keep all files from my calendar file. I will use codex to help me remove unnessecary files. 24.) yes, but I will not touch the streamlit code. I will leave it there just in case I have to return back to the streamlit version. I will only delete that streamlit code and libraries once I have a working web application. 25.) No, no domain. I dont even know how to creat a web application 26.) not sure why we need a domainf or the back end? I thought I just have to get a link for the front end so that the user can use the web application? And that the backend just comes with the front end so tha tthe user cannot see any code just the ui? 27.) Remember I just want a good prompt so that i can use it, make sure the prompt describes details of the background information on this project. Also go over the project overview .md file to get more details, make sure to include detailed backgroud session. For the output I want short, summarized steps in high level. I will be asking questions often so keep your responses short dont dump in all steps at once. If you are going to do that just explain the steps in bullet points very high level.

**Prompt Architect for GPT-5 Thinking said:**

Thought for 12s

got it — thanks! i won’t drop the prompt yet. here are the **only remaining essentials** I need so I can generate a tight, tailored “combine FE+BE on GCP (no Firebase)” prompt for you.

**last key clarifications (super short)**

1. **Audience & IAP**: Is this app **public to anyone with Google login** or **restricted to a small allow-list (internal)**?

note: **IAP** is great for internal/allow-list. For public consumer apps, standard Google OAuth inside the app is typical (not IAP).

1. **Target architecture (pick one for v1)**
   * **A. FastAPI (Python) on Cloud Run** for backend + **Next.js on Cloud Run** for frontend, both on **one custom domain** (e.g., app.yourdomain.com + api.yourdomain.com).
   * **B. Next.js only** with **Route Handlers (TS) as API**, on Cloud Run (simpler, but moves backend logic to TS).  
     Which do you want?
2. **Custom domain**: Do you want me to assume we’ll **buy a new domain** (e.g., via Google Domains/Cloud Domains) and point it to Cloud Run via HTTPS Load Balancer?
3. **Postgres region/size**: Any preferred **GCP region** (e.g., us-central1) and tiny starter tier for **Cloud SQL for Postgres** (cheapest to use free credits)?
4. **OAuth consent**: Is your Google OAuth app intended for **external production users** (will need scopes like Calendar; may require app verification) or **internal/testing** for now?
5. **Token storage approach**: OK to store **Google refresh tokens** in Postgres encrypted with **Cloud KMS** (envelope encryption), with **client secrets in Secret Manager**?
6. **Session strategy**: Prefer **server sessions (httpOnly cookies)** handled by backend (FastAPI or Next.js API) after OAuth callback?
7. **LLM for now**: Stick with **Gemini** via Google AI Studio / Vertex AI? (I’ll stub a provider layer so you can swap later.)
8. **Monorepo layout**: OK with
9. /apps/frontend (Next.js)
10. /apps/backend (FastAPI)
11. /packages/shared (schemas/types)

so you can keep your Python code mostly intact?

1. **Short-term goal**: For the first milestone, is success = **Sign in with Google → list user’s Google Calendars → show events** (read-only) from DB+Calendar, with tokens stored in Postgres?

Reply with quick picks (just numbers + letters are fine). Once I have these, I’ll deliver a **short, high-level, bullet-point Improved Prompt** (with background context embedded) that you can drop into Codex/GPT to scaffold the repo, wiring, and deploy steps.

**You said:**

1.) I will be using standard one but I want to know about both since I will use the IAP later with a different project. but for this project we should use google oauth for consumer apps. 2.) target architecture-> for modern websites, I dont see two links you know? If I go to some other web application I just see the GUI. That is what I want. 3.) yea, I want to test that out. I would prefer a free one for testing purposes. 4.) I have no clue, but us central is good .Not sure about the database, this is my first time building this web app with a database. I was going to use docker containers, for db, backend, frontend so its better to upload my web app? 5.) OAuth consent, would need app verification... but for now lets not keep anything complicated. Just do internal testing for now, so people with testing access can only view it i think? right? its inteded for external production users. It will require app verification. 6.) not sure, is it okay? what do other web application do tha trequires google tokens and shit? Do what they do 7.) No clue what this means, just do whatever is necessary and whatever the other web applicaiton developers use. 8.) Stick with gemini via google ai studio... idk what vertex ai is. I will deal with this later. 9.) what is packages shared chemas? is this the database? 10.) yea something like that should be good for now.

**Prompt Architect for GPT-5 Thinking said:**

Thought for 7s

awesome — based on your answers, here’s a tight plan and the prompt you asked for (kept short + high-level). we’ll use **Next.js on Cloud Run** (single URL GUI) with **Google OAuth**, **Postgres (Cloud SQL)**, **Secret Manager**, and **KMS-encrypted refresh tokens**. I also added a mini note on **IAP** for later.

**Strengths & Weaknesses (super short)**

**Strengths**

* Clear requirement: single URL GUI, Google sign-in first, Calendar access.
* Willing to use standard, proven GCP pieces (Cloud Run, Cloud SQL, Secret Manager, KMS).
* Keep Python code around (safe fallback).

**Gaps / risks**

* First deployment: OAuth setup + callback URLs + DB networking to Cloud Run.
* Token security: must do refresh-token storage correctly (KMS + Postgres).
* App verification later (when you go public).

**Improved Prompt (XML Sandwich + Perfection Loop)**

<context> You are building a modern web app for consumer users that signs in with Google, reads their Google Calendar, and shows events. The app should be a \*\*single Next.js site\*\* (TypeScript) deployed to \*\*Cloud Run\*\* behind HTTPS at a \*\*custom domain\*\* (test domain is fine). No Firebase. Use \*\*Google OAuth\*\* (internal/testing for now), store \*\*refresh tokens\*\* in \*\*Cloud SQL for Postgres\*\* with \*\*envelope encryption via Cloud KMS\*\*, and keep secrets in \*\*Secret Manager\*\*. Target region: \*\*us-central1\*\*. Costs should be minimal (smallest Cloud SQL tier, autoscaling Cloud Run). Existing Python/Streamlit code stays in the repo but is not used. Short-term success = user signs in → we fetch their Google Calendars → display events; DB stores users, sessions, and encrypted tokens. </context> <task> 1) Scaffold a \*\*monorepo\*\* with: - `/apps/frontend` → Next.js 14/15 (App Router), Tailwind/shadcn, route handlers under `/app/api/\*`. - `/packages/shared` → shared Zod schemas & TS types (DTOs), small token crypto wrapper (KMS client). 2) Implement \*\*Google OAuth flow\*\*: - Next.js route handlers for `/api/auth/login`, `/api/auth/callback`, `/api/auth/logout`. - Use \*\*httpOnly secure cookies\*\* for session; store session in Postgres with expiry. 3) \*\*Token storage\*\*: - On callback: store `refresh\_token` encrypted with \*\*KMS (envelope)\*\* in Postgres table `oauth\_tokens(user\_id, provider, enc\_blob, created\_at, updated\_at)`. 4) \*\*Calendar API\*\*: - `/api/calendar/list` returns calendars; `/api/calendar/events?calendarId=...&from=...&to=...` returns events. - Use Google Calendar API with refreshed access tokens on server only. 5) \*\*DB (Cloud SQL Postgres)\*\*: - Tables: `users`, `sessions`, `oauth\_tokens`, `usage\_events` (for simple analytics). - Migrations via Prisma or SQL files (keep it minimal). 6) \*\*Secrets & config\*\*: - Store `GOOGLE\_CLIENT\_ID`, `GOOGLE\_CLIENT\_SECRET` in \*\*Secret Manager\*\*. - KMS key: `projects/<proj>/locations/us-central1/keyRings/app/cryptoKeys/tokens`. - Next runtime reads via \*\*GCP Secret Manager Access\*\*. 7) \*\*Deploy\*\*: - Dockerfile for Next.js (Node 20+), build & start server. - Cloud Run service `calendar-web` (min instances 0/1, cpu-throttling). - Private Cloud SQL connector or TCP with IAM DB auth. 8) \*\*Testing hooks\*\* (lightweight): - Route handler unit tests (auth, calendar) with mocked Google APIs. - Health check `/api/healthz`. </task>

<output\_format>

* Output code changes as file trees with concise snippets.
* Provide shell commands for: GCP setup, OAuth credentials, KMS, Secret Manager, Cloud SQL, Cloud Run deploy.
* Keep explanations terse, bullets only.  
  </output\_format>

<quality\_bar>

* Security: httpOnly cookies, CSRF on OAuth state, no tokens in client JS.
* Reliability: graceful error messages, retries on token refresh, timeouts.
* Cost: smallest SQL tier; Cloud Run min instances = 0 or 1; region us-central1.
* DX: .env.example for local dev; scripts for migrate/seed.  
  </quality\_bar>

<verbosity>concise; bullet points; no walls of text.</verbosity>

<process> 1) Create monorepo structure and minimal pages: `/app/page.tsx` (signed-out), protected `/app/manage-events`. 2) Implement OAuth routes, session cookie, Postgres adapters. 3) Add Google Calendar client with token refresh helper. 4) Wire Secret Manager + KMS, create SQL schema + migrations. 5) Local run with Docker Compose (web + Postgres), then switch to Cloud SQL on deploy. 6) CI (GitHub Actions): lint, typecheck, test, build; deploy on `main`. </process> <finalize> - Print a short “Runbook” with: create GCP project, enable APIs, create OAuth client (Web), set authorized redirect URI (`/api/auth/callback`), create KMS key, create SQL instance, store secrets, deploy Cloud Run, map temp domain; later attach custom domain via HTTPS LB. - Provide a one-screen checklist for go-live. </finalize> <!-- Router nudge -->

If the request needs Python APIs instead, propose a parallel /apps/backend FastAPI on Cloud Run and keep the same DB/KMS/Secret Manager; otherwise proceed with Next.js-only API routes.  
</process>

**Rationale (why these choices)**

* **Single URL experience** ⇒ Next.js on Cloud Run with route handlers keeps UI+API together (no separate “api subdomain”).
* **Security best practice** ⇒ refresh tokens live **only on server**, encrypted with **KMS**, secrets in **Secret Manager**, sessions via **httpOnly cookies**.
* **GCP-native + low cost** ⇒ Cloud Run (scale to zero), Cloud SQL smallest tier, us-central1, autoscaling.
* **Future-proof** ⇒ you can add an optional /apps/backend FastAPI later without changing OAuth/DB/KMS patterns.
* **IAP later** ⇒ for internal apps, place Cloud Run behind HTTPS LB + IAP and remove app-level OAuth UI; for consumer apps, keep in-app OAuth and don’t use IAP.

**Critical Questions (minimal)**

1. Pick **ORM**: ok to use **Prisma with pg** (familiar DX) or prefer raw SQL?
2. **Cookie domain**: once you buy a domain, we’ll need it for secure cookies (for now use Cloud Run default domain).
3. **Scopes**: confirm https://www.googleapis.com/auth/calendar.readonly for v1?
4. **LLM stub**: ok to keep a placeholder server route for Gemini calls (disabled for now)?

**Suggested Micro-Add-Ons**

* **(Clarity)** “BLUF first” section in README: “Sign in → list calendars → view events.”
* **(Risk Control)** “Security pass” checklist: no tokens in client; KMS enforced; rotate secrets quarterly.
* **(Quality)** “Sanity check” route /api/self-test attempts DB + Secret Manager + KMS decrypt.
* **(Performance)** Enable gzip & cache control headers on static assets.
* **(Depth)** Add minimal audit table usage\_events (who/when hit which route).

**ultra-high-level steps (what you’ll do next)**

* create GCP project → enable APIs (Cloud Run, Cloud SQL, Secret Manager, KMS, OAuth, Calendar).
* set up Cloud SQL (Postgres), KMS key, Secret Manager entries.
* scaffold Next.js app with API routes for auth + calendar.
* wire Postgres (Prisma or SQL), sessions, encrypted token storage.
* dockerize → deploy to Cloud Run → test on default domain → (later) attach custom domain.

If this looks right, I’ll generate the **code-producing version** of this prompt (same structure, with concrete file trees, commands, and skeleton handlers).

**You said:**

are you sure the prompt you just gave me is a good prompt?

**Prompt Architect for GPT-5 Thinking said:**

Thought for 8s

short answer: it’s *good*, but we can make it **tighter and safer** with a few surgical tweaks (clearer success criteria, stricter security knobs, and fully unambiguous deploy/run steps). here’s the upgraded version, kept compact.

**Strengths & Weaknesses (super short)**

**Strengths**

* Single-URL Next.js on Cloud Run (simple UX).
* Google OAuth + Postgres + Secret Manager + KMS (sane, secure defaults).
* Minimal, testable v1 scope (signin → list calendars → view events).

**Weaknesses (fixed below)**

* A few ambiguities (ORM, cookie config, scopes, Node/Next version).
* Missing crisp runbook & env contract.
* Didn’t explicitly pin costs/tiers.

**Improved Prompt (XML Sandwich + Perfection Loop) — v2**

<context> Goal: Consumer-facing calendar web app, single URL, \*\*Next.js (App Router) on Cloud Run\*\*. Users must sign in with \*\*Google OAuth\*\* before any UI. App fetches user’s Google Calendars and events. Infra: \*\*GCP us-central1\*\*, \*\*Cloud SQL for Postgres (starter tier)\*\*, \*\*Secret Manager\*\*, \*\*Cloud KMS\*\* (envelope encryption for refresh tokens). No Firebase. Keep old Streamlit repo untouched (not used). Cost: minimize (Cloud Run scale-to-zero; smallest Postgres). Testing phase uses Google OAuth \*\*Internal\*\*; later, External with verification. Success (v1): sign in → list calendars → view events (read-only). </context> <task> 1) \*\*Repo & structure\*\* - Monorepo: ``` /apps/frontend (Next.js 15, Node 20, Tailwind/shadcn, API routes in /app/api) /packages/shared (zod schemas, DTOs, token crypto helpers) ``` - Dockerize Next.js for Cloud Run. Add `.env.example` and `Makefile`/npm scripts.

1. **Auth (Google OAuth)**
   * Routes: /api/auth/login, /api/auth/callback, /api/auth/logout.
   * Store session via **httpOnly, Secure, SameSite=Lax** cookies; server-side session table in Postgres.
   * OAuth state + PKCE; scopes: openid email profile + https://www.googleapis.com/auth/calendar.readonly.
   * On callback: exchange code → get refresh token → **encrypt with KMS** → store in Postgres.
2. **Calendar API**
   * Server-only fetchers with on-demand access-token refresh.
   * Endpoints:
     + GET /api/calendar/list
     + GET /api/calendar/events?calendarId=...&from=...&to=...
3. **Database (Postgres)**
   * Use **Prisma** (explicit schema + migrations). Tables: users, sessions, oauth\_tokens(user\_id, provider, enc\_blob, created\_at), usage\_events.
   * Local dev uses Docker Postgres; prod uses **Cloud SQL**. Provide DATABASE\_URL patterns for both.
4. **Secrets & keys**
   * Store GOOGLE\_CLIENT\_ID, GOOGLE\_CLIENT\_SECRET in **Secret Manager**.
   * Create **KMS key** projects/<proj>/locations/us-central1/keyRings/app/cryptoKeys/tokens.
   * App reads secrets at runtime via Google ADC; no secrets in .env in prod.
5. **Deploy**
   * Enable APIs: Cloud Run, Cloud SQL Admin, Secret Manager, KMS, Service Usage, IAM, Calendar API, Identity Toolkit (OAuth).
   * Build & deploy calendar-web to Cloud Run (min=0 or 1). Attach Cloud SQL via Connector.
   * Use default Cloud Run URL for testing; later add **custom domain** + HTTPS LB.
6. **UX**
   * / shows “Sign in with Google”. After auth: /manage-events page with calendars and events (SSR or RSC fetch).
   * Handle error states (no calendars, token refresh failure, 401/403 redirect to login).
7. **Testing & ops**
   * Unit tests for route handlers (mock Google APIs).
   * /api/healthz (liveness) + /api/self-test (DB + Secret Manager + KMS round-trip).
   * Basic audit logs into usage\_events (user, route, ts).

</task>

<output\_format>

* Provide: file tree, key code snippets, and exact shell commands for GCP setup, KMS, Secret Manager, Cloud SQL, and Cloud Run deploy.
* Keep explanations terse (bullets).  
  </output\_format>

<quality\_bar>

* **Security**: httpOnly cookies; no tokens client-side; OAuth state/PKCE; KMS encryption; least-privileged service accounts.
* **Reliability**: timeouts, retries on token refresh; graceful 401/403.
* **Cost**: Cloud Run scale-to-zero; smallest Cloud SQL tier; single region us-central1.
* **DX**: npm run dev, npm run migrate, npm run deploy.  
  </quality\_bar>

<verbosity>concise; bullets; no long paragraphs.</verbosity>

<process> 1) Scaffold repo + Next.js + Prisma schema + auth routes. 2) Implement KMS wrapper + Secret Manager loader + token vault. 3) Add Calendar endpoints + protected pages. 4) Local Docker Compose (web + postgres) → run migrations → smoke test. 5) Create GCP resources, set IAM, bind Cloud SQL, deploy → smoke test on Cloud Run URL. 6) (Later) map custom domain; plan OAuth External verification. </process> <finalize> - Print a 1-page \*\*Runbook\*\*: GCP project creation, enable APIs, create OAuth credentials (redirect: `/api/auth/callback`), create KMS key, create Cloud SQL, load secrets, deploy Cloud Run, verify healthz/self-test, add domain. - Output a \*\*go/no-go checklist\*\* for v1. </finalize> <!-- router nudge -->

If Python APIs are required later, add /apps/backend FastAPI on Cloud Run; reuse same DB/KMS/Secret Manager.  
</process>

**Rationale (why v2 is better — brief)**

* Pins **Next.js 15 / Node 20**, scopes, cookie flags, and Prisma—reduces guesswork.
* Explicit **PKCE + state** and **KMS envelope** pattern—industry-standard OAuth security.
* Clear **deploy runbook** and **health/self-test** endpoints—faster debugging.
* Keeps **single URL** while remaining extensible (can add IAP or FastAPI later).

**Critical Questions (only what’s still needed)**

1. OK with **Prisma** as ORM? (else: raw SQL)
2. Accept **calendar.readonly** for v1?
3. We’ll start with **Cloud Run default domain** (no custom domain) for the first deploy—good?

**Suggested Micro-Add-Ons**

* **(Clarity)** BLUF in README: “Sign in → list calendars → view events.”
* **(Risk Control)** “Security pass” checklist in repo root.
* **(Quality)** /api/self-test verifies DB + Secret Manager + KMS.
* **(Performance)** Cache static assets; short TTL for calendar responses.
* **(Depth)** Add usage\_events for basic analytics.

if you’re happy with this, say “ship v2” and i’ll produce the **code-scaffolding prompt** (same structure, but with concrete file trees, env vars, and gcloud/docker commands) in a short, bullet-only format.

**You said:**

now generate the code-producing version of this prompt with same structure with concrete file trees commands and skeleton handlers. Dont generate code yet just make sure the user agrees with your reasoning and method then generate code. Write a prompt again so that it considers everything we talked about right now.

**Prompt Architect for GPT-5 Thinking said:**

Thought for 12s

great — here’s the **code-producing prompt** (no code yet). It’s tight, short, and reflects everything we agreed on. If you’re good with this, say **“agree — generate code”** and I’ll use it to produce the scaffold.

**Strengths & Weaknesses Summary (super short)**

**Strengths:** single-URL UX; GCP-native (Cloud Run, Cloud SQL, Secret Manager, KMS); Google OAuth; minimal v1 scope.  
**Weaknesses to watch:** OAuth callback/config drift; Cloud SQL connectivity; token security; app verification later.

**Improved Prompt (XML Sandwich + Perfection Loop) — Code-Producing Version (No code yet)**

<context> You are generating a production-ready scaffold for a consumer web app that requires Google Sign-In before accessing a calendar UI. Single URL site using \*\*Next.js (App Router)\*\* deployed on \*\*Cloud Run\*\* in \*\*us-central1\*\*. Infra: \*\*Cloud SQL for Postgres (starter tier)\*\*, \*\*Secret Manager\*\* for client secrets, \*\*Cloud KMS\*\* envelope encryption for Google \*\*refresh tokens\*\*, OAuth \*\*Internal\*\* for testing now (later External). No Firebase. Old Streamlit/Python code remains in repo but is unused. Short-term success: user signs in → list calendars → view events (read-only). Incorporate background from the project’s \*\*project overview .md\*\* (if present) into README sections. </context> <task> Produce the following (concise, deterministic, copy-pastable):

1. **Monorepo file tree** (no code, just paths)

/apps/frontend (Next.js 15, Node 20, Tailwind/shadcn, API routes in /app/api)

/packages/shared (zod schemas, DTOs, token crypto helpers)

/infra (Dockerfile, docker-compose for local dev, GitHub Actions, IaC placeholders)

/docs (Runbook.md, Security.md, Architecture.md)

1. **Skeleton handlers & pages** (list files with brief comment of purpose; no code):

* /apps/frontend/app/page.tsx (signed-out landing with “Sign in with Google”).
* /apps/frontend/app/manage-events/page.tsx (protected).
* /apps/frontend/app/api/auth/login/route.ts
* /apps/frontend/app/api/auth/callback/route.ts
* /apps/frontend/app/api/auth/logout/route.ts
* /apps/frontend/app/api/calendar/list/route.ts
* /apps/frontend/app/api/calendar/events/route.ts
* /apps/frontend/app/api/healthz/route.ts
* /apps/frontend/app/api/self-test/route.ts (DB + Secret Manager + KMS round-trip).

1. **Config & env contracts**

* .env.example keys (local only): DATABASE\_URL, GOOGLE\_CLIENT\_ID, GOOGLE\_CLIENT\_SECRET, SESSION\_SECRET, GCP\_PROJECT\_ID, KMS\_KEY\_RESOURCE, CLOUD\_SQL\_INSTANCE.
* Production: secrets retrieved via **Secret Manager** and **ADC**; no plaintext secrets in env.

1. **Database schema (Prisma)** — provide the **Prisma schema text** and **migrations** in the final output (when generating code). Tables:

* users(id, email, name, picture, createdAt)
* sessions(id, userId, sessionToken, expires)
* oauth\_tokens(id, userId, provider, enc\_blob, createdAt)
* usage\_events(id, userId, route, ts)

1. **Security**

* OAuth **state + PKCE**, scopes: openid email profile + https://www.googleapis.com/auth/calendar.readonly.
* **httpOnly, Secure, SameSite=Lax** session cookies; server-side session store in Postgres.
* Token vault: **KMS envelope encryption** (generate DEK per token; store ciphertext + metadata in enc\_blob).
* Never expose tokens to client JS.

1. **Google Calendar client (server-only)**

* Helper to refresh access tokens using stored encrypted refresh token; timeouts, retries, 401→relogin.

1. **Local dev**

* docker-compose.yml running Postgres + web.
* NPM scripts: dev, build, start, migrate, lint, test.
* Seed script creates a test user row (optional).

1. **GCP setup (exact commands)** — output copy-pastable **gcloud** commands in the final code response:

* Enable services: run, sqladmin, secretmanager, cloudkms, iam, servicemanagement, serviceusage, calendar-json.googleapis.com.
* Create **Cloud SQL Postgres** (us-central1, smallest tier) + user + DB; print DATABASE\_URL template.
* Create **KMS** keyring app + key tokens in us-central1; assign **Cloud Run SA** cloudkms.cryptoKeyEncrypterDecrypter.
* Create **Secret Manager** secrets for GOOGLE\_CLIENT\_ID, GOOGLE\_CLIENT\_SECRET, SESSION\_SECRET.
* Create **OAuth** web client (console steps + redirect URI /api/auth/callback).
* Grant **Secret Manager Accessor** to Cloud Run SA.
* Deploy **Cloud Run** service (calendar-web) with SQL connector env + min instances 0/1.

1. **CI/CD (GitHub Actions)**

* Lint, typecheck, test, build; on main → deploy to Cloud Run using Workload Identity Federation or a JSON key secret (WIF preferred).

1. **Docs**

* docs/Runbook.md (setup & deploy checklist).
* docs/Security.md (token handling, KMS/Secrets, cookie flags).
* docs/Architecture.md (request flows, components).

Output format: **file tree + code blocks for config and handlers + shell commands**, all in concise bullets.  
</task>

<output\_format>

* Provide:
  1. File trees
  2. Minimal code snippets for handlers/pages/helpers (TypeScript/Next.js)
  3. Prisma schema + first migration SQL
  4. docker-compose.yml and Dockerfile
  5. gcloud commands and IAM bindings
  6. Short README with BLUF and run steps
* Keep explanations terse (bullets). No walls of text.  
  </output\_format>

<quality\_bar>

* **Security:** httpOnly cookies, CSRF via OAuth state, PKCE, KMS envelope, least-privileged roles, no tokens in client.
* **Reliability:** retries on token refresh, 401/403 redirects, healthz & self-test endpoints.
* **Cost:** Cloud Run scale-to-zero; smallest Cloud SQL tier; single region **us-central1**.
* **DX:** fast npm run dev, prisma migrations, clear README, commented configs.  
  </quality\_bar>

<verbosity>concise; bullet lists; compact code blocks only where necessary.</verbosity>

<process> 1) Scaffold monorepo & Next.js app; add Tailwind/shadcn, route handlers, protected page. 2) Add Prisma schema, migrations; session adapter; token vault (KMS). 3) Implement Calendar endpoints using refreshed access tokens. 4) Prepare local Docker Compose; smoke test. 5) Generate GCP commands for services, SQL, KMS, secrets, deploy. 6) Emit CI workflow yaml. 7) Print Runbook checklist + go/no-go. </process> <finalize> Return a \*\*single response\*\* containing: - Tree, configs, minimal handler code, Prisma schema, Docker & Compose, gcloud commands, CI yaml, and README stubs. All content must be directly runnable with minimal edits (project ID/SQL instance names), with TODOs clearly marked. </finalize> <!-- router nudge -->

If the user later requests a Python backend, also scaffold /apps/backend (FastAPI on Cloud Run) sharing the same DB/KMS/Secrets pattern and leaving the Next.js API routes intact for UI auth proxy.  
</process>

**Rationale (why this prompt will yield what you want)**

* Forces a **single, self-contained deliverable** (code + infra commands + docs) with security and cost controls.
* Keeps **single URL** UX while enabling scale and future IAP/FastAPI add-ons.
* Uses **industry-standard** token storage (KMS + Secret Manager), reducing risk when you later go External.

**Critical Questions (last tiny confirmations)**

1. **Prisma** as ORM — OK?
2. Start with **Cloud Run default domain** (custom domain later) — OK?
3. Use scopes openid email profile + calendar.readonly — OK?

**Suggested Micro-Add-Ons**

* **(Clarity)** BLUF in README (“Sign in → list calendars → view events”).
* **(Risk Control)** /api/self-test performs DB + KMS encrypt/decrypt + Secret Manager fetch.
* **(Quality)** simple rate limiter on /api/calendar/\*.
* **(Performance)** cache static assets; short TTL for calendar list.

**If you agree**, reply: **“agree — generate code”** and I’ll emit the scaffold per this prompt.

**You said:**

agree

**Prompt Architect for GPT-5 Thinking said:**

Thought for 10s

awesome — shipping the scaffold now. it’s compact but complete: file tree, minimal handlers, prisma schema, docker, gcloud cmds, CI, and a tiny runbook. swap placeholders like YOUR\_PROJECT\_ID.

**Monorepo layout**

.

├─ apps/

│ └─ frontend/

│ ├─ app/

│ │ ├─ api/

│ │ │ ├─ auth/

│ │ │ │ ├─ login/route.ts

│ │ │ │ ├─ callback/route.ts

│ │ │ │ └─ logout/route.ts

│ │ │ ├─ calendar/

│ │ │ │ ├─ list/route.ts

│ │ │ │ └─ events/route.ts

│ │ │ ├─ healthz/route.ts

│ │ │ └─ self-test/route.ts

│ │ ├─ manage-events/page.tsx

│ │ ├─ layout.tsx

│ │ └─ page.tsx

│ ├─ lib/

│ │ ├─ auth.ts

│ │ ├─ google.ts

│ │ ├─ kms.ts

│ │ ├─ secrets.ts

│ │ └─ db.ts

│ ├─ prisma/

│ │ └─ schema.prisma

│ ├─ package.json

│ ├─ next.config.mjs

│ ├─ tsconfig.json

│ ├─ tailwind.config.ts

│ ├─ postcss.config.mjs

│ ├─ Dockerfile

│ └─ .env.example

├─ packages/

│ └─ shared/

│ ├─ package.json

│ ├─ src/

│ │ ├─ schemas.ts

│ │ └─ types.ts

│ └─ tsconfig.json

├─ infra/

│ ├─ docker-compose.yml

│ ├─ github/

│ │ └─ workflows/deploy.yml

│ └─ README.md

└─ docs/

├─ Runbook.md

├─ Security.md

└─ Architecture.md

**Minimal code & configs**

**apps/frontend/package.json**

{

"name": "calendar-web",

"private": true,

"version": "0.1.0",

"scripts": {

"dev": "next dev",

"build": "prisma generate && next build",

"start": "next start -p 8080",

"migrate": "prisma migrate deploy",

"lint": "eslint .",

"test": "node -e \"console.log('tests TBD')\""

},

"dependencies": {

"@google-cloud/kms": "^5.7.0",

"@google-cloud/secret-manager": "^5.6.0",

"@prisma/client": "^5.20.0",

"cookie": "^0.6.0",

"next": "15.0.3",

"zod": "^3.23.8"

},

"devDependencies": {

"eslint": "^9.13.0",

"prisma": "^5.20.0",

"typescript": "^5.6.3"

}

}

**apps/frontend/next.config.mjs**

/\*\* @type {import('next').NextConfig} \*/

const nextConfig = {

experimental: { serverActions: { allowedOrigins: ['\*'] } },

output: 'standalone'

};

export default nextConfig;

**apps/frontend/tsconfig.json**

{

"compilerOptions": {

"target": "ES2022",

"lib": ["ES2022", "DOM"],

"module": "ESNext",

"moduleResolution": "Bundler",

"strict": true,

"baseUrl": ".",

"paths": { "@shared/\*": ["../../packages/shared/src/\*"] }

}

}

**apps/frontend/.env.example (local only)**

DATABASE\_URL=postgresql://postgres:postgres@localhost:5432/appdb?schema=public

GOOGLE\_CLIENT\_ID=your\_local\_client\_id.apps.googleusercontent.com

GOOGLE\_CLIENT\_SECRET=your\_local\_secret

SESSION\_SECRET=dev-session-secret

GCP\_PROJECT\_ID=YOUR\_PROJECT\_ID

KMS\_KEY\_RESOURCE=projects/YOUR\_PROJECT\_ID/locations/us-central1/keyRings/app/cryptoKeys/tokens

CLOUD\_SQL\_INSTANCE=YOUR\_PROJECT\_ID:us-central1:calendar-postgres

OAUTH\_REDIRECT\_URL=http://localhost:3000/api/auth/callback

**apps/frontend/prisma/schema.prisma**

generator client {

provider = "prisma-client-js"

}

datasource db {

provider = "postgresql"

url = env("DATABASE\_URL")

}

model users {

id String @id @default(cuid())

email String @unique

name String?

picture String?

createdAt DateTime @default(now())

sessions sessions[]

tokens oauth\_tokens[]

}

model sessions {

id String @id @default(cuid())

userId String

sessionToken String @unique

expires DateTime

user users @relation(fields: [userId], references: [id], onDelete: Cascade)

@@index([userId])

}

model oauth\_tokens {

id String @id @default(cuid())

userId String

provider String

enc\_blob Bytes

createdAt DateTime @default(now())

user users @relation(fields: [userId], references: [id], onDelete: Cascade)

@@unique([userId, provider])

}

model usage\_events {

id String @id @default(cuid())

userId String?

route String

ts DateTime @default(now())

user users? @relation(fields: [userId], references: [id])

}

**apps/frontend/lib/db.ts**

import { PrismaClient } from '@prisma/client';

const globalForPrisma = globalThis as unknown as { prisma?: PrismaClient };

export const prisma = globalForPrisma.prisma ?? new PrismaClient();

if (process.env.NODE\_ENV !== 'production') globalForPrisma.prisma = prisma;

**apps/frontend/lib/secrets.ts**

import { SecretManagerServiceClient } from '@google-cloud/secret-manager';

const isGcp = !!process.env.K\_SERVICE; // Cloud Run

const client = isGcp ? new SecretManagerServiceClient() : null;

export async function getSecret(name: string): Promise<string> {

if (!isGcp) return process.env[name] ?? '';

const [v] = await client!.accessSecretVersion({

name: `projects/${process.env.GCP\_PROJECT\_ID}/secrets/${name}/versions/latest`

});

return v.payload?.data?.toString() ?? '';

}

**apps/frontend/lib/kms.ts (envelope encryption helper)**

import { KeyManagementServiceClient } from '@google-cloud/kms';

const isGcp = !!process.env.K\_SERVICE;

const kms = isGcp ? new KeyManagementServiceClient() : null;

// Encodes: { iv, dekCiphertext, dataCiphertext, tag }

export async function encryptWithKms(plaintext: Buffer) {

if (!isGcp) return { enc\_blob: plaintext }; // dev mode stores raw (dev only)

const name = process.env.KMS\_KEY\_RESOURCE!;

const [resp] = await kms!.generateRandomBytes({ location: 'us-central1', lengthBytes: 32 });

const dek = resp.data!; // 32B DEK

const { createCipheriv, randomBytes } = await import('crypto');

const iv = randomBytes(12);

const cipher = createCipheriv('aes-256-gcm', dek, iv);

const dataCiphertext = Buffer.concat([cipher.update(plaintext), cipher.final()]);

const tag = cipher.getAuthTag();

const [wrap] = await kms!.encrypt({ name, plaintext: dek });

return { enc\_blob: Buffer.concat([iv, tag, Buffer.from(wrap.ciphertext!), dataCiphertext]) };

}

export async function decryptWithKms(enc\_blob: Buffer) {

if (!isGcp) return enc\_blob; // dev mode passthrough

const name = process.env.KMS\_KEY\_RESOURCE!;

const iv = enc\_blob.subarray(0, 12);

const tag = enc\_blob.subarray(12, 28);

// next is KMS-wrapped DEK; assume 256+ bytes; take len from KMS RSA length via config? keep simple:

const wrappedLenGuess = 256; // works for RSA-2048; adjust if using AES-KWP

const dekWrapped = enc\_blob.subarray(28, 28 + wrappedLenGuess);

const dataCiphertext = enc\_blob.subarray(28 + wrappedLenGuess);

const [unwrapped] = await kms!.decrypt({ name, ciphertext: dekWrapped });

const { createDecipheriv } = await import('crypto');

const decipher = createDecipheriv('aes-256-gcm', unwrapped.plaintext!, iv);

decipher.setAuthTag(tag);

const plaintext = Buffer.concat([decipher.update(dataCiphertext), decipher.final()]);

return plaintext;

}

Note: in real use, store lengths/metadata; here we keep a compact blob for brevity. You can refine later.

**apps/frontend/lib/auth.ts (session helpers)**

import { cookies } from 'next/headers';

import { randomBytes } from 'crypto';

import { prisma } from './db';

const COOKIE = 'sid';

const MAX\_AGE = 60 \* 60 \* 24 \* 7; // 7d

export async function createSession(userId: string) {

const sessionToken = randomBytes(32).toString('base64url');

const expires = new Date(Date.now() + MAX\_AGE \* 1000);

await prisma.sessions.create({ data: { userId, sessionToken, expires } });

cookies().set(COOKIE, sessionToken, { httpOnly: true, secure: true, sameSite: 'lax', path: '/', maxAge: MAX\_AGE });

}

export async function getSessionUser() {

const sid = cookies().get(COOKIE)?.value;

if (!sid) return null;

const sess = await prisma.sessions.findUnique({ where: { sessionToken: sid }, include: { user: true } });

if (!sess || sess.expires < new Date()) return null;

return sess.user;

}

export async function clearSession() {

const sid = cookies().get(COOKIE)?.value;

if (sid) await prisma.sessions.delete({ where: { sessionToken: sid } }).catch(() => {});

cookies().delete(COOKIE);

}

**apps/frontend/lib/google.ts (OAuth + token refresh)**

import { getSecret } from './secrets';

import { prisma } from './db';

import { encryptWithKms, decryptWithKms } from './kms';

const OAUTH\_AUTH = 'https://accounts.google.com/o/oauth2/v2/auth';

const OAUTH\_TOKEN = 'https://oauth2.googleapis.com/token';

const CAL\_LIST = 'https://www.googleapis.com/calendar/v3/users/me/calendarList';

const EVENTS = (id: string, timeMin?: string, timeMax?: string) =>

`https://www.googleapis.com/calendar/v3/calendars/${encodeURIComponent(id)}/events?singleEvents=true${timeMin ? `&timeMin=${encodeURIComponent(timeMin)}`: ''}${timeMax ? `&timeMax=${encodeURIComponent(timeMax)}`: ''}`;

export function oauthUrl(state: string, codeChallenge: string, redirectUri: string) {

const params = new URLSearchParams({

client\_id: process.env.GOOGLE\_CLIENT\_ID ?? '',

redirect\_uri: redirectUri,

response\_type: 'code',

scope: 'openid email profile https://www.googleapis.com/auth/calendar.readonly',

access\_type: 'offline',

prompt: 'consent',

state,

code\_challenge: codeChallenge,

code\_challenge\_method: 'S256'

});

return `${OAUTH\_AUTH}?${params.toString()}`;

}

export async function exchangeCode(code: string, redirectUri: string, codeVerifier: string) {

const client\_id = process.env.GOOGLE\_CLIENT\_ID ?? await getSecret('GOOGLE\_CLIENT\_ID');

const client\_secret = process.env.GOOGLE\_CLIENT\_SECRET ?? await getSecret('GOOGLE\_CLIENT\_SECRET');

const resp = await fetch(OAUTH\_TOKEN, {

method: 'POST',

headers: { 'content-type': 'application/x-www-form-urlencoded' },

body: new URLSearchParams({

code, client\_id, client\_secret, redirect\_uri: redirectUri, grant\_type: 'authorization\_code', code\_verifier: codeVerifier

})

});

if (!resp.ok) throw new Error(`token exchange failed ${resp.status}`);

return resp.json() as Promise<{ access\_token: string; refresh\_token?: string; id\_token?: string; expires\_in: number }>;

}

export async function refreshAccessToken(userId: string) {

const tok = await prisma.oauth\_tokens.findFirst({ where: { userId, provider: 'google' } });

if (!tok) throw new Error('no token');

const refresh = await decryptWithKms(Buffer.from(tok.enc\_blob));

const client\_id = process.env.GOOGLE\_CLIENT\_ID ?? await getSecret('GOOGLE\_CLIENT\_ID');

const client\_secret = process.env.GOOGLE\_CLIENT\_SECRET ?? await getSecret('GOOGLE\_CLIENT\_SECRET');

const resp = await fetch(OAUTH\_TOKEN, {

method: 'POST',

headers: { 'content-type': 'application/x-www-form-urlencoded' },

body: new URLSearchParams({

client\_id, client\_secret, refresh\_token: refresh.toString(), grant\_type: 'refresh\_token'

})

});

if (!resp.ok) throw new Error('refresh failed');

return resp.json() as Promise<{ access\_token: string; expires\_in: number }>;

}

export async function listCalendars(access\_token: string) {

const r = await fetch(CAL\_LIST, { headers: { Authorization: `Bearer ${access\_token}` } });

if (!r.ok) throw new Error('calendar list failed');

return r.json();

}

export async function listEvents(access\_token: string, calendarId: string, timeMin?: string, timeMax?: string) {

const r = await fetch(EVENTS(calendarId, timeMin, timeMax), { headers: { Authorization: `Bearer ${access\_token}` } });

if (!r.ok) throw new Error('events failed');

return r.json();

}

export async function storeRefreshToken(userId: string, refresh\_token: string) {

const { enc\_blob } = await encryptWithKms(Buffer.from(refresh\_token));

await prisma.oauth\_tokens.upsert({

where: { userId\_provider: { userId, provider: 'google' } },

update: { enc\_blob },

create: { userId, provider: 'google', enc\_blob }

});

}

**Route handlers (skeletons)**

**apps/frontend/app/api/auth/login/route.ts**

import { NextResponse } from 'next/server';

import { oauthUrl } from '@/app/lib/google';

import crypto from 'crypto';

export async function GET() {

const state = crypto.randomBytes(16).toString('base64url');

const verifier = crypto.randomBytes(32).toString('base64url');

const challenge = crypto.createHash('sha256').update(verifier).digest().toString('base64url').replace(/\+/g,'-').replace(/\//g,'\_').replace(/=+$/,'');

const redirectUri = process.env.OAUTH\_REDIRECT\_URL!;

const res = NextResponse.redirect(oauthUrl(state, challenge, redirectUri));

res.cookies.set('oauth\_state', state, { httpOnly: true, secure: true, sameSite: 'lax', path: '/', maxAge: 600 });

res.cookies.set('code\_verifier', verifier, { httpOnly: true, secure: true, sameSite: 'lax', path: '/', maxAge: 600 });

return res;

}

**apps/frontend/app/api/auth/callback/route.ts**

import { NextResponse } from 'next/server';

import { exchangeCode, storeRefreshToken } from '@/app/lib/google';

import { prisma } from '@/app/lib/db';

import { createSession } from '@/app/lib/auth';

export async function GET(req: Request) {

const url = new URL(req.url);

const code = url.searchParams.get('code');

const state = url.searchParams.get('state');

const cookies = (await import('next/headers')).cookies();

const savedState = cookies.get('oauth\_state')?.value;

const verifier = cookies.get('code\_verifier')?.value;

if (!code || !state || state !== savedState || !verifier) return NextResponse.redirect('/', { status: 303 });

const redirectUri = process.env.OAUTH\_REDIRECT\_URL!;

const token = await exchangeCode(code, redirectUri, verifier);

const idToken = token.id\_token ?? '';

const [, payloadB64] = idToken.split('.');

const payload = JSON.parse(Buffer.from(payloadB64, 'base64').toString());

const email = payload.email as string;

const name = payload.name as string | undefined;

const picture = payload.picture as string | undefined;

const user = await prisma.users.upsert({

where: { email },

update: { name, picture },

create: { email, name, picture }

});

if (token.refresh\_token) await storeRefreshToken(user.id, token.refresh\_token);

await createSession(user.id);

const res = NextResponse.redirect(new URL('/manage-events', req.url));

res.cookies.delete('oauth\_state'); res.cookies.delete('code\_verifier');

return res;

}

**apps/frontend/app/api/auth/logout/route.ts**

import { NextResponse } from 'next/server';

import { clearSession } from '@/app/lib/auth';

export async function POST() { await clearSession(); return NextResponse.json({ ok: true }); }

**apps/frontend/app/api/calendar/list/route.ts**

import { NextResponse } from 'next/server';

import { getSessionUser } from '@/app/lib/auth';

import { refreshAccessToken, listCalendars } from '@/app/lib/google';

export async function GET() {

const user = await getSessionUser();

if (!user) return NextResponse.json({ error: 'unauthorized' }, { status: 401 });

const { access\_token } = await refreshAccessToken(user.id);

const data = await listCalendars(access\_token);

return NextResponse.json(data);

}

**apps/frontend/app/api/calendar/events/route.ts**

import { NextResponse } from 'next/server';

import { getSessionUser } from '@/app/lib/auth';

import { refreshAccessToken, listEvents } from '@/app/lib/google';

export async function GET(req: Request) {

const user = await getSessionUser();

if (!user) return NextResponse.json({ error: 'unauthorized' }, { status: 401 });

const { searchParams } = new URL(req.url);

const calendarId = searchParams.get('calendarId') ?? 'primary';

const from = searchParams.get('from') ?? undefined;

const to = searchParams.get('to') ?? undefined;

const { access\_token } = await refreshAccessToken(user.id);

const data = await listEvents(access\_token, calendarId, from, to);

return NextResponse.json(data);

}

**apps/frontend/app/api/healthz/route.ts**

export async function GET() { return new Response('ok'); }

**apps/frontend/app/api/self-test/route.ts**

import { prisma } from '@/app/lib/db';

import { getSecret } from '@/app/lib/secrets';

import { encryptWithKms, decryptWithKms } from '@/app/lib/kms';

export async function GET() {

const dbOk = await prisma.$queryRaw`SELECT 1 as ok`;

const secret = await getSecret('GOOGLE\_CLIENT\_ID');

const roundtrip = await decryptWithKms((await encryptWithKms(Buffer.from('hello'))).enc\_blob);

return new Response(JSON.stringify({ dbOk, secretPresent: !!secret, roundtrip: roundtrip.toString() }), { headers: { 'content-type': 'application/json' }});

}

**Pages**

**apps/frontend/app/page.tsx**

export default function Page() {

return (

<main className="p-8">

<h1 className="text-2xl font-semibold mb-4">Calendar App</h1>

<a className="px-4 py-2 rounded bg-black text-white" href="/api/auth/login">Sign in with Google</a>

</main>

);

}

**apps/frontend/app/manage-events/page.tsx**

'use client';

import { useEffect, useState } from 'react';

export default function ManageEvents() {

const [cals, setCals] = useState<any[]>([]);

const [events, setEvents] = useState<any[]>([]);

const [selected, setSelected] = useState<string>('primary');

useEffect(() => { fetch('/api/calendar/list').then(r=>r.json()).then(d=>setCals(d.items ?? [])); }, []);

useEffect(() => { fetch(`/api/calendar/events?calendarId=${encodeURIComponent(selected)}`).then(r=>r.json()).then(d=>setEvents(d.items ?? [])); }, [selected]);

return (

<main className="p-6">

<h2 className="text-xl mb-4">Your Calendars</h2>

<select className="border p-2 mb-4" value={selected} onChange={e=>setSelected(e.target.value)}>

{cals.map((c:any)=><option key={c.id} value={c.id}>{c.summary}</option>)}

</select>

<ul className="space-y-2">{events.map((e:any)=><li key={e.id} className="border p-2 rounded">{e.summary}</li>)}</ul>

</main>

);

}

**Docker & Compose**

**apps/frontend/Dockerfile**

# --- build ---

FROM node:20-alpine AS builder

WORKDIR /app

COPY package.json package-lock.json\* ./

RUN npm ci

COPY . .

RUN npx prisma generate && npm run build

# --- run ---

FROM node:20-alpine

WORKDIR /app

ENV PORT=8080 NODE\_ENV=production

COPY --from=builder /app ./

EXPOSE 8080

CMD ["npm", "start"]

**infra/docker-compose.yml (local dev)**

version: "3.9"

services:

db:

image: postgres:16

environment:

POSTGRES\_PASSWORD: postgres

POSTGRES\_USER: postgres

POSTGRES\_DB: appdb

ports: ["5432:5432"]

healthcheck:

test: ["CMD-SHELL", "pg\_isready -U postgres"]

interval: 5s

timeout: 3s

retries: 10

web:

build:

context: ../apps/frontend

environment:

DATABASE\_URL: postgres://postgres:postgres@db:5432/appdb

GOOGLE\_CLIENT\_ID: ${GOOGLE\_CLIENT\_ID}

GOOGLE\_CLIENT\_SECRET: ${GOOGLE\_CLIENT\_SECRET}

SESSION\_SECRET: dev-session-secret

OAUTH\_REDIRECT\_URL: http://localhost:3000/api/auth/callback

ports: ["3000:8080"]

depends\_on: [db]

**GCP setup (copy/paste)**

Set shell vars:

PROJECT\_ID=YOUR\_PROJECT\_ID

REGION=us-central1

SQL\_INSTANCE=calendar-postgres

DB\_NAME=appdb

DB\_USER=appuser

gcloud config set project $PROJECT\_ID

gcloud services enable run.googleapis.com sqladmin.googleapis.com secretmanager.googleapis.com cloudkms.googleapis.com iam.googleapis.com serviceusage.googleapis.com servicemanagement.googleapis.com calendar-json.googleapis.com

Create Cloud SQL (Postgres):

gcloud sql instances create $SQL\_INSTANCE --database-version=POSTGRES\_16 --cpu=1 --memory=3840MB --region=$REGION --storage-size=10 --tier=db-f1-micro --availability-type=zonal

gcloud sql databases create $DB\_NAME --instance=$SQL\_INSTANCE

gcloud sql users create $DB\_USER --instance=$SQL\_INSTANCE --password="$(openssl rand -base64 16)"

# Compose DATABASE\_URL (note: use Cloud SQL Connector in Cloud Run, not public IP)

echo "postgresql://$DB\_USER:REDACTED@localhost:5432/$DB\_NAME?sslmode=disable"

Create KMS key:

gcloud kms keyrings create app --location=$REGION

gcloud kms keys create tokens --keyring=app --location=$REGION --purpose=encryption

KMS\_KEY="projects/$PROJECT\_ID/locations/$REGION/keyRings/app/cryptoKeys/tokens"

echo $KMS\_KEY

Secrets:

gcloud secrets create GOOGLE\_CLIENT\_ID --replication-policy=automatic

gcloud secrets create GOOGLE\_CLIENT\_SECRET --replication-policy=automatic

gcloud secrets create SESSION\_SECRET --replication-policy=automatic

# Add values:

echo -n "YOUR\_OAUTH\_CLIENT\_ID" | gcloud secrets versions add GOOGLE\_CLIENT\_ID --data-file=-

echo -n "YOUR\_OAUTH\_CLIENT\_SECRET" | gcloud secrets versions add GOOGLE\_CLIENT\_SECRET --data-file=-

echo -n "$(openssl rand -base64 32)" | gcloud secrets versions add SESSION\_SECRET --data-file=-

**OAuth client (Console UI):**

* Type: **Web application**
* Authorized redirect URI: https://<cloud-run-url>/api/auth/callback (later) and http://localhost:3000/api/auth/callback for local.
* Copy client ID & secret into Secret Manager above.

Build & deploy Cloud Run:

# Build container from apps/frontend

pushd apps/frontend

gcloud builds submit --tag gcr.io/$PROJECT\_ID/calendar-web

popd

# Create service account

gcloud iam service-accounts create calendar-web-sa --display-name="Calendar Web SA"

# Grant needed roles

gcloud projects add-iam-policy-binding $PROJECT\_ID --member="serviceAccount:calendar-web-sa@$PROJECT\_ID.iam.gserviceaccount.com" --role="roles/secretmanager.secretAccessor"

gcloud projects add-iam-policy-binding $PROJECT\_ID --member="serviceAccount:calendar-web-sa@$PROJECT\_ID.iam.gserviceaccount.com" --role="roles/cloudkms.cryptoKeyDecrypter"

gcloud projects add-iam-policy-binding $PROJECT\_ID --member="serviceAccount:calendar-web-sa@$PROJECT\_ID.iam.gserviceaccount.com" --role="roles/cloudkms.cryptoKeyEncrypter"

gcloud projects add-iam-policy-binding $PROJECT\_ID --member="serviceAccount:calendar-web-sa@$PROJECT\_ID.iam.gserviceaccount.com" --role="roles/cloudsql.client"

# Deploy to Cloud Run and attach Cloud SQL

gcloud run deploy calendar-web \

--image gcr.io/$PROJECT\_ID/calendar-web \

--region $REGION \

--allow-unauthenticated \

--service-account calendar-web-sa@$PROJECT\_ID.iam.gserviceaccount.com \

--set-env-vars "GCP\_PROJECT\_ID=$PROJECT\_ID,KMS\_KEY\_RESOURCE=$KMS\_KEY,CLOUD\_SQL\_INSTANCE=$PROJECT\_ID:$REGION:$SQL\_INSTANCE" \

--add-cloudsql-instances "$PROJECT\_ID:$REGION:$SQL\_INSTANCE" \

--port 8080 \

--min-instances=0

# After deploy, set OAUTH\_REDIRECT\_URL to the Cloud Run URL

CLOUD\_RUN\_URL=$(gcloud run services describe calendar-web --region $REGION --format='value(status.url)')

gcloud run services update calendar-web --region $REGION --set-env-vars "OAUTH\_REDIRECT\_URL=$CLOUD\_RUN\_URL/api/auth/callback"

echo "Cloud Run URL: $CLOUD\_RUN\_URL"

For Prisma to connect to Cloud SQL in Cloud Run, set DATABASE\_URL to use the **connector socket**. Easiest path: use host=/cloudsql/PROJECT:REGION:INSTANCE via Unix socket; or run the JavaScript connector. To keep this short, you can initially expose a **public IP** with authorized networks for testing, then lock down and use connector.

**CI/CD (GitHub Actions) — infra/github/workflows/deploy.yml**

name: deploy

on:

push:

branches: [ main ]

jobs:

build-deploy:

permissions:

contents: read

id-token: write

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v4

- uses: actions/setup-node@v4

with: { node-version: '20' }

- name: Install deps

working-directory: apps/frontend

run: npm ci

- name: Build

working-directory: apps/frontend

run: npm run build

- uses: google-github-actions/auth@v2

with:

workload\_identity\_provider: ${{ secrets.WIF\_PROVIDER }}

service\_account: calendar-web-sa@${{ secrets.GCP\_PROJECT\_ID }}.iam.gserviceaccount.com

- uses: google-github-actions/setup-gcloud@v2

- name: Build & Push

working-directory: apps/frontend

run: |

gcloud builds submit --tag gcr.io/${{ secrets.GCP\_PROJECT\_ID }}/calendar-web

- name: Deploy

run: |

gcloud run deploy calendar-web \

--image gcr.io/${{ secrets.GCP\_PROJECT\_ID }}/calendar-web \

--region us-central1 \

--allow-unauthenticated \

--service-account calendar-web-sa@${{ secrets.GCP\_PROJECT\_ID }}.iam.gserviceaccount.com \

--set-env-vars "GCP\_PROJECT\_ID=${{ secrets.GCP\_PROJECT\_ID }},KMS\_KEY\_RESOURCE=projects/${{ secrets.GCP\_PROJECT\_ID }}/locations/us-central1/keyRings/app/cryptoKeys/tokens" \

--add-cloudsql-instances "${{ secrets.GCP\_PROJECT\_ID }}:us-central1:calendar-postgres" \

--port 8080 \

--min-instances=0

**Docs stubs**

**docs/Runbook.md**

# Runbook (v1)

- Create GCP project; enable services (Run, SQL, Secret Manager, KMS, Calendar).

- Create Cloud SQL Postgres (us-central1); create DB/user.

- Create KMS keyring `app` + key `tokens`.

- Create Secrets: GOOGLE\_CLIENT\_ID/SECRET, SESSION\_SECRET.

- OAuth client (Web): add redirect `${CLOUD\_RUN\_URL}/api/auth/callback` and `http://localhost:3000/api/auth/callback`.

- Build & deploy Cloud Run; set env vars; run `/api/healthz` and `/api/self-test`.

- Sign in, list calendars, view events.

**docs/Security.md**

- Sessions: httpOnly+Secure cookies; server-side store with expiry.

- OAuth state+PKCE; scope limited to `calendar.readonly`.

- Refresh tokens: encrypted via KMS envelope; never sent to client.

- Secrets fetched from Secret Manager via SA; least-privileged roles.

- Rotate SESSION\_SECRET quarterly; enforce TLS (Cloud Run).

**docs/Architecture.md**

- Next.js App Router on Cloud Run (UI + API routes).

- Storage: Cloud SQL Postgres via Prisma.

- Token vault: KMS envelope encryption; DEK per token.

- Google APIs: server-only fetch with refreshed access tokens.

- Health: /api/healthz; /api/self-test (DB+Secrets+KMS).

**Tiny README (root excerpt)**

# Calendar App (GCP, Next.js, Cloud Run)

\*\*BLUF:\*\* Sign in with Google → list calendars → view events (read-only).

## Dev

- `cp apps/frontend/.env.example apps/frontend/.env`

- `docker compose -f infra/docker-compose.yml up --build`

- visit http://localhost:3000

## Deploy

- follow `docs/Runbook.md` + gcloud commands in repo.

**Go/No-Go checklist (v1)**

* /api/healthz = ok
* /api/self-test returns dbOk + secretPresent + roundtrip
* Google OAuth internal app configured; local + Cloud Run redirect URIs set
* Sign-in works; calendars and events render
* No tokens in browser storage or network payloads

Want me to generate a repo-ready ZIP (all files with code as above) next? If yes, say: **“generate full repo files”** and I’ll output the complete file contents in-place.