Project Context: Github Issue Automation

Objective

Build a simple app that integrates **Devin** with **GitHub Issues** so a developer can:

1) view a repo's open issues,
2) trigger a **Devin scoping** session that produces a concrete plan + confidence,
3) iteratively refine the plan with follow-ups, and
4) **execute the plan** to implement changes and open a PR.

Scope (What to build)

- **Frontend:** Vite + TypeScript + Tailwind + Shadcn UI.
- **Backend:** FastAPI (Python).
- **Secrets:** Use an in-memory approach for this POC; accept a **GitHub PAT** from the UI only when needed (never store or log it).
- **Devin:** Use Devin Sessions for scoping → follow-ups → execution. Poll status **every 3 seconds** to update UI progress and confidence.
- **No persistence / CDC / discovery:** Limit to **latest 100 issues** per repo. No DB. No webhooks or event-driven updates. No repo discovery—connect by **URL**.

Constraints / Non-Goals

- No database; **in-memory** cache only (repos + downloaded issues).
- No background jobs or queues; rely on **manual re-sync** and polling.
- No deployment required (local run is sufficient).
- PAT is per-action, **never persisted**; redact in errors/logs.

User Experience & Flows

A) Repo Navigator

- Shows a grid of previously **connected repositories** (name, URL, open-issues count), with a row action to **Delete**.
- **Empty state** encourages connecting the first repo.
- **Connect repo** opens a modal to input:
    - **Repo URL** (e.g., `https://github.com/owner/repo`)
    - **GitHub Personal Access Token** (masked with *reveal* eye icon)

- On **Connect**:
    - Validate the repo exists and the PAT has permission to **open PRs**.
    - If valid, fetch up to **100 open issues**, cache them, and navigate to **Issue Dashboard** for that repo.

B) Issue Dashboard (per repo)

- Table columns: **Status**, **Summary** (clickable), **Labels** (show up to 3; tooltip for overflow), **Issue ID**, **Author**, **Age**.
- Defaults: **Open issues**, **sorted by most recent**.
- Controls: **Filter by label**, **keyword search** (with Reset), **Re-sync** button (re-pulls and **overwrites** any cached analysis).
- Clicking **Summary** opens **Issue Detail** modal.

C) Issue Detail → Scope & Triage

- Modal shows the issue content/metadata.
- Primary action: **"Scope & Triage"** → navigates to a screen collecting optional **Additional Context**.
- On submit:
    - Start a **Devin session** to produce a **developer-ready plan** for this issue.
    - UI **polls every ~3s** to show Devin's **progress** and **confidence (Low/Medium/High)** using *Structured Output* (see schema).
- You can send **Follow-up instructions**; Devin updates the plan, risks, and confidence; polling continues.

D) Execute Plan

- When satisfied, click **Execute Plan**.
- Prompt for **branch name** (pre-fill from `branch_suggestion`).
- In the **same Devin session**, instruct Devin to:
    - create branch from `main`,
    - implement changes + tests,
    - run locally / capture evidence if relevant,
    - open a **PR against main** using the repo's PR template.
- UI continues polling; on completion, show the **PR URL** and mark the issue as **"PR Submitted"** (with link) in the dashboard.

API Surface (Backend)

Repos

- `POST /api/repos/connect` → `{ repoUrl, githubPat }`
  - Validate repo + PAT permissions (needs PR capability).
  - Cache repo record: `{ id, owner, name, url, connectedAt }`.
  - Fetch and cache **open issues (≤100)** immediately.
- `GET /api/repos` → list connected repos + last known open-issue count.
- `DELETE /api/repos/{id}` → remove repo + caches.
- `POST /api/repos/{id}/resync` → re-pull open issues (overwrite any cached analysis).

Issues

- `GET /api/repos/{id}/issues?q=&label=&page=&pageSize=`
  - Query/filter/search/paginate against **cached** issues (do not call GitHub every time).
  - Each item includes: status, title, labels (≤3), `issue_number`, `author`, `created_at` (and derived age).

Devin Sessions

- `POST /api/issues/{issueId}/scope`
  - Body: `{ additionalContext }`
  - Creates a **Devin session** with the *Scoping Prompt* (below), requesting **Structured Output** and storing the `sessionId` in the issue cache.
- `GET /api/devin/{sessionId}`
  - Proxy → returns `status`, `structured_output` (see schema), and a `url` to open the run.
- `POST /api/devin/{sessionId}/message`
  - Body: `{ message }` → sends follow-up instruction; structured output should update.
- `POST /api/issues/{issueId}/execute`
  - Body: `{ branchName }` → sends an *Execute Prompt* to the **same** session. Expect `pr_url` in structured output / final message.

Structured Output — Required JSON Schema

Devin should **continuously** maintain this JSON in `structured_output`:

```
{
  "progress_pct": 0,                  // 0–100 overall
  "confidence": "low|medium|high",    // feasibility of plan
  "summary": "one-paragraph plan",
  "risks": ["list of notable risks"],
  "dependencies": ["list of dependencies/external calls"],
  "estimated_hours": 0,
  "action_plan": [
    {"step": 1, "desc": "specific, testable step", "done": false}
  ],
  "branch_suggestion": "feat/issue-<num>-<slug>",
  "pr_url": ""                        // filled after Execute completes
}
```