Tech Design Document – GitHub Actions Pipeline Dashboard

This document explains the system in an engineer-friendly **and non-technical** way. It covers how the dashboard works, what talks to what, which APIs exist, what we store in the database, and how the user interface is organized.

---

1) High-level architecture

Purpose in one line

A small web app that **collects GitHub Actions run data**, stores it in a database, **shows live metrics** (success rate, average build time, last build status), and **alerts on failures** (Slack/email).

Main building blocks

1. **Frontend (React + Vite, served by Nginx in Docker)**
   The web page you open in the browser. It shows cards, charts, and a table of recent runs, and calls the backend's REST APIs.

2. **Backend (Node.js + Express, Docker)**
   The application server. It:

   - Polls GitHub's REST API for workflow runs at a fixed interval.
   - Calculates metrics (success/failure rate, average build time, latest run status).
   - Stores runs in **PostgreSQL**.
   - Exposes REST APIs that the frontend calls.
   - Sends **Slack** alerts when a run fails.

3. **Database (PostgreSQL, Docker)**
   Stores the raw runs and basic computed fields like duration (in seconds). This enables historical views and reliable metrics.

4. **Alerting (Slack webhook)**
   When the backend sees a failed run, it posts a concise message to a Slack channel. Optional email integration can be added similarly.

5. **Containerization (Docker Compose)**
   Brings up Postgres, backend, and frontend in a single command. Each component is isolated and can be restarted independently.

How data flows end-to-end

1. A GitHub Action workflow finishes (success/failure).

2. On a schedule (e.g., every 60 seconds), the backend **polls GitHub** for the latest runs.
3. The backend **upserts** those runs into Postgres (insert if new, update if already seen).
4. If any run has **conclusion = failure**, the backend posts an **alert to Slack**.
5. The frontend calls the backend **metrics and runs APIs** and renders the dashboard.

Why this architecture

- **Simple & reliable**: polling avoids webhook setup hurdles; Slack alerts provide immediate visibility.
- **Portable**: all components are Dockerized and can run on a laptop, VM, or server.
- **Extensible**: can add Jenkins, CircleCI, or multiple GitHub repos later with minimal changes.

---

2) API structure (routes & sample responses)

Base URL: `/api`

> Notes: • All responses are JSON. • Errors follow `{ "error": "message" }` with appropriate HTTP status codes.

2.1 Health

GET `/api/health`
Checks that the API is up.

```
{ "ok": true }
```

2.2 Metrics

GET `/api/metrics?windowHours=24`
Returns summary metrics for the selected time window.

```
{
  "windowHours": 24,
  "success": 42,
  "failure": 8,
  "completed": 50,
  "successRate": 0.84,
  "averageBuildTimeSec": 315,
  "latestRun": {
    "id": 1234567890,
```

```
    "status": "completed",
    "conclusion": "success",
    "name": "CI",
    "branch": "main",
    "event": "push",
    "actor": "octocat",
    "created_at": "2025-08-21T10:15:04.000Z",
    "run_started_at": "2025-08-21T10:15:05.000Z",
    "updated_at": "2025-08-21T10:20:20.000Z",
    "html_url": "https://github.com/org/repo/actions/runs/1234567890",
    "duration_seconds": 315
  }
}
```


2.3 Runs (paginated)

GET `/api/runs?page=1&pageSize=20`
Lists the most recent runs.


```
{
  "page": 1,
  "pageSize": 20,
  "items": [
    {
      "id": 1234567890,
      "status": "completed",
      "conclusion": "failure",
      "name": "CI",
      "branch": "feature-x",
      "event": "push",
      "actor": "jane",
      "created_at": "2025-08-21T09:03:10.000Z",
      "run_started_at": "2025-08-21T09:03:12.000Z",
      "updated_at": "2025-08-21T09:05:56.000Z",
      "html_url": "https://github.com/org/repo/actions/runs/1234567890",
      "duration_seconds": 164
    }
  ]
}
```


2.4 Latest run (shortcut)

GET `/api/latest-run`
Returns a single most recent run.


```
{
  "id": 1234567890,
  "status": "completed",
  "conclusion": "success",
  "name": "CI",
```

```
  "branch": "main",
  "event": "push",
  "actor": "octocat",
  "created_at": "2025-08-21T10:15:04.000Z",
  "run_started_at": "2025-08-21T10:15:05.000Z",
  "updated_at": "2025-08-21T10:20:20.000Z",
  "html_url": "https://github.com/org/repo/actions/runs/1234567890",
  "duration_seconds": 315
}
```

2.5 Manual poll

POST `/api/poll`
Forces an immediate GitHub poll (useful during demos).

```
{ "ingested": 37 }
```

2.6 Errors (examples)

- `401 Unauthorized` from GitHub → the backend will log a clear message like *"GitHub token missing or invalid"* and respond with an error if the endpoint proxies anything.
- `500 Internal Server Error` → generic unexpected error (e.g., DB not reachable).

---

3) Database schema

3.1 Tables

Table: `runs` (one row per GitHub Actions workflow run)

Column	Type	Notes
`id`	BIGINT (PK)	GitHub Actions run ID (globally unique per run)
`status`	TEXT	`queued`, `in_progress`, `completed`
`conclusion`	TEXT	`success`, `failure`, `cancelled`, `skipped`, `null` (if not done)
`name`	TEXT	Workflow name
`event`	TEXT	e.g., `push`, `pull_request`
`branch`	TEXT	Derived from `head_branch`
`actor`	TEXT	GitHub username who triggered it
`created_at`	TIMESTAMPTZ	When the run record was created by GitHub
`run_started_at`	TIMESTAMPTZ	When the run actually began

Column	Type	Notes
`updated_at`	TIMESTAMPTZ	Last update time (finish time for completed runs)
`html_url`	TEXT	Link to the run in the GitHub UI
`run_attempt`	INT	1 for first attempt, increments on retries
`duration_seconds`	INT	Calculated: `updated_at - run_started_at` in seconds

Indexes: - `idx_runs_created_at` on `(created_at DESC)` → fast latest queries. - `idx_runs_branch` on `(branch)` → filter by branch quickly. - `idx_runs_conclusion` on `(conclusion)` → reporting by outcome.

Initialization DDL (excerpt):

```
CREATE TABLE IF NOT EXISTS runs (
  id BIGINT PRIMARY KEY,
  status TEXT NOT NULL,
  conclusion TEXT,
  name TEXT,
  event TEXT,
  branch TEXT,
  actor TEXT,
  created_at TIMESTAMPTZ,
  run_started_at TIMESTAMPTZ,
  updated_at TIMESTAMPTZ,
  html_url TEXT,
  run_attempt INT,
  duration_seconds INT
);

CREATE INDEX IF NOT EXISTS idx_runs_created_at ON runs(created_at DESC);
CREATE INDEX IF NOT EXISTS idx_runs_branch ON runs(branch);
CREATE INDEX IF NOT EXISTS idx_runs_conclusion ON runs(conclusion);
```

3.2 Data lifecycle

- **Insert/Update:** on each poll, new runs are inserted; existing runs are updated (retry count, conclusion, timestamps, duration).
- **Retention (optional):** you may periodically archive old rows to S3 or prune after N days.
- **Aggregation:** metrics are computed at query time (no separate summary tables needed for this scope).

---

4) UI layout (explanation)

Overview

The UI focuses on **clarity and at-a-glance health**.

Header
- Title: *GitHub Actions Dashboard*
- Subtext: repository/owner and selected time window

Metric cards (top row) - ✅ **Success Rate** — percentage of completed runs that ended in `success` within the window.
- 🕒 **Average Build Time (s)** — average of `duration_seconds` across completed runs in the window.
- 📌 **Last Build Status** — status and branch of the most recent run; includes a link to GitHub. - **Window selector** — dropdown to choose 1h/6h/12h/24h/48h/72h/7d. - **Actions** — a **Poll Now** button to trigger immediate refresh via `/api/poll`.

Charts & tables (main area) - **Bar chart: Success vs Failure** — helps spot trends (e.g., sudden spike in failures).
- **Recent Runs table** — sortable columns: Run ID, Name, Branch, Status, Conclusion, Duration (s), Link.
- Clicking the **Logs** link opens the run in GitHub for detailed logs.

Alerting feedback (optional panel) - Shows latest Slack alert timestamp ("Last failure alert sent …").

Empty/Loading states - Clear copy like "No runs in the selected window yet" and skeleton loaders during fetch.

Responsiveness & accessibility - Cards stack on small screens; table scrolls horizontally if needed.
- Color + text indicators (not color-only) for status to support color-blind users.

Interaction model

- The UI polls **only the backend**; the backend in turn polls GitHub.
- Time window changes trigger a fresh `/api/metrics` fetch.
- "Poll Now" calls `/api/poll` and then reloads metrics and runs.

Non-functional notes (quick)

- **Security**: GitHub PAT stored as environment variable; never committed. Backend rejects startup if token missing. CORS restricted to the frontend origin

(configurable).

- **Performance**: pagination for runs; indexes on time/branch/outcome; polling interval configurable (e.g., 60s).

- **Reliability**: if GitHub is temporarily unavailable, backend logs the error and retries on next interval; Postgres connection pooled.

- **Extensibility**: add org-wide or multi-repo monitoring by storing `owner`, `repo` columns and looping over a configured list.
