CENTRAL SQUARE FOUNDATION

CREATOR'S GUIDE V1.0

ScoutEd

A complete guide to building, deploying, and maintaining the education funding opportunity discovery platform.

Partnerships & Strategic Initiatives Team

Built with React · Supabase · Vercel · GitHub Actions

February 2026

Table of Contents

1. Introduction & What Is ScoutEd

ScoutEd is a web-based dashboard built for the Partnerships and Strategic Initiatives team at **Central Square Foundation (CSF)**. It automatically scouts, aggregates, scores, and displays education funding and grant opportunities from across the web — delivered daily at 8:00 AM IST.

The Problem

Manually tracking education grants across dozens of websites is time-consuming and error-prone. Opportunities are missed, deadlines are forgotten, and the team spends hours browsing instead of building partnerships.

The Solution

ScoutEd automates this entire workflow:

1. A **scraper** runs every morning, fetching opportunities from 10+ sources
2. Each opportunity is **scored** for relevance to CSF's mission (0–100)
3. Results appear on a **branded dashboard** with filtering, bookmarking, and search
4. A **daily email digest** delivers the top opportunities to subscribers at 8:30 AM IST

Design Constraints

CONSTRAINT	DECISION
Zero infrastructure cost	Free tiers of Vercel, Supabase, GitHub Actions, Resend, OpenRouter
3-user limit	No authentication system needed (localStorage for state)
CSF Brand Guidelines	Strict colour, typography, and logo usage compliance

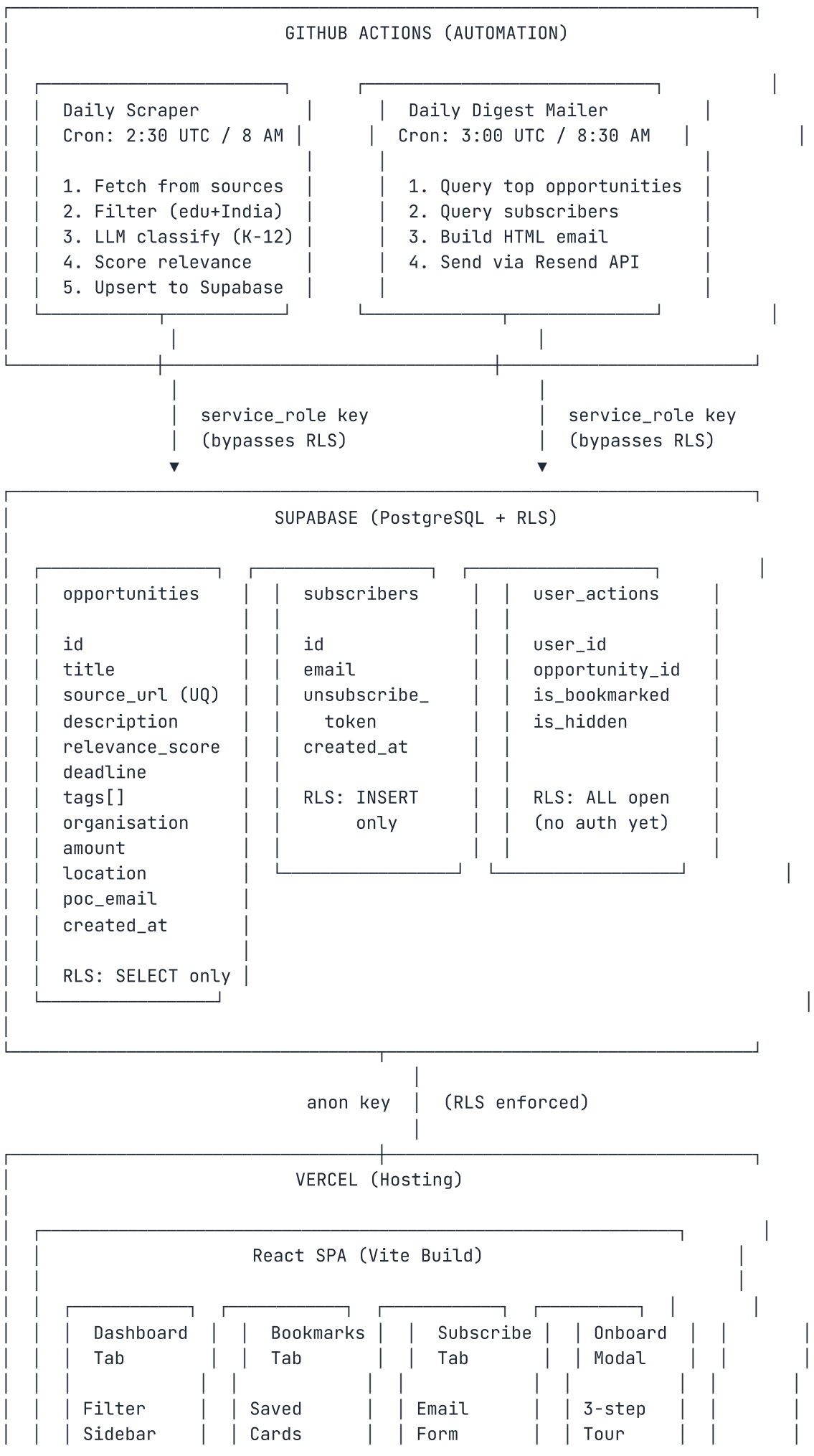
Tech Stack at a Glance

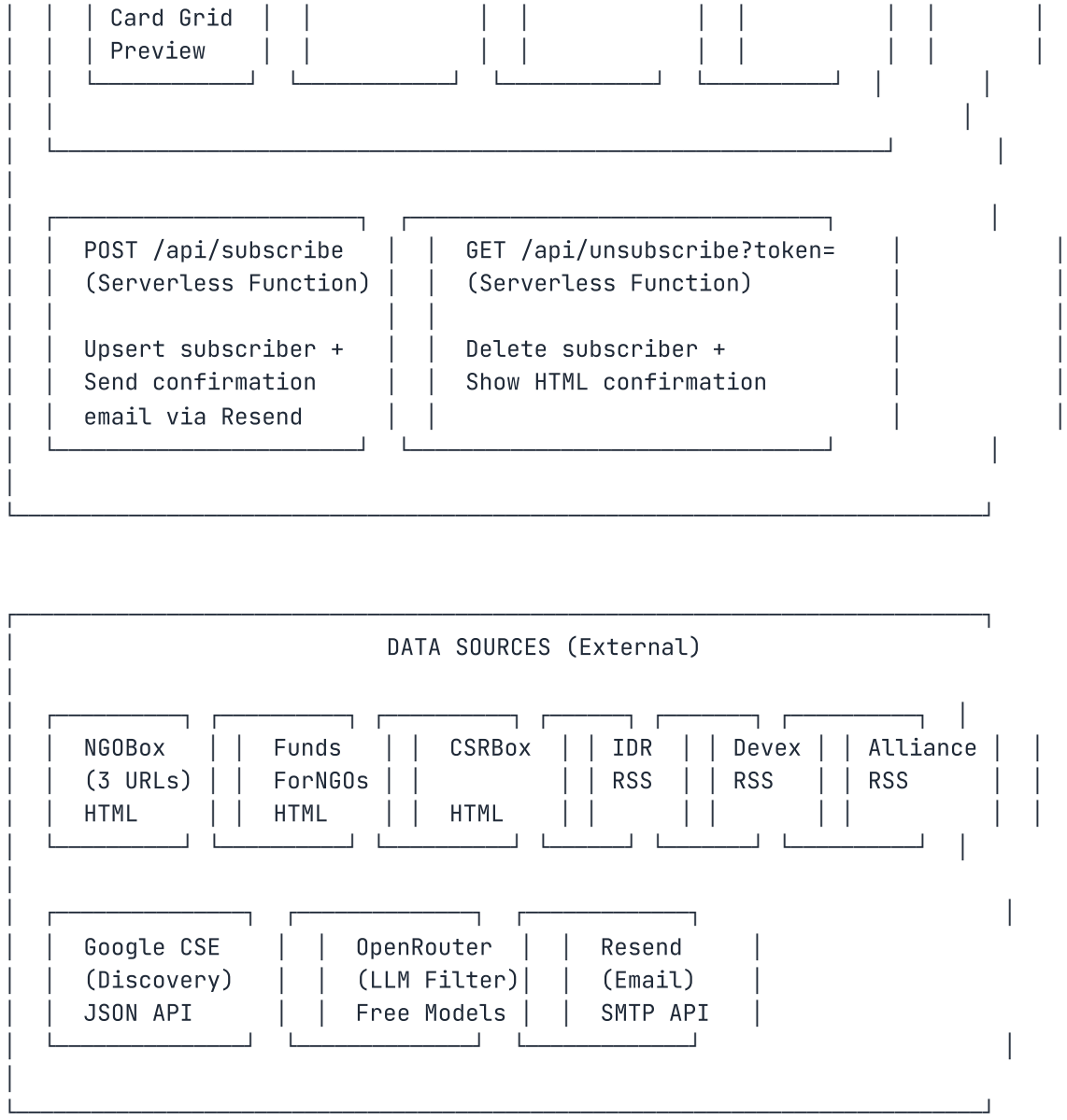
React 19 TypeScript Tailwind CSS v4 Supabase Vercel GitHub Actions Node.js 20

2. Architecture Overview

ScoutEd follows a **serverless, zero-cost architecture** pattern. There is no traditional backend server. The frontend talks directly to Supabase, automation runs via GitHub Actions, and serverless functions handle email operations.

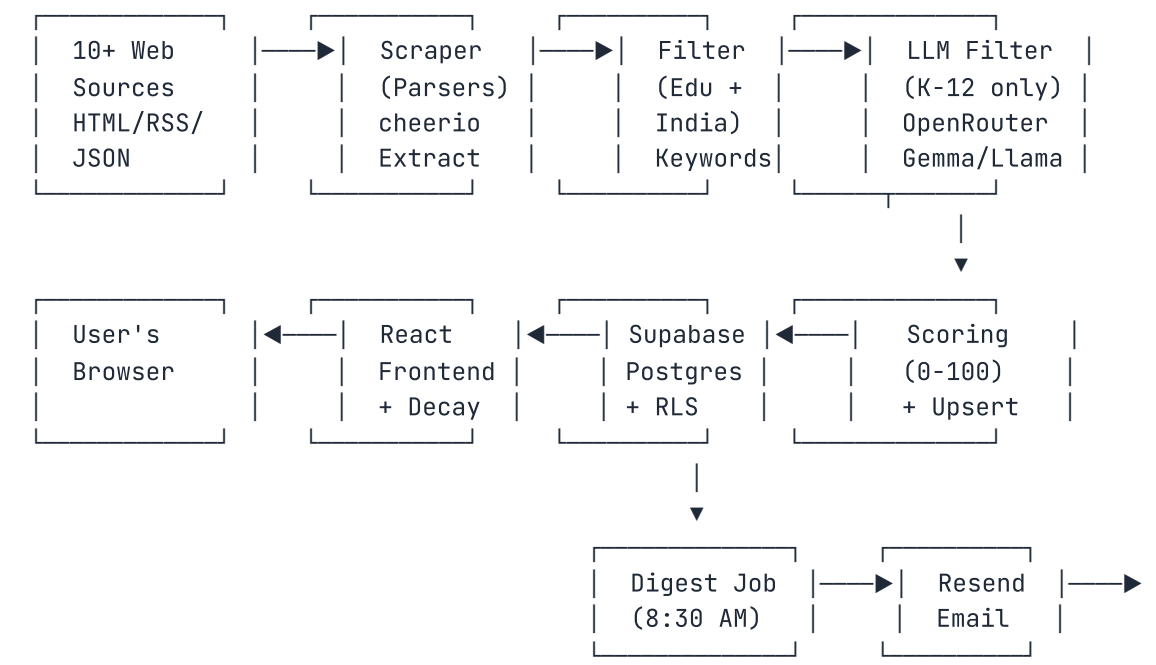
System Architecture Diagram


GITHUB ACTIONS (AUTOMATION)
Daily Scraper
Cron: 2:30 UTC / 8 AM
1. Fetch from sources
2. Filter (edu+India)
3. LLM classify (K-12)
4. Score relevance
5. Upsert to Supabase
Daily Digest Mailer
Cron: 3:00 UTC / 8:30 AM
1. Query top opportunities
2. Query subscribers
3. Build HTML email
4. Send via Resend API
service_role key
(bypasses RLS)
service_role key
(bypasses RLS)
SUPABASE (PostgreSQL + RLS)
opportunities
id
title
source_url (UQ)
description
relevance_score
deadline
tags[]
organisation
amount
location
poc_email
created_at
RLS: SELECT only
subscribers
id
email
unsubscribe_
token
created_at
RLS: INSERT
only
user_actions
user_id
opportunity_id
is_bookmarked
is_hidden
RLS: ALL open
(no auth yet)
anon key
(RLS enforced)
VERCEL (Hosting)
React SPA (Vite Build)
Dashboard
Tab
Filter
Sidebar
Bookmarks
Tab
Saved
Cards
Subscribe
Tab
Email
Form
Onboard
Modal
3-step
Tour



Data Flow Summary

DATA FLOW: From Web Sources to User's Screen & Inbox



3. Prerequisites & Tool Installation

3a. Node.js, Git & VS Code

You need the following installed on your machine before starting:

TOOL	VERSION	PURPOSE
Node.js	20+ (LTS)	JavaScript runtime for frontend and scraper
npm	10+ (comes with Node)	Package manager
Git	2.40+	Version control
VS Code	Latest	Code editor (recommended)

Installing Node.js

```
# Download and install from https://nodejs.org (LTS version)
# Verify installation:
$ node --version      # Should show v20.x.x or higher
$ npm --version       # Should show 10.x.x or higher
```

Installing Git

```
# Download from https://git-scm.com/downloads
# Verify:
$ git --version       # Should show 2.40+
```

Recommended VS Code Extensions

- **ESLint** — JavaScript/TypeScript linting
- **Tailwind CSS IntelliSense** — Autocomplete for Tailwind classes
- **Prettier** — Code formatting
- **Claude Code** — AI-powered coding assistant (see next section)

3b. Claude Code (AI Coding Assistant)

Claude Code is the AI assistant used to build and maintain ScoutEd. It runs in your terminal and can read, write, and execute code with your approval.

Step 1: Install Claude Code

```
# Install globally via npm
$ npm install -g @anthropic-ai/claude-code
```

Step 2: Authenticate

```
# Navigate to your project directory
$ cd path/to/Scouted

# Launch Claude Code (opens browser for authentication)
$ claude
```

On first run, Claude Code will open your browser for authentication with your Anthropic account. Follow the prompts to authorise.

Step 3: Using Claude Code with ScoutEd

```
# Start Claude Code in the project directory
$ cd path/to/Scouted
$ claude

# Claude reads the CLAUDE.md file automatically for project context
# You can now ask it to:
#   - Add new scraper sources
#   - Fix bugs
#   - Modify the UI
#   - Update scoring rules
#   - Create commits (/commit)
```

> **HOW CLAUDE CODE KNOWS ABOUT SCOUTED**
>
> The `CLAUDE.md` file in the project root contains all architecture decisions, brand guidelines, and implementation rules. Claude Code reads this automatically and follows the instructions strictly. You do not need to re-explain the project each time.

Key Claude Code Commands

COMMAND	WHAT IT DOES
`/help`	Show available commands
`/commit`	Stage changes and create a git commit
`/clear`	Clear conversation history
`Ctrl+C`	Cancel current operation
`Escape` (x2)	Exit Claude Code

VS Code Integration (Optional)

Install the **Claude Code** extension from the VS Code marketplace. This embeds Claude Code directly in the VS Code sidebar so you can use it without switching to a separate terminal.

4. Project Setup & Local Development

Cloning the Repository

```
$ git clone https://github.com/your-org/Scouted.git
$ cd Scouted
```

Installing Dependencies

```
# Install frontend dependencies
$ npm install

# Install scraper dependencies
$ cd scraper
$ npm install
$ cd ..
```

Environment Configuration

Create a `.env` file in the project root:

```
# .env (Frontend)
VITE_SUPABASE_URL=your_supabase_project_url
VITE_SUPABASE_ANON_KEY=your_supabase_anon_key
```

> **MOCK DATA FALLBACK**
>
> If `VITE_SUPABASE_URL` is not set, the frontend automatically uses built-in mock data (`src/lib/mockData.ts`) with 15+ realistic Indian education grant examples. This means you can work on the UI without a database connection.

Starting the Development Server

```
$ npm run dev

# Output:
#   VITE v7.3.1  ready in 300 ms
#   ➜  Local:   http://localhost:5173/
#   ➜  Network: http://192.168.x.x:5173/
```

Open `http://localhost:5173` in your browser. You should see the ScoutEd dashboard with the onboarding modal.

Available Scripts

COMMAND	DESCRIPTION
`npm run dev`	Start Vite dev server with hot reload
`npm run build`	TypeScript check + production build → `dist/`
`npm run preview`	Preview the production build locally
`npm run lint`	Run ESLint on all source files
`cd scraper && npm run scrape`	Run the scraper manually (requires env vars)

Project Structure

```
Scouted/
├── src/                          # Frontend React application
│   ├── App.tsx                   # Main app with routing & state
│   ├── main.tsx                  # Entry point
│   ├── index.css                 # Tailwind v4 theme & animations
│   ├── types/
│   │   └── index.ts              # TypeScript interfaces
│   ├── lib/
│   │   ├── constants.ts          # CSF sectors, states, score weights
│   │   ├── scoring.ts            # Frontend decay logic
│   │   ├── supabase.ts           # Database client init
│   │   └── mockData.ts           # Fallback mock data (15+ entries)
│   ├── hooks/
│   │   ├── useOpportunities.ts   # React Query infinite scroll
│   │   ├── useBookmarks.ts       # localStorage bookmarks/hidden
│   │   └── useOnboarding.ts      # First-visit tour flag
│   └── components/
│       ├── layout/     Header, MobileNav
│       ├── cards/      OpportunityCard
│       ├── filters/    FilterSidebar, MobileFilterDrawer
│       ├── ui/         Badge, RelevanceScore, SearchBar
│       ├── subscription/ EmailSubscribe
│       ├── onboarding/   OnboardingModal
│       └── preview/      PreviewPane
│
├── scraper/                      # Daily scraper (GitHub Actions)
│   ├── index.ts                  # Main orchestration
│   ├── config.ts                 # Sources, sectors, funders
│   ├── scoring.ts                # Relevance scoring algorithm
│   ├── supabase.ts               # DB upsert logic
│   ├── llm-filter.ts             # LLM classification (OpenRouter)
│   ├── send-digest.ts            # Daily email builder
│   └── parsers/
│       ├── utils.ts              # Shared utilities
│       ├── detail-fetcher.ts     # Batch detail page scraping
│       ├── ngobox.ts             # NGOBox grant parser
│       ├── ngobox-rfp.ts         # NGOBox RFP/EOI parser
│       ├── fundsforngos.ts       # FundsForNGOs parser
│       ├── csrbox.ts             # CSRBox CSR data parser
│       ├── idr.ts                # IDR RSS parser
│       ├── devex.ts              # Devex RSS + article parser
│       ├── alliance.ts           # Alliance Magazine parser
│       └── google-cse.ts         # Google Custom Search parser
│
├── api/                          # Vercel serverless functions
│   ├── subscribe.ts              # POST /api/subscribe
│   └── unsubscribe.ts            # GET /api/unsubscribe?token=
│
├── supabase/
│   └── schema.sql                # Database schema + RLS policies
│
├── .github/workflows/
│   ├── scraper.yml               # Cron: daily scraper at 8 AM IST
│   └── daily-digest.yml          # Cron: email digest at 8:30 AM IST
│
├── package.json                  # Frontend deps & scripts
├── vite.config.ts                # Vite + React + Tailwind config
```

```
├── vercel.json                    # URL rewrite rules
├── CLAUDE.md                      # AI assistant instructions
└── .env.example                   # Environment variable template
```

5. Supabase Database Setup

Creating a Supabase Project

1. Go to supabase.com and sign up for a free account.
2. Click **"New Project"**. Choose a name (e.g., "scouted"), select a region close to India (Mumbai or Singapore), and set a database password.
3. Once the project is created, navigate to **Settings** → **API**. Note down the **Project URL** and **anon (public) key** for frontend use, and the **service_role key** for scraper use.

Running the Schema

4. In the Supabase dashboard, go to **SQL Editor**. Paste the entire contents of `supabase/schema.sql` and click **"Run"**.

This creates three tables with Row Level Security:

TABLE	PURPOSE	ANON ACCESS
`opportunities`	Grant/funding listings from scraper	SELECT only
`subscribers`	Email digest subscribers	INSERT only
`user_actions`	Bookmarks and hidden items	Full (no auth yet)

Schema Details

```sql
-- Key table: opportunities
CREATE TABLE opportunities (
  id           UUID DEFAULT uuid_generate_v4() PRIMARY KEY,
  title        TEXT NOT NULL,
  source_url   TEXT NOT NULL UNIQUE,    -- Dedup key
  description  TEXT NOT NULL DEFAULT '',
  relevance_score INTEGER NOT NULL DEFAULT 0,
  deadline     DATE,
  poc_email    TEXT,
  tags         TEXT[] DEFAULT '{}',
  organisation TEXT,
  amount       TEXT,
  location     TEXT,
  created_at   TIMESTAMPTZ DEFAULT NOW()
);
```

> **IMPORTANT: SOURCE_URL IS THE DEDUP KEY**
>
> The `source_url` column has a `UNIQUE` constraint. The scraper uses `UPSERT` with `onConflict: 'source_url'` to prevent duplicates — if the same URL is scraped twice, the existing row is updated rather than duplicated.

Row Level Security Rationale

- **Opportunities:** Public read access. The scraper writes using the `service_role` key which bypasses RLS.
- **Subscribers:** Anyone can subscribe (INSERT). No one can list emails (prevents harvesting). Digest uses `service_role` to read.
- **User Actions:** Fully open for now. When Supabase Auth is added, this will be locked to `auth.uid()` .

6. Frontend Deep Dive

Component Architecture

```
App.tsx
 │
 ├── Header (sticky top nav, hamburger menu)
 │
 ├── [Dashboard Tab]
 │   ├── FilterSidebar (desktop: score, sectors, states, deadline)
 │   ├── SearchBar (full-text search)
 │   ├── OpportunityCard[] (grid of cards)
 │   │   ├── RelevanceScore (animated SVG circle)
 │   │   ├── Badge[] (sector tags with colours)
 │   │   └── Actions (bookmark, hide, external link)
 │   ├── "Load More" (infinite pagination)
 │   └── EmailSubscribe (sidebar, desktop only)
 │
 ├── [Bookmarks Tab]
 │   └── OpportunityCard[] (filtered to bookmarked items)
 │
 ├── [Subscribe Tab]
 │   └── EmailSubscribe (email form + confirmation)
 │
 ├── PreviewPane (slide-over detail panel)
 ├── MobileNav (bottom tab bar)
 ├── MobileFilterDrawer (sheet overlay)
 └── OnboardingModal (3-step first-visit tour)
```

State Management

ScoutEd uses **React hooks and local state** — no global state library:

HOOK	STATE	STORAGE
`useOpportunities`	Opportunity data + pagination	React Query (in-memory, 5 min stale time)
`useBookmarks`	Bookmarked & hidden item IDs	localStorage
`useOnboarding`	Has user seen the tour?	localStorage (`scouted_onboarding_complete`)
`useState` in App	Active tab, filters, selected card	In-memory (resets on refresh)

Data Fetching Pattern

Uses **TanStack React Query** with infinite pagination. The hook queries Supabase directly (no REST API layer), fetching 20 items per page:

```
// useOpportunities.ts — simplified
const { data, fetchNextPage, hasNextPage } = useInfiniteQuery({
  queryKey: ['opportunities', filters],
  queryFn: async ({ pageParam = 0 }) => {
    // Server-side filtering (score threshold, deadline)
    let query = supabase
      .from('opportunities')
      .select('*')
      .order('relevance_score', { ascending: false })
      .range(pageParam, pageParam + 19)

    // Client-side filtering (sectors, states, text search)
    return clientFilter(data, filters)
  },
  staleTime: 5 * 60 * 1000  // 5 minutes
})
```

Styling: Tailwind CSS v4

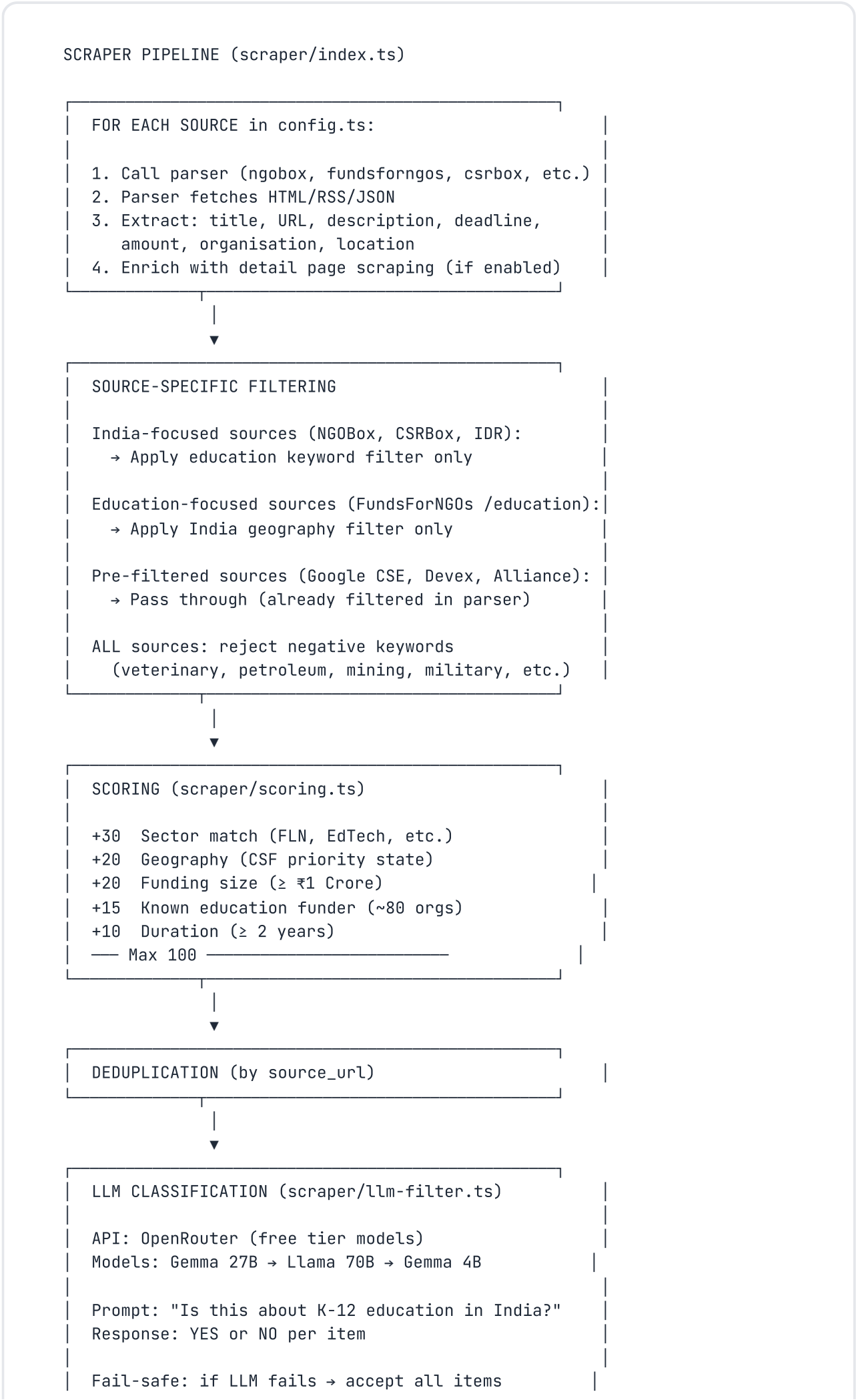
Theme colours are defined in `src/index.css` using Tailwind v4's `@theme` directive. This makes CSF brand colours available as utility classes throughout the app:

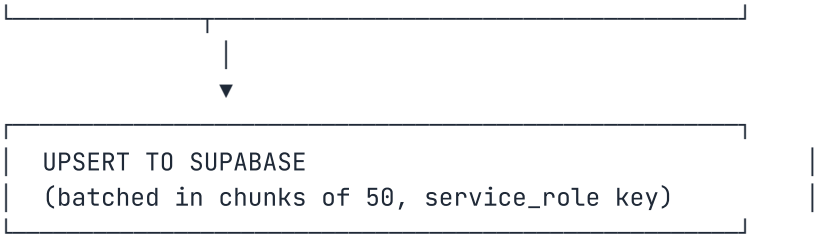
```css
/* src/index.css */
@theme {
  --color-csf-blue: #00316B;
  --color-csf-yellow: #FFD400;
  --color-csf-light-blue: #8FBAFF;
  --color-csf-orange: #C93F13;
  --color-csf-lime: #87FF38;
  --color-csf-purple: #7030A0;

  --font-heading: "Gill Sans MT", "Gill Sans", sans-serif;
  --font-body: "Cambria", Georgia, serif;
}

/* Usage in components: */
/* bg-csf-blue, text-csf-yellow, font-heading, font-body */
```

7. Scraper Deep Dive

How the Scraper Works

The scraper is a Node.js TypeScript script that runs as a GitHub Actions cron job every day at 8:00 AM IST.


SCRAPER PIPELINE (scraper/index.ts)
FOR EACH SOURCE in config.ts:
1. Call parser (ngobox, fundsforngos, csrbox, etc.)
2. Parser fetches HTML/RSS/JSON
3. Extract: title, URL, description, deadline, amount, organisation, location
4. Enrich with detail page scraping (if enabled)
SOURCE-SPECIFIC FILTERING
India-focused sources (NGOBox, CSRBox, IDR):
→ Apply education keyword filter only
Education-focused sources (FundsForNGOs /education):
→ Apply India geography filter only
Pre-filtered sources (Google CSE, Devex, Alliance):
→ Pass through (already filtered in parser)
ALL sources: reject negative keywords
(veterinary, petroleum, mining, military, etc.)
SCORING (scraper/scoring.ts)
+30 Sector match (FLN, EdTech, etc.)
+20 Geography (CSF priority state)
+20 Funding size (≥ ₹1 Crore)
+15 Known education funder (~80 orgs)
+10 Duration (≥ 2 years)
── Max 100 ──
DEDUPLICATION (by source_url)
LLM CLASSIFICATION (scraper/llm-filter.ts)
API: OpenRouter (free tier models)
Models: Gemma 27B → Llama 70B → Gemma 4B
Prompt: "Is this about K-12 education in India?"
Response: YES or NO per item
Fail-safe: if LLM fails → accept all items



Active Sources

SOURCE	PARSER FILE	TYPE	EXPECTED YIELD
NGOBox Education Grants	`ngobox.ts`	HTML scraping (paginated)	5–20/run
NGOBox All Grants	`ngobox.ts`	HTML scraping (paginated)	5–20/run
NGOBox RFP/EOI	`ngobox-rfp.ts`	HTML scraping	5–10/run
FundsForNGOs Education	`fundsforngos.ts`	HTML (WordPress)	3–10/run
FundsForNGOs India	`fundsforngos.ts`	HTML (WordPress)	3–10/run
CSRBox	`csrbox.ts`	HTML scraping	10–30 companies
IDR (India Development Review)	`idr.ts`	RSS feed	5–10/run
Devex News	`devex.ts`	RSS + article pages	2–5/run
Alliance Magazine	`alliance.ts`	RSS + search pages	2–5/run
Google Custom Search	`google-cse.ts`	JSON API	5–20/run

Shared Parser Utilities

All parsers import from `scraper/parsers/utils.ts` :

FUNCTION	PURPOSE	EXAMPLE
`stripTags(html)`	Remove HTML, collapse whitespace	`"<b>Hello</b>"` → `"Hello"`
`extractTags(text)`	Map text to CSF sector tags	→ `["EdTech", "FLN / Foundational Literacy"]`
`extractLocation(text)`	Find Indian state names	→ `"Haryana"`
`parseDate(text)`	Parse various date formats	`"16 Feb 2026"` → `"2026-02-16"`
`extractAmount(text)`	Parse Indian/international amounts	→ `"₹5 Crore"`
`dedup(items)`	Deduplicate by `source_url`	Removes duplicate entries
`isEducationRelevant(text)`	K-12 keyword check	`true` / `false`

Detail Page Enrichment

After initial scraping, `detail-fetcher.ts` visits each opportunity's source URL to extract the full description and POC email. Rate-limited to prevent blocks:

- Max 30 items per source
- 3 concurrent requests
- 1 second delay between batches
- 10 second timeout per page
- Falls back to original data on failure

Running the Scraper Manually

```
# Requires SUPABASE_URL and SUPABASE_SERVICE_ROLE_KEY env vars
$ cd scraper
$ npx tsx index.ts
```

8. Relevance Scoring Algorithm

Base Score (Calculated by Scraper)

Each opportunity starts at 0 and earns points across five dimensions:

COMPONENT	POINTS	CONDITION
Sector Match	+30	Title, description, or tags contain: FLN, Foundational Literacy, School Governance, EdTech, Early Childhood, Classroom Instruction, High Potential Students
Geography Match	+20	Location or text mentions a CSF priority state: Punjab, Haryana, Uttar Pradesh, Telangana, Odisha, Assam, Bihar, Himachal Pradesh, Gujarat
Funding Size	+20	Amount ≥ ₹1 Crore OR ≥ $1 Million (international equivalent)
Known Donor	+15	Organisation matches one of ~80 known education funders (Gates Foundation, Tata Trusts, Wipro, etc.)
Duration	+10	Project duration ≥ 2 years
	Maximum Score: 100	

Decay (Applied by Frontend)

To keep the feed fresh, the **frontend applies a decay of −5 points per week** since the opportunity was created:

```
function applyDecay(baseScore: number, createdAt: string): number {
  const weeks = Math.floor(
    (Date.now() - new Date(createdAt).getTime()) / (7 * 24 * 60 * 60 * 1000)
  )
  return Math.max(0, baseScore - weeks * 5)
}
```

> **WHY DECAY IS FRONTEND-ONLY**

> The scraper stores the *base score* in the database. Decay is applied on render so that the score dynamically decreases as time passes without needing a database update. Supabase also has a `decayed_score()` function for server-side sorting if needed.

Visual Score Indicator

SCORE RANGE	COLOUR	LABEL
75–100	**Green**	High relevance
50–74	**Yellow**	Medium relevance
0–49	**Red**	Low relevance

9. Email Digest System

How It Works

1. **Subscription:** Users enter their email on the dashboard. A `POST /api/subscribe` serverless function upserts the email into the `subscribers` table and sends a confirmation email via Resend.
2. **Daily Digest:** A GitHub Actions cron job runs at 3:00 UTC (8:30 AM IST). It queries the top 10 opportunities from the last 48 hours, builds a branded HTML email, and sends it to all subscribers.
3. **Unsubscribe:** Every email includes a personalised unsubscribe link. Clicking it hits `GET /api/unsubscribe?token=<uuid>`, which deletes the subscriber and shows an HTML confirmation page.

Email Template

The digest email is a professionally designed HTML email using table-based layout for maximum client compatibility:

- **Header:** ScoutEd branding with CSF Blue (#00316B) background and date
- **Body:** Up to 10 opportunity cards with score badge, title, description preview, metadata
- **Footer:** "View All on ScoutEd" button, mission statement, personalised unsubscribe link
- **Sender:** `digest@scouted.whybe.ai` (verified domain via Resend)

Serverless API Functions

ENDPOINT	METHOD	PURPOSE
`/api/subscribe`	POST	Add email to subscribers + send confirmation
`/api/unsubscribe?token=`	GET	Remove subscriber + show confirmation HTML

Both functions live in the `api/` directory and are automatically deployed as Vercel serverless functions.

> **RESEND FREE TIER LIMIT**

> Resend's free tier allows 100 emails/day and 3,000 emails/month. With 3 subscribers and one digest per day, this is more than sufficient.

10. Deploying to Vercel

Initial Deployment

1 **Install Vercel CLI**

```
$ npm install -g vercel
```

2 **Link Your Project**

```
$ cd path/to/Scouted
$ vercel link
# Follow prompts to connect to your Vercel account
```

3 **Set Environment Variables**

In the Vercel dashboard (Settings → Environment Variables), add:

VARIABLE	ENVIRONMENT	PURPOSE
`VITE_SUPABASE_URL`	All	Supabase project URL (used at build time)
`VITE_SUPABASE_ANON_KEY`	All	Supabase public key (used at build time)
`SUPABASE_URL`	All	For serverless functions
`SUPABASE_SERVICE_ROLE_KEY`	All	For serverless function writes
`RESEND_API_KEY`	All	For confirmation/unsubscribe emails
`RESEND_DOMAIN_VERIFIED`	All	Set to `true` to use verified domain

4 Deploy

```
# Preview deployment
$ vercel

# Production deployment
$ vercel --prod
```

> **AUTO-DEPLOYMENT WITH GIT**
>
> Once linked, Vercel automatically deploys on every `git push` to the `master` branch. Pull requests get preview deployments with unique URLs.

Vercel Configuration

The `vercel.json` file handles URL routing:

```
{
  "rewrites": [
    { "source": "/api/(.*)", "destination": "/api/$1" },
    { "source": "/(.*)",     "destination": "/index.html" }
  ]
}
```

- `/api/*` routes go to serverless functions in the `api/` directory
- All other routes serve the React SPA (client-side routing)

Build Settings

SETTING	VALUE
Framework Preset	Vite
Build Command	`tsc -b && vite build`
Output Directory	`dist`
Node.js Version	20.x

11. GitHub Actions (Automation)

Two workflows automate the daily scraping and email digest.

Workflow 1: Daily Scraper

File: `.github/workflows/scraper.yml`

SETTING	VALUE
Schedule	`cron: '30 2 * * *'` (2:30 UTC = 8:00 AM IST)
Runner	`ubuntu-latest`
Timeout	10 minutes
Node Version	20
Command	`npx tsx index.ts` (in `scraper/` directory)

Workflow 2: Daily Email Digest

File: `.github/workflows/daily-digest.yml`

SETTING	VALUE
Schedule	`cron: '0 3 * * *'` (3:00 UTC = 8:30 AM IST)
Runner	`ubuntu-latest`
Timeout	5 minutes
Node Version	20
Command	`npx tsx send-digest.ts` (in `scraper/` directory)

Setting Up GitHub Secrets

Go to your GitHub repository → **Settings** → **Secrets and variables** → **Actions**

Add the following repository secrets:

SECRET NAME	PURPOSE
`SUPABASE_URL`	Supabase project URL
`SUPABASE_SERVICE_ROLE_KEY`	Service role key (bypasses RLS)
`OPENROUTER_API_KEY`	OpenRouter API key (for LLM filter; optional)
`RESEND_API_KEY`	Resend email API key (for digest)

Manual Trigger

Both workflows support `workflow_dispatch`, meaning you can trigger them manually:

```
# Via GitHub CLI
$ gh workflow run "ScoutEd Daily Scraper"
$ gh workflow run "ScoutEd Daily Email Digest"

# Or via GitHub UI: Actions tab → Select workflow → "Run workflow"
```

12. Environment Variables Reference

VARIABLE	WHERE USED	REQUIRED	PURPOSE
`VITE_SUPABASE_URL`	Frontend (.env)	No*	Supabase project URL. *Falls back to mock data if unset.
`VITE_SUPABASE_ANON_KEY`	Frontend (.env)	No*	Supabase public (anon) key. *Falls back to mock data if unset.
`SUPABASE_URL`	Scraper, API functions, GitHub Actions	Yes	Supabase project URL
`SUPABASE_SERVICE_ROLE_KEY`	Scraper, API functions, GitHub Actions	Yes	Service role key (bypasses RLS for writes/reads)
`OPENROUTER_API_KEY`	Scraper (GitHub Actions)	No	OpenRouter key for LLM classification. If unset, LLM filter is skipped.
`GOOGLE_CSE_API_KEY`	Scraper	No	Google Custom Search API key. If unset, CSE source is skipped.
`GOOGLE_CSE_ID`	Scraper	No	Programmable Search Engine ID (50-domain limit)
`RESEND_API_KEY`	API functions, Digest (GitHub Actions)	Yes (digest)	Resend email API key
`RESEND_DOMAIN_VERIFIED`	API functions	No	Set to `"true"` to use verified domain sender

SECURITY: NEVER COMMIT CREDENTIALS

> Never commit `.env` files, API keys, or service role keys to the repository. Use `.env.example` as a template (already in the repo) and `.gitignore` to exclude `.env`. Store secrets in GitHub Actions Secrets and Vercel Environment Variables.

13. Adding New Sources

To add a new data source to the scraper, follow these five steps:

1 **Create a Parser File**

Create `scraper/parsers/<name>.ts` that exports a parse function:

```
import { stripTags, extractTags, extractLocation, parseDate,
         extractAmount, dedup } from './utils.js'

interface RawOpportunity {
  title: string
  source_url: string
  description: string
  deadline: string | null
  poc_email: string | null
  tags: string[]
  organisation: string | null
  amount: string | null
  location: string | null
}

export async function parseMySource(url: string): Promise<RawOpportunity[]> {
  const res = await fetch(url)
  const html = await res.text()
  // Parse HTML with cheerio, extract fields
  // Use shared utilities for tags, location, dates, amounts
  return dedup(opportunities)
}
```

2 **Add to Config**

Add a new entry to `SOURCES` array in `scraper/config.ts` :

```
{
  name: 'My New Source',
  url: 'https://example.com/grants',
  parser: 'my-source',
}
```

3 **Register Parser**

Add to the `PARSERS` map in `scraper/index.ts`:

```
import { parseMySource } from './parsers/my-source.js'

const PARSERS = {
  // ... existing parsers
  'my-source': parseMySource,
}
```

4 **Classify Filtering**

In the filter block of `scraper/index.ts`, decide whether the source is India-focused, education-focused, or pre-filtered:

```
// India-focused → apply education filter only
const isIndiaSource = source.url.includes('example.com')

// Education-focused → apply India filter only
const isEducationSource = source.url.includes('/education/')
```

5 **Test Locally**

```
$ cd scraper
$ npx tsx index.ts
```

Watch the console output to verify your parser fetches and filters correctly.

Recommended Future Sources

SOURCE	TYPE	NOTES
NITI Aayog	Government policy	Indian government think tank
MyGov.in	Government engagement	Citizen participation platform
Give2Asia	Philanthropy	Asia-focused grant listings
data.gov.in	Open data	CSR expenditure datasets
csr.gov.in	National CSR portal	Requires browser automation

14. Brand Guidelines Reference

Colour Palette

COLOUR	HEX	CSS VARIABLE	USAGE
CSF Blue	`#00316B`	`csf-blue`	Backgrounds, headers, primary buttons
CSF Yellow	`#FFD400`	`csf-yellow`	Accents, highlights, CTA buttons
Light Blue	`#8FBAFF`	`csf-light-blue`	Tags, secondary elements
Burnt Orange	`#C93F13`	`csf-orange`	Deadline urgency indicators
Lime Green	`#87FF38`	`csf-lime`	High relevance, positive indicators
Purple	`#7030A0`	`csf-purple`	Charts, additional categorisation

Typography

ELEMENT	FONT	TAILWIND CLASS	FALLBACK
Headings (H1, H2, Card titles)	Gill Sans MT	`font-heading`	Gill Sans, sans-serif
Body text, descriptions	Cambria	`font-body`	Georgia, serif

Formatting Rules

- **Capitalise subjects:** Math, Science, English (not math, science, english)
- **British English:** Programme (not Program), Colour (not Color)
- **Indian numbering:** Use Lakh/Crore (not Million/Billion) for Indian grants
- **Logo clear space:** Always maintain 1/2 logo height as clear space around the CSF logo

15. Troubleshooting

Common Issues

"No opportunities found" on dashboard

- Check that `VITE_SUPABASE_URL` and `VITE_SUPABASE_ANON_KEY` are set in `.env`
- Verify the database has data by checking the Supabase dashboard → Table Editor → opportunities
- If no env vars are set, mock data should appear. If even mock data doesn't show, check browser console for errors

Scraper returns 0 results

- Source websites may have changed their HTML structure. Check the parser against the current page
- Verify environment variables (`SUPABASE_URL` , `SUPABASE_SERVICE_ROLE_KEY`) are correct
- Check GitHub Actions logs: Actions tab → Select failed run → Read logs

Email digest not sending

- Verify `RESEND_API_KEY` is set in GitHub Secrets
- Check Resend dashboard for delivery logs and errors
- Ensure there are subscribers in the `subscribers` table
- Ensure there are recent opportunities (last 48 hours) to include in the digest

LLM filter rejecting everything

- The LLM filter is designed to be strict about K-12 education. Higher education and non-education items are intentionally rejected
- If OpenRouter free tier quota is exhausted, the filter accepts all remaining items (fail-safe)
- Check if `OPENROUTER_API_KEY` is valid by testing a manual API call

Vercel deployment fails

- Run `npm run build` locally first to catch TypeScript errors

- Ensure all `VITE_*` environment variables are set in Vercel dashboard
- Check Vercel deployment logs for the specific error

Supabase RLS errors

- **"permission denied"** on INSERT to opportunities → The scraper must use `service_role` key, not the anon key
- **"permission denied"** on SELECT from subscribers → Anon users cannot read subscriber emails (by design)
- To debug RLS, temporarily check the Supabase dashboard → Authentication → Policies

Bookmarks lost on different device

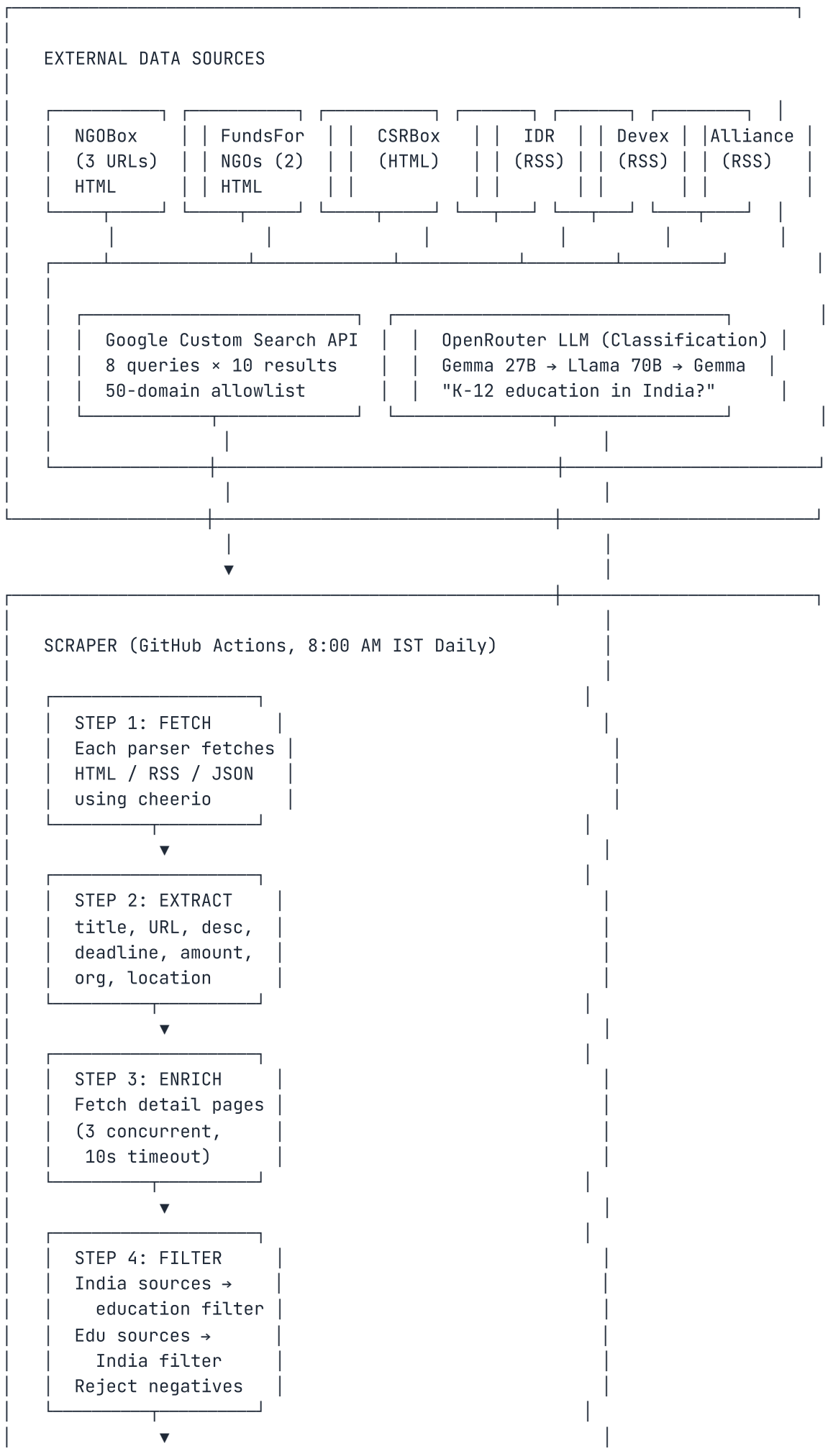
This is by design. Bookmarks are stored in `localStorage` and are device-specific. When Supabase Auth is added (future work), bookmarks will sync across devices.

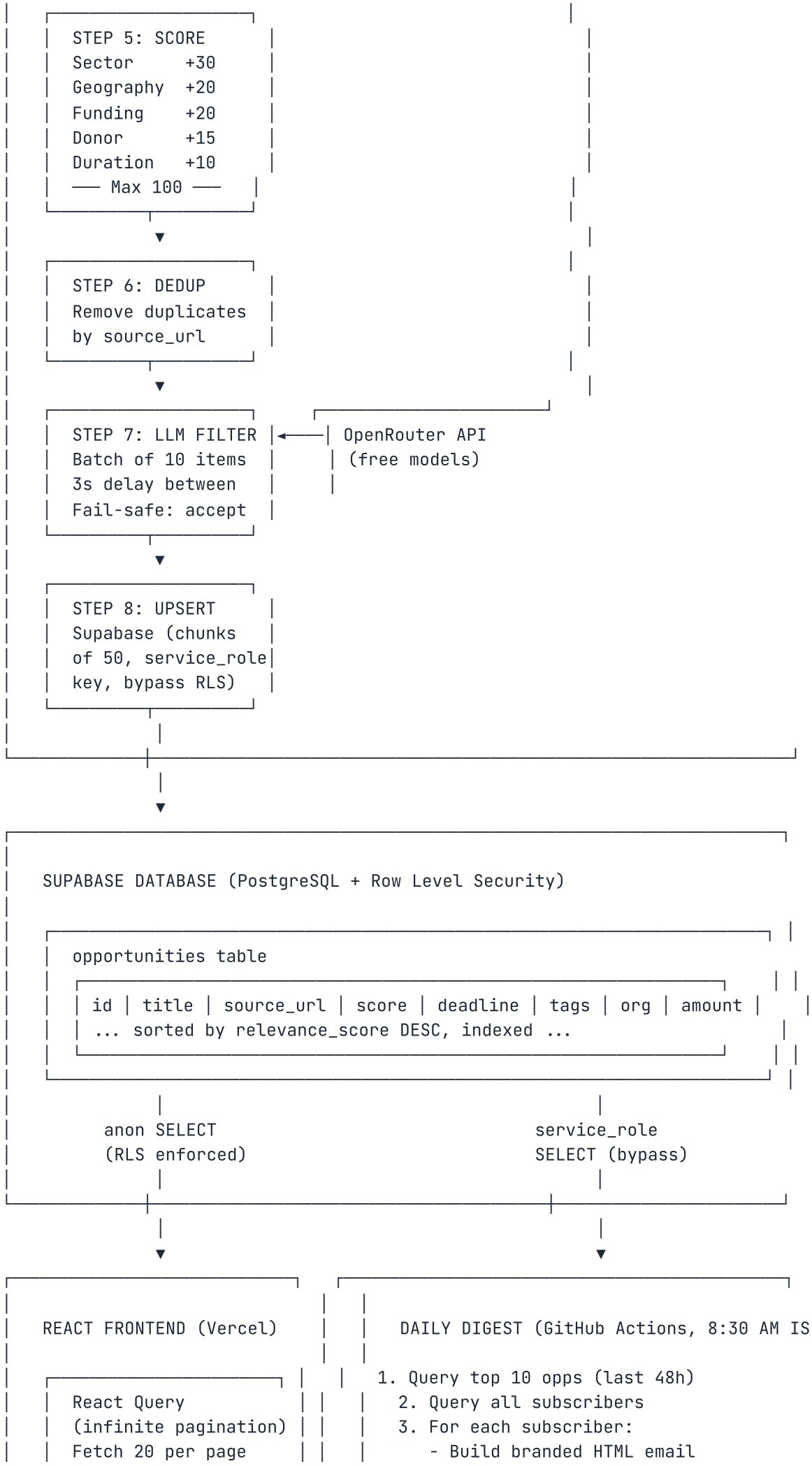
Useful Debug Commands

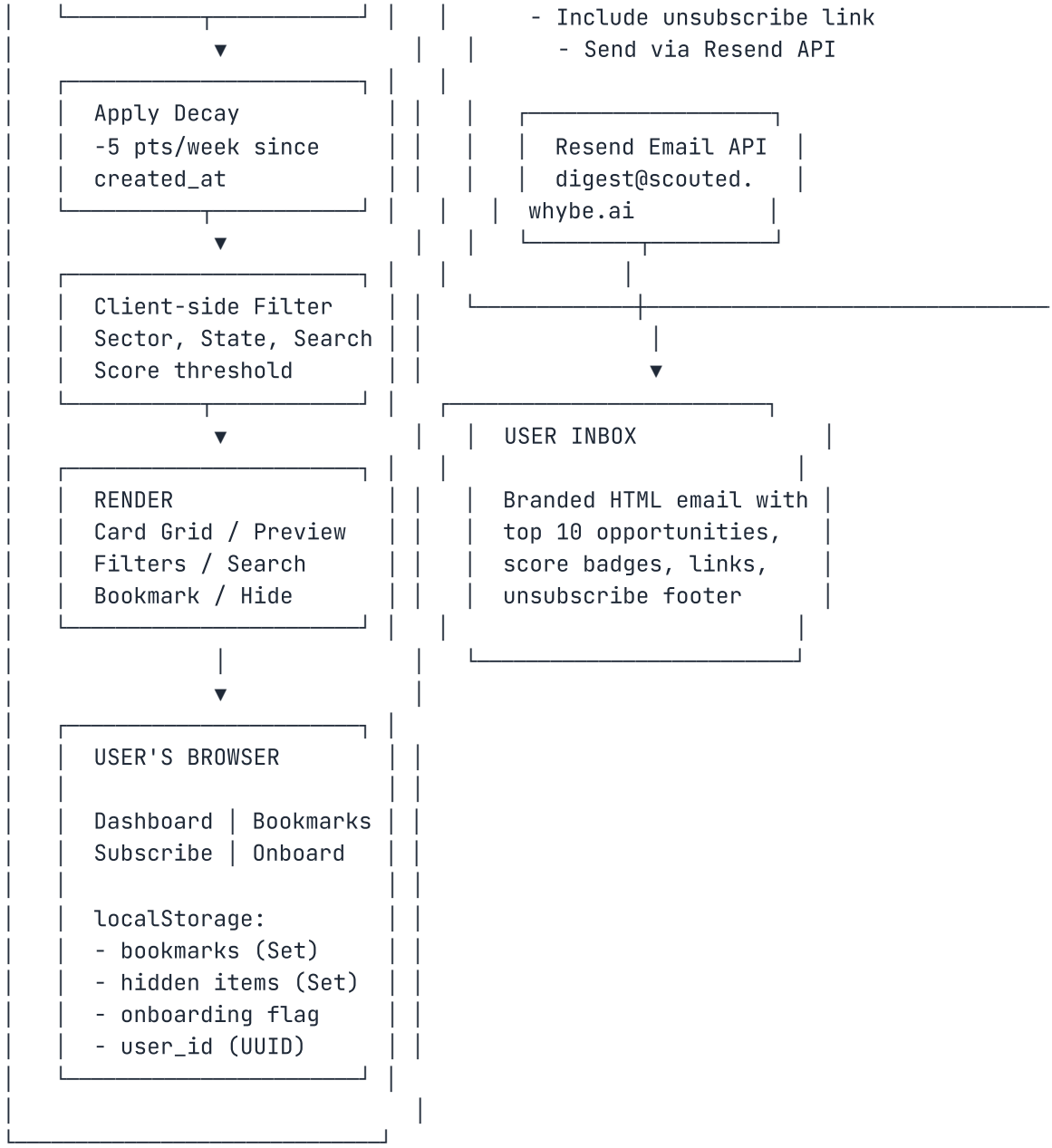
```
# Check GitHub Actions status
$ gh run list --workflow=scraper.yml --limit=5

# View logs from last scraper run
$ gh run view --log <run-id>

# Trigger scraper manually
$ gh workflow run "ScoutEd Daily Scraper"

# Check Vercel deployment status
$ vercel ls

# View Vercel function logs
$ vercel logs <deployment-url>
```

16. Appendix: Full Data Flow Diagram

This diagram shows every step of data flowing from external web sources to the user's browser and inbox, including all intermediate processing stages.


EXTERNAL DATA SOURCES
NGOBox (3 URLs) HTML
FundsFor NGOs (2) HTML
CSRBox (HTML)
IDR (RSS)
Devex (RSS)
Alliance (RSS)
Google Custom Search API
8 queries × 10 results
50-domain allowlist
OpenRouter LLM (Classification)
Gemma 27B → Llama 70B → Gemma
"K-12 education in India?"
SCRAPER (GitHub Actions, 8:00 AM IST Daily)
STEP 1: FETCH
Each parser fetches HTML / RSS / JSON using cheerio
STEP 2: EXTRACT
title, URL, desc, deadline, amount, org, location
STEP 3: ENRICH
Fetch detail pages (3 concurrent, 10s timeout)
STEP 4: FILTER
India sources → education filter
Edu sources → India filter
Reject negatives


STEP 5: SCORE
Sector +30
Geography +20
Funding +20
Donor +15
Duration +10
Max 100
STEP 6: DEDUP
Remove duplicates by source_url
STEP 7: LLM FILTER
Batch of 10 items
3s delay between
Fail-safe: accept
OpenRouter API
(free models)
STEP 8: UPSERT
Supabase (chunks of 50, service_role key, bypass RLS)
SUPABASE DATABASE (PostgreSQL + Row Level Security)
opportunities table
id | title | source_url | score | deadline | tags | org | amount |
... sorted by relevance_score DESC, indexed ...
anon SELECT
(RLS enforced)
service_role
SELECT (bypass)
REACT FRONTEND (Vercel)
React Query
(infinite pagination)
Fetch 20 per page
DAILY DIGEST (GitHub Actions, 8:30 AM IS
1. Query top 10 opps (last 48h)
2. Query all subscribers
3. For each subscriber:
- Build branded HTML email



ScoutEd

Built for the Partnerships & Strategic Initiatives Team

Central Square Foundation