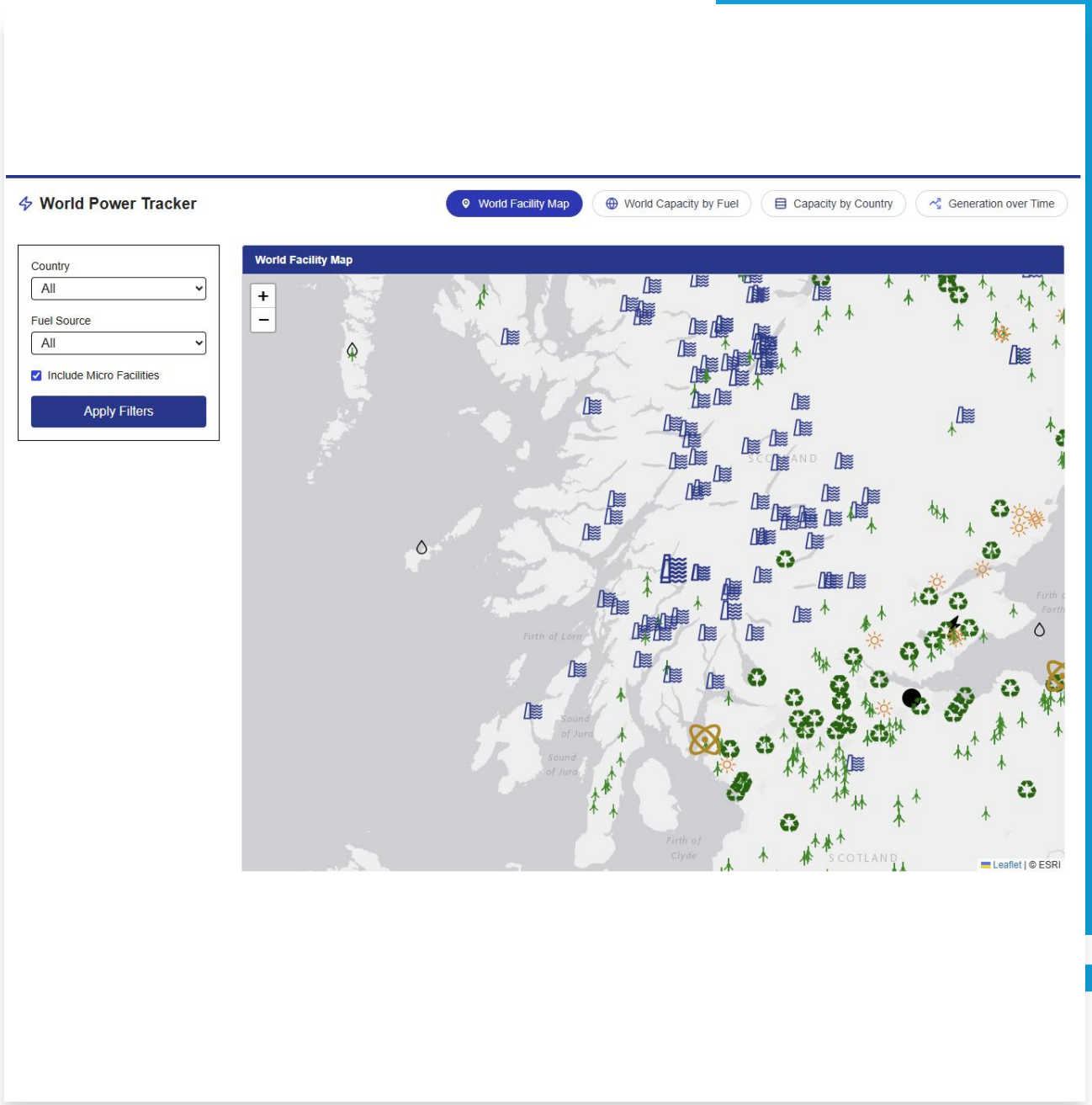
Software Engineering Coding Exercise

Ian Brown



Agenda

1	Overview of Tech Stack
2	Dataset, Database and ER Diagram
3	Building the Back-End API
4	Testing the Back End
5	Building the Front-End UI
6	Improvements

Squashing the Stack with Next.js + SQLite

Layer	Traditional Full Stack	'Squashed' Stack
Frontend	React / Angular	**Next.js** (/app/*)
Backend API	Express / Node.js	**Next.js API Routes** (/app/api/*)
Database	External DB (Postgres, MySQL)	**Embedded SQLite** (/lib/sqlite.db)

Advantages of this architecture

- **Unified Codebase** — Frontend, backend, and database logic live in one repo
- **Simplified Deployment** — Ship as a single Node.js app; no external services required
- **Embedded Database** — SQLite lives inside the app (/lib/power_generation.sqlite); no credentials or networked DB required
- **Local = Production** — What you test locally is exactly what runs in production
- **Lower Maintenance Overhead** — No need to manage separate services or sync environments, objects can be shared with backend and front end without code duplication
- **Ideal for dashboarding or static dataset apps**— Perfect for internal apps, prototypes, and lightweight analytics such as this exercise

Drawbacks

- The database will get overwritten with every new deployment of the app. Updates will be lost
- The database is in source control

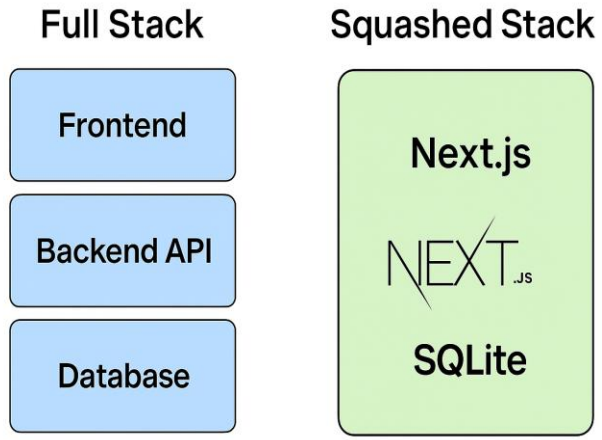


Diagram generated by AI

Transforming the Excel data into a normalized schema

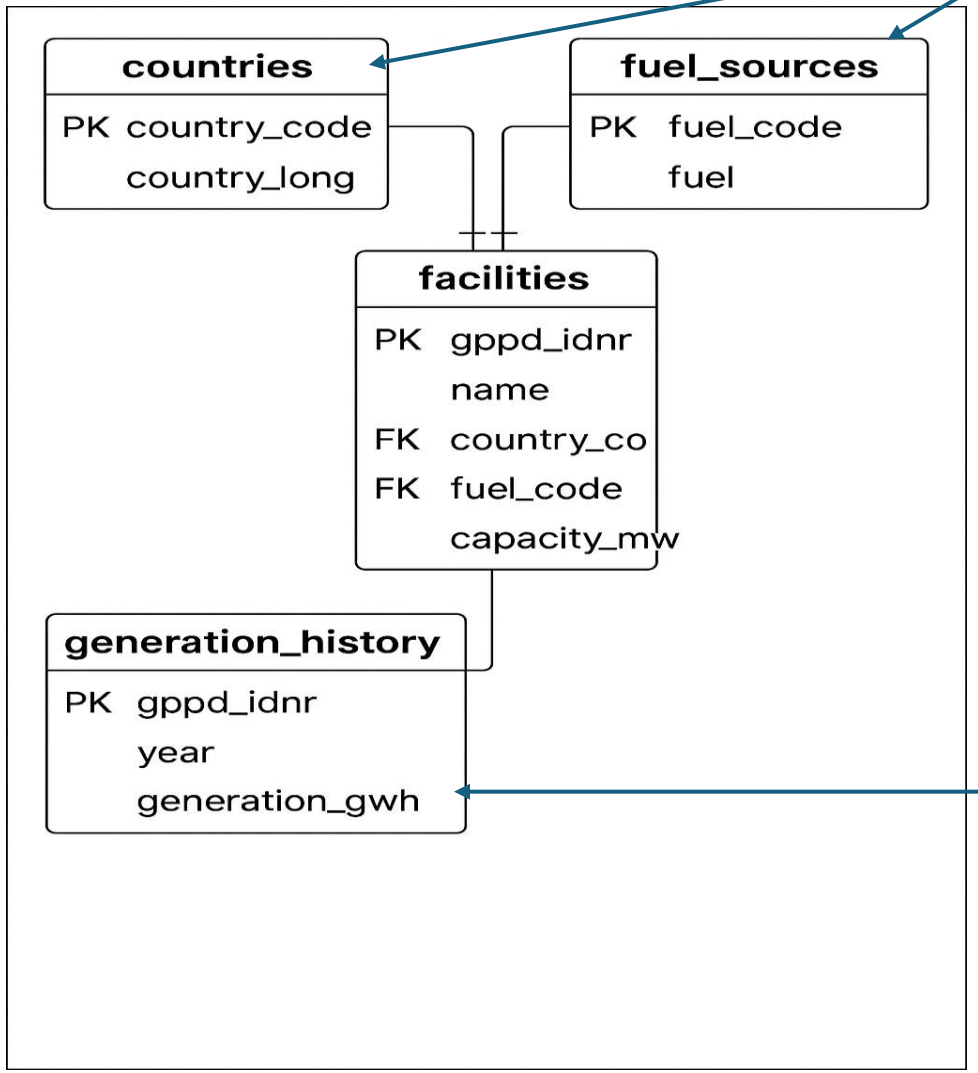


Diagram generated by AI

Used '**DISTINCT**' statements to get list of countries and fuels

Used '**COLEASE**' statements to replace NULL generation numbers with 0s

Used **UNION** to union the _2013, _2014… columns from wide history format into narrow format

Used **CREATE TABLE** with nested UNION within a **correlated sub-query** to create a new table called generation_history

```
CREATE TABLE generation_history AS
SELECT gppd_idnr, '2013' AS year, generation_gwh_2013 AS value FROM facilities
UNION ALL
SELECT gppd_idnr, '2014', generation_gwh_2014 FROM facilities
UNION ALL
SELECT gppd_idnr, '2015', generation_gwh_2015 FROM facilities
UNION ALL
SELECT gppd_idnr, '2016', generation_gwh_2016 FROM facilities
UNION ALL
SELECT gppd_idnr, '2017', generation_gwh_2017 FROM facilities
UNION ALL
SELECT gppd_idnr, '2018', generation_gwh_2018 FROM facilities
UNION ALL
SELECT gppd_idnr, '2019', generation_gwh_2019 FROM facilities;
```

Using views for Denormalization

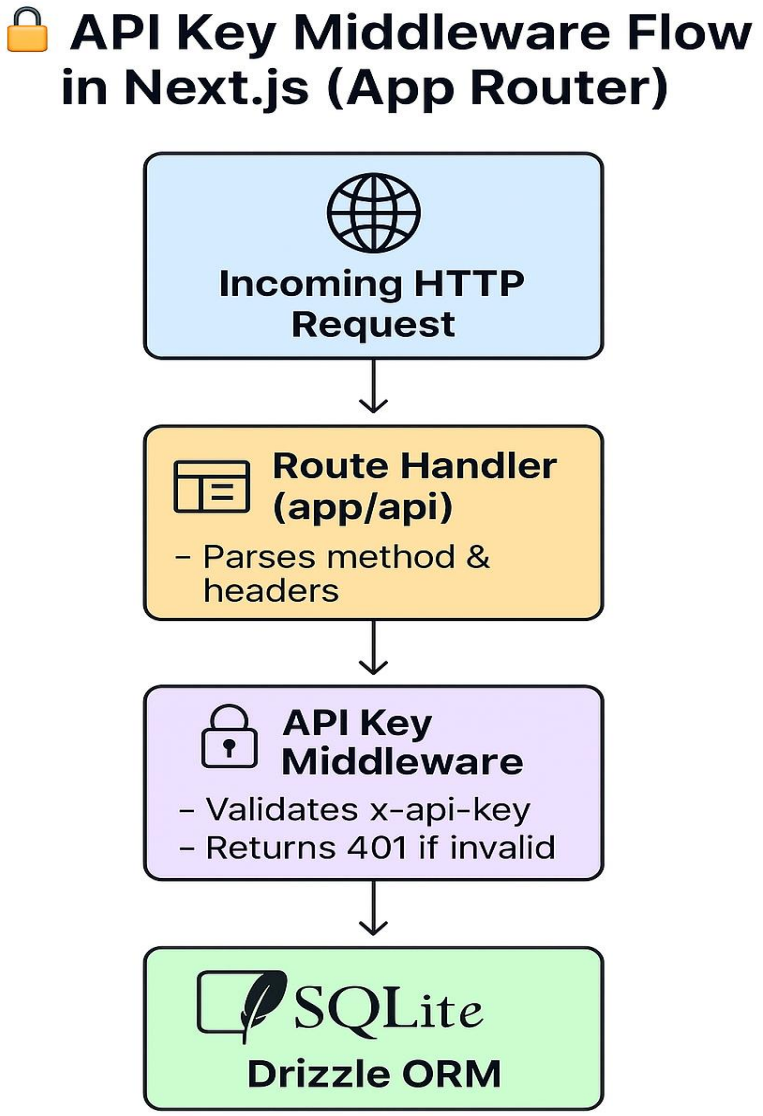
	gppd_idnr	latitude	longitude	name	capacity_mw	owner	of_capacity	fuel_code	fuel	country_code	country_long
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
25714	USA0001482	43.6877	-70.6116	Bonny Eagle	7.2	Brookfield White Pine Hyd...	2019	1	Hydro	USA	United States of ...
25715	USA0060413	33.053	-111.34	Bonnybrooke PV	50	Apple Inc	2019	2	Solar	USA	United States of ...
25716	USA0003000	36.1438	-97.0681	Boomer Lake Station	6	Stillwater Utilities Authority	2019	5	Oil	USA	United States of ...
25717	USA0003395	36.4403	-82.4381	Boone Dam	105.9	Tennessee Valley Authority	2019	1	Hydro	USA	United States of ...
25718	USA0010556	42.6525	-71.3224	Boott Hydropower	22.9	Boott Hydropower Inc	2019	1	Hydro	USA	United States of ...
25719	USA0060734	40.0389	-74.7766	Bordentown Solar	6.8	Marina Energy LLC	2019	2	Solar	USA	United States of ...
25720	USA0059200	48.9647	-99.6139	Border Winds Wind Farm	150	Northern States Power C...	2019	6	Wind	USA	United States of ...
25721	USA0000328	35.5881	-118.5248	Borel	12	Southern California Ediso...	2019	1	Hydro	USA	United States of ...
25722	USA0050067	35.6647	-101.4354	Borger Plant	37.5	Sid Richardson Carbon Ltd	2019	3	Gas	USA	United States of ...
25723	USA0062169	45.431	-122.285	Boring Solar LLC	2.2	Boring Solar LLC	2019	2	Solar	USA	United States of ...
25724	USA0060565	33.2701	-116.3496	Borrego Springs Energy Storage	1.5	SDGE Batteries	2019	14	Storage	USA	United States of ...
25725	USA0057625	41.1153	-87.2328	Bos Dairy LLC	1.1	Bos Dairy LLC	2019	3	Gas	USA	United States of ...
25726	USA0058807	36.0081	-77.81	Boseman Solar Center LLC	5	Cypress Creek Renewables	2019	2	Solar	USA	United States of ...
25727	USA0055172	31.8594	-97.3586	Bosque County Peaking	807	Calpine Bosque Energy ...	2019	3	Gas	USA	United States of ...
25728	USA0056547	35.3619	-81.8344	Bostic Delivery No 2	1	North Carolina Mun Powe...	2019	5	Oil	USA	United States of ...
25729	USA0061186	42.3347	-71.0742	Boston Medical Center CHP Plant	2	Boston Medical Center	2019	3	Gas	USA	United States of ...
25730	USA0057992	42.2933	-71.0339	Boston Scientific Solar	1.1	Conso					
25731	USA0000902	38.8348	-122.7677	Bottle Rock Power	55	Bottle					

```
CREATE VIEW vw_facilities AS
SELECT
  f.gppd_idnr, latitude, longitude, name, capacity_mw, owner, year_of_capacity_data
  fs.fuel_code, fs.fuel,
  c.country_code, c.country_long
FROM facilities f
LEFT JOIN fuel_sources fs ON f.fuel_code = fs.fuel_code
LEFT JOIN countries c ON f.country_code = c.country_code
```

Used **CREATE VIEW AND LEFT JOIN** commands to create a de-normalized view

Building out the back end

Component	Purpose
Typescript	Verb based RESTful API Routing
Typescript	API Key authentication ‘middleware’ to validate ‘x-api-key’ in header of request
HTTP Status codes	To standardize Reponses from API Routing - Unauthorized (401), Forbidden (403), Error (500) and OK (200)
Drizzle ORM	Object-Relational-Mapping to simplify database query management
SQLite	The database where data is stored
JEST	Run integration tests on API routes



Use of an ORM in this application

Discussion: Object Relational Mapping tools map SQL queries to code-based objects. Thus, reducing the need to write SQL code or code to map raw data into list of objects. ORM's can be very helpful to speed up development and maintain clean code. However,... They generally become more of a hinderance and a hack if you are building analytical applications where complex queries are being used to generate datasets that require multiple joins or aggregations – this is because there is no direct Object-to-database relation mapping!, although most ORMS support raw SQL and the ability to create custom objects to map to the SQL query

Advantages of using an ORM in this app

- The ORM generates objects (schemas) which can be reused across the database, API and front-end UI to ensure consistency and type safety. This is a great time saver and code reducer

- I could swap out SQLite for Postgres, MySQL and this would work without changing any code other than the connection string.

Disadvantages of using an ORM in this app

- ORM specific queries took me longer to learn than using native SQL. I could have used the ORM with native SQL instead of the ORM query language, but I chose to honor using the full ORM features.

Building the API route and API Security middleware

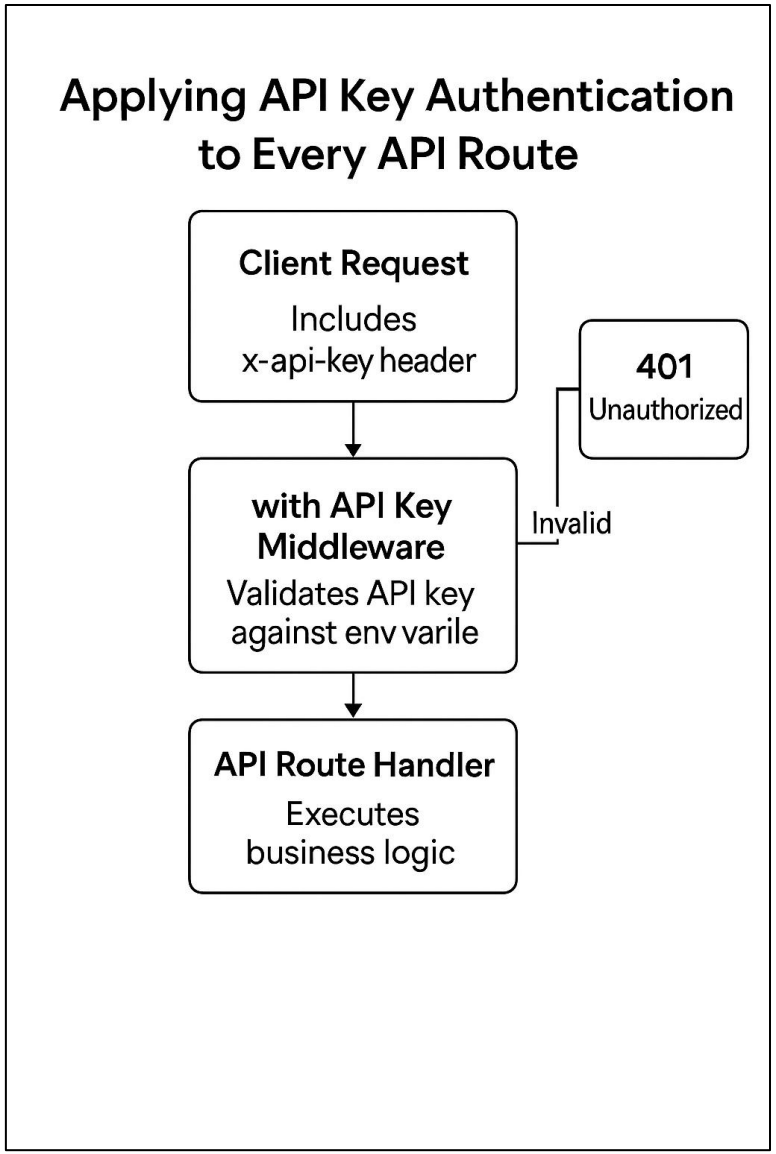


Diagram generated by AI

Route.ts

```ts
import { NextResponse } from 'next/server';
import { getCountryCapacity } from '@/lib/dbQueries';
import { withApiKey } from '@/app/api/api-authenticator';

/**
 * API route handler for fetching country-level power capacity d
 * Protected by API key middleware via `withApiKey`.
 *
 * Optional query parameter: `fuel=1`
 * Returns an array of capacity data per country.
 */
export const GET = withApiKey(async (req: Request) => {
  try {
    const { searchParams } = new URL(req.url);
    const fuelParam = searchParams.get('fuel');

    // Convert fuel param to number if present
    const fuelCode = fuelParam !== null ? parseInt(fuelParam) :

    // If fuel is not a valid number, treat as null
    const fuel = Number.isNaN(fuelCode) ? null : fuelCode;

    // Get data from the database
    let data;
    try {
      data = await getCountryCapacity(fuel);
    } catch (dbError: any) {
      // Handle database errors gracefully
      return NextResponse.json(
        {
          error: 'Failed to fetch country capacity data',
          details: dbError?.message ?? 'database error',
        },
        { status: 500 }
      );
    }
```

Api-Authenticator.ts

```ts
app > api > TS api-authenticator.ts > ...
import { NextResponse } from 'next/server';
type RouteHandler = (req: Request) => Promise<Response>;

/**
 * Middleware that enforces API key validation before executing the route handler.
 * Applies to App Router-style handlers using the Web API `Request` object.
 */
export const withApiKey = (handler: RouteHandler): RouteHandler => {
  return async (req: Request) => {
    // Reject request if API key is missing or invalid. 403 Forbidden.
    if (!validateApiKey(req)) {
      return NextResponse.json(
        { error: 'Forbidden: Invalid API Key' },
        { status: 403 }
      );
    }
    // Proceed to the original handler if API key is valid
    return handler(req);
  };
};


export const validateApiKey = (req: Request): boolean => {
  // Key from environment variables
  const validKey = process.env.NEXT_PUBLIC_REST_API_KEY
  // Key from request headers
  const incomingKey = req.headers.get('x-api-key');
  // Check if the incoming key matches the valid key
  return typeof incomingKey === 'string' && incomingKey === validKey;
};
```

Testing the Backend – Jest Testing Framework

Discussion: Unit testing occurs at the lowest unit of work (e.g. a function, a database call etc). Integration testing occurs at a level which achieves an outcome (e.g. A web API call which validates a key and fetches data from the database). Evangelical testing would require tests for each unit of work and for each integration. In the full-stack this rarely adds value as more code is written to test the app than to build the app. Finding the right level of testing to ensure quality should always generally the goal

Jest Testing framework

- Supports unit and integration testing
- Runs test in parallel
- Easy config
- Very popular and widely used

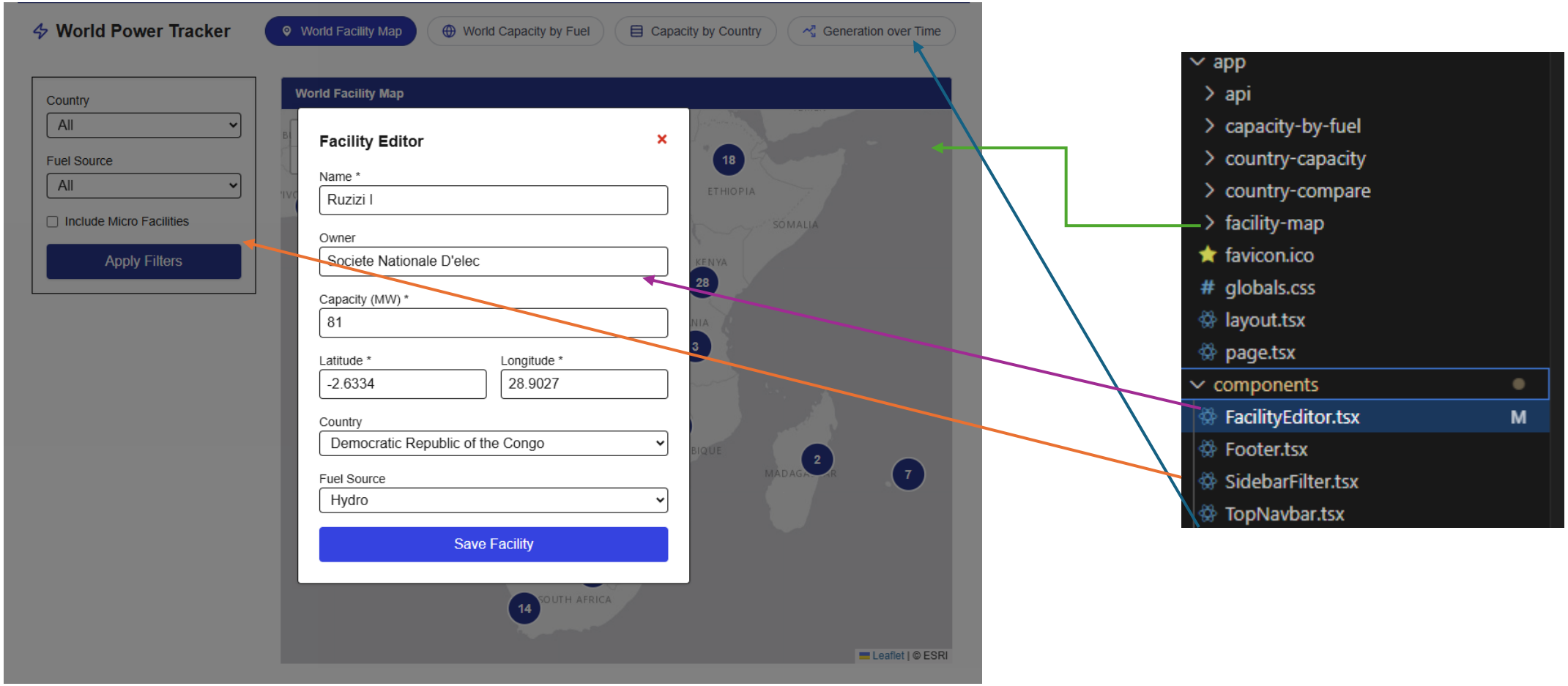
I created Jest **integration tests for each API** in the application

- This ensured that the database queries work
- That the API was secure
- That the API error handling was working
- That the API returned valid data
- That the API returned data in the correct data format

The front end – UI Scaffolding

Component	Purpose	Tool/Library Used
Routing & Pages	Navigation and page structure	Next.js using App Router
UI Framework	Layout, typography, spacing	Tailwind CSS
Map Component	Visualize geographic data	react-leaflet + leaflet
Chart Component	Display time-series or categorical data	react-chartjs-2 + chart.js
Country Filter	Select and filter by country	react-select
Input validation	Inline form validation	Zod

Next JS app. Pages, Components etc.



Improvements to the solution

Component	Immediate action	Improvement	Enterprise Application
Security	Use SSL on App and API		Use enterprise SSO
RESTful API	Remove API key from source code.	Remove key based security. Whitelist IP or domain to access the API	Use enterprise SSO
RESTful API	Remove REST entirely. Replace with tRPC		
Database	Remove SQLite Move to MySql	Whitelist database access to API only. Use SSL to access database.	Set granular read/write permissions specifically for API
ORM	Use SQL Queries over ORM query language.	Defend against SQL injection	Possibly drop use of the ORM
UI	Improve mobile accessibility	Swap tailwind for a more component driven UI framework (Fluent etc.)	Adopt components from a corporate design system
UI	Caching of filters on app load		
Containerization	Drop containerization and publish direct to cloud hosted app service		Enterprise security configured on app services