Name: Bhagyashree R
University: SSMRV College
University Registration Number: U18GO22S0050
Degree Program: Bachelor’s in Computer Science
Internship Role: AI/ML Intern
Organization: Rubixe
Internship Duration: 24-03-2025 to 23-04-2025
Project Name: Country Data Aggregator and Insights Dashboard

Problem Statement

- The Challenge:

In today's data-driven world, reliable country-level information is crucial for research, policy-making, and education. However, such data is often scattered across various APIs and platforms, each offering partial or unstructured insights. This fragmentation makes it time-consuming and inefficient for users to gather and analyze meaningful information.

- The Objective:

To address this, the project aims to develop a centralized web dashboard that aggregates country-specific data from trusted sources, processes it for consistency, stores it in a structured format, and makes it easily accessible through a user-friendly interface.

- Real-World Application:

This dashboard can serve as a valuable tool for:

Analysts, looking to compare demographic and geographic statistics.
Educators and Students, needing real-time data for academic projects.
Researchers, exploring regional trends and development metrics.

Project Overview

The Country Data Aggregator and Insights Dashboard is a full-stack web application designed to simplify access to structured, real-time data about countries across the world. This project integrates multiple components — from backend APIs to frontend interfaces — to deliver a seamless and insightful user experience.

🔹 What is a Full-Stack Dashboard?

A full-stack application includes both:

- A backend (server-side) that handles business logic, data processing, storage, and communication with external APIs.
- A frontend (client-side) that presents this processed data to users in an interactive and visually appealing way.

This ensures a complete solution — from raw data retrieval to end-user visualization — within a single unified application.

Backend Responsibilities

The backend of this project is built using Python and Flask and performs the following key functions:

- Data Fetching:
  Connects to the REST Countries API to retrieve up-to-date data about countries.
  Extracts relevant fields such as country name, capital, population, area, and region.
- Data Cleaning & Processing:
  Cleans the data to remove inconsistencies or null values.
  Formats it for uniformity and readability using Pandas.
- Database Integration:
  Stores the cleaned data in a local SQLite database.
  Utilizes SQLAlchemy ORM for interacting with the database seamlessly.
- RESTful API Exposure:
  Creates API endpoints such as /countries and /countries/<name> using Flask Blueprints, allowing the frontend or other applications to access country data in real-time.
- Error Handling & Logging:
  Implements mechanisms to catch and log errors (e.g., API downtime, invalid input) to improve reliability and debuggability.

◆ Frontend Responsibilities

The frontend is developed using React.js, which provides a responsive and modular architecture for creating dynamic user interfaces.

- User Interface:

Built with Tailwind CSS, ensuring a modern, clean, and mobile-responsive design.

- Core Features:

Cards: Displays metrics like total countries, global population, and average area.

Search & Filter: Allows users to instantly filter countries by name, region, or size.

Charts & Graphs: Integrates Recharts to visualize country populations and regional distributions with interactive bar and pie charts.

- API Integration:

Uses Axios to fetch real-time data from the Flask backend and dynamically update the interface based on user actions (e.g., typing in the search box or clicking filters).

Deployment & Scalability

The entire project is containerized using Docker, which packages both backend and frontend components into isolated environments:

Simplifies the deployment process across various machines.

Makes the application portable and ready for production use.

Utilizes docker-compose to run the frontend and backend as services with a single command.

This ensures a developer-friendly and scalable deployment workflow.

Summary of Technology Stack

- Layer Technologies Used

Backend Python, Flask, SQLAlchemy, SQLite, Pandas

Frontend React.js, Tailwind CSS, Recharts, Axios

Deployment Docker, Docker Compose

Dev Tools Git, GitHub, VS Code

Backend – Folder Structure

main.py: Starts Flask server.

app.py: App factory setup.

routes/: REST API routes (countries.py).

services/: Logic for fetching, processing, and storing data.

database.py: SQLAlchemy setup for SQLite.

models.py: Country model class.

utils.py: Utility functions (e.g., data cleaning).

API Development

Flask Application:

Created RESTful endpoints to serve processed data.

Implemented routes for retrieving all countries and specific country details.

Error Handling:

Managed exceptions and provided meaningful error messages.

Ensured API robustness and reliability.

Frontend – Tech Stack

React.js for component-based UI.

Tailwind CSS for responsive design.

Recharts for dynamic graphs (bar chart, pie chart).

FRONTEND-Dashboard Features

Search Functionality: Enabled users to search for specific countries.
Filtering Options: Allowed filtering based on regions or other criteria.
Responsive Design: Ensured compatibility across devices and screen sizes.
Real-time Updates: Reflected the latest data fetched from the API.

Frontend – Code Overview

- App.jsx: Main container, routes, layout.
- components/:
  - Card.jsx, SearchBar.jsx, CountryTable.jsx, PieChart.jsx, etc.
- Data fetched via Axios from backend API.
- State management via React useState and useEffect

Database Schema

SQLite with a single table: Country

id (PK), name, capital, region, population, area

ORM mapping with SQLAlchemy (models.py)

Docker & Deployment

- Dockerfile:
  - Backend: Python image with all dependencies from requirements.txt
  - Frontend: Node image to serve React app
- docker-compose.yml:
  - Manages both containers with shared network
- Deployment:
  - Local: docker-compose up
  - Future: Ready for deployment on AWS/GCP/DigitalOcean

Key Challenges

API inconsistencies (missing data fields)

State management in dynamic filters

Cross-Origin issues during integration

Data volume optimization (filter/sort without reloading)

Solutions Implemented

- Validation logic for null/missing fields.
- Custom React hooks for filtering.
- Used CORS headers in Flask to allow frontend requests.
- Lazy loading and map-based data render logic.

Security & Optimization

Data sanitization before DB insert.

Error handling in API endpoints.

Logging added for API errors and exceptions.

Minimized bundle size in frontend for performance.

Future Scope

- Add user authentication (JWT).
- Cloud deployment with CI/CD (e.g., GitHub Actions).
- Export data as CSV/Excel.
- Add more datasets (e.g., GDP, languages, currencies).

Real-World Applications

- Used by data journalists for statistical summaries.
- Educational dashboards for geography and economics classes.
- NGO or policy dashboards for development tracking.

Results & Impact

- Successfully built and deployed full-stack application.
- Clean, responsive UI with real-time data.
- 100+ countries aggregated and visualized.

Learnings & Reflections

- Gained hands-on experience in:
  - API development and testing
  - React component structuring
  - Deployment using Docker
- Improved skills in debugging, REST design, UI planning

Conclusion & Acknowledgment

- Thanks to the Rubixe mentors for guidance.
- Thanks to college faculty for support.
- Proud to have built a complete, real-world project from scratch!

Thank You