Sri Lanka Institute of Information Technology

Assignment 2

Application Frameworks

Date of submission: 2025/05/01

Submitted By: IT22056870 – Prasad H G A T

Information Technology
B.Sc. (Hons) in Information Technology

Table of Contents

Introduction 2
Technology Stack 3
Features Implemented 4
API Endpoints Used 4
GET /all 4
GET /name/{name}?fullText=true 4
GET /region/{region} 5
GET /currency/{currencyCode} 5
Application Design 6
Testing 12
Approach: 12
Tools and Technologies: 12
Test Report 13
Challenges Faced & Solutions 13
Complex and Inconsistent API Data Structure 13
Regional Filtering / Search Not Updating State Properly 14
UI Design Conflicts Between Tailwind CSS and Ant Design 15
Conclusion 15

Introduction

In today's world, access to comprehensive and up to date information about countries is vital for education, research and travel planning. The **NationScope** application was developed as a React-Based web application that presents users with structured and visually appealing data about countries across the globe by consuming and displaying information from the Rest Countries API.

The primary objective of this project is to apply modern frontend development techniques using React functional components and to gain practical experience in integrating RESTful APIs, managing application state and building responsive, user-friendly interfaces. Through Nationscope, users can search for countries by name, filter by region or language and view detailed information about each country such as its capital, population, flag, region and spoken languages.

This application serves as both a technical exercise in applying concepts of application frameworks and as a demonstration of best practices in frontend development, including component-based architecture, responsive design, asynchronous data handling, and user session management.

Scope of the Application

Nationscope is designed to:

- Dynamically fetch and render real-time country data from REST Countries API endpoints.
- Allow each user to search specific countries, filter them by region or language and explore detailed views of each country.
- Favorite and save their favorite countries and search them as needed from the Favorite library.

By developing Nationscope, this project also aims to reflect real-world application development practices, such as using version control with GitHub, performing meaningful unit and integration testing, and maintaining comprehensive documentation.

GitHub Repository: https://github.com/SE1020-IT2070-OOP-DSA-25/af-2-devkyoshi

Vercel Deployment: https://af-project-deploy.vercel.app/

Technology Stack

The Nationscope application was developed using a modern web development stack that prioritizes performance, scalability, and ease of development. The technologies were carefully selected to meet the assignment objectives and ensure a smooth user experience. The following tools and frameworks were utilized:

1. **Frontend Framework**

   - **React (with Vite):** The core of the application is built using React with the Vite build tool for faster development and optimized builds. React's functional components and hooks were used for modular design and efficient state management.

2. **Styling and UI Frameworks**

   - **Tailwind CSS:** A utility-first CSS framework used to build a highly responsive and customizable user interface quickly and consistently across the application.
   - **Ant Design:** A professional React UI library used to enhance the UI components such as dropdowns, modals, input fields, buttons, and tables—providing a polished, production-ready look and feel.

3. **Authentication and Backend Services**

   - **Firebase Authentication:** Firebase was used to implement secure user authentication, allowing users to sign up, log in, and manage sessions effectively.
   - **Firebase Realtime Database / Firestore:** additional user-specific data (e.g., favorites) were stored, Firebase's database services were used to handle this backend functionality.

4. **Hosting and Deployment**

   - **Vercel:** The application is deployed on Vercel, a reliable and free hosting platform that supports continuous deployment directly from the GitHub repository.

5. **Version Control**

   - **Git & GitHub:** Version control was maintained using Git, with the source code hosted on GitHub. Regular and descriptive commits were made throughout the development process to ensure traceability and maintain a clean codebase.

Features Implemented

- Display of country data including Name, Capital, Region, Population, Flag, and Languages.
- Search functionality (by country name).
- Region filter (via dropdown).
- Language filter
- Dynamic updates (no page reload).
- Country detail view on clicks
- User login and favorite countries

API Endpoints Used

The Nationscope application integrates deeply with the REST Countries API to retrieve and display comprehensive data about countries worldwide. The integration focuses on making dynamic, real-time requests using axios, with responses handled efficiently to populate and filter country data in the UI.

GET /all

Purpose:
Fetches a summary list of all countries.

Fields Extracted:

Country Name, Capital, Population, Currencies, Flag, Google/OpenStreet Maps links

Usage in Nationscope:
This data populates the initial country list and is used to render the main overview interface.

GET /name/{name}?fullText=true

Purpose:
Fetches detailed information about a specific country using its common name.

Fields Extracted:

Official and native names, Country codes (CCA2, CCA3, CIOC), Flags and Coat of Arms, Capital, Region, Subregion, Area, Population, Coordinates, Languages and Currencies, Calling code, Timezones, Borders, Postal codes and Car info, Google Maps and OpenStreetMap links, Gini index, Demonyms, Start of week

Usage in Nationscope:

This endpoint powers the country detail view page/modal, offering users a complete breakdown of a selected country.

GET /region/{region}

Purpose:
Retrieves countries filtered by a specific region (e.g., Asia, Europe).

Fields Extracted:

Name, Capital, Population, Currencies, Flag, Map links

Usage in Nationscope:
Enables regional filtering through dropdown or button selection. The view updates dynamically based on the selected region.

GET /currency/{currencyCode}

Purpose:
Retrieves countries filtered by a specific currency Code (e.g., LKR, USD).

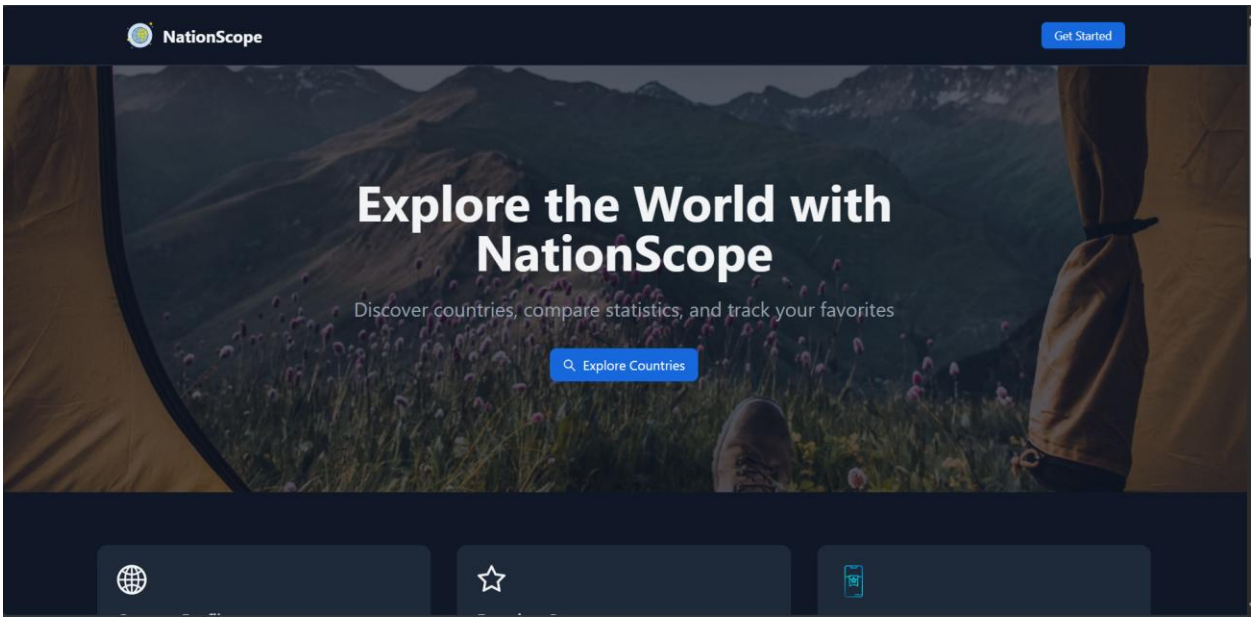
Fields Extracted:

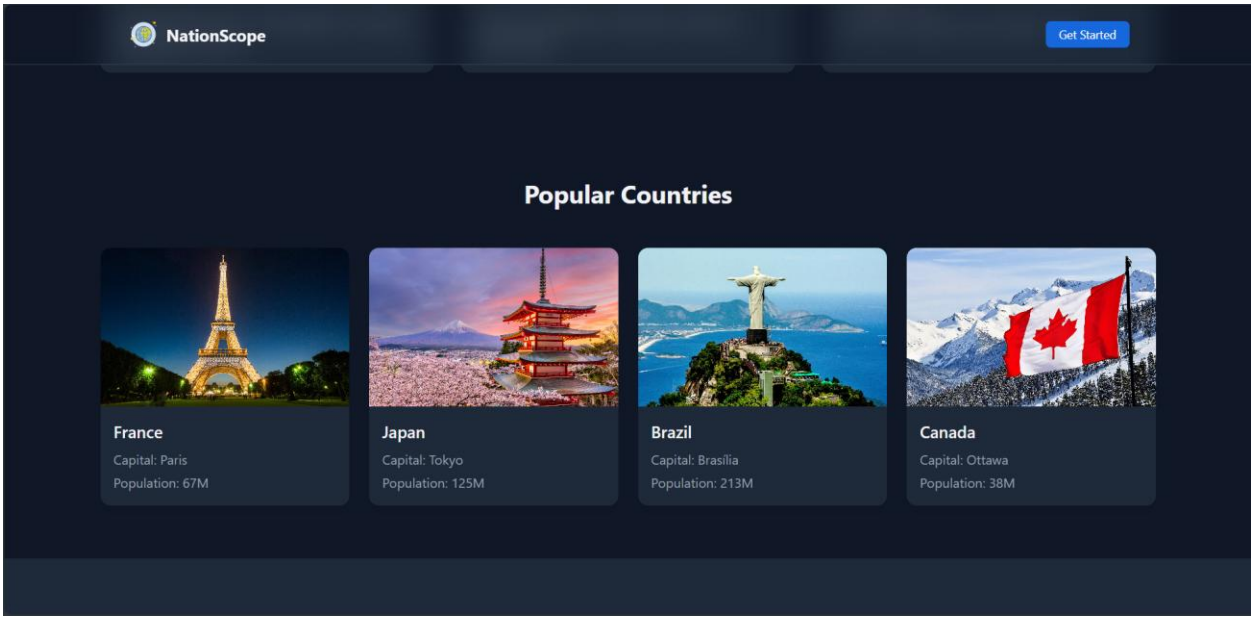
Name, Capital, Population, Currencies, Flag, Map links

Usage in Nationscope:
Enables currency filtering through search. The view updates dynamically based on the selected currency

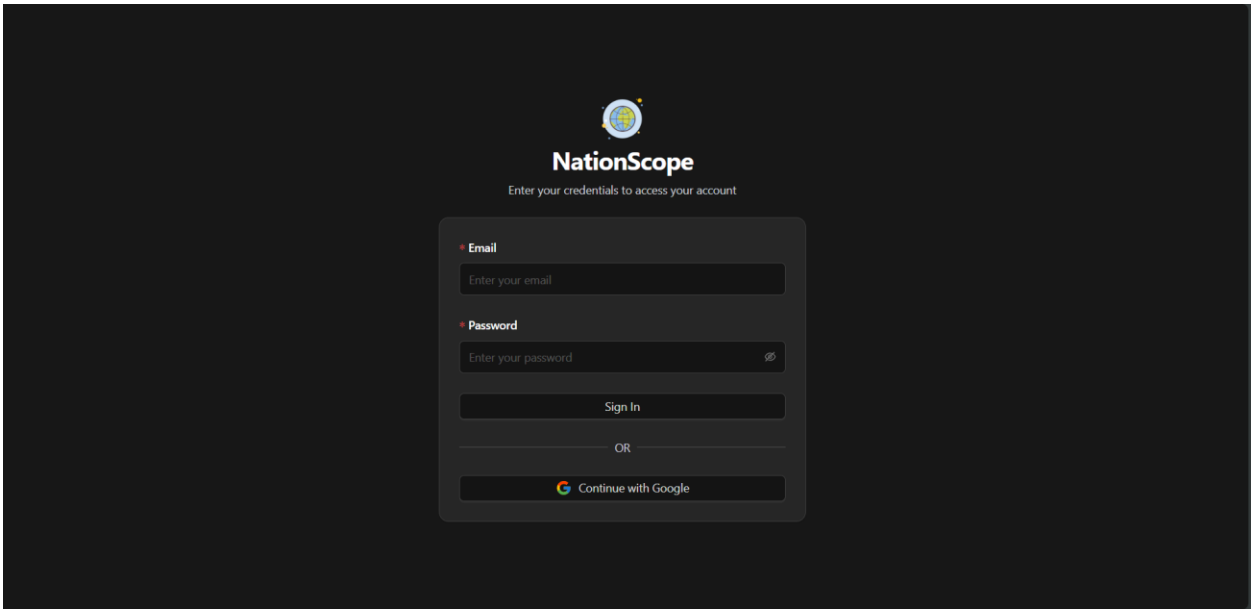
Application Design

Landing Page:

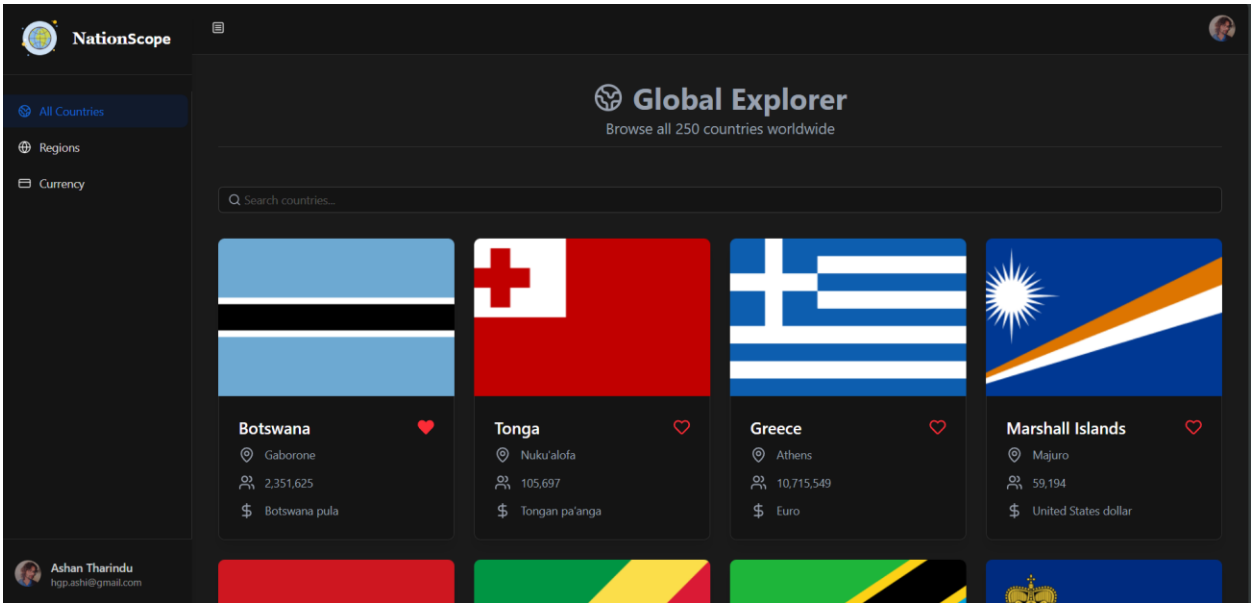




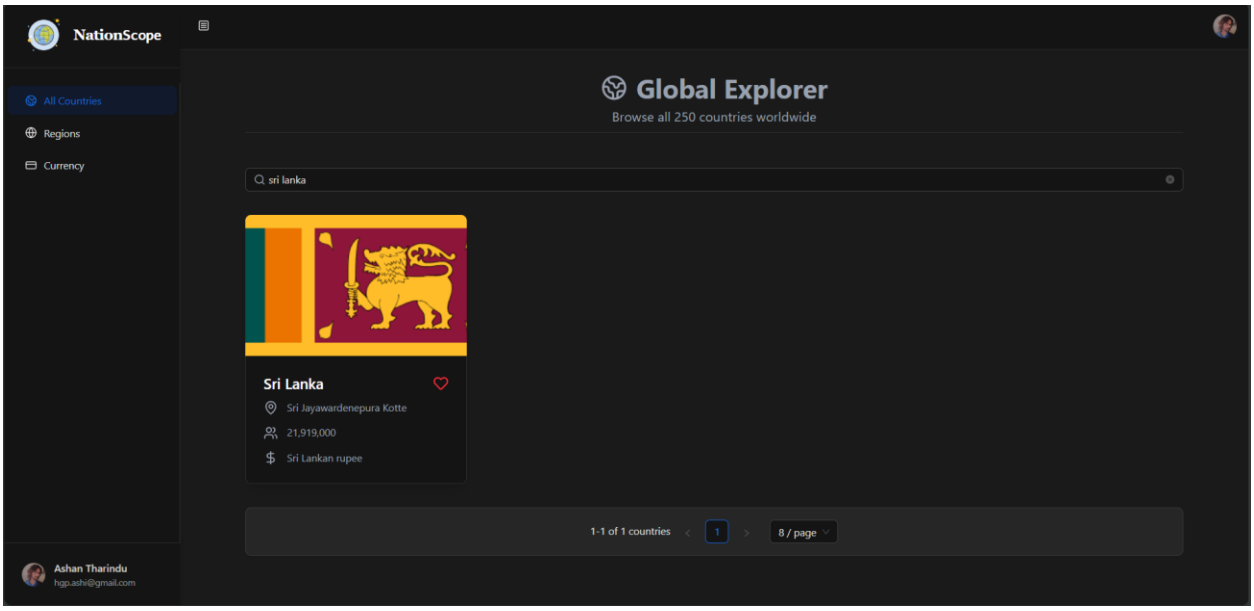
Login Page



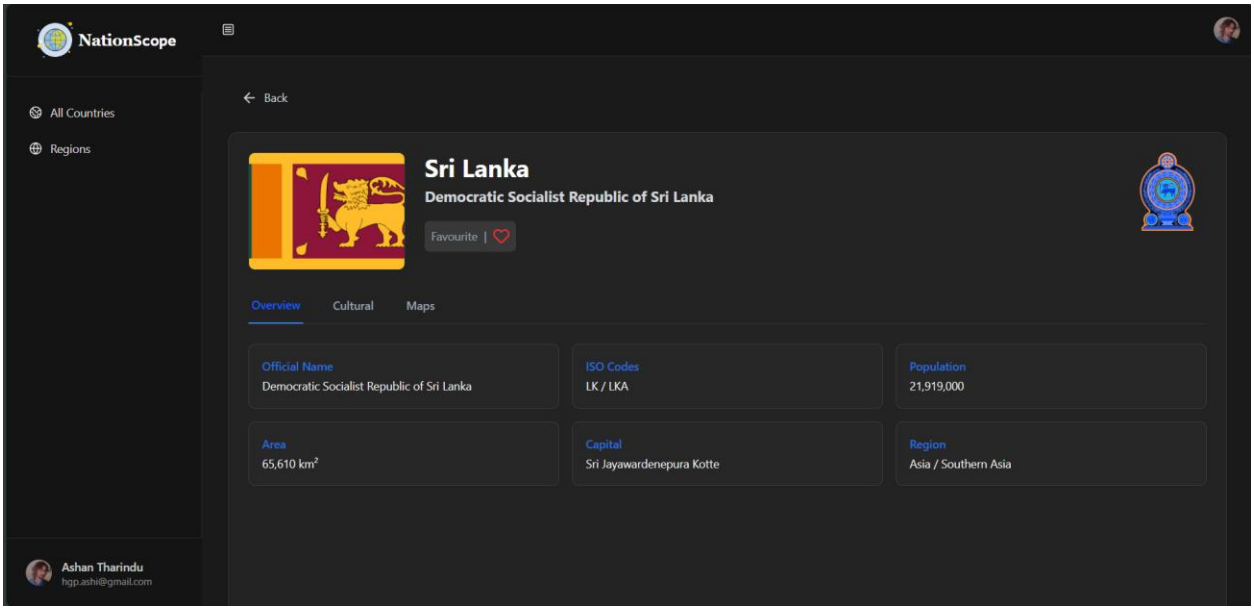
All Countries

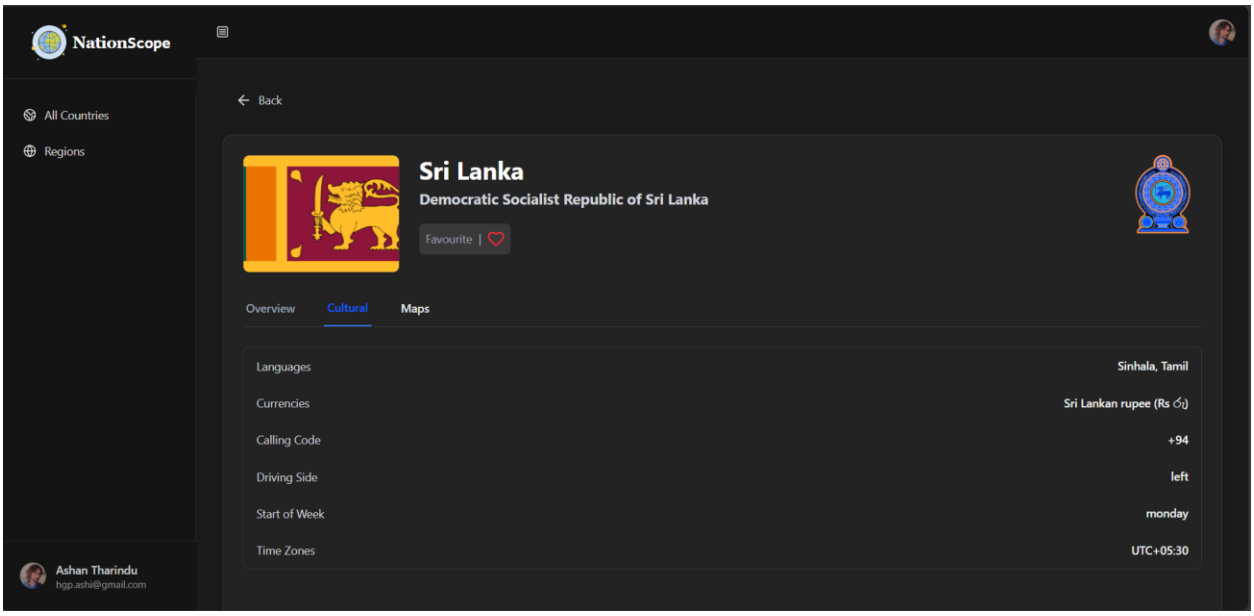


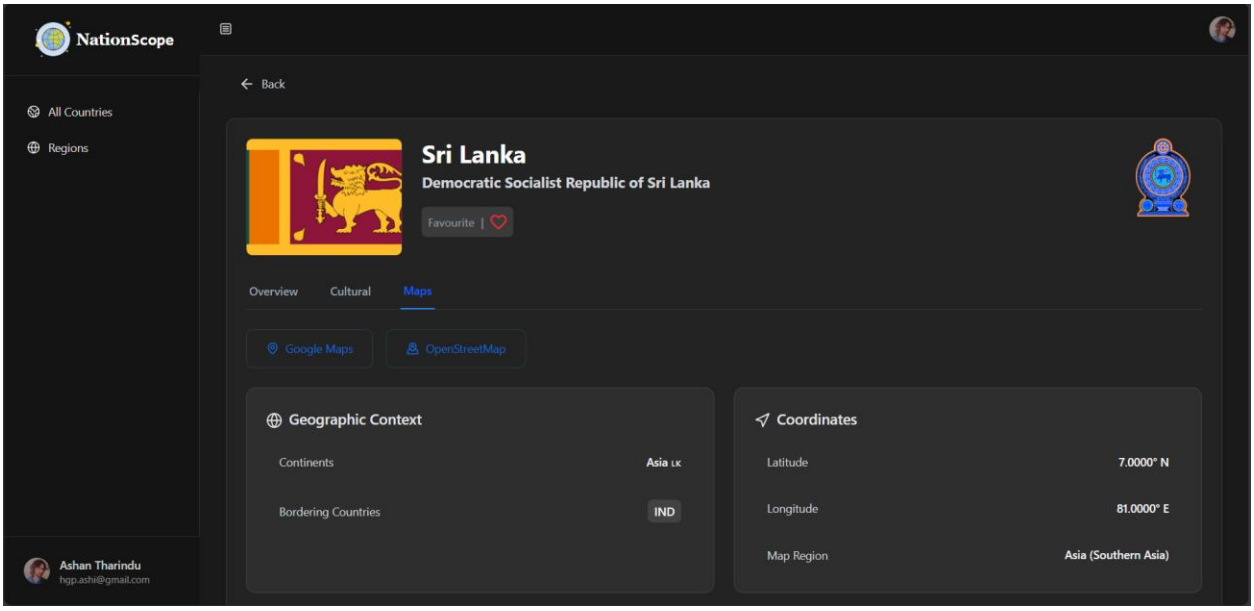
Search Country by name



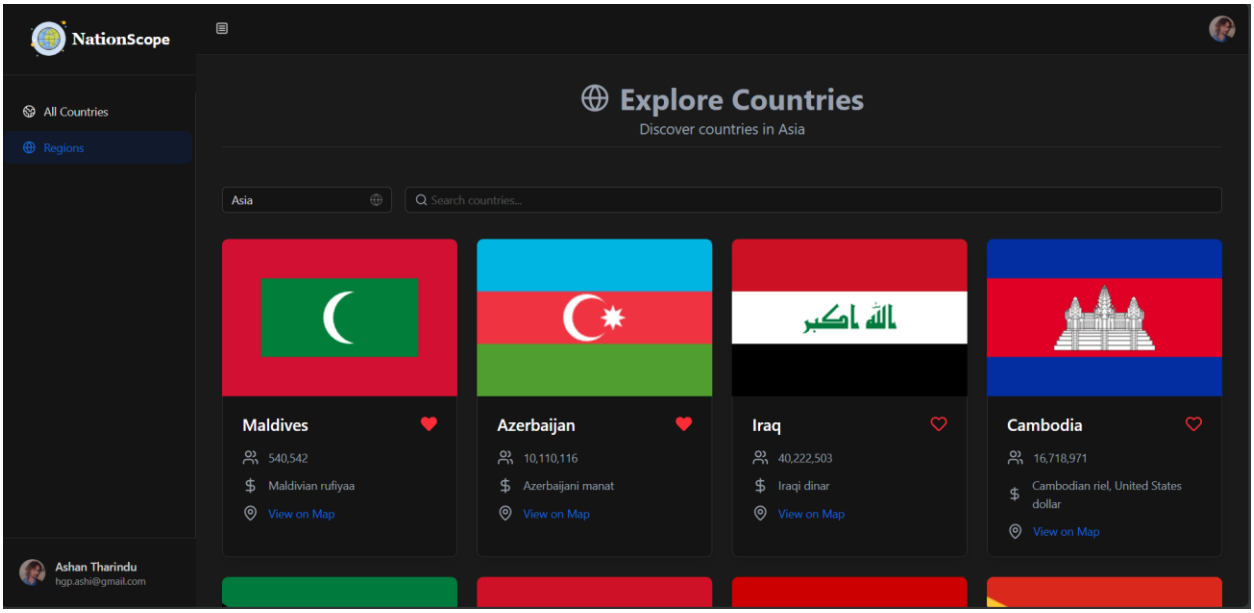
Details of a Specific Country



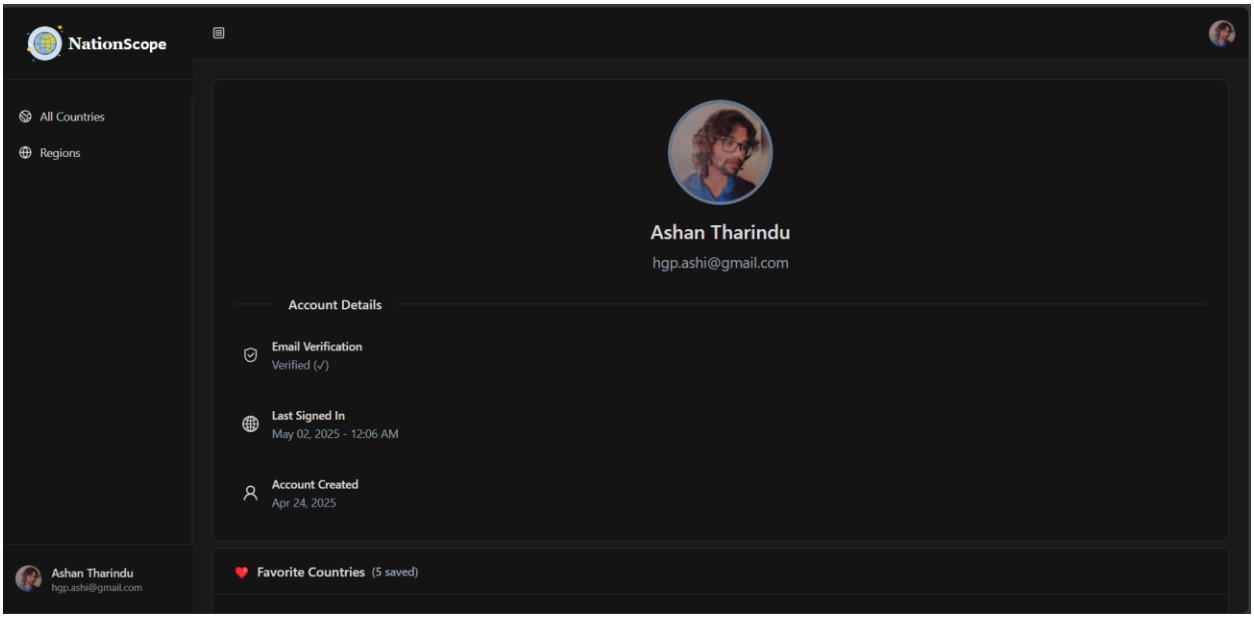

NationScope
All Countries
Regions
Back
Sri Lanka
Democratic Socialist Republic of Sri Lanka
Favourite
Overview
Cultural
Maps
Languages
Sinhala, Tamil
Currencies
Sri Lankan rupee (Rs ඊ)
Calling Code
+94
Driving Side
left
Start of Week
monday
Time Zones
UTC+05:30
Ashan Tharindu
hgp.ashi@gmail.com


NationScope
All Countries
Regions
Back
Sri Lanka
Democratic Socialist Republic of Sri Lanka
Favourite
Overview
Cultural
Maps
Google Maps
OpenStreetMap
Geographic Context
Continents
Asia
Bordering Countries
IND
Coordinates
Latitude
7.0000° N
Longitude
81.0000° E
Map Region
Asia (Southern Asia)
Ashan Tharindu
hgp.ashi@gmail.com

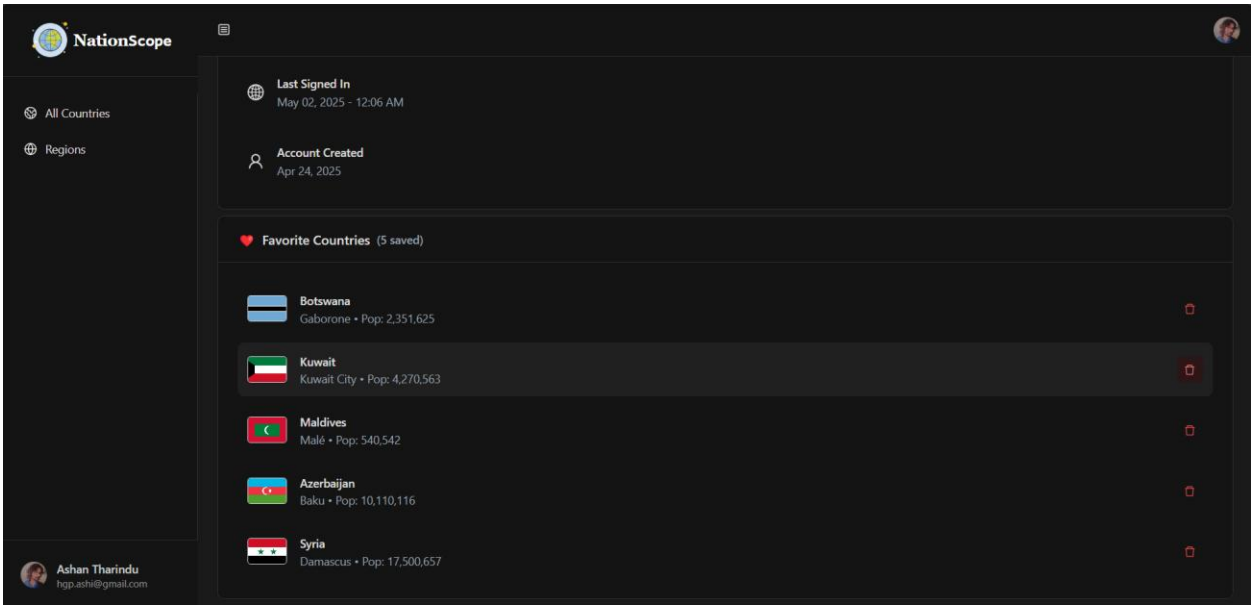
Search Countries by Region



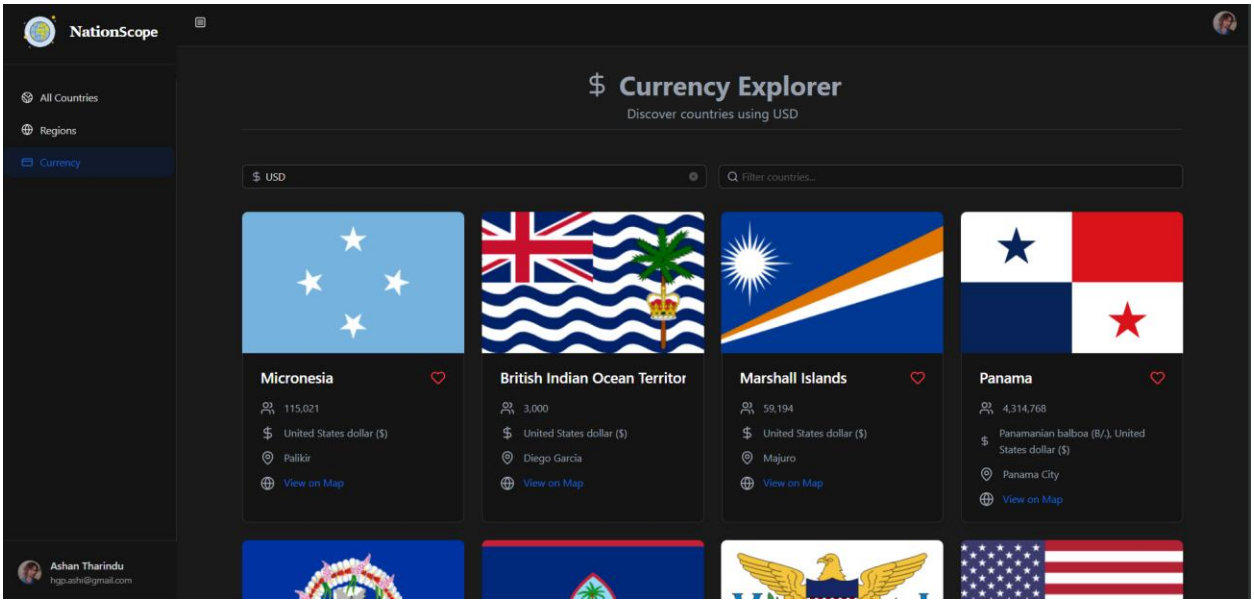
User Profile Page



Favorite Countries



Currency Page



Testing

Followed an **automated testing approach** to ensure the functionality, reliability, and usability of the application. Our strategy focused on validating key features through repeatable and maintainable tests.

Approach:

- **Functional and UI Testing**: Used **Selenium WebDriver** with **Python** to simulate user interactions and verify application behavior.
- **Test Organization**: The **pytest** framework was used to structure test cases, manage test execution, and handle setup/teardown processes.
- **Maintainable Test Code**: Implemented the **Page Object Model (POM)** design pattern to separate page structure from test logic, promoting code reuse and simplifying maintenance.

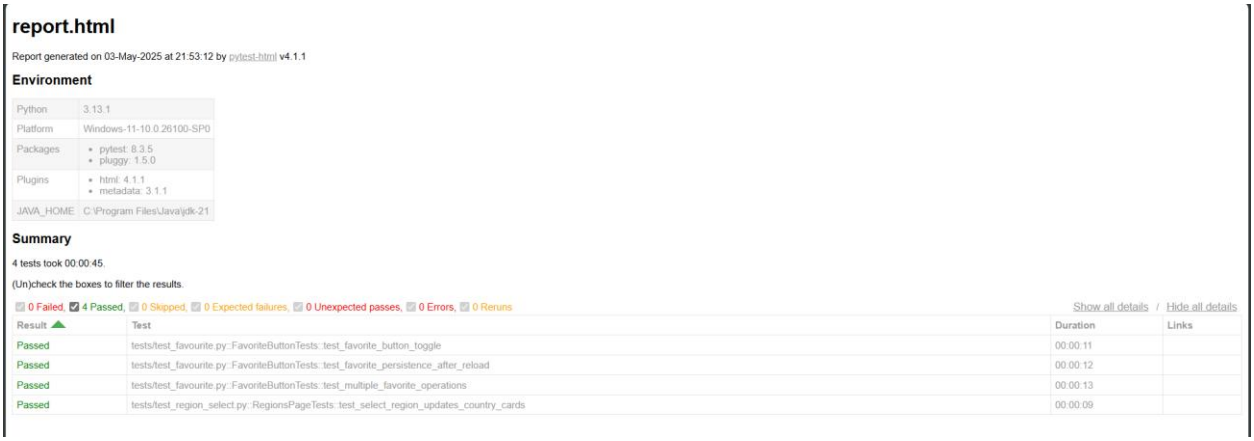
Tools and Technologies:

- **Python 3.10** with **pytest** – for writing and executing test cases
- **Selenium WebDriver** – for browser automation
- **Microsoft Edge** and its corresponding WebDriver – for real browser testing
- **pytest-html** – to generate interactive and readable test reports

This combination allowed for clear, automated validation of key application behaviors across different scenarios.

Feature Area	Test Coverage
Authentication	- Valid and invalid login scenarios - Registration with new and existing emails
Favorite System	- Toggle favorite on/off - Verify persistence after page reload - Multiple toggle operations
Region Selection	- Region-based filtering - Dynamic content updates based on selection - UI validation
Navigation	- Access control to protected routes - Redirect behavior (e.g., after login) - Handling unexpected or broken routes

Test Report



report.html

Report generated on 03-May-2025 at 21:53:12 by pytest-html v4.1.1

Environment

Python	3.13.1
Platform	Windows-11-10.0.26100-SP0
Packages	• pytest: 8.3.5 • pluggy: 1.5.0
Plugins	• html: 4.1.1 • metadata: 3.1.1
JAVA_HOME	C:\Program Files\Java\jdk-21

Summary

4 tests took 00:00:45.

(Un)check the boxes to filter the results.

0 Failed, 4 Passed, 0 Skipped, 0 Expected failures, 0 Unexpected passes, 0 Errors, 0 Reruns — Show all details / Hide all details

Result	Test	Duration	Links
Passed	tests/test_favourite.py::FavoriteButtonTests::test_favorite_button_toggle	00:00:11	
Passed	tests/test_favourite.py::FavoriteButtonTests::test_favorite_persistence_after_reload	00:00:12	
Passed	tests/test_favourite.py::FavoriteButtonTests::test_multiple_favorite_operations	00:00:13	
Passed	tests/test_region_select.py::RegionsPageTests::test_select_region_updates_country_cards	00:00:09	

Challenges Faced & Solutions

Complex and Inconsistent API Data Structure

The REST Countries API returns deeply nested and sometimes inconsistent data structures. For example, languages, currencies, and native-name objects vary greatly between countries and require transformation for UI rendering.

Solution

Custom utility functions and manual data restructuring were used to normalize the data. Optional chaining and null checks (?.) were applied to safely access nested fields.

For example,

```
export const getAllCountries = async () => {  Show usages  Ashan Tharindu
  try {
    const res = await api.get(
      url: "/all?fields=name,capital,currencies,flags,population,maps",
    );

    return (
      res.data &&
      res.data.map((country) => ({
        name: country.name,
        currencies: country.currencies,
        capital: country.capital,
        flags: country.flags,
        population: country.population,
      }))
    );
  } catch (e) {
    console.error("Error fetching countries:", e);
    return [];
  }
};
```

Regional Filtering / Search Not Updating State Properly

When implementing the region filter using the /region/{region} endpoint, the filtered list sometimes failed to re-render correctly. This was due to stale props and unnecessary re-renders caused by frequent changes in the selected region, which affected component performance and led to inconsistent UI updates.

Solution

To optimize performance and ensure that filtering logic ran only, when necessary, useMemo was introduced to memoize the filtered country list based on the selected region. This avoided redundant computations and kept the component rendering efficient. In addition, useEffect and useState were used correctly to synchronize the region selection and trigger data fetching only when dependencies changed. This combination ensured both correctness and performance while maintaining a clean separation of concerns.

```
useEffect( effect: () => {
  if (selectedRegion) {
    setLoading( value: true);
    getCountriesByRegion(selectedRegion) Promise<...>
      .then((data) => setCountries(data)) Promise<void>
      .finally( onFinally: () => setLoading( value: false));
  }
}, deps: [selectedRegion]);

const filteredCountries : any[] = useMemo(
  factory: () =>
    countries.filter(
      (country) =>
        country.name.common
          .toLowerCase()
          .includes(searchQuery.toLowerCase()) ||
        country.name.official
          .toLowerCase()
          .includes(searchQuery.toLowerCase()),
    ),
  deps: [countries, searchQuery],
);
```

UI Design Conflicts Between Tailwind CSS and Ant Design

Combining Tailwind CSS and Ant Design led to conflicting styles and layout issues, especially when using form controls or modal components.

Solution:
Careful scoping and custom CSS overrides were used to avoid conflicts. Where possible, layout and responsiveness were handled with Tailwind, while Ant Design was used for higher-level UI components (e.g., dropdowns, tables, modals).

Conclusion

The *Nationscope* project was a valuable exercise in modern web application development, combining the power of React with real-time data integration from the REST Countries API. Through this project, key objectives such as API consumption, responsive UI design, state management, and optional user authentication were successfully met.

The development process involved integrating multiple tools and technologies including React with Vite for a fast and optimized development experience, Tailwind CSS and Ant Design for styling and UI components, and Firebase for backend services like authentication. The use of Axios and RESTful endpoints allowed clean, modular API handling, while React's useState, useEffect, and useMemo hooks ensured a performance and reactive user interface.

Numerous real-world challenges were encountered, ranging from API data inconsistencies and UI framework conflicts to deployment issues and performance optimizations. However, each challenge provided learning opportunities that enhanced understanding of advanced React patterns, asynchronous data handling, and best practices in frontend architecture.

The final product is a responsive, well-structured, and user-friendly web application that provides rich information about countries, enables dynamic search and filtering, and is deployed on a reliable platform for public access. While there are opportunities for future enhancements (such as adding maps, saving favorite countries, or internationalization), the current version of *Nationscope* fulfills the assignment requirements comprehensively and reflects a solid grasp of application frameworks.

Overall, this project reinforced essential frontend development skills and highlighted the importance of writing maintainable, scalable foundation for building more complex, real-world applications in the future.