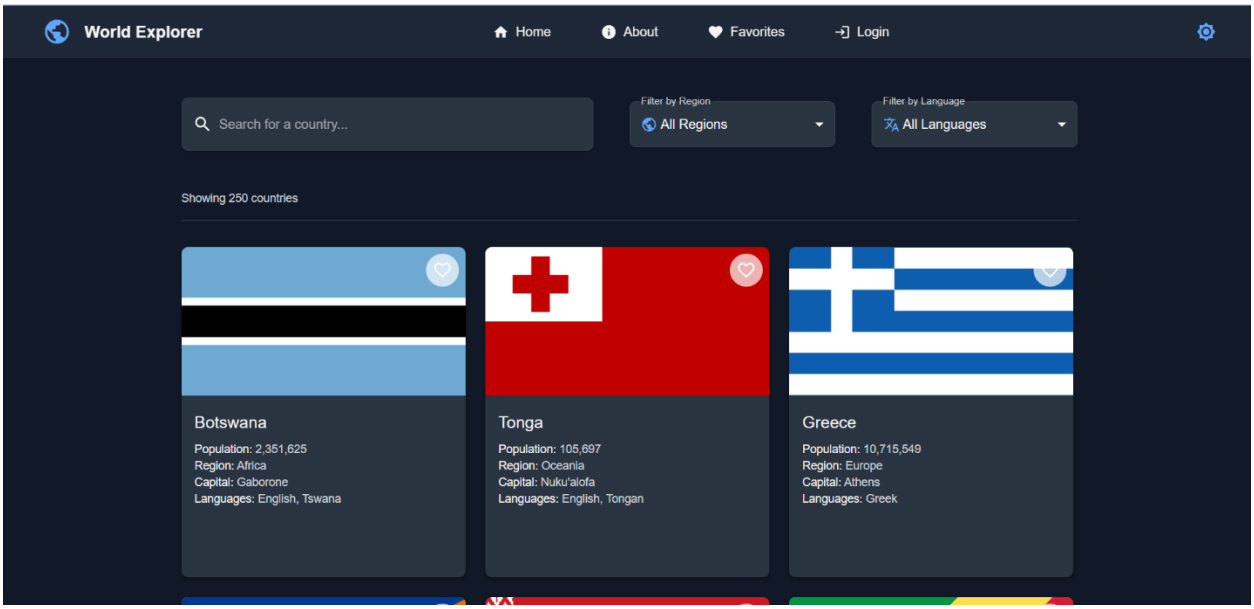
World Explorer Application



IT22195548

Adhikari A. A. J. T.

Introduction

Country Explorer is a responsive web application built with React that allows users to explore detailed information about countries worldwide. The application integrates with the REST Countries API to fetch and display country data, with additional features for user authentication and favorites management.

Hosted Application URL: https://rest-countries-app-deploy.vercel.app/

Technology Stack

Frontend

- **React** (Functional Components)
- **Vite** (Build Tool)
- **Material UI** (CSS Framework)
- **Axios** (HTTP Client)
- **React Router DOM** (Routing)
- **React Testing Library** (Testing)

Backend

- **Node.js** with Express
- **MongoDB** (Database)
- **JWT** (Authentication)

Deployment

- **Frontend:** Vercel

API Integration

REST Countries API Endpoints Used

Endpoint	Description
GET /all	Fetches all countries
GET /name/{name}	Searches countries by name
GET /region/{region}	Filters countries by region
GET /alpha/{code}	Gets detailed country information by alpha code

Application Features

Core Functionality

- Browse all countries
- Search countries by name
- Filter by region and language
- Detailed country view
- Responsive design for all devices
- Dark/Light mode toggle

Advanced Features

- User registration and login
- JWT-based authentication
- Favorite countries system
- Session persistence
- Error handling and loading states

Challenges and Solutions

Challenge 1: Handling Async API Calls

Issue:

- API calls were not properly synchronized, leading to race conditions where data could load out of order or fail to update correctly.
- Loading states were inconsistent, causing UI flickering.

Solution:

- Used React's useEffect with proper dependency arrays to control when API calls execute.
- Implemented loading states with a centralized context to prevent UI flickering.
- Added error boundaries and fallback UI for failed API requests.
- Utilized Axios interceptors to handle errors globally.

Challenge 2: Search Not Filtering Correctly

Issue:

- The search bar did not update the results dynamically as the user typed.
- Filtering was case-sensitive, making searches unreliable.

Solution:

- Implemented debouncing to avoid excessive API calls while typing.
- Added client-side filtering as a fallback if the API search (/name/{name}) fails.
- Normalized search input (converted to lowercase) for case-insensitive matching.

Challenge 3: Filtering by Region/Language

Issue:

- Language filtering was difficult because the API nests language data inside objects (e.g., languages: { eng: "English" }).
- Region filters sometimes return unexpected results due to API inconsistencies.

Solution:

- Flattened language data by extracting values from nested objects (e.g., Object.values(country.languages)).
- Cached API responses for /region/{region} to reduce redundant calls.
- Added a fallback filter in case the API response was incomplete.

Challenge 4: JWT Authentication (Backend)

Issue:

- Token expiration was not handled gracefully, causing users to be logged out unexpectedly.
- Storing JWTs securely.

Solution:

- Used HTTP-only cookies for JWT storage instead of localStorage.
- Implemented token refresh logic to silently renew expired tokens.
- Added middleware to verify tokens on protected routes.
- Used React Context to manage global auth state.

Challenge 5: UI Not Responsive (Layout Breaks on Mobile/Tablets)

Issue:

- The Material UI grid system sometimes collapsed on smaller screens.
- Images (flags) overflowed containers on mobile.

Solution:

- Replaced fixed-width layouts with flexible units **(**%**,** fr**,** minmax()**)**.
- Added responsive breakpoints in Material UI's sx prop.
- Used object-fit: contain for flag images to prevent distortion.

Installation Guide

Prerequisites

- Node.js (v18 or higher)
- npm or yarn
- MongoDB (for local backend)

Setup Instructions

1. Clone the repository:

   ```
   git clone https://github.com/SE1020-IT2070-OOP-DSA-25/af-2-JanudiAdhikari.git
   cd af-2-JanudiAdhikari
   ```

2. Install frontend dependencies:

   ```
   cd frontend
   npm install
   ```

3. Install backend dependencies:

   ```
   cd ../backend
   npm install
   ```

4. Create .env files with required environment variables.
5. Start the development servers:
   - Backend: npm run dev (from backend directory)
   - Frontend: npm run dev (from frontend directory)

Testing Strategy

Unit Tests

- Component rendering tests
- Utility function tests
- State management tests

Integration Tests

- API call tests
- Authentication flow tests
- Filtering functionality tests

Test Execution

```
cd frontend

npm test
```

Deployment Process

Frontend Deployment (Vercel)

1. Connected GitHub repository to Vercel
2. Configured environment variables
3. Set up automatic deployments on push to main branch

Conclusion

Country Explorer successfully demonstrates the integration of React with external APIs while implementing modern web development practices. The application provides a user-friendly interface for exploring country data with additional features that enhance user engagement through authentication and personalization.