EXPLORING COUNTRIES WITH REST API

Exploring Countries with Rest API using full stack

A MINI PROJECT REPORT

Submitted by

Kavyapriya S 312321205089

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

St. JOSEPH'S COLLEGE OF ENGINEERING

(An Autonomous Institution)

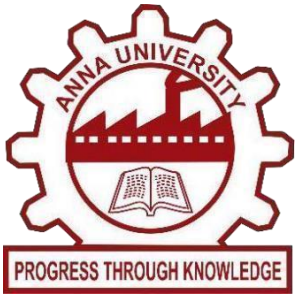
St. Joseph's Group of Institution

OMR, Chennai 600 119

ANNA UNIVERSITY: CHENNAI 600 025

April - 2024

ANNA UNIVERSITY: CHENNAI 600 025



BONAFIDE CERTIFICATE

Certified that this mini project report **"Exploring countries with Rest API**" is the bonafide work of **KAVYAPRIYA – 312321205089** who carried out the mini project under my supervision, for the partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Information Technology.

SIGNATURE

SUPERVISOR

Mrs. Sinthuja, M.Tech.,(Ph.D).,
Assistant Professor,
Department of Information Technology,
St. Joseph's College of Engineering,
OMR, Chennai- 600119.

SIGNATURE

HEAD OF THE DEPARTMENT

Ms. G Lathaselvi, B.E.,M.E.,(Ph.D).,
Associate Professor,
Department of Information Technology,
St. Joseph's College of Engineering,
OMR, Chennai- 600119.

CERTIFICATE OF EVALUATION

College name : St. Joseph's College of Engineering

Branch : Information Technology

Semester : VI

SL.NO	NAME OF THE STUDENTS	TITLE OF THE PROJECT	NAME OF THE SUPERVISOR
1	Kavyapriya S (312321205089)	**COUNTRIES API**	Mrs. Sinthuja

The report of the mini project work submitted by the above students in partial fulfillment for the award of Bachelor of Technology Degree in Information Technology of Anna University were evaluated and confirmed to be reports of the work done by the above students.

Submitted for mini project and Viva Examination held on____.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

At the outset we would like to express our sincere gratitude to the beloved **Chairman, Dr. Babu Manoharan, M.A., M.B.A., Ph.D.,** for his constant guidance and support.

We would like to express our heartfelt thanks to our respected **Managing Director, Mr. B. Shashi Sekar, M.Sc** for his kind encouragement and blessings.

We wish to express our sincere thanks to our **Executive Director, Mrs. S. Jessie Priya, M.Com.,** for providing ample facilities in the institution. We express our deepest gratitude and thanks to our beloved **Principal, Dr.Vaddi Seshagiri Rao, B.E.,M.E., M.B.A., Ph.D., F.I.E.,** for his inspirational ideas during the course of the project.

We wish to express our sincere thanks and gratitude to **Mrs. G. Lathaselvi, B.E.,M.E., (Ph.D)., Head of the Department**, Department of Information Technology, St. Joseph's College of Engineering for her guidance and assistance in solving the various intricacies involved in the project.

It is with deep sense of gratitude that we acknowledge our supervisor **Mrs. Sinthuja, M.Tech.,(Ph.D).,** for his expert guidance and connoisseur suggestion.

Finally we thank our department staff members who helped us in the successful completion of this mini project.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

This project presents a comprehensive exploration of countries worldwide through a web-based application utilizing a RESTful API and a full-stack development approach. The application offers users an intuitive platform to access detailed information about countries, including demographics, languages, currencies, and geographic data. Leveraging RESTful principles, the backend seamlessly communicates with the frontend, enabling efficient data retrieval and presentation. With React for dynamic frontend interfaces, Bootstrap for responsive design, and Node.js/Express.js for robust backend services, the application delivers a user-friendly experience. Through this integration, users can explore countries effortlessly, catering to diverse needs such as travel planning, research, and cultural discovery.

CHAPTER 1

INTRODUCTION

In today's interconnected world, access to information about different countries is invaluable for various purposes, from travel planning to research and education. Leveraging the power of APIs (Application Programming Interfaces) and full-stack development, we can create sophisticated web applications that provide users with comprehensive insights into countries worldwide. This introduction delves into the concept of exploring countries through an API model, employing a full-stack approach to build a robust and dynamic platform.

The Exploring Countries API Model utilizes a full-stack architecture to deliver a seamless and immersive user experience. At the core of this model is a RESTful API that serves as the gateway to vast repositories of country data. This API follows the REST principles, ensuring standardized communication between the frontend and backend components. By adhering to REST conventions, developers can efficiently retrieve, update, and manipulate country data, enhancing the scalability and maintainability of the application.

On the frontend, modern frameworks like React are employed to build dynamic user interfaces that facilitate intuitive interaction with the API. React's component-based architecture allows for the creation of reusable UI elements, streamlining development and promoting code organization. Additionally, frameworks like Bootstrap are utilized for styling, ensuring responsive and visually appealing designs that adapt seamlessly to various screen sizes and devices.

Meanwhile, on the backend, technologies such as Node.js and Express.js are utilized to create robust server-side applications. These frameworks enable efficient handling of HTTP requests, data processing, and business logic implementation. With Node.js's event-driven architecture and Express.js's minimalist approach, developers can build scalable and high-performance backend services that power the application's functionality.

The Exploring Countries API Model harnesses the capabilities of full-stack development to provide users with a comprehensive platform for accessing country information. Through a RESTful API, frontend technologies like React, and backend frameworks such as Node.js and Express.js, the model ensures efficient communication, dynamic user interfaces, and responsive designs.

CHAPTER 2

LITERATURE REVIEW

[1] The article "A Systematic Review of API Evolution Literature" is a comprehensive review of the literature on API evolution, which is the process of changing and updating APIs over time. The review identifies three main research questions: how the field of API evolution research has evolved, what the current state-of-the-art in API evolution research is, and what the current and future challenges related to API evolution are. The review also includes a systematic literature search and selection process, as well as an analysis of publication trends and common research goals, evaluation methods, metrics, and subjects in API evolution research. The review concludes that the main remaining challenges related to APIs and API evolution are automatically identifying and leveraging factors that drive API changes, creating and using uniform benchmarks for research evaluation, and understanding the impact of API evolution on API developers and users with respect to various programming languages

[2] Application Programming Interfaces (APIs) are perceived as major enablers of digital transformation, as such they have attracted the attention of both practitioners and researchers. This notwithstanding, past research on APIs have focused largely on the technical dimensions, neglecting the social and cultural contexts. The purpose of this study is therefore to understand how regulative, normative and cognitive institutions affect the development and integration of APIs in Ghana. Drawing on the new institutional theory as a lens and an interpretive case study methodology, our findings show that normative institutions such as business strategy, customers need, relationships, and experience of vendors enabled the development and integration of APIs. However, regulative institutions in the form of regulations and laws (unwillingness of some institutions to integrate with other applications), security concerns, were regarded as constraining factors to API integration. Also, Cognitive forces in the form of non-disclosure issues and carelessness constrained the integration of APIs.

[3] There are 69 urban villages in Tasikmalaya, and each one has its own information system including demographic and geografic information. The fundamental issue with this study is that the demographic and geography data for each urban village have not been integrated. The answer is to use Restful API to build a comprehensive urban village information system. The technology is used to

integrate spatial and population data by connecting them. The performance of access to demographic and geographic data information in each urban village in Tasikmalaya City can be improved by an integrated system. For population data, JSON is utilized, and for geographic data, GeoJSON. Each REST API is put through tests using Jmeter to gauge how well it performs in terms of response time and the volume of data the urban village information system receives. The study's findings, specifically how well each urban village information system integrated geography and population data. According to Jmeter test findings, it typically takes 14.7 seconds for the REST API SIKEL to load population data. The loading of geographic data in GeoJSON format via the typical GIS REST API takes 38.4 seconds.

[4] The purpose of this study is to perform a synthesis of API research. The study took stock of literature from academic journals on APIs with their associated themes, frameworks, methodologies, publication outlets and level of analysis. The authors draw on a total of 104 articles from academic journals and conferences published from 2010 to 2018. A systematic literature review was conducted on the selected articles. The findings suggest that API research is primarily atheoretical and largely focuses on the technological dimensions such as design and usage; thus, neglecting most of the social issues such as the business and managerial applications of APIs, which are equally important. Future research

directions are provided concerning the gaps identified.

[5] Application Programming Interfaces (APIs) are perceived as major enablers of digital transformation, as such they have attracted the attention of both practitioners and researchers. This notwithstanding, past research on APIs have focused largely on the technical dimensions, neglecting the social and cultural contexts. The purpose of this study is therefore to understand how regulative, normative and cognitive institutions affect the development and integration of APIs in Ghana. Drawing on the new institutional theory as a lens and an interpretive case study methodology, our findings show that normative institutions such as business strategy, customers need, relationships, and experience of vendors enabled the development and integration of APIs. However, regulative institutions in the form of regulations and laws (unwillingness of some institutions to integrate with other applications), security concerns, were regarded as constraining factors to API integration. Also, Cognitive forces in the form of non-disclosure issues and carelessness constrained the integration of APIs.

[6] As industry continues to expand, developers have created numerous application programming interfaces (APIs), including libraries, frameworks, web services, and other reusable code. For example, in the popular Node Package Manager (NPM), there are more than a half a million APIs available. Students learn APIs

in classes or in coding bootcamps; developers learn and use them to build software in their jobs. These APIs are what allow developer to make sophisticated software quickly by reusing the work others have done and shared, and developers consider available APIs as the most important factor when choosing a programming language.

[7] Modern web services increasingly rely on REST APIs. Effectively testing these APIs is challenging due to the vast search space to be explored, which involves selecting API operations for sequence creation, choosing parameters for each operation from a potentially large set of parameters, and sampling values from the virtually infinite parameter input space. Current testing tools lack efficient exploration mechanisms, treating all operations and parameters equally (i.e., not considering their importance or complexity) and lacking prioritization strategies. Furthermore, these tools struggle when response schemas are absent in the specification or exhibit variants. To address these limitations, we present an adaptive REST API testing technique that incorporates reinforcement learning to prioritize operations and parameters during exploration. Our approach dynamically analyzes request and response data to inform dependent parameters and adopts a sampling-based strategy for efficient processing of dynamic API feedback. We evaluated our technique on ten RESTful services, comparing it against state-of-the-art REST testing tools with respect to code coverage achieved,

requests generated, operations covered, and service failures triggered. Additionally, we performed an ablation study on prioritization, dynamic feedback analysis, and sampling to assess their individual effects. Our findings demonstrate that our approach outperforms existing REST API testing tools in terms of effectiveness, efficiency, and fault-finding ability.

[8] This discusses the use of News API, a web service that provides access to news articles from various sources, with React, a popular front-end framework. With News API, developers can quickly retrieve news articles from over 30,000 sources and customize the API response format to their needs. React provides a powerful and efficient way to create user interfaces, making it an ideal framework for building news applications. By using React with News API, developers can create a user-friendly and responsive news application that can be tailored to the user's requirements. This research paper provides an overview of News API and its features, discusses how to use News API with React, and includes an example code snippet. Developers can make use of the News API and React to create innovative and feature-rich news applications that can cater to a diverse range of users.

[9] Pro REST API Development with Node.js is your guide to managing and understanding the full capabilities of successful REST development. API design is a hot topic in the programming world, but not many resources exist for

developers to really understand how you can leverage the advantages. This book will provide a brief background on REST and the tools it provides (well known and not so well known). Understand how there is more to REST than just JSON and URLs. You will then cover and compare the maintained modules currently available in the npm community, including Express, Restify, Vatican, and Swagger. Finally you will code an example API from start to finish, using a subset of the tools covered. The Node community is currently flooded with modules; some of them are published once and never updated again - cluttering the entire universe of packages. Pro REST API Development with Node.js shines light into that black hole of modules for the developers trying to create an API. Understand REST API development with Node.js using this book today.

[10] Back-End is a logical space with the functionality and operation of a software application or information system. Currently, SMA Xaverius 1 Palembang does not yet have a rest server-based application whose background can be accessed by a multi-platform rest client. Therefore, an application based on the RESTful rest api based back-end was built. This RESTful application or REST Server provides data to be accessed by the REST Client using data exchange in JSON format using the HTTP protocol built using the Laravel Framework. The Laravel framework provides a mechanism for building REST Servers via the Rest Server library that supports a full RESTful server implementation. The results of this

research have been successful. A Laravel 7-based REST Server application has been built which provides endpoints in the form of json and status codes which can later be accessed by multi-platform rest clients. Through the application based on the back-end rest api, the results from the endpoint get a response from the server in the form of json that supports interoperability.

[11] We have created REST APIs ready to be consumed by client apps and to render the user interface using these APIs. The next step is to add security enhancements, and we'll be ready to distribute our APIs to third-party apps as well. That's exactly what we'll be doing in this chapter. In order to integrate these APIs with a React app, we will first create a basic todo application.

[12] Understanding the behaviour of a system's API can be hard. Giving users access to relevant examples of how an API behaves has been shown to make this easier for them. In addition, such examples can be used to verify expected behaviour or identify unwanted behaviours. Methods for automatically generating examples have existed for a long time. However, state-of-the-art methods rely on either white-box information, such as source code, or on formal specifications of the system behaviour. But what if you do not have access to either? e.g., when interacting with a third-party API. In this paper, we present an approach to automatically generate relevant examples of behaviours of an API, without requiring either source code or a formal specification of behaviour.

CHAPTER 3

SYSTEM ANALYSIS

Embarking on the development of a RESTful API for exploring countries is an exciting opportunity to put theoretical knowledge into practice. The main goal is to gain practical experience in designing, implementing, and testing a web service following RESTful principles. Building an API that provides detailed country information allows for a deeper understanding of concepts like HTTP methods, URI structures, data modeling, and API documentation. Challenges include selecting appropriate data sources, designing effective data models, and implementing robust error handling and authentication mechanisms. Throughout this project, I aim to develop proficiency in various areas such as RESTful architecture, backend development using technologies like Node.js, data modeling, API documentation, and testing. To enhance the learning experience, collaboration among peers, real-world use cases, feedback loops, and peer review sessions are essential. Ultimately, developing this Countries REST API is not just about creating a functional product but also about gaining valuable skills and insights that will be invaluable in my future pursuits in web development.

3.1 EXISTING SYSTEM

The existing system harnesses the REST Countries API to access comprehensive global country data, covering population statistics, languages spoken, currencies used, and geographical coordinates. With its RESTful architecture, the API simplifies data retrieval via HTTP requests, offering developers a convenient and efficient resource. The frontend, developed using React, capitalizes on the API's dynamic nature to furnish users with an intuitive interface for exploring country information. By seamlessly integrating with the API's endpoints, the frontend delivers real-time data updates as users navigate the application, facilitating tasks like country searches, regional browsing, and demographic analysis. Supporting the frontend is a full-stack architecture, employing frameworks like Express.js or Flask on the backend to manage communication between frontend and API. This setup ensures smooth data flow, fortified by robust error handling, authentication measures, and data formatting routines, thus enhancing system reliability and security. Integration between frontend and backend components is seamless, facilitated by HTTP requests. User interactions with the frontend trigger requests to the backend, which in turn communicates with the REST Countries API, culminating in the delivery of accurate and up-to-date information to users. In sum, the existing system, leveraging the REST Countries API, exemplifies the efficacy of integrating external resources to enrich web applications. By amalgamating React's flexibility with a full-stack approach, the system furnishes users with a compelling interface, empowering them to effortlessly explore and

learn about countries worldwide.

3.2 PROPOSED SYSTEM

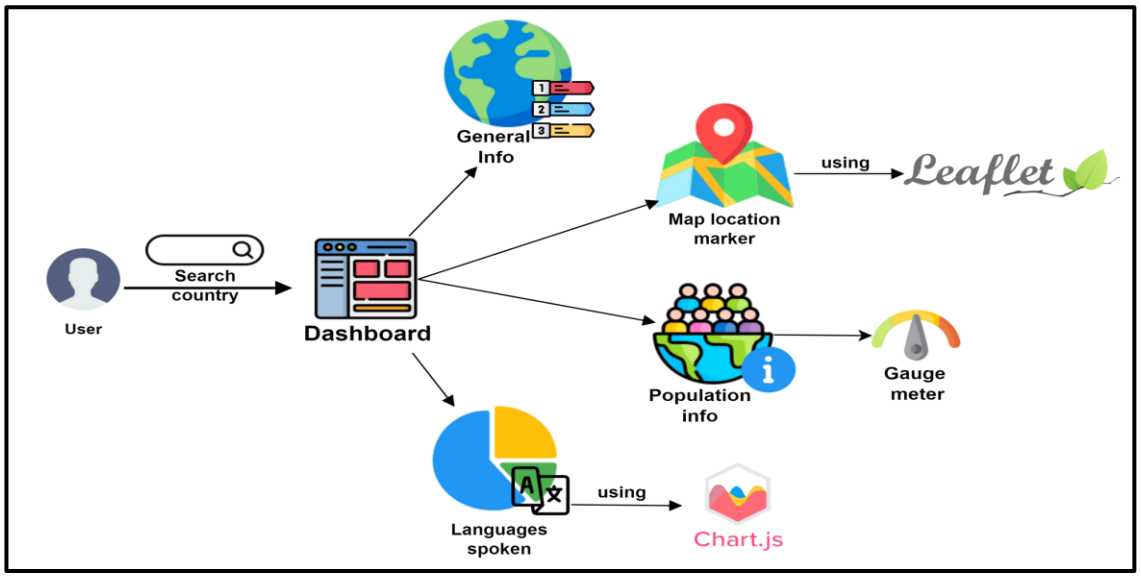
The proposed system design integrates the REST Countries API with a full-stack architecture and utilizes React for the frontend, Bootstrap for UI components, and Leaflet for interactive maps. The frontend, crafted with React and Bootstrap, offers an intuitive interface for users to explore country data. Leveraging the REST Countries API, the frontend dynamically fetches and displays comprehensive information, while Bootstrap ensures a responsive and visually appealing layout. Leaflet, an open-source mapping library, enriches the user experience by providing interactive maps for geographical visualization. On the backend, frameworks like Express.js or Flask manage communication between the frontend and the REST API, handling requests, and ensuring data reliability and security. The system architecture seamlessly integrates frontend and backend components, enabling smooth data flow and interaction, ultimately delivering a user-friendly platform for effortless exploration of country data worldwide.

CHAPTER 4

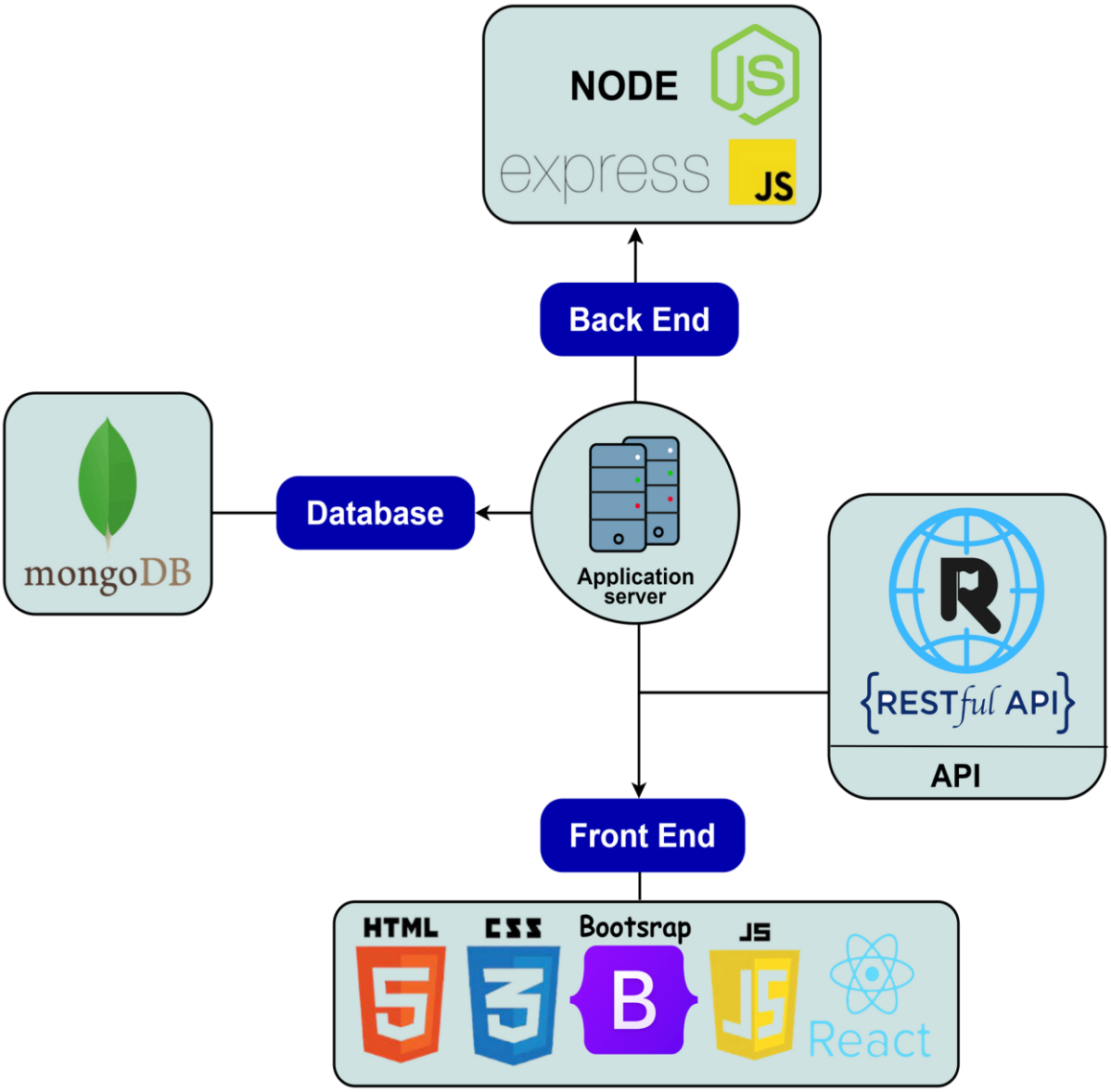
SYSTEM DESIGN

Software design is a process of problem-solving and planning for a software solution. After the purpose and specifications of software is determined, software developers will design or employ designers to develop a plan for a solution. It includes construction component and algorithm implementation issues which shown in as the architectural view. During this chapter we will introduce some principles that are considered through the Software design

4.1 WORKFLOW DIAGRAM



4.2 DEPENDENCY DIAGRAM



CHAPTER 5

SYSTEM IMPLEMENTATION

5.1 REST COUNTRIES API

The REST Countries API is a public API that provides information about countries around the world. It allows developers to retrieve data such as country names, capital cities, population, currencies, languages spoken, and more. This API follows RESTful principles, meaning it uses standard HTTP methods like GET requests to access and manipulate the data. Developers can use the REST Countries API to incorporate country-related information into their applications, such as educational tools, travel apps, or statistical analysis.

5.2 REACT

React is a popular JavaScript library for building user interfaces, particularly single-page applications. It allows developers to create interactive and dynamic UI components by breaking down complex UIs into smaller, reusable pieces called components. React uses a virtual DOM to efficiently update the UI, resulting in improved performance and a better user experience.When exploring the REST Countries API with React, developers can utilize React's component-based architecture to create a frontend interface for interacting with the API. They can design components such as search bars, dropdown menus, and result displays to fetch and showcase country data from the API. React's state

management and lifecycle methods enable seamless data fetching and updating, ensuring that the UI remains in sync with the API responses. Additionally, React's ecosystem offers tools like Axios for making HTTP requests, allowing developers to integrate API calls effortlessly. Overall, React empowers developers to build responsive, interactive, and efficient interfaces for exploring the REST Countries API.

5.3 BOOTSTRAP

Bootstrap is a front-end framework that facilitates the creation of responsive and visually appealing websites and web applications. It offers a collection of pre-designed templates, CSS styles, and JavaScript components, allowing developers to build user interfaces quickly and efficiently. When exploring the REST Countries API, Bootstrap can be utilized to enhance the user experience of consuming and displaying the retrieved data. Developers can leverage Bootstrap's grid system to structure and organize the layout of the webpage or application, ensuring consistency across different screen sizes and devices. Additionally, Bootstrap's styling components can be applied to various elements to improve the overall visual presentation of the country information obtained from the API. This might include styling buttons, tables, forms, and other UI elements to provide a cohesive and professional look. By integrating Bootstrap with the REST Countries API, developers can create user-friendly interfaces for browsing and interacting with country-related data seamlessly.

5.4 CHARTJ

Chart.js is a JavaScript library used for creating interactive and visually appealing charts and graphs on web pages. With Chart.js, developers can easily generate various chart types such as line charts, bar charts, pie charts, and more to visualize data in a user-friendly manner. When exploring the REST Countries API, Chart.js can be utilized to present the retrieved country data in a graphical format. For instance, developers can create bar charts to compare population sizes or pie charts to illustrate language distributions across different countries. By integrating Chart.js with the REST Countries API, users can gain insights into global demographics, economics, and geographic distributions through interactive visualizations. This enhances the user experience by making complex data more understandable and engaging, facilitating better decision-making and analysis. Overall, Chart.js serves as a valuable tool in presenting and exploring data from the REST Countries API in a comprehensible and insightful manner.

5.5 CHARTJS-2

Chart.js-2 is an enhanced version of this library, offering additional features and improvements. When exploring the REST Countries API, developers can leverage Chart.js-2 to visualize various country-related data. For instance, they can create bar charts to compare population sizes, pie charts to illustrate language distributions, or line charts to track historical trends in GDP.

By fetching relevant data from the REST API and integrating it with Chart.js-2, developers can provide users with insightful visualizations that make it easier to understand and interpret information about different countries. These charts enhance the user experience by presenting data in a more digestible format, facilitating analysis, and enabling users to gain valuable insights into global demographics, economics, and cultural aspects.

5.6 LEAFLET

Leaflet is an open-source JavaScript library used for creating interactive maps on web pages. It provides developers with a lightweight and flexible solution for displaying geographical data with various layers, markers, and popups. Leaflet is highly customizable and supports a wide range of mapping providers, making it suitable for diverse applications. When exploring the REST Countries API, Leaflet can be employed to visualize country-related data on an interactive map. For example, developers can retrieve geographical coordinates from the API along with other country information such as name, capital, and population. By integrating Leaflet with the REST Countries API, developers can dynamically generate markers on the map to represent each country and display additional information in popups when users interact with the markers. This allows users to visually explore the distribution of countries across the globe and access

relevant details about each country directly on the map interface, enhancing their overall browsing experience.

5.7 AXIOS

Axios is a popular JavaScript library used for making HTTP requests from web browsers and Node.js environments. It provides a simple and easy-to-use interface for sending asynchronous requests to servers and handling responses. In the context of exploring the REST Countries API, Axios can be utilized to interact with the API endpoints and retrieve information about different countries. To use Axios with the REST Countries API, developers typically make GET requests to specific endpoints provided by the API, such as /all to retrieve information about all countries, or /name/{country_name} to retrieve information about a specific country. Axios allows developers to handle the response data in various formats, such as JSON, and perform further processing or display it in their applications. By integrating Axios with the REST Countries API, developers can create dynamic and interactive applications that fetch and display country-related data to users seamlessly.

5.8 EXPRESS

Express is a popular web application framework for Node.js, designed to simplify the process of building web applications and APIs. It provides a robust set of features for handling HTTP requests, routing, middleware, and more, making it ideal for developing RESTful APIs.

When exploring the REST Countries API using Express, developers can leverage its routing capabilities to define endpoints that correspond to different API resources and actions. They can create routes to handle HTTP requests such as GET, POST, PUT, and DELETE, allowing them to retrieve, create, update, and delete country data from the REST Countries API. Express middleware can be used to preprocess requests, handle authentication, or perform data validation before sending requests to the API. developers can use Express to structure their application logic, separate concerns, and organize code effectively. By combining Express with other tools like Axios for making HTTP requests and Mongoose for database interaction, developers can create powerful and scalable REST API solutions for accessing and manipulating data from the REST Countries.

5.9 MONGOOSE

Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js, allowing developers to define data schemas and interact with MongoDB databases in a structured manner. When exploring the REST Countries API, Mongoose can be utilized to model the data retrieved from the API. Developers can create schemas to define the structure of country data, including attributes such as name, capital, population, currencies, and languages. Mongoose facilitates data validation, ensuring that only valid data is stored in the database according to the specified schema. Additionally, Mongoose simplifies the process of querying and manipulating data, providing a convenient interface for CRUD (Create, Read, Update, Delete) operations. By integrating Mongoose with the REST Countries API, developers can effectively manage and organize country data within their Node.js applications, enhancing scalability, maintainability, and overall development efficiency.

5.10 NODEMON

Nodemon is a utility tool for Node.js that automatically restarts the server when changes are detected in the code. It simplifies development by eliminating the need to manually restart the server. When exploring the REST Countries API, Nodemon can be used to ensure real-time updates while modifying and testing code for API integration.

CHAPTER 6

CONCLUSION

The web application, built on a full-stack architecture with Bootstrap for styling and React for frontend interactivity, offers an intuitive platform for exploring countries. Leveraging REST principles ensures efficient data retrieval, while Bootstrap's responsive design enhances user experience across devices. React's dynamic interfaces and comprehensive search features enable easy country discovery, complemented by interactive visualizations for deeper insights. Whether for travel planning, research, or curiosity, the application provides valuable country information. In conclusion, the integration of these technologies results in a robust, user-centric platform that caters to diverse exploration needs with efficiency and ease of use.

FUTURE SCOPE

1. Enhanced User Experience: Integrating the REST Countries API with a full-stack application using React can lead to a more immersive and interactive user experience. Users can explore detailed country information with dynamic UI components.

2. Global Data Analysis: Leveraging the vast dataset provided by the API, developers can implement data analysis features to gain insights into global demographics, economics, and social trends.

3. Personalized Dashboards: With React's component-based architecture and state management capabilities, developers can create personalized dashboards for users to track favorite countries, compare statistics, and visualize data using charts and graphs.

4. Mobile Applications: Utilizing React Native, developers can extend the application's reach by developing mobile apps that consume the REST Countries API. This opens up opportunities to target a broader audience across various platforms.

5. Localization and Internationalization: By incorporating language and specific features, developers can cater to diverse user bases. React's support for localization libraries can facilitate seamless translation and adaptation of content.

6. Real-time Updates: Implementing real-time updates using technologies like WebSockets or GraphQL subscriptions can enable live data feeds and notifications for users, ensuring they stay informed about changes in countries' data.

7. Community Engagement: Building features such as user reviews, ratings, and discussions around countries can foster community engagement within the application. React's ecosystem offers various plugins and components for implementing social features.

8. Scalability and Performance: Implementing efficient caching mechanisms, optimizing API requests, and employing serverless architectures can ensure scalability and high performance, accommodating growing user bases and traffic.

By leveraging a full-stack approach, developers can create robust, feature-rich applications that harness the wealth of data provided by the REST Countries API, offering users a comprehensive platform for exploring, analyzing, and engaging with global information.

APPENDIX 1

CODE

Server side:

```
const express = require('express');
const mongoose = require("mongoose");
const cors = require('cors');
const axios = require('axios');

const app = express();
app.use(cors());
app.use(express.json());

mongoose.connect("mongodb://localhost:27017/country_api", { useNewUrlParser: true,
useUnifiedTopology: true });

const countrySchema = new mongoose.Schema({
    name: String,
    capital: String,
    currency: String,
    nativeName: String,
    timeZone: String,
    languages: [String],
    latitude: Number,
    longitude: Number,
    population: Number,
    flag: String,
    populationCategory: { type: String, enum: ['high', 'medium', 'low'] }
});

const Country = mongoose.model('Country', countrySchema);

function categorizePopulation(population) {
    if (population > 100000000) {
        return 'high';
    } else if (population > 10000000) {
        return 'medium';
    } else {
        return 'low';
```

```
    }
}

app.get('/fetchAndSaveCountryInfo', async (req, res) => {
    const countryName = req.query.countryName;
    try {
        const response = await
axios.get(`https://restcountries.com/v3.1/name/${countryName}?fullText=true`);
        const countryData = response.data[0];

        const newCountry = new Country({
            name: countryData.name.common,
            capital: countryData.capital[0] || 'not available',
            currency: countryData.currencies ?
countryData.currencies[Object.keys(countryData.currencies)[0]].name : 'not
available',
            nativeName: countryData.name.nativeName ?
countryData.name.nativeName.common : 'none',
            timeZone: countryData.timezones ? countryData.timezones[0] : 'none',
            flag: countryData.flags.png,
            languages: countryData.languages ? Object.values(countryData.languages) :
[],
            latitude: countryData.latlng ? countryData.latlng[0] : 0,
            longitude: countryData.latlng ? countryData.latlng[1] : 0,
            population: countryData.population || 0,
            populationCategory: categorizePopulation(countryData.population)
        });

        await newCountry.save();

        res.status(201).json({ message: 'Country information saved successfully' });
    } catch (error) {
        console.error('Error fetching and saving country information:', error);
        res.status(500).json({ error: 'Internal server error' });
    }
});

app.get('/getLastSavedCountryInfo', async (req, res) => {
    try {
        const lastSavedCountry = await Country.findOne().sort({ _id: -1 }).limit(1);
        if (lastSavedCountry) {
            res.status(200).json(lastSavedCountry);
        } else {
            res.status(404).json({ message: 'No country information found' });
```

```
        }
    }
    catch (error) {
        console.error('Error retrieving last saved country information:', error);
        res.status(500).json({ error: 'Internal server error' });
    }
});
app.listen(30001, () => {
    console.log("Server is running on port 30001");
});
```

Client side:
index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import 'bootstrap/dist/css/bootstrap.min.css';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

App.js

```
import React, { useState, useEffect } from 'react';
import './App.css';
import { Button, Container, Form, Row, Col, Spinner } from 'react-bootstrap';
import axios from 'axios';
import PieChart from './components/PieChart';
import Info from './components/Info';
import Gauge from './components/Gauge.js';
import MarkerMap from './components/MarkerMap.js';

function App() {
  const [countryName, setCountryName] = useState('');
  const [showComponents, setShowComponents] = useState(false);
  const [loading, setLoading] = useState(false);
  const [lastSavedCountry, setLastSavedCountry] = useState(null);
  const [pieChartData, setPieChartData] = useState(null);
```

```
  useEffect(() => {
    getLastSavedCountry();
  }, []);

  const getLastSavedCountry = async () => {
    try {
      setLoading(true);
      const response = await
axios.get('http://localhost:30001/getLastSavedCountryInfo');
      console.log('Last saved country:', response.data);
      setLastSavedCountry(response.data);
      const labels = response.data.languages.map(lang => lang);
      const data = response.data.languages.map(() => response.data.languages.length);
      setPieChartData({
        labels: labels,
        datasets: [{
          label: 'Languages Spoken',
          data: data,
          backgroundColor: [
            "rgba(75,192,192,1)",
            "#f3ba2f",
            "#50AF95",
            "#2a71d0",
            "#ecf0f1",
          ],
          borderColor: "black",
          borderWidth: 2,

        }]
      });
      setLoading(false);
    }
    catch (error) {
      setLoading(false);
      console.error('Error fetching last saved country information:', error);
    }
  };

  const handleSubmit = async (event) => {
    event.preventDefault();
    try {
      setLoading(true);
      const response = await
```

```
axios.get(`http://localhost:30001/fetchAndSaveCountryInfo?countryName=${countryName}`
);
      console.log(response.data.message);
      setShowComponents(true);
      getLastSavedCountry();
    } catch (error) {
      setLoading(false);
      console.error('Error submitting country name:', error);
    }
  };

  const handleChange = (event) => {
    setCountryName(event.target.value);
  };

  return (
    <div>
      <Container className='mt-5 d-flex justify-content-center align-items-center'>
        <Row className='d-flex justify-content-center align-items-center'>

          <Form onSubmit={handleSubmit} className="d-flex align-items-center">
            <Form.Group controlId="countryName" className="mr-2 mb-0">
              <Form.Control
                type="text"
                value={countryName}
                onChange={handleChange}
                placeholder="Enter country name"
              />
            </Form.Group>
            <Button className='mx-3' variant="primary" type="submit">
              {loading ? <Spinner animation="border" variant="light" size="sm" /> :
'Search'}
            </Button>
          </Form>
        </Row>
      </Container>
      {
        showComponents && lastSavedCountry && (
          <div>
            <Container>
              <Row className='justify-content-center align-items-center'>
                <Col lg={5} md={12} sm={12} className='d-flex px-5 m-3 justify-
content-between align-items-center box'>
                  <Info {...lastSavedCountry} />
```

```
                </Col>
                <Col lg={5} md={12} sm={12} className='d-flex justify-content-center
align-items-center box'>
                  <Gauge level={lastSavedCountry.populationCategory} />
                </Col>
              </Row>
              <Row className='justify-content-center align-items-center' >
                <Col lg={5} md={12} sm={12} className='m-3 d-flex justify-content-
center align-items-center box'>
                  {pieChartData && <PieChart data={pieChartData} />}
                </Col>
                <Col lg={5} md={12} sm={12} className='d-flex justify-content-center
align-items-center box'>
                  <MarkerMap latitude={lastSavedCountry.latitude}
longitude={lastSavedCountry.longitude} />
                </Col>
              </Row>
            </Container>
          </div>
        )
      }
    </div >
  );
}

export default App;
```

Components:
Info.js

```
import React from 'react';
import '../App.css';

function Info(lastSavedCountry) {
    return (
        <div className="d-flex align-items-center">
            <img className="me-4 flag" src={lastSavedCountry.flag} alt="Flag" />
            <div>
                <h4>General Info:</h4>
                <h6>Country Name: {lastSavedCountry.name}</h6>
                <h6>Capital: {lastSavedCountry.capital}</h6>
                <h6>Currency: {lastSavedCountry.currency}</h6>
                <h6>Timezone: {lastSavedCountry.timeZone}</h6>
```

```
            </div>
        </div>
    )
}
export default Info;
```

Gauge.js

```
import React, { useEffect, useRef, useState } from 'react';
import '../App.css';

const Gauge = ({ level }) => {
    const gaugeRef = useRef(null);
    const [value, setValue] = useState(0);

    useEffect(() => {
        setValue(mapLevelToValue(level));
    }, [level]);

    useEffect(() => {
        setGaugeValue(gaugeRef.current, value);
    }, [value]);

    const mapLevelToValue = (level) => {
        switch (level) {
            case 'low':
                return 0.25;
            case 'medium':
                return 0.5;
            case 'high':
                return 0.9;
            default:
                return 0;
        }
    };

    const setGaugeValue = (gauge, value) => {
        if (value < 0 || value > 1) {
            return;
        }
        gauge.querySelector(".gauge__fill").style.transform = `rotate(${value /
2}turn)`;
```

```
        gauge.querySelector(".gauge__cover").textContent = `${level}`;
    };

    return (
        <div className="gauge" ref={gaugeRef}>
            <h4>Population: </h4>
            <div className="gauge__body">

                <div className={`gauge__fill ${level}`}></div>
                <div className="gauge__cover"></div>
            </div>
        </div>
    );
};

export default Gauge;
```

PieChart.js

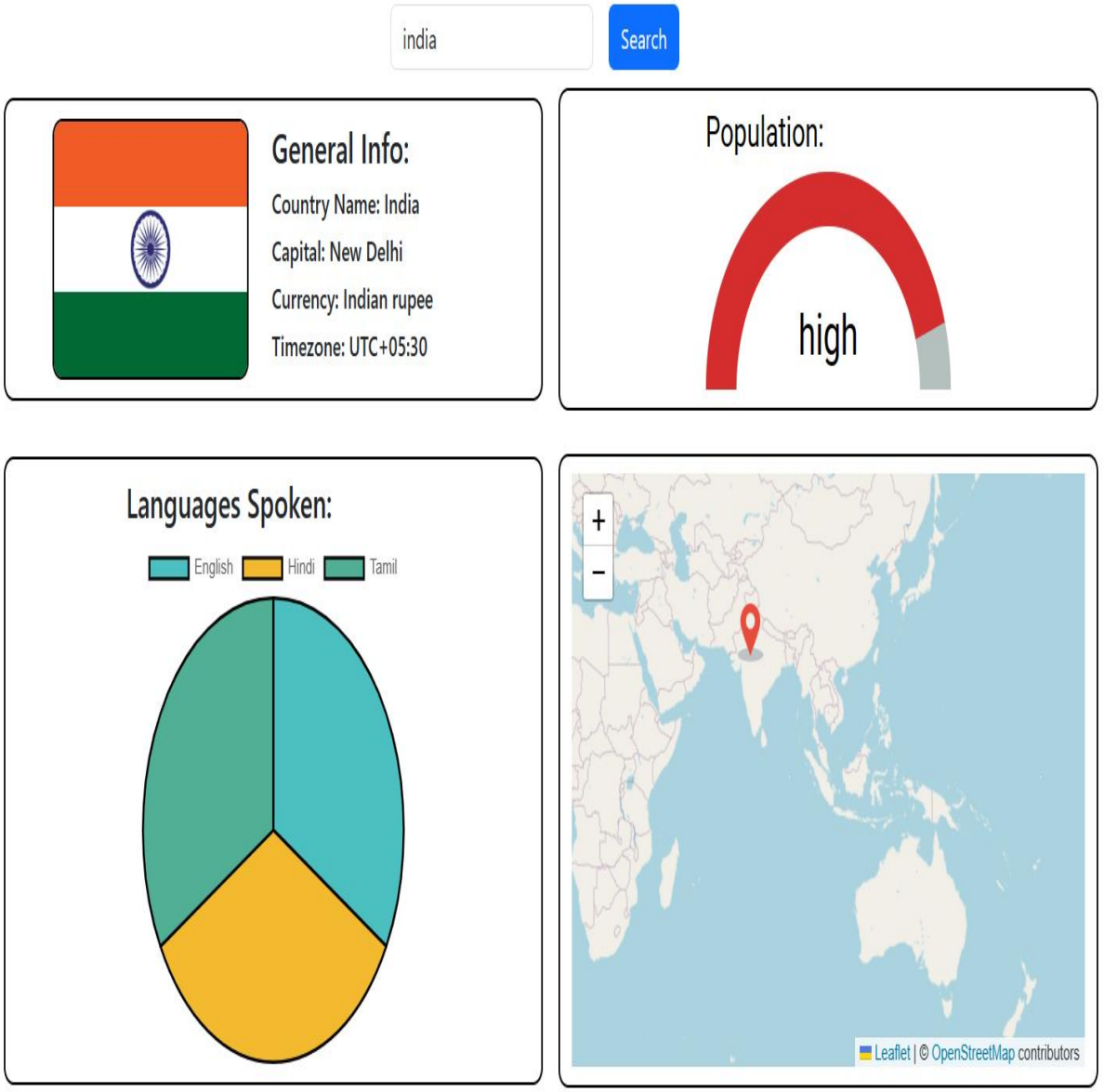
```
import React from 'react'
import { Pie } from "react-chartjs-2";
import { Chart as ChartJS } from "chart.js/auto";

function PieChart({ data }) {
    return <div style={{ width: 300 }}>
        <h4>Languages Spoken:</h4>
        <Pie data={data} />
    </div>
}

export default PieChart;
```

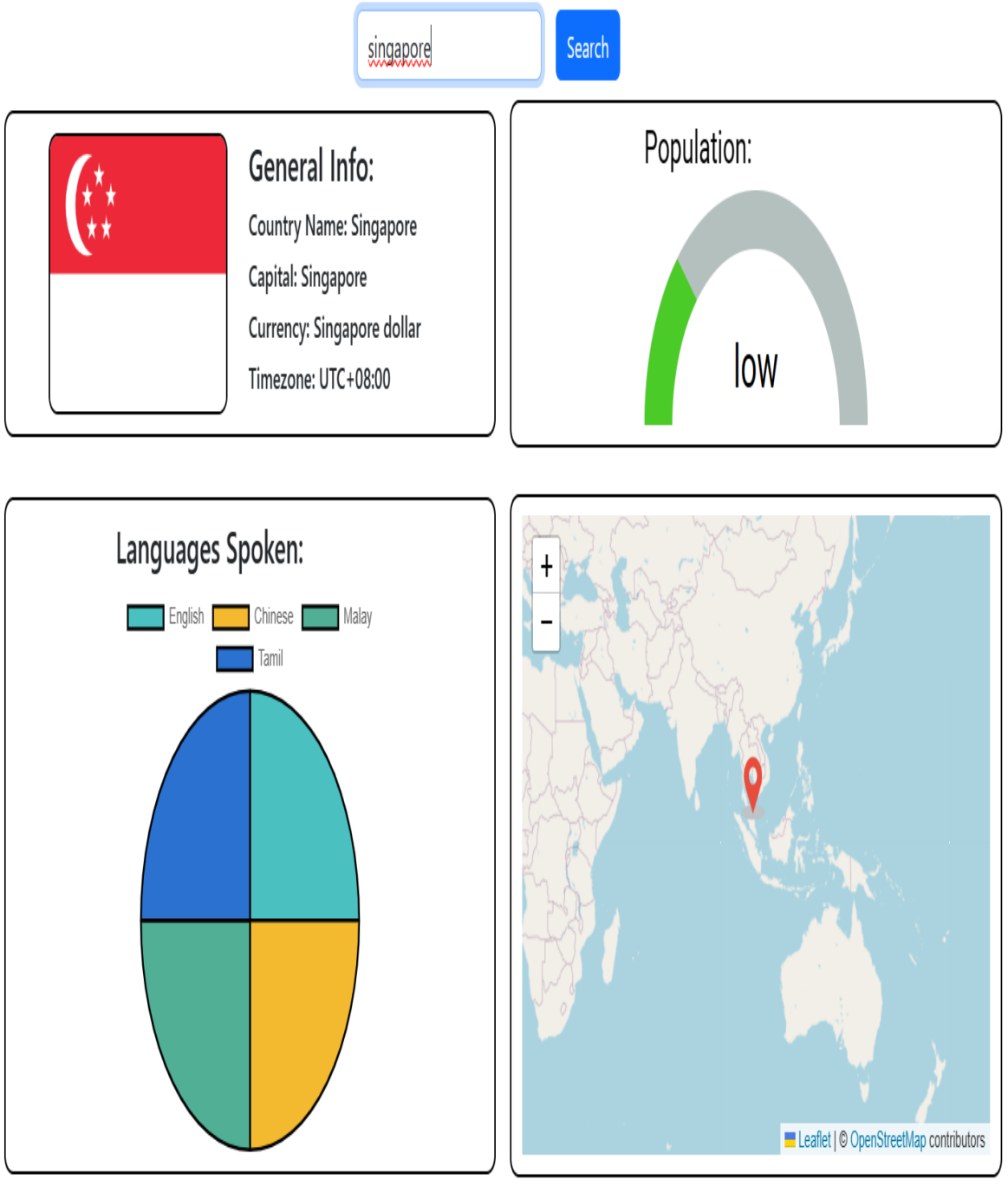
MarkerMap.js

```
import React from "react";
import { MapContainer, TileLayer, Marker, Popup } from "react-leaflet";
import "leaflet/dist/leaflet.css";
import L from "leaflet";
import pointer from './location.png';

const customIcon = L.icon({
    iconUrl: pointer,
```

```
    iconSize: [38, 38],
    iconAnchor: [22, 44],
    popupAnchor: [-3, -46]
});

const MarkerMap = ({ latitude, longitude }) => {
    return (
        <MapContainer center={[latitude, longitude]} zoom={2} style={{ height:
"340px", width: "100%" }}>
            <TileLayer
                attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
                url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
            />
            <Marker position={[latitude, longitude]} icon={customIcon}>
                <Popup>
                    Latitude: {latitude}, Longitude: {longitude}
                </Popup>
            </Marker>
        </MapContainer>
    );
};
export default MarkerMap;
```

APPENDIX 2
OUTPUT




singapore
Search
General Info:
Country Name: Singapore
Capital: Singapore
Currency: Singapore dollar
Timezone: UTC+08:00
Population:
low
Languages Spoken:
English
Chinese
Malay
Tamil
+
−
Leaflet | © OpenStreetMap contributors

malaysia Search

General Info:

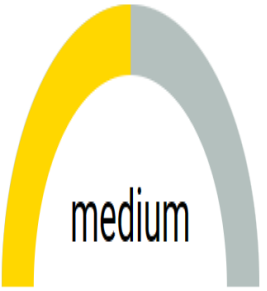
Country Name: Malaysia

Capital: Kuala Lumpur

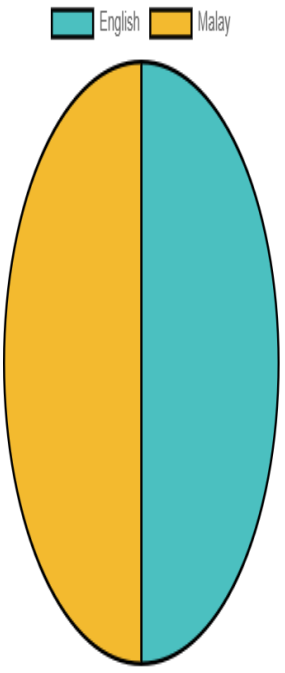
Currency: Malaysian ringgit

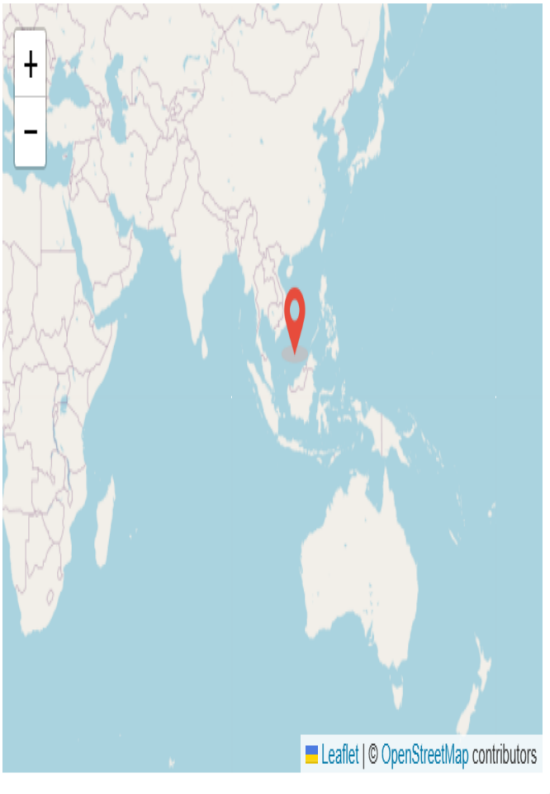
Timezone: UTC+08:00

Population:

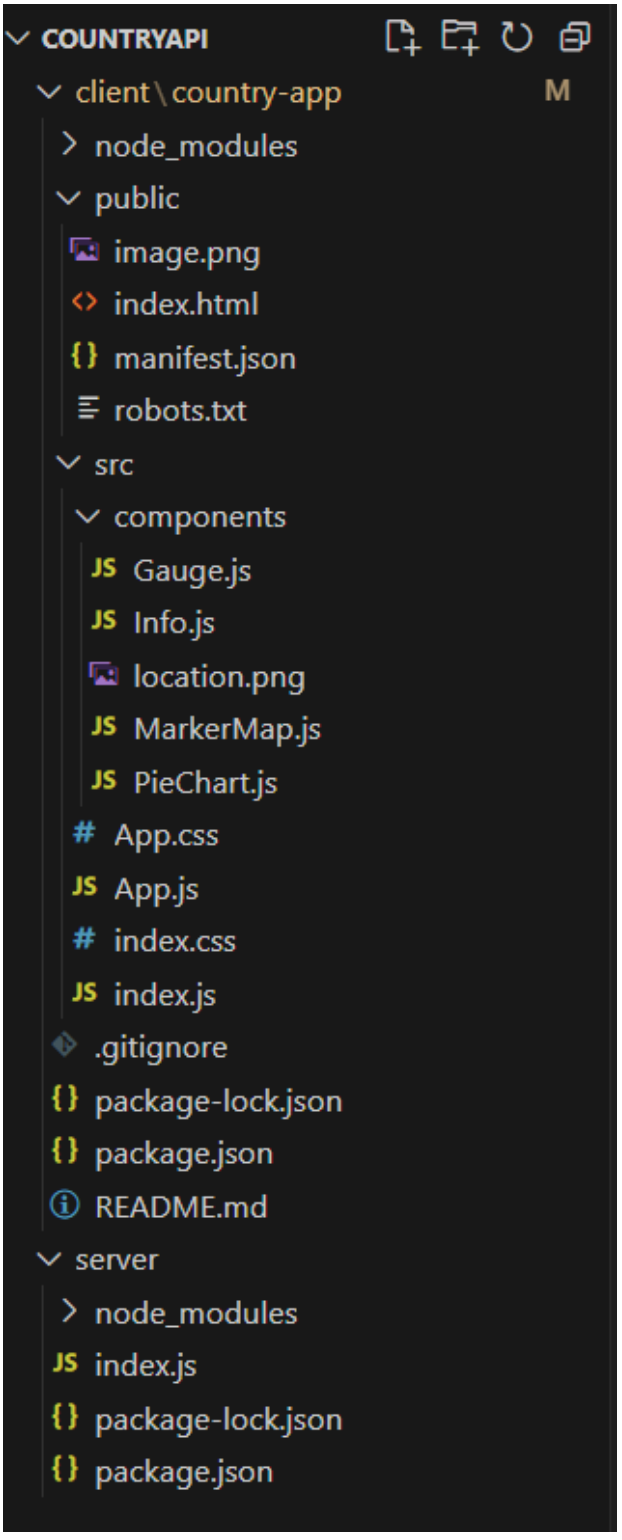


Languages Spoken:





File Structure



References

[1] https://www.mongodb.com/docs/

[2] https://react-bootstrap.netlify.app/

[3] https://react.dev/blog/2023/03/16/introducing-react-dev

[4] https://mongoosejs.com/docs/

[5] https://axios-http.com/docs/intro

[6] https://restcountries.com/

[7] https://ieeexplore.ieee.org/abstract/document/10006893

[8] https://www.researchgate.net/publication/371962386_A_Research_Paper_on_React_News_-_API?_sg=gODGE9th4KUYBys57IpCThaB3lZX_DZcK

[9] https://www.researchgate.net/publication/376200521_Revolutionizing_Communication_Crafting_a_CuttingEdge_Chat_App_with_the_latest_ChatGPT_API_and_Full_Stack_Technologies?_sg=wKxK6I8HHwlHrcQtk3EKwYuxaKDeFMwaEOOiGPkibro4936c1CdEKQIqc5tEyiVDSRV_XjaYR0hah_s&_tp=eyJjb250ZXh0Ijp7ImZpcnN0UGFnZSI6Il9kaXJlY3QiLCJwYWdlIjoiX2RpcmVjdCJ9fQ

[10] https://www.researchgate.net/publication/326494482_REST_API_Development_with_Nodejs_Manage_and_Understand_the_Full_Capabilities_of_Successful_REST_Development

[11] https://www.researchgate.net/publication/372618199_Power_BI_REST_API

[12] https://www.researchgate.net/publication/378528850_Microservices_implementation_using_Rest_API_for_MSMEs_based_on_Raspberry_PI

[13] https://www.researchgate.net/publication/379999428_Exploring_API_behaviours_through_generated_examples?_sg=oXT2rVdIFccd7Ys9h3XoMpFtN_7dir60IP14niU9eBOO8y3CArfUX3DZnownd8pilc0_mJWRUi7ALc&_tp=eyJjb250ZXh0Ijp7ImZpcnN0UGFnZSI6Il9kaXJlY3QiLCJwYWdlIjoiX2RpcmVjdCJ9fQ

[14] https://www.researchgate.net/publication/380066492_REST_API

[15] https://www.researchgate.net/publication/373487191_Exploring_API_Behaviours_Through_Generated_Examples

[16] https://www.researchgate.net/publication/359324736_CRAFTS_Configurable_REST_APIs_for_Triple_Stores

[17] https://www.researchgate.net/publication/290747064_How_Do_Developers_React_to_RESTful_API_Evolution

[18] https://www.researchgate.net/publication/321523861_Pro_REST_API_Development_with_Nodejs

[19] https://www.researchgate.net/publication/362321044_BACK-END_SYSTEM_DESIGN_BASED_ON_REST_API

[20] https://www.researchgate.net/publication/372801152_Integrating_REST_APIs_with_a_Frontend_React_App

[21] https://www.researchgate.net/publication/378552167_RESTlogic_Detecting_Logic_Vulnerabilities_in_Cloud_REST_APIs