

Marco Tidu

COUNTRIES DATA VISUALIZATION SYSTEM

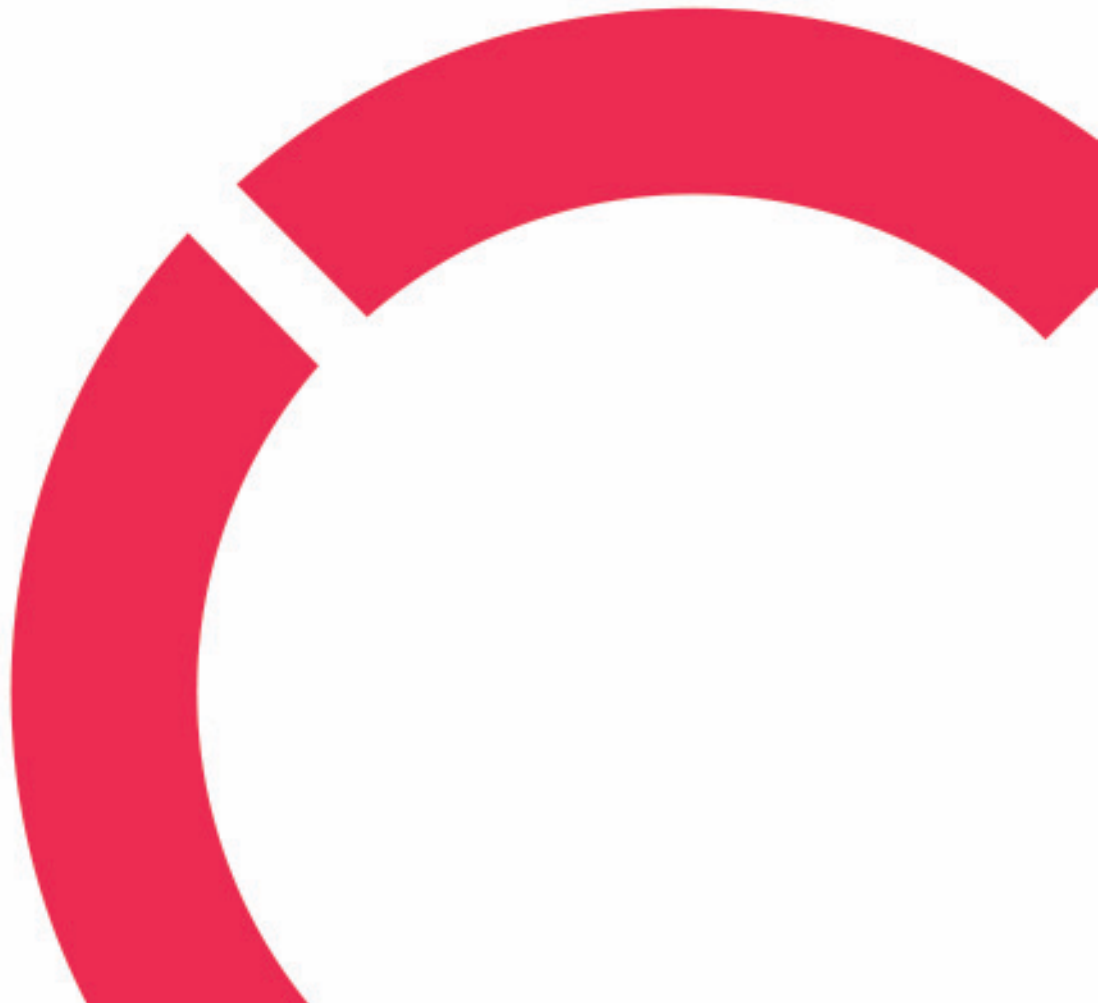
Exploring Country Data through interactive visualizations

Thesis

CENTRIA UNIVERSITY OF APPLIED SCIENCES

Information Technology

June 2023



ABSTRACT

Centria University of Applied Sciences	Date 4.6.2023	Author Marco Tidu
Degree programme Information Technology		
Name of thesis Countries Data Visualization System – Exploring Country Data through interactive visualizations		
Centria supervisor Aliashgar Khavasi	Pages 25 + 3	
Instructor representing commissioning institution or company Aliashgar Khavasi		
<p>This project aimed to develop a web application that provides users with information about countries worldwide. The application utilizes the REST Countries APIⁱ to fetch and display country data, including population, area, and borders. The project's main objectives were to create an user-friendly interface with data visualization tools, implement state management, and ensure application responsiveness across different devices.</p> <p>The development process involved using Reactⁱⁱ for the frontend, Reduxⁱⁱⁱ for state management, Express.js^{iv} and Node.js^v for the backend side and Tailwind^{vi} and Flowbite^{vii} for custom styling. Data visualization tools like tables, graphs, and live calculations were implemented to aid users in understanding the information using Chart.js^{viii} as visualization library. The project also incorporated best practices for state management and responsive design to provide a smooth and efficient user experience.</p> <p>The project's outcome was a functional and interactive web application that achieved the objectives set out in the beginning. The application allows users to search for specific countries, view country data, and compare countries based on different criteria. The project's significance lies in its contribution to making information about countries more accessible and easily digestible for users. The skills gained during the project, including proficiency in React, Redux, Express, Node and Tailwind, are valuable for building dynamic and responsive web applications.</p>		
Key words Chart.js, Express.js, Fullstack Application, JavaScript, React, TypeScript		

CONCEPT DEFINITIONS

FRONT END

User interface accessible via computer, tablet, or mobile phone.

BACK END

Application paired with the front end that enables the communication between user and server.

SERVER

Computer designed to provide a service to other computers called clients and their users.

CLIENT

Device such as computer, tablet or mobile phone used to consume data.

FULL STACK APPLICATION

Application consisting of front end and back end sides.

REACT

JavaScript based UI development library developed by Facebook widely used in web development.

JAVASCRIPT

High level programming language that enables the user to create content on the web, control multimedia files and communicate with other computers.

TYPESCRIPT

Strongly typed programming language based on JavaScript, giving better developing tools.

ABSTRACT

CONCEPT DEFINITIONS

CONTENTS

1	INTRODUCTION.....	1
2	LITERATURE REVIEW.....	3
2.1	Visualization Libraries	3
2.2	Interactive Dashboards	4
2.3	Virtual Reality (VR) and Augmented Reality (AR)	5
2.4	World Data Bank	6
2.5	Worldometer	7
3	Methodology.....	9
3.1	React.....	9
3.2	Chart.js	9
3.3	Tailwind	10
3.4	Flowbite.....	10
3.5	React Router.....	11
3.6	Axios.....	11
3.7	Typescript	12
3.8	Express.js	12
3.9	ESLint	13
4	Backend Development	14
5	Frontend Development	15
5.1	Data By Country	15
5.2	Data By Region.....	17
5.3	Data Visualization.....	20
6	Implementation	23
7	Results and evaluation.....	24
8	Conclusion.....	25

FIGURES

Figure 1 Screenshot from a chart built with Chart.js, the library of choice for my project.....	3
Figure 2 Screenshot from Power BI, interactive dashboard from Microsoft.....	4
Figure 3 Which technology to choose between virtual reality and augmented reality for a professional application?	5
Figure 4 Screenshot from World Data Bank web application	6
Figure 5 Screenshot from Worldometer web application	7
Figure 6 Tailwind made my design planning much easier than expected. Image credits: (https://blog.logrocket.com/comparing-tailwind-css-bootstrap-time-ditch-ui-kits/)	19
Figure 7 Screenshot from Data By Region section showing a bar chart from Chart.js library.	20
Figure 8 Screenshot from Data By Region section, showing a pie chart from Chart.js library.....	21
Figure 9 Data By Country table	22

CODE SNIPPETS

Code 1 Code snippet of thunk function in the application.....	15
Code 2 Code snippet from of onChange function from the application.	15
Code 3 Code snippet from the conditional rendering section from the application	17
Code 4 Code snippet showing the useEffect hook dispatching the fetchCountries thunk.....	17
Code 5 Two different useState handling the state of the two chart component.....	17
Code 6 Here the actual conditional rendering is shown.....	18

1 INTRODUCTION

In this part the reader will find details regarding the background, motivation, and context of the project, mentioning its objectives and intended outcomes, explaining its significance and utilization, and defining its scope and limitations. The Model-View-Controller^{ix} (MVC) architectural pattern is a widely used design pattern in software engineering that separates an application's components into three interconnected parts - the Model, View, and Controller. In the context of this project, the MVC pattern can be applied to describe the background, motivation, and context of the full-stack application that renders data from the REST Countries API. The Model component of the application involves the data storage and management system, which in this case, is the REST Countries API. The API provides data about countries worldwide, including information such as population, area, and borders. The motivation for using this API is to provide users with access to relevant and up-to-date information about countries in a user-friendly manner.

The View component is responsible for rendering the data from the Model component to the user interface. The user interface in this project is built using React, a popular JavaScript library for building user interfaces. The use of React allows for the efficient rendering of dynamic data and the implementation of data visualization tools such as tables, graphs, and live calculations. The Controller component manages the communication between the View and Model components and controls the application's behaviour. In this project, the Controller is implemented using the Express framework, which handles server-side logic and API requests. Express provides a robust and efficient way to manage API calls to the REST Countries API and handle data processing and manipulation before serving it to the frontend.

In conclusion, the MVC pattern provides a useful framework for understanding the project's background, motivation, and context. The use of the REST Countries API as the Model component, React as the View component, and Express as the Controller component demonstrates the effective use of the MVC pattern in building a full-stack application that renders data from the REST Countries API. This project aims to develop a functional and interactive React application that will retrieve data from the restcountries.com API and analyze it with the MERN stack^x. I plan to use this opportunity to improve my skills in React, Node.js, and MongoDB while also learning about data visualization techniques. The intended outcomes of the project are fourfold. Firstly, the application should be able to efficiently retrieve data from the restcountries.com API and display it on the user interface. This will involve us-

ing various data fetching techniques such as RESTful APIs, Axios^{xi}, and Promises. Secondly, the application should include a set of effective graphs and charts that can represent the data to the user. Popular data visualization libraries such as Chart.js will be used to create different visualizations such as line charts, bar charts, pie charts, and scatter plots. The goal is to provide users with a clear understanding of the data. Thirdly, the application should have a user-friendly interface that allows users to interact with the data and customize the visualizations. This means implementing features like search filters, sorting, and pagination to make the data more accessible and customizable for users.

Finally, the codebase should be well-organized, follow best practices, and be easy to maintain and scale. This includes adhering to coding standards and following modular design patterns to make the code more modular and maintainable. The scope of the project is to develop a React application that utilizes the MERN stack to fetch and analyze data from restcountries.com API and visualize it in the form of graphs and charts. The project aims to create a user-friendly interface that allows users to interact with the data and customize the visualizations, and a well-organized codebase that follows best practices and is easy to maintain and scale. The project's limitations include the data provided by the restcountries.com API. While the API provides a wealth of data on countries worldwide, it may not be comprehensive or up to date in certain areas. Additionally, the project's visualizations are limited to the data provided by the API, and any insights or conclusions drawn from the visualizations must be taken with caution. Another limitation of the project is that it may not be able to handle large amounts of data efficiently. While the project utilizes modern web technologies to provide efficient data visualizations, there may be limitations on the amount of data that can be processed and displayed in real-time.

Finally, the project's scope may be limited by the resources and expertise of the development team. While the project aims to follow best practices and provide a well-organized codebase, there may be limitations on the team's knowledge and experience with certain technologies or development practices. This may impact the project's scalability and maintainability in the long term.

2 LITERATURE REVIEW

Existing research and technologies related to data visualization have significantly advanced the field, enabling the effective representation and analysis of complex data. For example, visualization libraries have become increasingly efficient, offering users high-quality tools to visualize their data effectively. Interactive dashboards have emerged as the preferred choice for modern business professionals, particularly in financial and banking sectors. On the other hand, Virtual Reality (VR) and Augmented Reality (AR) technologies are focused on providing immersive experiences with data, rather than emphasizing data manipulation. These technologies offer solutions that allow users to immerse themselves in data environments and gain unique perspectives on their datasets.

2.1 Visualization Libraries

There are several powerful open-source libraries available, such as D3.js, Plotly.js, and Chart.js, which provide a wide range of customizable charts, graphs, and interactive visualizations. These libraries offer flexibility and ease of use, allowing developers to create compelling visual representations of data.

My choice for this project fell to Chart.js, a popular open-source JavaScript library that provides a simple and flexible way to create interactive and visually appealing charts and graphs on web pages. It offers a wide variety of chart types, including line charts, bar charts, pie charts, radar charts, and more.

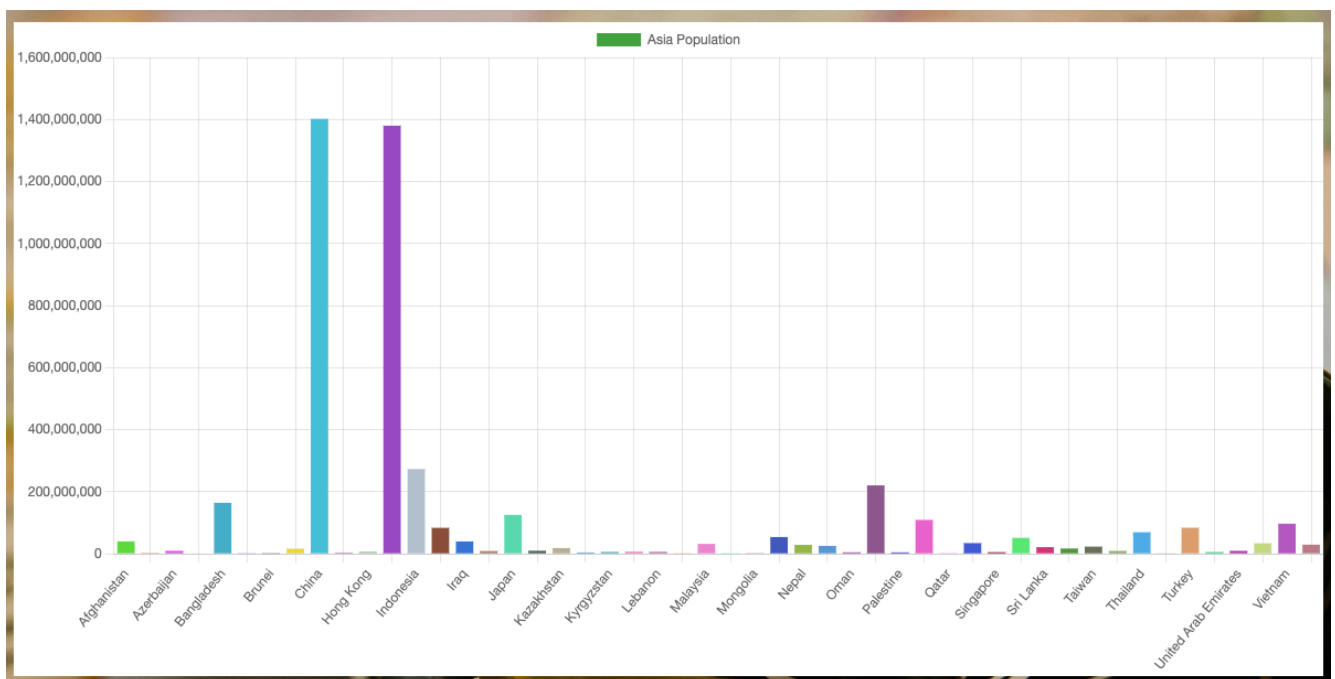


Figure 1 Screenshot from a chart built with Chart.js, the library of choice for my project.

2.2 Interactive Dashboards

Tools like Tableau, Power BI, and QlikView have gained popularity in creating interactive dashboards that enable users to explore and analyze data visually. These platforms provide intuitive interfaces, drag-and-drop functionality, rich visualization options and various visualization options to create dynamic and interactive data presentations.



Figure 2 Screenshot from Power BI, interactive dashboard from Microsoft.

Power BI has gained popularity due to its user-friendly interface, extensive visualization options, and integration capabilities with other Microsoft products. It caters to a wide range of users, from business analysts and data professionals to executives and decision-makers, enabling them to gain actionable insights from their data through interactive and visually compelling dashboards and reports.

2.3 Virtual Reality (VR) and Augmented Reality (AR)

VR and AR technologies are being increasingly used in data visualization. They provide immersive experiences, allowing users to visualize and interact with data in three-dimensional space. These technologies offer new perspectives for data exploration, particularly in fields like scientific research, architecture, and medical imaging.



Figure 3 Which technology to choose between virtual reality and augmented reality for a professional application?

2.4 World Data Bank

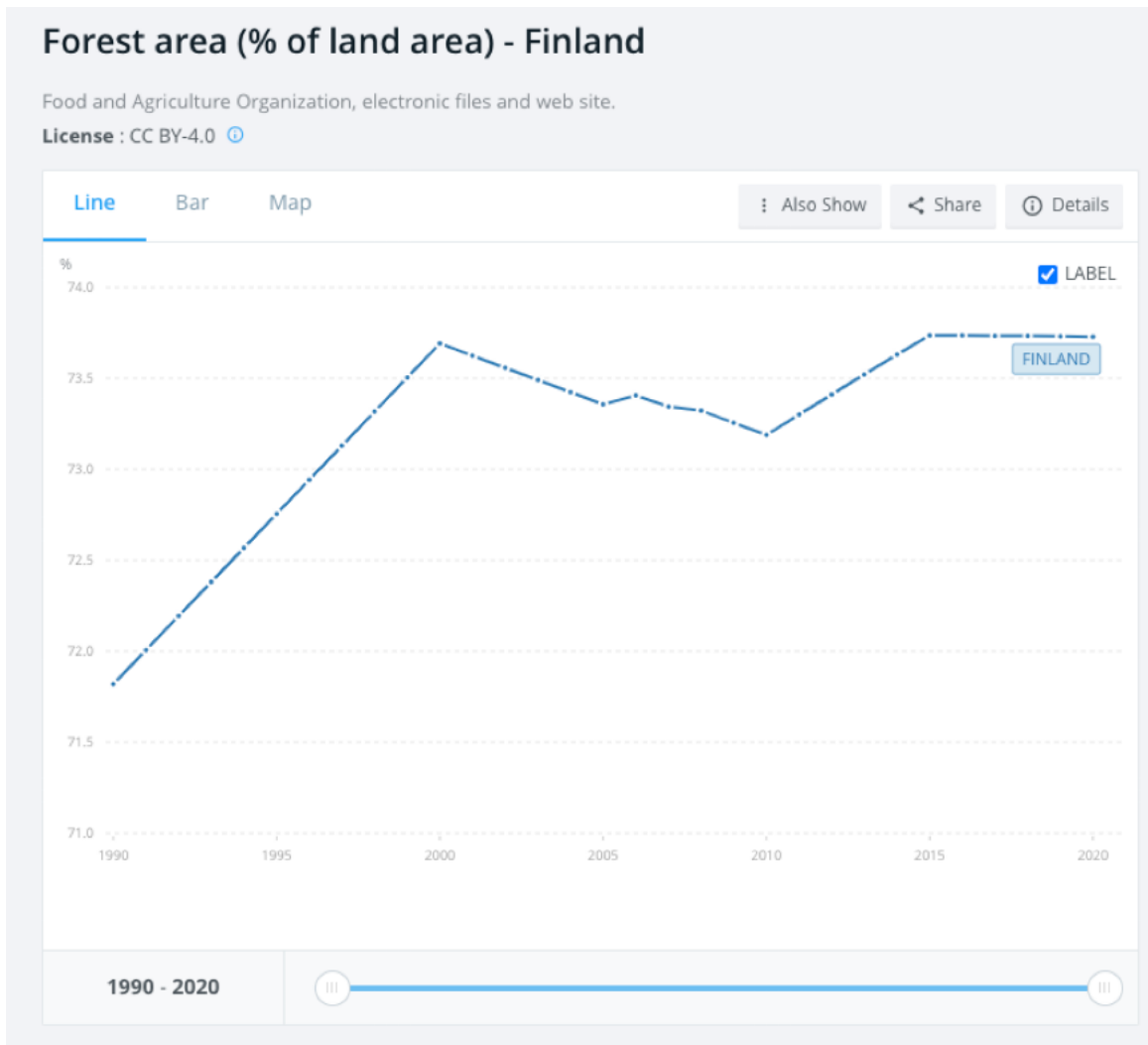


Figure 4 Screenshot from World Data Bank web application

World Data Bank provides users with data on various indicators across countries worldwide. The application allows users to select countries, indicators, and time periods, and then presents the data in various formats, including line charts, bar charts, and maps. However, the application is primarily focused on economic data and lacks data on cultural and social aspects of countries.

2.5 Worldometer

Another notable example in the realm of data visualization and global statistics is Worldometer. Worldometer is a comprehensive platform that offers real-time statistics on various global issues, including population, health, education, environment, economy, and more. It serves as a valuable resource for individuals, researchers, and organizations seeking up-to-date information on worldwide trends and metrics.



Figure 5 Screenshot from Worldometer web application

One of the key features of Worldometer is its use of charts and graphs to present data in a visually engaging and easily understandable manner. By employing various types of visualizations, such as line charts, bar graphs, and pie charts, Worldometer effectively communicates complex data sets and enables users to grasp significant trends and patterns briefly. These visualizations enhance the accessibility and interpretability of the information, facilitating data-driven insights and decision-making processes.

Furthermore, Worldometer provides users with customization options to tailor the visualizations according to their preferences. Users can often adjust parameters such as time frames, geographical regions, and specific metrics of interest to focus on the data that is most relevant to them. This flexibility allows users to personalize their data exploration and gain a deeper understanding of the global issues that matter to them.

It is important to note, however, that Worldometer primarily focuses on global statistics and aggregates data from various sources. While it offers a comprehensive view of worldwide trends and metrics, it does not provide detailed data on individual countries. For more specific country-level data, other resources such as the REST Countries API or dedicated country-specific databases would be more suitable.

These applications highlight the importance of data visualization tools for understanding complex global issues. However, there are gaps in the existing applications that can be addressed by the proposed project. The proposed project aims to provide users with a comprehensive set of country data visualization tools with a sleek and easy to approach interface. The application also allows users to interact with the data, customize visualizations, and compare countries based on different criteria. In this context, the proposed project is positioned as a valuable resource for researchers, policymakers, educators, and students who need access to comprehensive and customizable country data visualization tools.

3 METHODOLOGY

3.1 React

React is a popular JavaScript library widely used for building modern and interactive user interfaces. It offers a component-based approach to UI development, allowing developers to create reusable and modular UI components. React efficiently manages the UI state and efficiently renders changes to the UI, resulting in fast and seamless user experiences. With its large community, extensive documentation, and thriving ecosystem, React is a go-to choice for building scalable and performant web applications.

3.2 Chart.js

Chart.js is a popular open-source JavaScript library that allows developers to create responsive and customizable charts and graphs for web applications. It provides a wide range of chart types, including line, bar, pie, radar, and scatter charts, and supports a variety of data formats, including JSON, CSV, and array. This library has a simple and intuitive API that makes it easy to create charts with customizable styles and animations. It also offers several useful features such as tooltips, legend, and data labels, and supports plugins that extend its functionality.

One of the key advantages of Chart.js is its responsiveness, which enables the charts to be easily adapted to different screen sizes and devices. Additionally, Chart.js is lightweight and fast, making it an ideal choice for creating dynamic and interactive data visualizations. Chart.js can be used with a wide range of web development technologies, including React, Angular, and Vue.js. It is widely adopted and has a large and active community, which provides support, documentation, and plugins to extend its functionality.

3.3 Tailwind

Tailwind CSS is a popular open-source utility-first CSS framework that allows developers to rapidly build responsive and modern user interfaces. It provides a comprehensive set of pre-designed CSS classes that can be used to style HTML elements and components without writing custom CSS. Tailwind CSS is based on a utility-first approach, which means that it provides many low-level classes that can be combined to create custom styles. These classes are designed to be easy to read and understand and can be applied directly to HTML elements using class attributes.

Tailwind CSS includes a wide range of pre-built styles for common user interface elements such as typography, colors, spacing, and layout. It also supports responsive design, allowing developers to easily create layouts that adapt to different screen sizes and devices. One of the key advantages of Tailwind CSS is its flexibility and customization options. Developers can easily create custom styles by modifying the existing classes or by creating their own custom classes. Tailwind CSS can be used with a wide range of web development technologies, including React, Angular, and Vue.js. It is widely adopted and has a large and active community, which provides support, documentation, and plugins to extend its functionality.

3.4 Flowbite

Flowbite is a front-end framework that provides a set of pre-built HTML, CSS, and JavaScript components for building responsive and modern web applications. It is based on the Bootstrap framework, but with added features and improvements. Flowbite includes a wide range of UI components such as buttons, forms, modals, tabs, and navigation bars. It also provides pre-designed layouts and templates that can be easily customized and adapted to different use cases. One of the key advantages of Flowbite is its ease of use and flexibility. Developers can quickly build and customize user interfaces using the pre-built components, without having to write complex custom code. The framework also provides responsive design features, making it easy to create interfaces that work well on different screen sizes and devices.

Flowbite is compatible with a variety of front-end development technologies, including React, Angular, and Vue.js. It also integrates with popular tools such as Webpack and Gulp, making it easy to use in existing development workflows.

3.5 React Router

React Router is a popular library for building client-side routing in React applications. It allows developers to create and manage multiple routes within a single-page application, allowing for a more seamless and dynamic user experience. React Router provides a simple and intuitive API for defining routes and handling navigation within a React application. It supports a range of routing options, including nested routes, dynamic routes, and route parameters. This allows developers to create complex user interfaces with multiple views and pages, while still maintaining a single-page application architecture. React Router also provides a range of advanced features, such as code splitting and lazy loading, which can improve application performance by reducing the initial load time and optimizing resource usage.

One of the key advantages of React Router is its tight integration with the React ecosystem. It is designed to work seamlessly with other React libraries and tools, such as Redux and React Native, allowing for a more integrated and streamlined development experience.

Overall, React Router is a powerful and flexible library for building client-side routing in React applications. Its intuitive API, advanced features, and tight integration with the React ecosystem make it a popular choice for developers looking to create dynamic and responsive user interfaces.

3.6 Axios

Axios is a popular JavaScript library used to make HTTP requests^{xiii} from web browsers or Node.js applications. It provides an easy-to-use API that allows developers to send asynchronous HTTP requests to RESTful endpoints and perform CRUD operations (Create, Read, Update, Delete) on data. With Axios, developers can configure and customize HTTP requests, handle HTTP responses, and intercept requests and responses to modify them. The library also supports various formats for data transmission, including JSON, XML, and form data. Axios is often used in combination with other popular web development libraries, such as React and Redux, as it seamlessly integrates with these tools. Its intuitive API and comprehensive documentation make it a preferred choice among developers for handling data fetching and HTTP requests in their web applications.

3.7 Typescript

TypeScript is a statically typed superset of JavaScript that provides developers with a range of powerful features and tools for building scalable and maintainable applications. By adding a type system to JavaScript, TypeScript enables developers to catch errors and bugs earlier in the development process and provides better tooling and code intelligence to improve productivity. TypeScript is designed to be compatible with existing JavaScript code, making it easy to integrate into new or existing projects.

Some of the key benefits of TypeScript include:

Type safety: TypeScript allows developers to define types for variables, functions, and other entities, which helps catch errors before runtime.

Improved productivity: TypeScript provides better code intelligence, auto-completion, and refactoring tools, which make developers more productive.

Scalability: TypeScript's static typing helps developers write code that is easier to maintain and scale over time.

Compatibility with JavaScript: TypeScript is a superset of JavaScript, which means it can be used with existing JavaScript codebases without major changes.

3.8 Express.js

Express.js is a popular backend web framework for Node.js. It is designed to help developers create web applications and APIs quickly and easily. Express provides a set of powerful features for routing, middleware^{xiii}, and error handling, making it a versatile tool for building complex web applications. One of the key features of Express is its routing system. It allows developers to define routes for handling incoming requests from clients and map them to specific functions or controllers that handle the logic and generate the response. This makes it easy to organize the application's endpoints and improve code readability. Express also provides a range of middleware functions that can be used to customize the behavior of the application. Middleware can perform various tasks, such as logging requests, parsing incoming data, and handling authentication and authorization. Developers can also create custom middleware to add specific functionality to their application.

Another advantage of Express is its support for a wide range of third-party packages and plugins. These can be used to add additional functionality to the application, such as database integration, caching, and security features. Express.js is a powerful and flexible framework that provides developers with the tools they need to build scalable and maintainable web applications. Its modular design, robust routing system, and extensive middleware support make it a popular choice for Node.js developers.

3.9 ESLint

ESLint ^{xiv} is a widely used static code analysis tool for identifying problematic patterns found in JavaScript code. It checks the code for potential errors, coding best practices, and syntax errors. ESLint is highly configurable, and it can be customized to suit a specific codebase's needs. It allows developers to define and enforce coding standards to maintain consistency throughout the codebase. ESLint can also be integrated into the development environment, enabling developers to detect errors as they write code, making the development process more efficient. By identifying potential issues in the codebase early on, ESLint helps prevent bugs and improves the overall quality of the code. Additionally, ESLint can be used with other tools such as code editors, continuous integration servers, and other build tools to provide automated code analysis and enforcement. ESLint is a valuable tool for developers working on large codebases, where maintaining consistency and avoiding common errors is crucial for efficient and reliable development.

4 BACKEND DEVELOPMENT

The server that I created is a Node.js server that uses the Express.js framework to handle HTTP requests. The server provides two endpoints that return data from the Rest Countries API:

`/countries` - This endpoint returns all countries in the API, with the following fields: name, region, capital, population, flags, languages, cca2, cca3, currencies, subregion, latlng, area, coatOfArms. When a GET request is made to this endpoint, the server sends a request to the Rest Countries API, gets the response data, and sends it back to the client as the response.

`/countries/:region` - This endpoint returns all countries in a specific region, with the name and population fields. The region is specified as a URL parameter. When a GET request is made to this endpoint, the server sends a request to the Rest Countries API, passing the region parameter in the URL, gets the response data, and sends it back to the client as the response.

The server uses the Axios library to make HTTP requests to the Rest Countries API. It uses `async/await` syntax to handle asynchronous operations and `try/catch` blocks to handle errors. The server also uses the `CORS` ^{xv}middleware to enable cross-origin resource sharing, allowing clients from different domains to access the API. The server listens on port 3001 and logs a message to the console when it starts running.

5 FRONTEND DEVELOPMENT

Let's dive into a detailed explanation of the React application, rendering of backend queries, user input handling, and user interface (UI) design. The application's frontend is built using React, a popular JavaScript library for building user interfaces. React allows developers to create reusable UI components that update dynamically based on changes in the application's state. In this section the two main components, Data By Country and Data By Region, will be found, with all their relatives high level functions explained.

5.1 Data By Country

```
dispatch(fetchCountriesByNameThunk(search));
```

Code 1 Code snippet of thunk function in the application.

In this code snippet, the application likely communicates with a backend system to fetch data. This is done using Redux thunk^{xvi} actions, which are asynchronous actions that are dispatched using the `useDispatch` hook from the React-Redux library. The `fetchCountriesByNameThunk` action is dispatched with the search input provided by the user, triggering an API request to the backend to fetch country data based on the search input. The retrieved data is then stored in the Redux store, which can be accessed using the `useSelector`^{xvii} hook to retrieve the updated state.

```
const onChange = (e: ChangeEvent<HTMLInputElement>) => {
  const MIN_SEARCH_LENGTH = 3;
  if (e.target.value.length < MIN_SEARCH_LENGTH) {
    setSearch("");
  } else {
    setSearch(e.target.value);
    dispatch(fetchCountriesByNameThunk(search));
  }
};
```

Code 2 Code snippet from of onChange function from the application.

In this code snippet the user input is handled using the `useState`^{xviii} hook from React. The `useState` hook allows the application to create and manage state variables. In this case, the search input provided by the user is stored in the `search` state variable, which is updated whenever the user types in the search bar. The `onChange` function is called whenever the input value changes, updating the `search` state variable. It also checks the length of the input against a minimum search length requirement before dispatching the `fetchCountriesByNameThunk` action to retrieve country data from the backend. The UI design in the code is implemented using various custom components that are imported from other files and styled using Tailwind CSS, a popular CSS utility framework. These components, such as `CountryTable`, `SearchAppBar`, `BackButton`, and `Footer`, are used to structure the UI and provide a seamless user experience with visually appealing styles. The UI design likely includes a header with a title, a search bar for inputting search queries, and a table to display the retrieved country data, all styled using Tailwind CSS classes.

Tailwind CSS makes it easy to create responsive and modern UI designs with its extensive set of pre-designed CSS classes that can be easily applied to HTML elements. For example, the search bar may have classes like `bg-white`, `rounded-lg`, and `shadow-md` to give it a clean and polished appearance. The table may have classes like `border-collapse`, `w-full`, and `table-auto` to provide a responsive and visually appealing layout. Furthermore, the UI design also accounts for cases where the user has not entered any search input or if the search input is invalid. For instance, when there is no search input, a prompt message is likely displayed using Tailwind CSS classes like `text-gray-500` and `text-sm` to instruct the user to search for a country.

When no matching countries are found based on the search input, an appropriate message is likely displayed using Tailwind CSS classes like `text-red-500` and `text-sm` to indicate that the country was not found, with proper spacing and alignment using classes like `mt-4` and `text-center`.

5.2 Data By Region

The component conditionally renders the `PopulationChart` and `AreaChart` components based on the values of `popChart` and `areaChart` state variables respectively. If `popChart` is `true`, the `PopulationChart` component is rendered, and if `areaChart` is `true`, the `AreaChart` component is rendered.

```
<div>
  {popChart ? (
    <div>
      <PopulationChart />
    </div>
  ) : null}
  {areaChart ? (
    <div>
      <AreaChart />
    </div>
  ) : null}
</div>
```

Code 3 Code snippet from the conditional rendering section from the application

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

Code 4 Code snippet showing the `useEffect` hook dispatching the `fetchCountries` thunk.

In the above code snippet, the `PopulationChart` and `AreaChart` components fetch data from the backend API using the `useEffect` React hook and render the retrieved data in the form of charts, such as bar chart and pie chart, respectively.

```
const [popChart, showPopChart] = useState(false);
const [areaChart, showAreaChart] = useState(false);
```

Code 5 Two different `useState` handling the state of the two chart component.

The component uses `useState` hook to manage the state of two boolean variables `popChart` and `areaChart`, which determine whether to show the `PopulationChart` and `AreaChart` components respectively.

```
const handlePopChart = () => {
  showPopChart(true);
  showAreaChart(false);
};

const handleAreaChart = () => {
  showAreaChart(true);
  showPopChart(false);
};
```

Code 6 Here the actual conditional rendering is shown.

The component defines two event handler functions, `handlePopChart` and `handleAreaChart`, which are triggered when the "Show Bar Chart" and "Show Pie Chart" buttons are clicked respectively. When the "Show Bar Chart" button is clicked, `handlePopChart` function is called, setting `popChart` to `true` and `areaChart` to `false`, which will render the `PopulationChart` component. When the "Show Pie Chart" button is clicked, `handleAreaChart` function is called, setting `areaChart` to `true` and `popChart` to `false`, which will render the `AreaChart` component.

The UI design of the `DataRegion` component includes a title, a description, and two buttons for selecting the type of chart to display.

The UI is structured using a flexbox layout with a main column that grows to fill the available space, and a footer that is aligned to the bottom of the page.

The UI uses classes from the Tailwind CSS framework to style various elements such as the title (`text-4xl`, `font-extrabold`, etc.), buttons (`bg-blue-700`, `rounded-lg`, etc.), and footer (`mt-auto`, etc.).

The UI also includes conditional rendering of the `PopulationChart` and `AreaChart` components based on the state of `popChart` and `areaChart` variables, allowing the user to switch between different chart views.



Figure 6 Tailwind made my design planning much easier than expected. Image credits: (<https://blog.logrocket.com/comparing-tailwind-css-bootstrap-time-ditch-ui-kits/>)

5.3 Data Visualization

In this section the details on how the data is rendered in the application will be discussed, mentioning specifically the components used from Chart.js library.

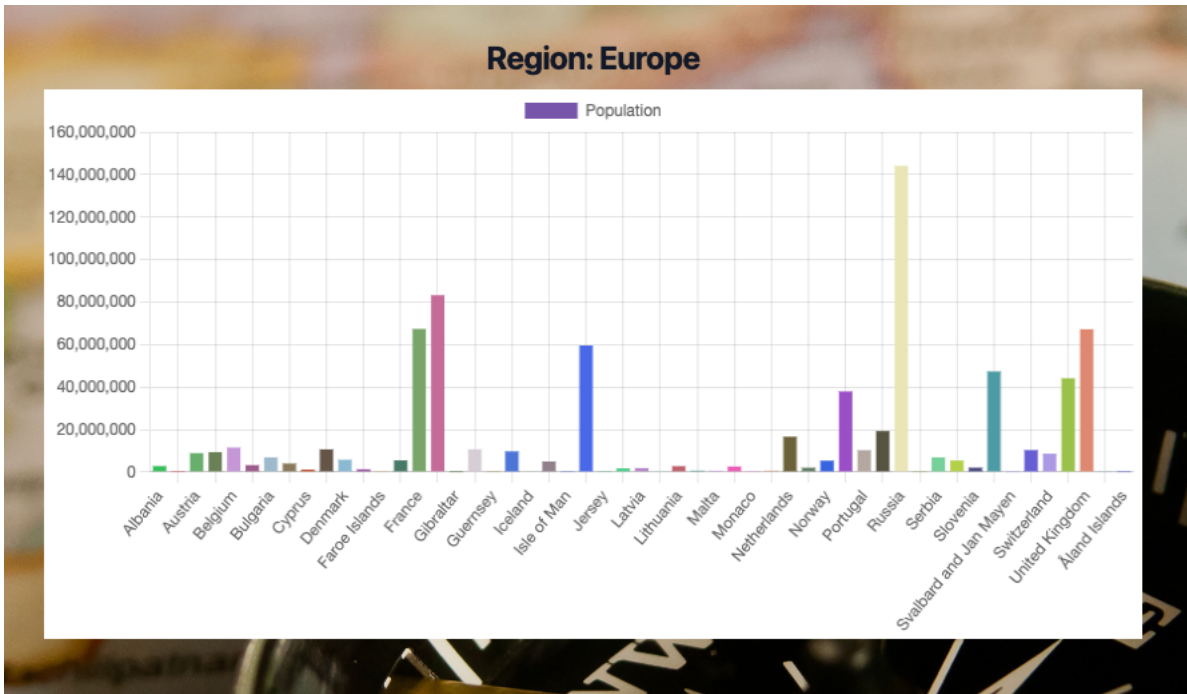


Figure 7 Screenshot from Data By Region section showing a bar chart from Chart.js library.

This component is a functional component called `PopulationChart` that contains a button group for selecting a region and a bar chart that displays the population data. The `useState` hook is used to manage the state of the selected region, and the `useEffect` hook is used to fetch data from an external API when the component is mounted. The `handleRegion` function is called when a region button is clicked and updates the selected region state. The code then maps the names and population data for each country in the selected region and creates an array of labels and data points for the bar chart. A random color function is used to generate a different color for each bar in the chart. Finally, the `Chart` component from the `react-chartjs-2` library is used to display the bar chart on the page.

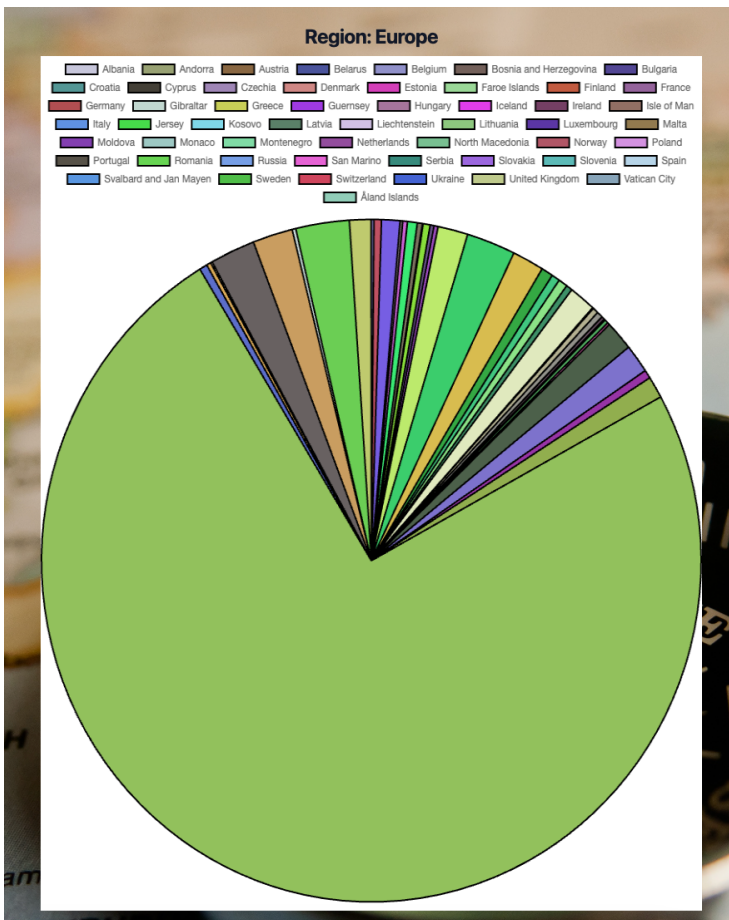


Figure 8 Screenshot from Data By Region section, showing a pie chart from Chart.js library.

This is a functional component called `AreaChart` that renders a chart showing the area of different countries based on the selected region. The chart is created using the `Chart.js` library and displayed using the `react-chartjs-2` component. The component uses the `useSelector` hook to access the `countries` object from the Redux store, which contains an array of countries. It also uses the `useDispatch` hook to dispatch actions to the Redux store, which are defined in the `countriesSlice` file. When the component is mounted, it dispatches a `fetchCountriesThunk` action to fetch the list of countries from an API. The `handleRegion` function is called when the user clicks on one of the region buttons,

which sets the `region` state to the selected region and dispatches a `handleSort` action with a value of 0 to sort the countries alphabetically.

The `names` and `area` arrays are created by mapping over the `countries` array and filtering by the selected `region`. The `names` array contains the names of the countries and the `area` array contains the area of the countries. The `undefined` and negative values are removed from both arrays using the `splice` method. The `data` object is used to configure the chart, setting the chart type to "Bar", the labels to the `names` array, and the data to the `area` array. A random color is generated for the chart using the `randomRGB` function. Finally, the component returns a div containing a group of buttons for selecting the region and the chart itself, displayed using the `Chart` component. The selected region is displayed above the chart.

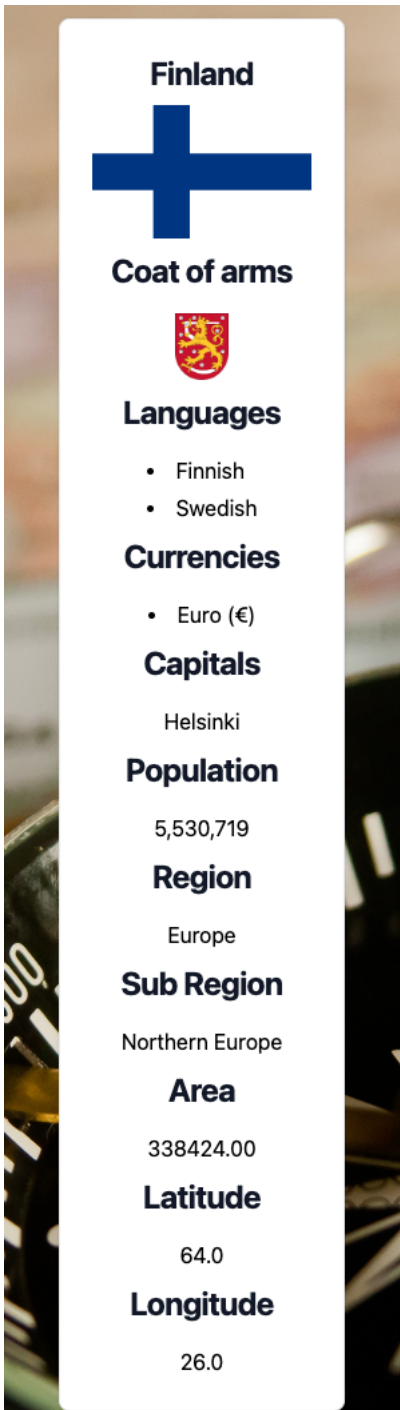


Figure 9 Data By Country table

This is a functional component that create a table of information about countries. The code exports a function called CountryTable, which takes in an object with a property called "filter" of type BasicTable. If the filter property is not provided, the function returns a loading message. Otherwise, it renders a div that contains information about each country in the filter array. Each country is displayed with its name, flag, coat of arms, languages, currencies, capital(s), population, region, sub-region, area, latitude, and longitude. These details are displayed in a series of divs, with headings for each piece of information and the information displayed underneath it.

In this component, Tailwind CSS classes are used to define the layout and appearance of the table, such as flex, flex-wrap, justify-center, etc. These classes are used in conjunction with regular HTML elements and React hooks like useState and useEffect to manage the state and lifecycle of the component.

6 IMPLEMENTATION

Throughout the development process of my project, I encountered various challenges that required creative solutions. One of the most significant challenges was handling and presenting large amounts of data from the REST Countries API in a meaningful and easy-to-understand way for the user. To overcome this challenge, I incorporated data visualization tools such as tables and graphs with Chart.js to help the user quickly comprehend the information. Another challenge that I encountered was managing the application's state and ensuring data consistency across components. To address this, I opted to use Redux for state management, which allowed me to store and manage all application data in a centralized location called a store, enabling me to manipulate that data using actions and reducers. Additionally, I utilized TypeScript for type checking and better code organization to ensure data consistency and prevent potential errors.

In addition, I faced difficulties with the UI design and responsiveness, particularly when integrating components from the Chart.js library. To overcome this challenge, I implemented Tailwind for custom styling and fine-tuned the CSS to achieve a consistent and responsive user interface that delivered an intuitive user experience. Throughout the development process, I focused on significant milestones such as implementing API requests, presenting data in tables and graphs, and creating a user-friendly interface with smooth navigation. This process provided me with valuable insights into effective data visualization techniques, best practices for state management, and the importance of responsive design for user experience.

7 RESULTS AND EVALUATION

As the developer of the application, I am thrilled to present the final product, which encompasses an array of features and functionalities that I have carefully crafted to provide users with a seamless and immersive experience for exploring data on different countries. One of the key highlights of the application is its responsive user interface design, which I have meticulously crafted to ensure that it looks visually appealing and functions flawlessly across various devices, including desktops, tablets, and mobile phones. The intuitive and user-friendly interface allows users to effortlessly navigate the application and access a myriad of features, including filtering and sorting data, viewing data in different formats such as tables and charts, and searching for specific countries based on their preferences or requirements. Performance has been a top priority in the development of the application. Leveraging cutting-edge technologies like React, Redux, and Chart.js, I have ensured that the application is optimized for speed and efficiency, with minimal loading times and lightning-fast response times, using API queries only when necessary. The use of Redux for state management has facilitated smooth and consistent data handling across different components, ensuring that users have a seamless experience while interacting with the application.

To ensure that the application meets the needs and expectations of our users, I conducted extensive user testing and collected feedback from a diverse group of users. The feedback received has been overwhelmingly positive, with users commending the application for its ease of use, comprehensive functionality, and exceptional responsiveness. This valuable feedback has served as a valuable source of insights, which I have utilized to make further improvements to the application, including adding new features, refining the user interface design, and addressing any issues or concerns raised by users.

In addition to its robust performance and user-friendly design, the application also boasts an array of data visualization techniques that facilitate effective data exploration. The integration of Chart.js has allowed users to visualize data in a meaningful and easy-to-understand way through visually appealing charts and graphs. This has enabled users to quickly comprehend and analyze data on different countries, making the application a powerful tool for data exploration and analysis. Furthermore, the application places a strong emphasis on data consistency and accuracy. Leveraging the power of TypeScript, I have implemented strong type checking and enforced strict data validation measures to ensure that the data presented to users is accurate, reliable, and up to date. This has helped to prevent potential errors and inconsistencies in the data, providing users with a trustworthy source of information on different countries.

8 CONCLUSION

In conclusion, the journey of developing this application has been a fulfilling and rewarding experience. The application has successfully achieved its objectives by providing users with a reliable, engaging, and feature-rich tool for exploring data on different countries. The integration of cutting-edge technologies such as React, Redux, and Chart.js has resulted in optimal performance, efficient data handling, and visually appealing data visualization. The application's user-centric approach, with a responsive interface design, intuitive navigation, and comprehensive functionality, has garnered overwhelmingly positive feedback from users, who have praised its ease of use and exceptional responsiveness. The robust implementation of Chart.js has facilitated effective data visualization, allowing users to quickly comprehend and analyze data on different countries, making the application a valuable tool for data exploration and analysis.

Furthermore, the application's strong emphasis on data consistency and accuracy, achieved through TypeScript and strict data validation measures, has ensured that users have access to reliable and up-to-date information. This has enhanced the credibility and trustworthiness of the application as a source of accurate data on different countries. As I reflect on this development journey, I am grateful for the invaluable learning experience it has provided me. I have honed my skills in front-end development, data visualization, and data handling, which have prepared me for the challenges and opportunities of pursuing a master's degree in Geoinformatics. I am committed to continually improving and expanding the capabilities of the application based on user feedback and evolving needs.

In conclusion, I am proud of the achievements of this application and excited about the future possibilities for growth and improvement. The development of this application has been a significant milestone in my professional journey, and I look forward to utilizing the skills and knowledge gained to further contribute to the field of Geoinformatics.

APPENDIX 1/1

Welcome to the GitHub repository for my bachelor's thesis project, "Countries Data Visualization System: exploring country data through interactive visualizations". The purpose of this repository is to provide a comprehensive demonstration of the project's code, features, and functionality. The project aims to develop a data visualization application using React that can fetch data from the REST Countries API and display it in user-friendly tables and graphs. The application also includes features such as search, filtering, and sorting to enable users to find specific information quickly.

To use and run this project, the user will need the following software and tools installed:

Node.js (v14 or higher)

npm (v7 or higher)

The user will also need access to the internet to download any necessary dependencies during the installation process. This repository already comes with all the needed packages. In case the user wants to install manually, please check the dependencies in the `package.json` file.

Here are the steps to clone and set up the project on a local machine:

Open the terminal or command prompt, navigate to the directory where the user wants to clone the repository and run the following command to clone the repository:

```
git clone https://github.com/tidumarco/countries-data-visualization-system.git
```

Once the repository is cloned, navigate to the project directory by running the following command:

```
cd countries-data-visualization-system
```

To install the client side of the application, run in the root folder:

```
yarn install
```

And then run it with:

```
yarn start
```

APPENDIX 1/2

To install the backend, open another terminal, navigate to:

```
cd server
```

And then install the required dependencies with:

```
yarn install
```

And to run it:

```
yarn start
```

Now the frontend will be accessible via the endpoint:

```
http://localhost:3000
```


APPENDIX 2/1 – USAGE GUIDE

After having successfully installed and launched the project, the user can access the application in a web browser by visiting <http://localhost:3000>.

In the homepage there will be two buttons:



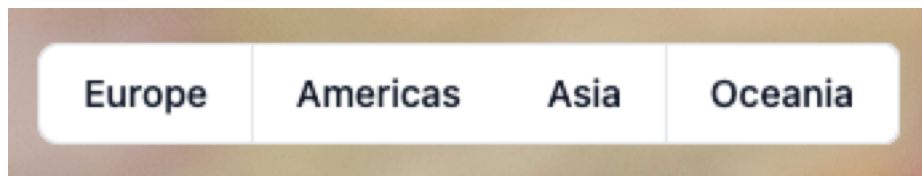
After clicking the “Data by region” button, the user will be presented with two choices:



Clicking on:



And then on one of the regions shown:



This chart will be displayed:

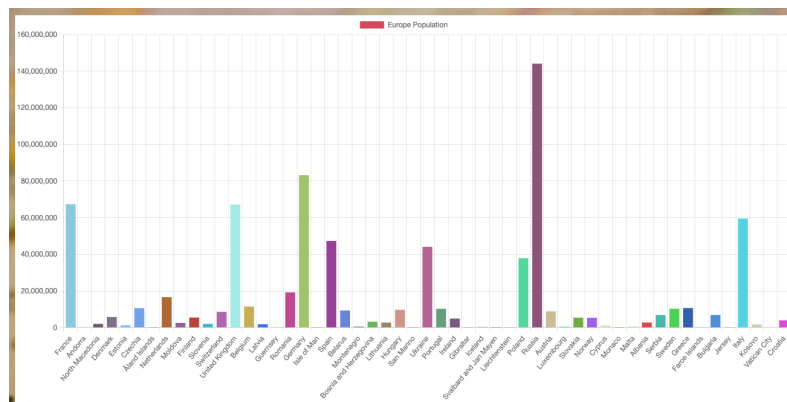


Figure 6 Bar chart showing Europe's population divided by country.

APPENDIX 2/2

Instead, if the user wants to move to the pie chart visualization click on:



And the user will be welcomed with something like this:

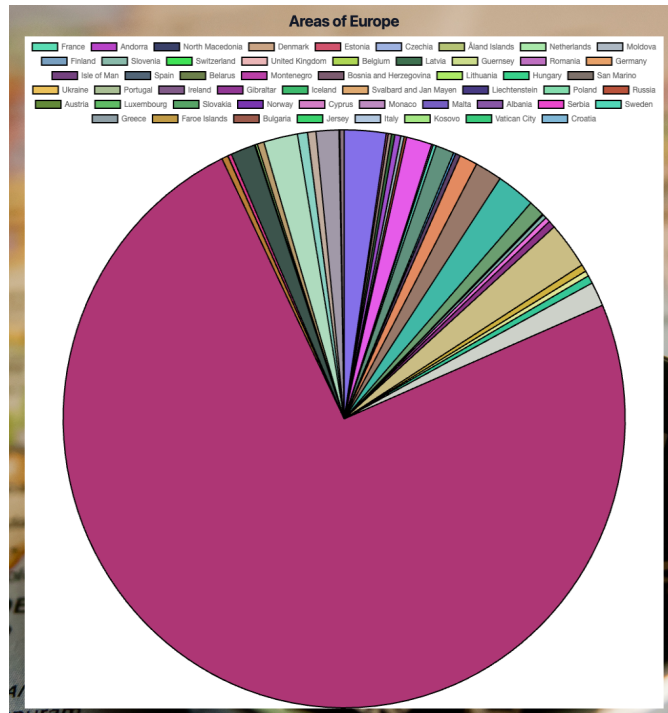
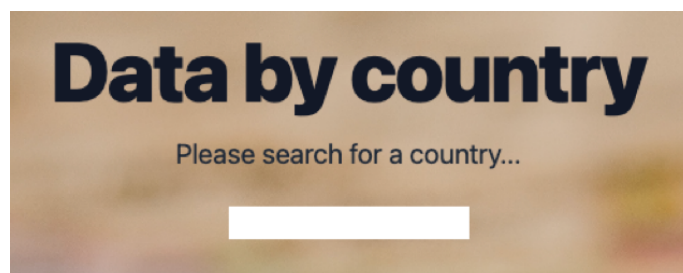


Figure 7 Pie chart showing Europe's countries areas.

From the homepage, after clicking on this button:

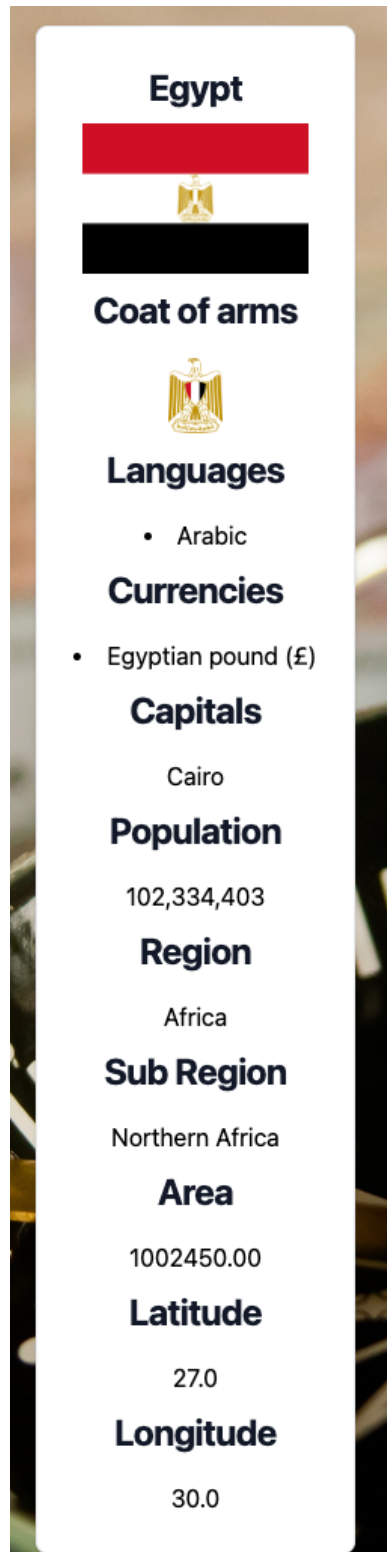


The user will be prompted to enter a country to search:




APPENDIX 2/3


And after inputting the country of choice this will be displayed:



Egypt



Coat of arms



Languages

- Arabic

Currencies

- Egyptian pound (£)

Capitals

Cairo

Population

102,334,403

Region

Africa

Sub Region

Northern Africa

Area

1002450.00

Latitude

27.0

Longitude

30.0

APPENDIX 3 – TROUBLESHOOTING AND SUPPORT

If the user encounters any issues while setting up or using the application, there are some troubleshooting steps that can be followed. For example, checking which npm and yarn version is installed, I warmly recommend using yarn as package manager, less prone to error and more reliable.

Another option is to clear cache and cookies. Sometimes, issues can arise due to cached data or stored cookies. Try clearing the browser's cache and cookies and then reload the application.

The user can also ensure that dependencies are installed correctly: make sure that all necessary dependencies are installed and up to date. Refer to the Prerequisites section of the Project Manual to ensure all required dependencies are installed.

Remember also to check the documentation such as the README .md file in the root directory of the project for additional guidance on setting up and running the application.

If the user is still experiencing issues or need further assistance, please reach out to me: my contacts can be found in the footer of the application and if not running they are in the README.md or create a new issue on the GitHub repository page. I will do my best to be of assistance as soon as possible.

REFERENCES

- ⁱ (Fayder Florez, 2023). REST Countries API documentation. Retrieved May 11, 2023, from <https://restcountries.com/>
- ⁱⁱ (Andrew Clark, 2023). In Official React Documentation. Retrieved May 11, 2023, from <https://reactjs.org/>
- ⁱⁱⁱ (Dan Abramov, 2023). In Redux Documentation. Retrieved May 11, 2023, from <https://redux.js.org/>
- ^{iv} (OpenJs Foundation, 2017). In Express.js Documentation. Retrieved May 11, 2023, from <https://expressjs.com/>
- ^v (OpenJs Foundation, 2023). In Node.js Documentation. Retrieved May 11, 2023, from <https://nodejs.org/>
- ^{vi} (Tailwind Labs, 2023). In Tailwind CSS Documentation. Retrieved May 11, 2023, from <https://tailwindcss.com/>
- ^{vii} (Flowbite, 2023). In Flowbite Documentation. Retrieved May 11, 2023, from <https://flowbite.com/>
- ^{viii} (Nick Downie, 2023). In Chart.js Documentation. Retrieved May 11, 2023, from <https://www.chartjs.org/>
- ^{ix} (Gamma, E., Helm, R., Johnson, R., & Vlissides, J. 1995). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional.
- ^x (Adhikari, P., & Shrestha, R., 2020). Building Web Applications with MERN Stack: A Comprehensive Guide to Implement the MERN Stack from Scratch. Packt Publishing.
- ^{xi} (Jim VanderHei, 2023). In Axios Documentation. Retrieved May 11, 2023, from <https://axios-http.com/>
- ^{xii} (Fielding, R. T., Gettys, J., Mogul, J. C., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T., 1999). Hypertext Transfer Protocol -- HTTP/1.1. IETF. RFC 2616.
- ^{xiii} (Klein, D., 2014). The Architecture of Open-Source Applications: The Middleware Web Framework Middleware. Lulu.com.
- ^{xiv} (Nicholas C. Zakas, 2013). In ESLint Documentation. Retrieved May 11, 2023, from <https://eslint.org/>
- ^{xv} (van Kesteren, A., & Barth, A., 2014). Cross-Origin Resource Sharing. W3C. Retrieved from <https://www.w3.org/TR/cors/>
- ^{xvi} (Rehor, D., 2020). Redux Thunk: What It Is and How to Use It. Retrieved from https://medium.com/@dave_rehor/redux-thunk-what-it-is-and-how-to-use-it-f7a9c7eff7d3
- ^{xvii} (Dan Abramov, 2023) useSelector API Documentation. Retrieved from <https://react-redux.js.org/api/hooks#useselector>
- ^{xviii} (Andrew Clark, 2023). useState API Documentation. Retrieved from <https://reactjs.org/docs/hooks-state.html>