Web Development using MERN Stack

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

DISHA GOYAL

20BCE2048

Under the guidance of

Dr. Parthiban K

School of Computer Science and Engineering,

VIT, Vellore.

May, 2024

DECLARATION

I hereby declare that the thesis entitled "**Web Development using MERN Stack**" submitted by me, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering* to VIT is a record of bonafide work carried out by me under the supervision of Dr. Parthiban K.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place : Vellore

Date : 08.05.2024

Signature of the Candidate

CERTIFICATE

This is to certify that the thesis entitled "**Web Development using MERN Stack**" submitted by **Disha Goyal (20BCE2048)**, School of Computer Science and Engineering, VIT, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering*, is a record of bonafide work carried out by him / her under my supervision during the period, 01.12.2023 to 30.04.2024, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The thesis fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place : Vellore

Date : 08.05.2024 **Signature of the Guide**

Internal Examiner **External Examiner**

Dr. Umadevi K S

Computer Science & Engineering

ACKNOWLEDGEMENTS

It is my pleasure to express with deep sense of gratitude to Dr. Parthiban K, Assistant Professor Sr. Grade 1, *School of Computer Science and Engineering,* Vellore Institute of Technology, for his constant guidance, continual encouragement, understanding; more than all, he taught me patience in my endeavor. My association with him is not confined to academics only, but it is a great opportunity on my part to work with an intellectual and expert in the field of Computer Science.

I would like to express my gratitude to Dr. G.Viswanathan, Mr. Sankar Viswanathan, Dr. Sekar Viswanathan, Dr. G V Selvam, Dr. V S Kanchana Bhaaskaran & Dr. Partha Sharathi Mallick, and Dr. Ramesh Babu K, School of Computer Science and Engineering, for providing an environment to work in and for his inspiration during the tenure of the course.

In jubilant mood I express ingeniously my whole-hearted thanks to all the teaching staff and members at VIT, working relentlessly at our university, for their enthusiasm coupled with timely encouragements showered on me with zeal, which prompted the acquirement of the requisite knowledge to finalize my course study successfully. I would also like to thank my parents for their support.

It is indeed a pleasure to thank my friends who persuaded and encouraged me to take up and complete this task. At last but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.

Disha Goyal

EXECUTIVE SUMMARY

In today's digital age, web development plays a crucial role in creating interactive and user-friendly applications. Among the myriad of technologies and frameworks available, the MERN (MongoDB, Express.js, React.js, Node.js) stack has emerged as a powerful tool for building modern web applications. This stack provides an end-to-end framework for the developers to work in and each of these technologies play a big part in the development of web applications.

For my capstone project, I am doing a non-pat internship at *Blackcoffer Pvt. Ltd.* My job role is that of a Web Development Intern. This internship is an exploration of the capabilities of the MERN stack and its role in shaping the future of web development. It focuses on the development of several applications like Weather Application, Food Recipe Application, Contact Management System, and a Visualization Dashboard App, using the MERN stack, each designed to showcase its unique features and functionalities.

The internship serves as a stepping stone towards a career in web development, providing me with valuable insights into the world of full-stack development and equipping me with the skills and knowledge needed to tackle complex web development challenges in the future. I look forward to applying these skills and experiences to contribute to innovative and dynamic applications in the future.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

MERN	MongoDB, Express.js, React, Node.js
CRUD	Create, Read, Update, Delete
API	Application programming interface
CMS	Contact Management System

1. INTRODUCTION

1.1. OBJECTIVES

The following objectives are outlined to help me to combine my theoretical knowledge with practical experience and problem-solving skills, and contribute to the innovations in the digital landscape, delivering solutions that meet the dynamic needs of users.

- **Explore the Capabilities of the MERN Stack:** Gain a comprehensive understanding of MongoDB, Express.js, React.js, and Node.js, and their roles in building modern web applications.
- **Develop Real-World Applications:** Build practical applications, such as a weather application, a Food Recipe App, and a Contact Management System, to showcase the versatility and effectiveness of the MERN stack.
- **Enhance Technical Skills:** Improve proficiency in full-stack web development by working with the MERN stack and implementing key features like real-time data updates, user authentication, and interactive user interfaces.
- **Understand Application Deployment:** Learn how to deploy MERN stack applications to hosting services, ensuring they are accessible to users over the internet.
- **Gain Practical Experience:** Apply theoretical knowledge of the MERN stack to real-world scenarios, gaining hands-on experience in developing and deploying web applications.
- **Demonstrate Problem-Solving Skills:** Encounter and overcome challenges faced during the development process, showcasing problem-solving abilities and adaptability in a dynamic web development environment.
- **Contribute to Innovation:** Utilize the MERN stack to create innovative and user-friendly web applications that cater to the evolving needs of users in the digital world.

Overall, the objectives of developing projects using the MERN stack aim to provide a holistic understanding of the MERN stack, empowering developers to

create modern, feature-rich web applications and deploy them effectively.

By achieving the objectives of this internship, I will have not only explored the MERN stack's capabilities but also honed my skills in full-stack web development. Through the development of innovative and user-friendly applications, I aim to showcase the potential of the MERN stack and prepare myself for future challenges in the field of web development.

1.2. MOTIVATION

The motivation behind this internship stems from a desire to explore and master the capabilities of the MERN stack, a powerful and widely-used framework for web development. By undertaking this internship, the aim is to not only gain practical experience in utilizing the MERN stack but also to deepen understanding of its individual components: MongoDB, Express.js, React.js, and Node.js. These technologies, when combined, offer a seamless and efficient framework for building modern web applications.

Moreover, the internship aims to address real-world challenges by developing applications like the Weather Application, Food Recipe App, Contact Management System, and Visualization Dashboard. Through these applications, I seek to demonstrate the versatility and effectiveness of the MERN stack in handling diverse requirements and providing innovative solutions to users and clients.

Overall, the motivation behind this project lies in the pursuit of knowledge, skill development (technical skills and problem-solving skills), and innovation in web development, with the ultimate goal of creating impactful and user-centric applications that enrich the online experience for users.

1.3. BACKGROUND

The background of this internship encompasses the development of four distinct web applications using the MERN stack, each designed to showcase different aspects of the MERN stack and its capabilities in web development.

The Weather Application will provide users with up-to-date weather information

for a specified location. Users will be able to search for a city and view the current weather conditions, including temperature, humidity, and wind speed. The app will use React.js to
display the weather information in a clear and visually appealing manner.

The Food Recipe App will allow users to explore a variety of recipes, search for specific recipes, and save their favorite recipes for future reference. The app will utilize React.js to create a responsive and intuitive user interface, making it easy for users to navigate and interact with the recipes.

The Contact Management System will enable users to organize and manage their contacts efficiently. It demonstrates the MERN stack's capabilities in handling CRUD (Create, Read, Update, Delete) operations and integrating with MongoDB for data storage.

The Visualization Dashboard provides various interactive charts to visualize data, enhancing data comprehension and analysis for users. It also allows users to query and filter data from a MongoDB database based on various parameters such as year, topic, sector, and region.

Overall, this internship aim is to gain practical experience in developing web applications using the MERN stack and to showcase the versatility and effectiveness of the MERN stack in creating dynamic and user-friendly web applications.

2. PROJECT DESCRIPTION AND GOALS

2.1. SURVEY ON EXISTING SYSTEM

In the first paper, the authors delve into React JS, a JavaScript library for front-end development. It discusses why React JS is a popular choice for developers. React JS is easy to learn and can be used to build complex UIs. It uses a virtual DOM to improve performance. React JS is also considered faster than Angular. However, React JS has some limitations. It requires additional libraries for routing and other functionalities. Also, the development environment changes quickly, so developers need to stay up-to-date. Overall, React JS is a powerful library that can be used to build modern web applications.

The second paper investigates the MERN stack and its advantages over previous technologies. It discusses what the MERN stack is and why it is popular. It also details the functionalities of each component of the MERN stack. Additionally, the article contrasts the MERN stack with HTML, CSS, SQL, and NoSQL. Some of the advantages of the MERN stack include that it is JavaScript-based, open-source, and efficient.

In the third paper, the authors talk about a full-stack web development technology called MERN. It discusses what full-stack development is and the different technologies used in a MERN stack. The article also details why someone might choose MERN over other stacks. Some of the important points are that MERN is easy to learn and uses JavaScript throughout the development process. Additionally, MERN stacks are good for mobile app development.

The fourth paper focuses on what CRUD operations are and how they are implemented in MongoDB. It also compares how CRUD operations are done in MongoDB to how they are done in a relational database. Some of the important points are that MongoDB uses JSON - like documents to store data and that indexes can be created to improve the performance of read queries. Schemas are not required in MongoDB. Indexes can be created to improve the performance of read queries.

The fifth paper discusses contact management systems. It discusses the challenges of

keeping track of contact information. People use a variety of methods, including address books, rolodexes, and digital tools. These tools can be helpful, but they also have limitations. The authors propose a new system that would help users manage their contacts and conversations more effectively.

The sixth paper highlights the challenges businesses face in managing their finances and offers a solution by providing a user-friendly platform for expense tracking, financial analytics, and data visualization. It also uses predictive analytics to forecast future financial trends and is built using the MERN stack.

2.2. RESEARCH GAP

Table 1 : Gaps Identified in existing system/technology

SNo.	Paper Title	Authors	Gaps Identified
1.	React JS – A Frontend Javascript Library (Nov, 2022)	Avinash Mishra, Arshita Gupta	1. React only deals with the View of MVC, so other tools are required for backend development. 2. Some developers find JSX programming difficult to learn. 3. The React environment changes quickly, so developers need to constantly update their skills.
2.	MERN: A Full-Stack Development (Jan, 2022)	Yogesh Baiskar, Priyas Paulzagade, Krutik Koradia, Pramod Ingole, Dhiraj Shirbhate	1. MERN can introduce complexity into your project, especially if you're not familiar with all of the technologies involved. 2. While MERN can be a great choice for small and medium-sized applications, it may not scale well for very large or complex applications. 3. Security is a major concern for any web application, and MERN stacks can be vulnerable to security attacks if not properly configured. 4. MERN stacks are primarily focused on front-end development and may

			not be the best choice for applications that require a lot of back-end functionality.
3.	Introduction to MERN Stack & Comparison with Previous Technologies (Jun, 2023)	Yogesh Kadam, Akhil Goplani, Shubit Mattoo, Shashank Kumar Gupta, , Darshan Amrutkar, Jyoti Dhanke	1. MERN stack can be complex for beginners to learn because it involves mastering four different technologies. 2. Setting up and configuring a MERN stack application can be complex and time-consuming. 3. MongoDB can be challenging to scale for very large applications. 4. React applications can be challenging to optimize for search engines. 5. React applications can generate large bundle sizes, which can slow down page load times.
4.	CRUD Operations in MongoDB (Jul, 2013)	Ciprian-Octavian Truică, Alexandru Boicea, Ionuț Trifan	1. Schema-less nature can lead to inconsistencies without application-level validation. 2. Limited support for granular updates within nested documents. 3. No automatic cascading deletes, requiring custom logic for dependent documents. 4. Optimistic locking can lead to race conditions, requiring proper locking mechanisms. 5. Limited transaction support, not ideal for complex operations across documents. 6. Updating large documents can be inefficient, consider partial updates or denormalization.
5.	Contact Management System (Jun, 2022)	Isha Bankar, Anjali Chiwande, Namita Ghadse, Trupti Shastrakar	1. Difficulty remembering precise details about contacts. 2. Inability to maintain reliable data retrieval. 3. Challenges managing an ever-increasing number of contacts. 4. Inability of current digital tools to reliably store and retrieve data. 5. Lack of an efficient cms system with search functionality.

6.	FinanceVue - A MERN Stack Finance Dashboard Application (Apr, 2024)	Kaarshnee Dewan, Aadith Lasar, Bhushan Bhokse	1. Existing applications are difficult to use, discouraging businesses from adopting them, thus highlighting a lack of user-friendly applications. 2. Traditional dashboard tools lack data analysis capabilities and features to identify trends and make informed decisions. 3. Businesses often rely on manual record-keeping methods, which can lead to errors and inefficiencies.

2.3. PROBLEM STATEMENT

In today's fast-paced world, convenience and accessibility are paramount. This is where these innovative MERN applications, a weather app, a recipe app, and a contact management system step in to empower users with information and personalized experiences that enrich their daily lives.

- **The Weather App:** In a world where weather conditions can drastically impact daily plans, existing weather apps often lack the intuitiveness and comprehensiveness needed for users to make quick and informed decisions. This react-based weather app aims to bridge that gap by offering a user-friendly platform that delivers real-time weather data for any city upon a simple search. By providing essential details like temperature, weather description, wind speed, and humidity, this app empowers users to navigate the elements with confidence, whether they're planning outdoor activities or simply deciding what to wear for the day.

- **The Food Recipe App:** Despite the abundance of online recipes, finding the perfect dish can be time-consuming and overwhelming. Existing recipe apps often lack user-friendly interfaces or fail to personalize recommendations. This react-based Food Recipe App tackles these issues by offering a searchable database powered by the Forkify API. Users can discover recipes by keyword, explore categorized results, and save favorites for easy access. By allowing users to build a personalized recipe collection based on their preferences, this app empowers food enthusiasts to streamline their culinary

exploration and rediscover the joy of cooking.

- **Contact Management System:** In today's digitally connected world, individuals and businesses manage numerous contacts. Keeping track of contact information, communication history, and contextual details can become overwhelming without an efficient system in place. The Contact Management System seeks to address this challenge by providing a user-friendly web application for organizing, storing, and interacting with contacts seamlessly. This system empowers users with CRUD operations (Create, Read, Update, Delete) for managing contacts, along with dedicated fields to store various contact details like names, email addresses, and phone numbers.

- **Visualization Dashboard:** In the era of big data, organizations deal with vast amounts of data across various domains. Analyzing and understanding this data can be challenging without effective visualization tools. The Visualization Dashboard App aims to tackle this challenge by offering a comprehensive web application for visualizing datasets in an intuitive and interactive manner. By leveraging functionalities such as filtering, searching, and dynamic chart generation, the Visualization Dashboard empowers users to gain valuable insights into diverse datasets through interactive visualizations, enabling users to identify trends, correlations, and patterns efficiently. Thus, users can derive actionable insights to make the decision-making process effective.

3. TECHNICAL SPECIFICATION

3.1. REQUIREMENTS

3.1.1 FUNCTIONAL

- **Food Recipe App:**
  - Users should be able to search for recipes by name or ingredient.
  - Users should be able to view detailed information about each recipe, including ingredients and instructions.
  - Users should be able to save their favorite recipes for future reference.
  - This web app should be responsive and work seamlessly.

- **Weather Application:**
  - Users should be able to search for weather information by city name.
  - Users should be able to view current weather conditions, including temperature, humidity, and wind speed.
  - Users should be able to view the weather forecast for any city for the current date.
  - The app should display weather information in a visually appealing and easy-to-understand manner.
  - The web app should be responsive and work well.

- **Contact Management System:**
  - Users should be able to add new contacts with details such as name, email, phone number, and address.
  - Users should be able to view a list of all contacts
  - Users should be able to search for specific contacts by name.
  - Users should be able to update contact information and delete contacts as needed.
  - The system should provide authentication and authorization features to ensure that only authorized users can access and manage contacts.

- ➢ The cms system should be responsive and work seamlessly.

❖ **Visualization Dashboard App:**

- ➢ The application should display all data points with different attributes on the screen.
- ➢ Users should be able to view various types of charts, including bar charts, line charts, pie charts, radar charts, and polar area charts.
- ➢ Users should have the option to filter data based on different criteria such as year, sector, region, topic, country, pestle, source, intensity, likelihood, title, or any keyword.
- ➢ Users should be able to interact with the charts dynamically, hover-over tooltips to display detailed information.
- ➢ Users should have the ability to customize the appearance of charts, including labels.
- ➢ The graphs should change dynamically in response to any search requests or filtering criteria applied by the users.
- ➢ The filters should be removed when the user clicks on the Reset Filter button.
- ➢ The system should be responsive and work seamlessly.

3.1.2 NON-FUNCTIONAL

❖ **Performance:**

- ➢ The applications should load quickly and respond to user interactions without delay.
- ➢ The applications should be able to handle a large number of users simultaneously without experiencing performance issues.
- ➢ The system should be able to handle a large number of contacts without experiencing performance issues.

❖ **Usability:**

- ➢ The user interface of the applications should be intuitive and easy to navigate.
- ➢ The application should provide clear and concise information to users, without overwhelming them with unnecessary details.

- ➢ The user interface of the system should be intuitive and easy to navigate.
- ➢ The system should provide clear feedback to users when actions are performed, such as adding or updating a contact.

❖ **Reliability:**

- ➢ The application and system should be reliable and available to users at all times, with minimal downtime for maintenance or updates.

❖ **Security:**

- ➢ The application should ensure the security and privacy of user data, including contact information and user login credentials.
- ➢ The application should implement secure authentication mechanisms to protect user accounts.
- ➢ The system should ensure the security and privacy of contact information, including implementing secure authentication mechanisms.

❖ **Compatibility:**

- ➢ The application and system should be compatible with a wide range of web browsers and devices, ensuring a consistent user experience.

❖ **Scalability:**

- ➢ The application should be designed to scale easily to accommodate an increasing number of users, data, and contacts.

❖ **Maintainability:**

- ➢ The application and system should be easy to maintain and update, with well-structured code and clear documentation for developers.

3.2. FEASIBILITY STUDY

3.2.1 TECHNICAL FEASIBILITY

The development of the Food Recipe App, Weather Application, Contact Management System, and Visualization Dashboard App using the MERN stack is technically feasible based on the following considerations:

Technology Stack: The MERN stack is well-established and widely used in the industry for developing modern web applications. Each component of the

stack (MongoDB, Express.js, React.js, Node.js) offers robust features and extensive documentation, making it suitable for building complex applications.

Development Tools: There is a wide range of development tools and libraries available for the MERN stack, which can expedite the development process and enhance the functionality of the applications. Also, React allows the creation and integration of various charts and visualizations using React Javascript Extension.

Scalability: The MERN stack is inherently scalable, allowing applications to handle a large number of users and data. MongoDB, as a NoSQL database, can easily scale horizontally to accommodate increasing data volume, while Node.js provides a non-blocking I/O model that can handle multiple concurrent requests efficiently.

Community Support: The MERN stack has a large and active community of developers who contribute to its development and provide support through forums, tutorials, and documentation. This community support ensures that developers can find solutions to any technical challenges they may encounter during the development process.

Overall, the technical feasibility of developing the Food Recipe App, Weather Application, Contact Management System, and Visualization Dashboard App using the MERN stack is high, given the maturity and robustness of the technologies involved.

3.2.2 ECONOMIC FEASIBILITY

The development of the Food Recipe App, Weather Application,Contact Management System, and Visualization Dashboard App using the MERN stack is economically feasible based on the following considerations:

Cost of Development: The MERN stack is open-source and free to use,

reducing the initial cost of development. Additionally, there is a wealth of resources and documentation available online, reducing the need for expensive training or consulting services.

Scalability and Maintenance Costs: The MERN stack is inherently scalable, allowing applications to grow with the user base without incurring significant additional costs. Maintenance costs are also relatively low, as the stack is well-maintained and supported by a large community of developers.

Return on Investment (ROI): The applications, once developed, have the potential to generate revenue through various means, such as advertisements, subscriptions, or premium features. The ROI can be significant, especially if the applications attract a large user base.

Overall, the economic feasibility of developing the Food Recipe App, Weather Application, Contact Management System, and Visualization Dashboard App using the MERN stack is high, given the low initial costs, potential for scalability and revenue generation, and competitive advantages that can be gained.

3.2.3 SOCIAL FEASIBILITY

The development of the Food Recipe App, Weather Application, Contact Management System, and Visualization Dashboard App using the MERN stack is socially feasible based on the following considerations:

User Engagement: The applications are designed to cater to the needs and interests of users, providing valuable features and functionality that enhance their daily lives. The Food Recipe App allows users to explore new recipes and expand their culinary skills, while the Weather Application provides essential weather information for planning outdoor activities. The Contact Management System helps users organize and manage their contacts efficiently. The Dashboard App allows users to analyze their data and make business decisions effectively.

Accessibility: The applications are accessible to a wide range of users. Efforts are made to ensure that the user interface is intuitive and easy to navigate, making the applications inclusive and user-friendly.

Community Impact: The applications have the potential to have a positive impact on the community by providing valuable resources and services. For example, the Weather Application can help users plan their day effectively and stay safe during unpleasant weather conditions.

User Privacy: The applications prioritize user privacy and data protection, ensuring that sensitive information is handled securely and in compliance with relevant regulations. Users can trust that their personal information is safe when using the applications.

Environmental Impact: While the applications themselves do not have a direct environmental impact, they can contribute to sustainability efforts by promoting digitalization and reducing the need for paper-based resources.

Overall, the social feasibility of developing the Food Recipe App, Weather Application, Contact Management System, and Visualization Dashboard App using the MERN stack is high, as they provide valuable services to users, are accessible to a wide audience, and prioritize user privacy and data protection.

3.3. SYSTEM SPECIFICATION

3.3.1 HARDWARE SPECIFICATION

Processor: The server hosting the applications should have a multicore processor with a clock speed of at least 2.0 GHz to ensure smooth performance under load.

Memory (RAM): A minimum of 4 GB of RAM is recommended for running the applications. However, for optimal performance, especially in handling large datasets, 8 GB or more of RAM is preferable.

Storage: The applications require storage for the application code, database, and any media assets (such as images or videos). A minimum of 50 GB of storage is recommended, with the ability to scale up as the application grows.

Operating System: The server should run a supported operating system, such as Linux (e.g., Ubuntu, CentOS) or Windows Server, with the necessary updates and security patches applied.

Network: A stable internet connection with sufficient bandwidth is essential for hosting the applications and ensuring fast and reliable access for users.

By meeting these system hardware specifications, the Food Recipe App, Weather Application, Contact Management System , and Visualization Dashboard App can be hosted and run efficiently, ensuring a seamless user experience and optimal performance.

3.3.2 SOFTWARE SPECIFICATION

Web Server: The applications can be deployed using a web server such as Apache or Nginx. The web server should be configured to serve the React.js frontend and proxy requests to the Node.js backend.

Node.js: Node.js is required to run the backend server for the applications. It should be installed on the server along with the necessary npm packages for the applications.

MongoDB: MongoDB is required to store the data for the applications. It should be installed on the server and configured to work with the Node.js backend.

React.js: React.js is required to build and run the frontend user interfaces for the applications. It should be included in the application codebase and built using npm scripts.

Express.js: Express.js is used to build the backend server for the applications. It should be included in the application codebase and configured to handle API requests from the frontend.

Additional Dependencies: Depending on the specific requirements of the application, additional dependencies have been used. Eg: bcrypt has been used for password hashing and JSON Web Token (JWT) for authentication purposes. Similarly, React Router DOM has been used as a Routing library for React applications and Axios is used for making API requests. These should be included in the application codebase and installed using npm.

By meeting these system software specifications, the Food Recipe App, Weather Application, Contact Management System, and Visualization Dashboard App can be deployed and run successfully, providing users with a seamless and reliable experience.

3.3.3 STANDARDS AND POLICIES

Coding Standards: Follow industry-standard coding practices and conventions for JavaScript, HTML, and CSS to ensure readability, maintainability, and consistency across the codebase. Use tools like Prettier to enforce coding standards.

Security Standards: Adhere to best practices for web application security, such as implementing secure authentication and authorization mechanisms, input validation, and protection against common vulnerabilities.

Data Protection: Ensure compliance with data protection regulations by implementing measures to protect user data, such as encryption and secure data storage practices.

Accessibility Standards: Design and develop the applications with accessibility in mind, to ensure that all users can access and use the applications.

Performance Standards: Optimize the applications for performance by following best practices for front-end and back-end development, such as minimizing network requests, using efficient algorithms, and caching static assets.

Version Control: Use version control systems (e.g., Git) to manage code changes and deployments. Follow best practices to ensure smooth and reliable working of the applications.

By adhering to these standards and policies, the Food Recipe App, Weather Application, and Contact Management System can meet industry best practices and ensure high quality, security, and usability for users.

4. DESIGN APPROACH AND DETAILS

4.1. SYSTEM ARCHITECTURE

The system architecture diagrams below provides a comprehensive overview of the contact management application's and the visualization dashboard's structure and functionality.

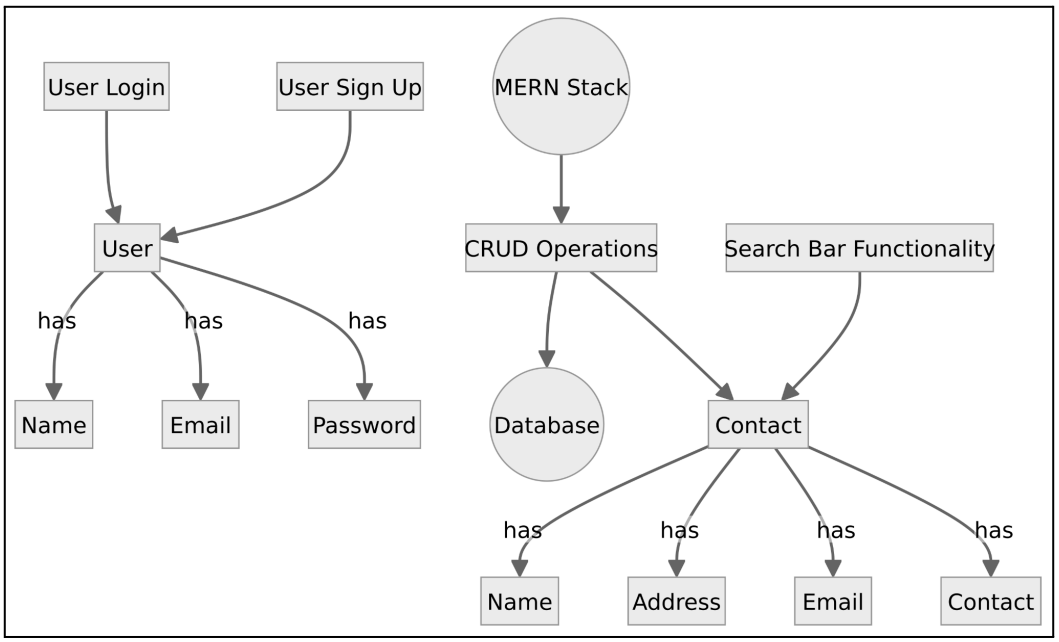


Fig 1 : System Architecture of the CMS App

The contact management application boasts a robust system architecture centered around the MERN stack, comprising MongoDB, Express.js, React, and Node.js. This technology stack facilitates seamless data management through CRUD operations, empowering users to Create, Read, Update, and Delete contact information efficiently. The system incorporates essential user authentication features with User Login and User Sign Up components, ensuring secure access to user accounts. Additionally, the application leverages MongoDB as its database component, providing scalable storage for user data. Entity relationships are carefully mapped out, highlighting interactions between user entities, authentication components, and database operations. The integration of these components within the MERN stack architecture ensures a user-friendly interface and robust functionality for effective contact management.

Furthermore, the architecture emphasizes the application's user-centric design,

with features like a Search Bar Functionality enhancing user experience by enabling quick access to desired contact information. The database component serves as the backbone, storing user-entered contact details securely. Seamless integration between components enables smooth data flow and interactions, ensuring a cohesive user experience. This integrated approach not only streamlines development but also enables the application to leverage the scalability and flexibility offered by the MERN stack.

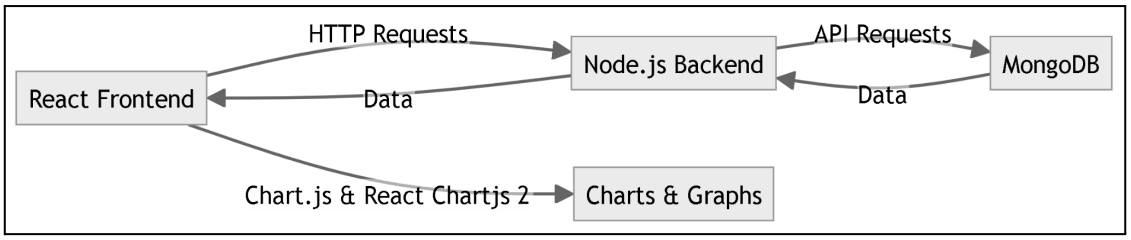


Fig 2 : System Architecture of the Dashboard App

The Data Visualization Dashboard project follows a typical client-server architecture, with a front-end React application serving as the client and an Express.js server handling the backend logic and database interactions. The React front-end is responsible for rendering the user interface, including various charts and visualizations, and facilitating user interactions. It communicates with the backend server via HTTP requests to fetch data from the database and perform operations like filtering and searching. The front-end utilizes libraries such as Chart.js and React Chartjs 2 to create interactive and visually appealing charts based on the fetched data. Additionally, the front-end routes user requests to the appropriate endpoints on the backend server based on the user's actions or input.

On the server side, the Express.js framework provides a robust and scalable platform for building RESTful APIs and handling HTTP requests. The server consists of multiple routes defined to handle different types of requests, such as fetching all data and filtering data based on various parameters like year, topic, or sector etc. Each route corresponds to a specific controller function, which contains the business logic for processing the request, querying the database using Mongoose, and returning the appropriate response. The Express server also establishes a connection to the MongoDB database, allowing seamless interaction

with the data stored in the database. This architecture ensures separation of concerns, scalability, and maintainability, making it easier to add new features or modify existing ones in the future.

Overall, the architecture diagram for both the applications provides a comprehensive overview of how each component contributes to the application's functionality, showcasing its ability in leveraging the strengths of the MERN stack for reliable and scalable development.

4.2. DESIGN

The application's architecture is defined with clarity and precision with the creation and integration of UML diagrams like Use Case Diagrams, Sequence Diagrams, Class Diagrams, and Data Flow Diagrams. The integration of these diagrams facilitates a clear understanding of system requirements, behaviors, and interactions, laying a solid foundation for the subsequent development and implementation phases for the contact management system and the visualization dashboard.

4.2.1 DATA FLOW DIAGRAM

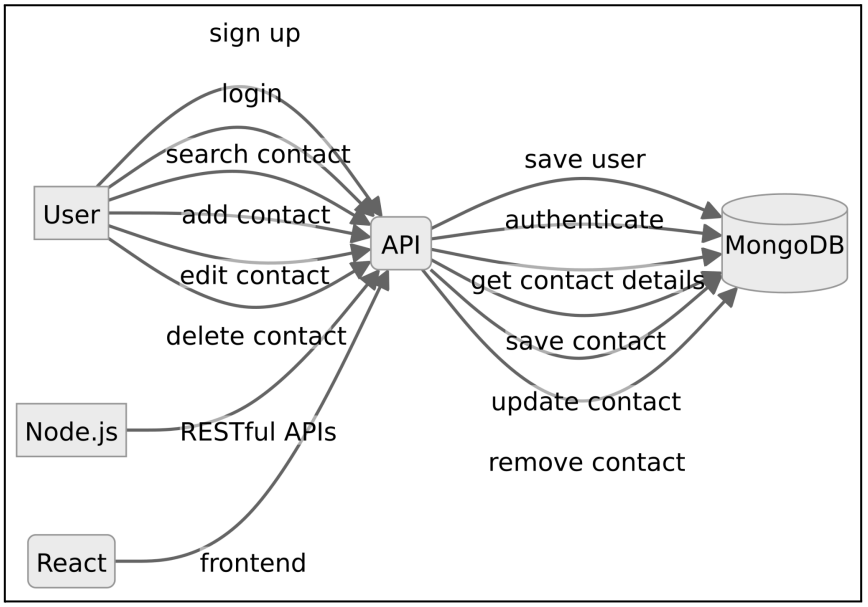


Fig 3 : Data Flow Diagram for CMS App

The data flow diagram above shows how users interact with a contact management app built with React (frontend), Node.js (backend APIs), and MongoDB (database).

Users manage contacts, triggering requests to the backend which interacts with the database for storage and management.

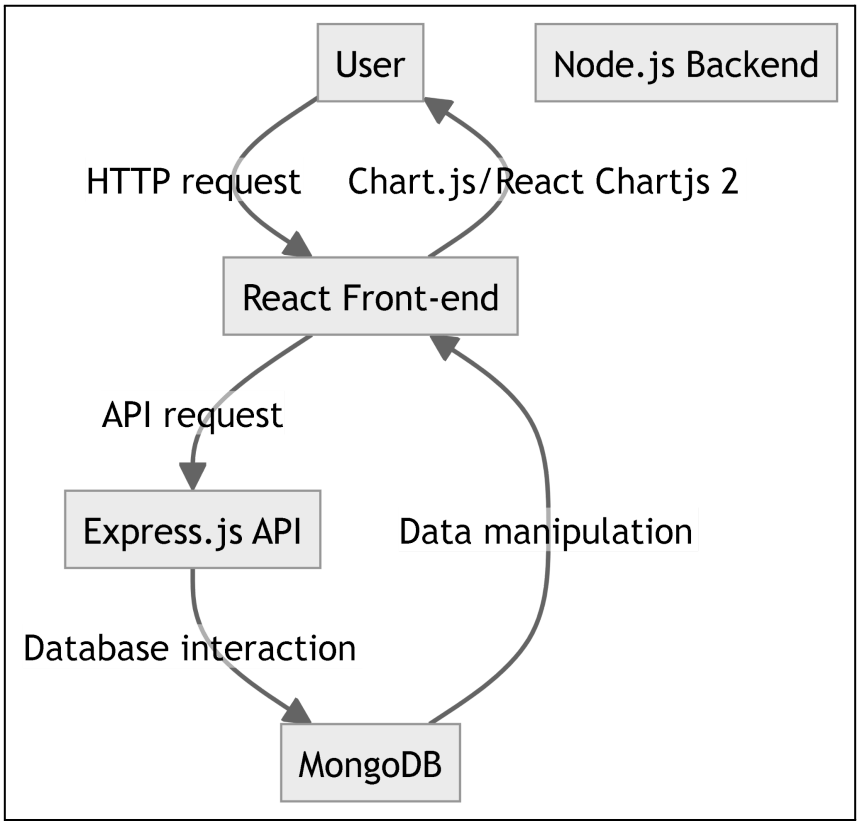


Fig 4 : Data Flow Diagram for Dashboard App

The data flow diagram above showcases how user interactions on the frontend trigger a series of requests and responses between the frontend and backend, ultimately resulting in the retrieval of data from the database and its presentation in a visual format on the user's dashboard.

4.2.2 USE CASE DIAGRAM

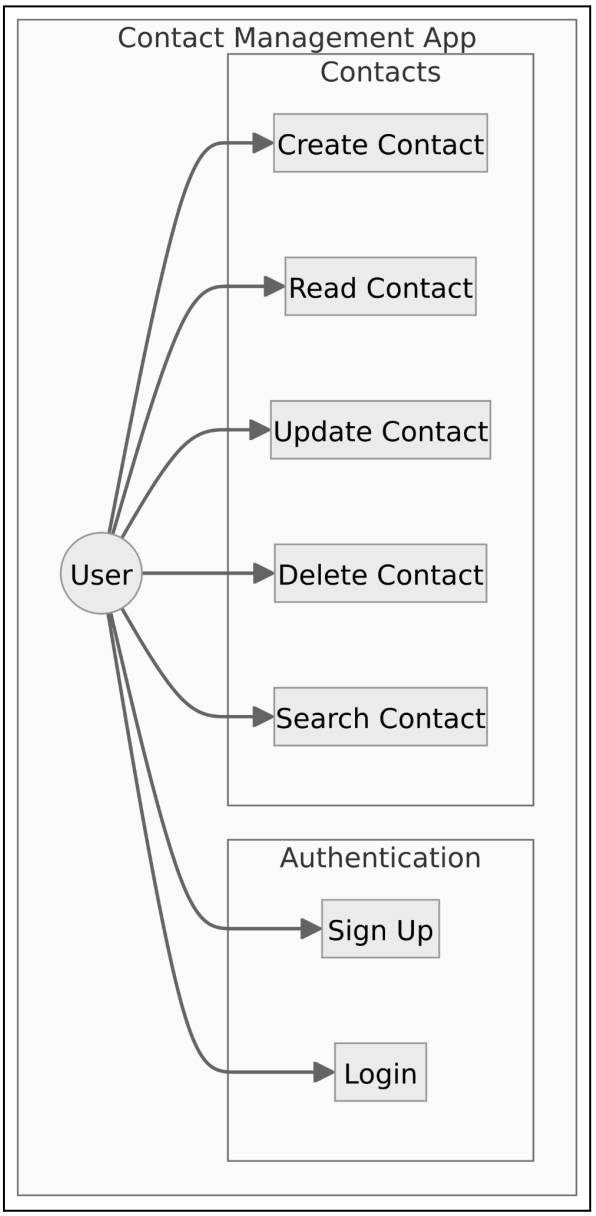


Fig 5 : Use Case Diagram for CMS App

The use case diagram above presents an overview of the contact management application's functionalities and interactions. Users are able to perform actions such as creating, reading, updating, and deleting contacts, as well as searching for specific contacts. Additionally, the application includes an authentication process, allowing users to sign up for new accounts or log in with existing credentials.

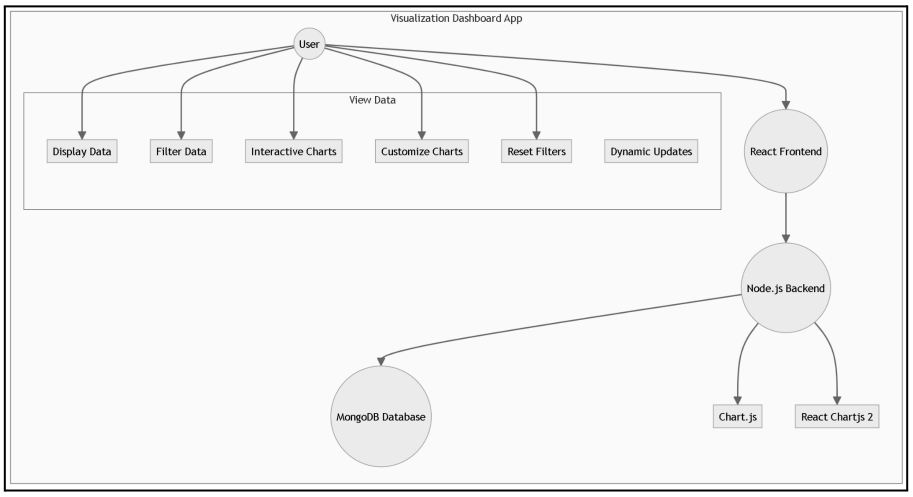


Fig 6 : Use Case Diagram for Dashboard App

This use case diagram above focuses on user interactions within a data visualization app. Users can view raw data, filter it, and interact with customizable charts for deeper insights. They can also reset filters and enjoy dynamic updates on the dashboard.

4.2.3 CLASS DIAGRAM

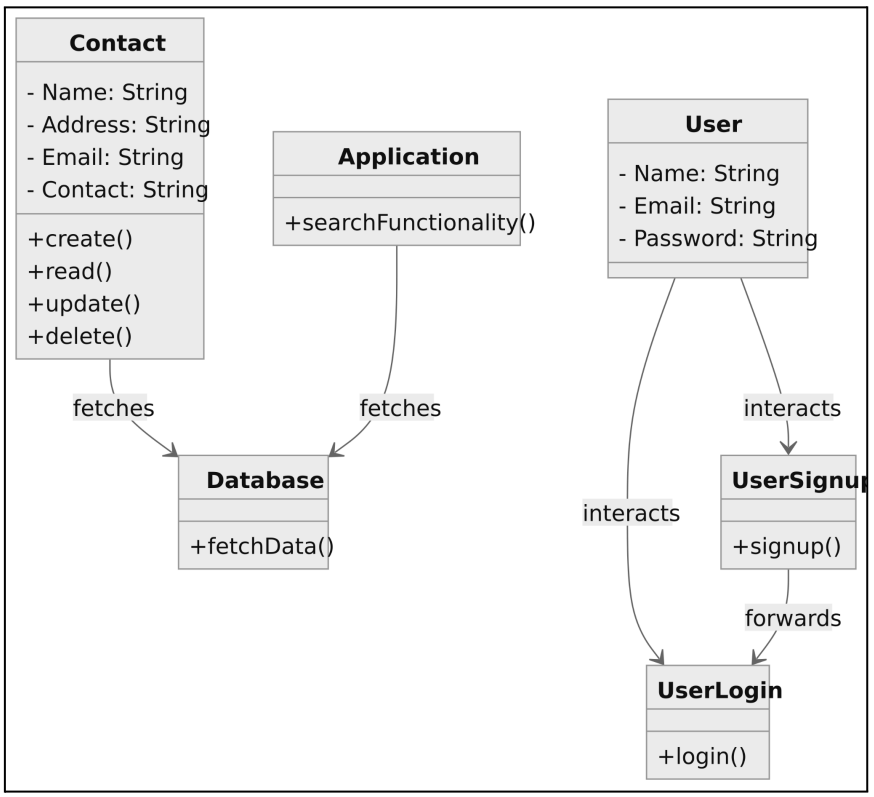


Fig 7 : Class Diagram for CMS App

The above class diagram shows how components like Contact (data), User (interaction), and Database (storage) work together. UserSignup and UserLogin handle authentication, while the Application class interacts with the database for searches.

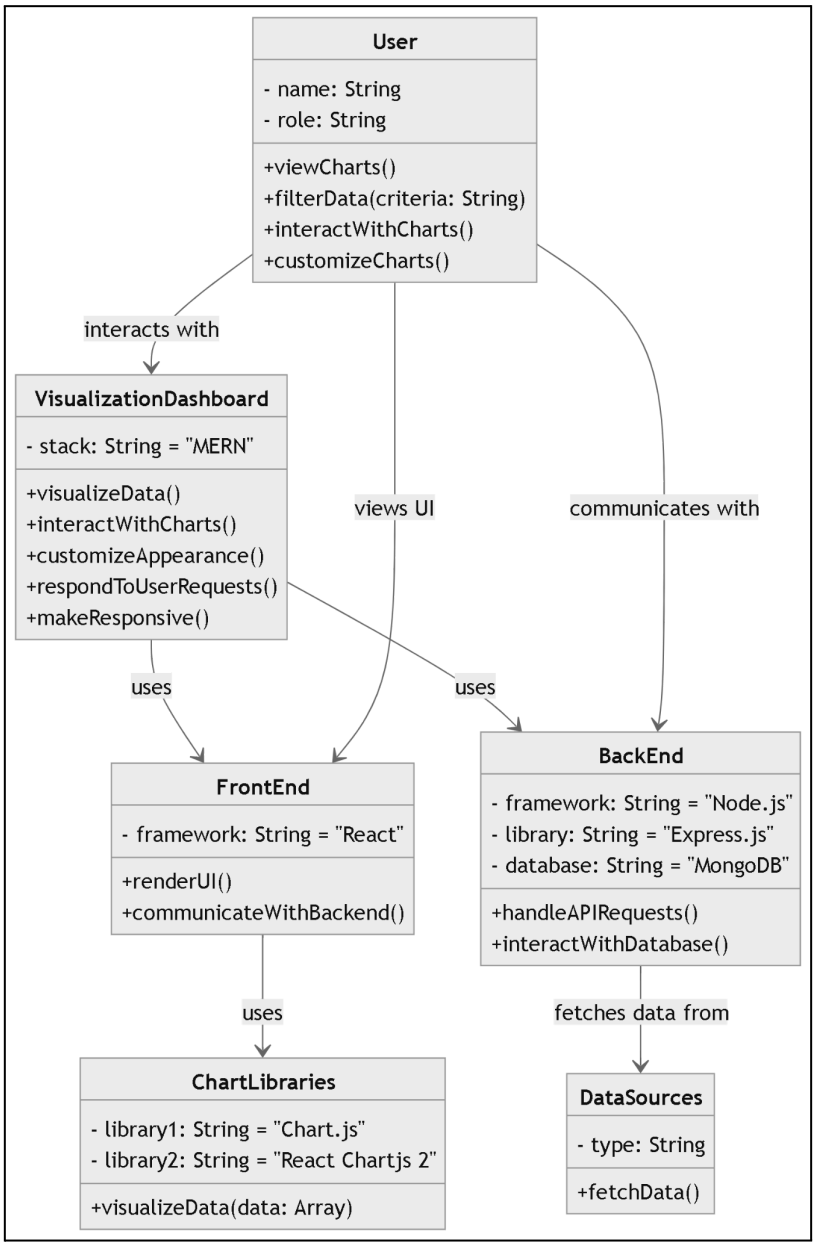


Fig 8 : Class Diagram for Dashboard App

The class diagram above shows how the data visualization app (MERN stack) works. It highlights classes like User, Visualization Dashboard, Backend, Frontend, and Data Sources. These classes work together to let users interact with charts, filter data, and see visualizations on the dashboard.

4.2.4 SEQUENCE DIAGRAM

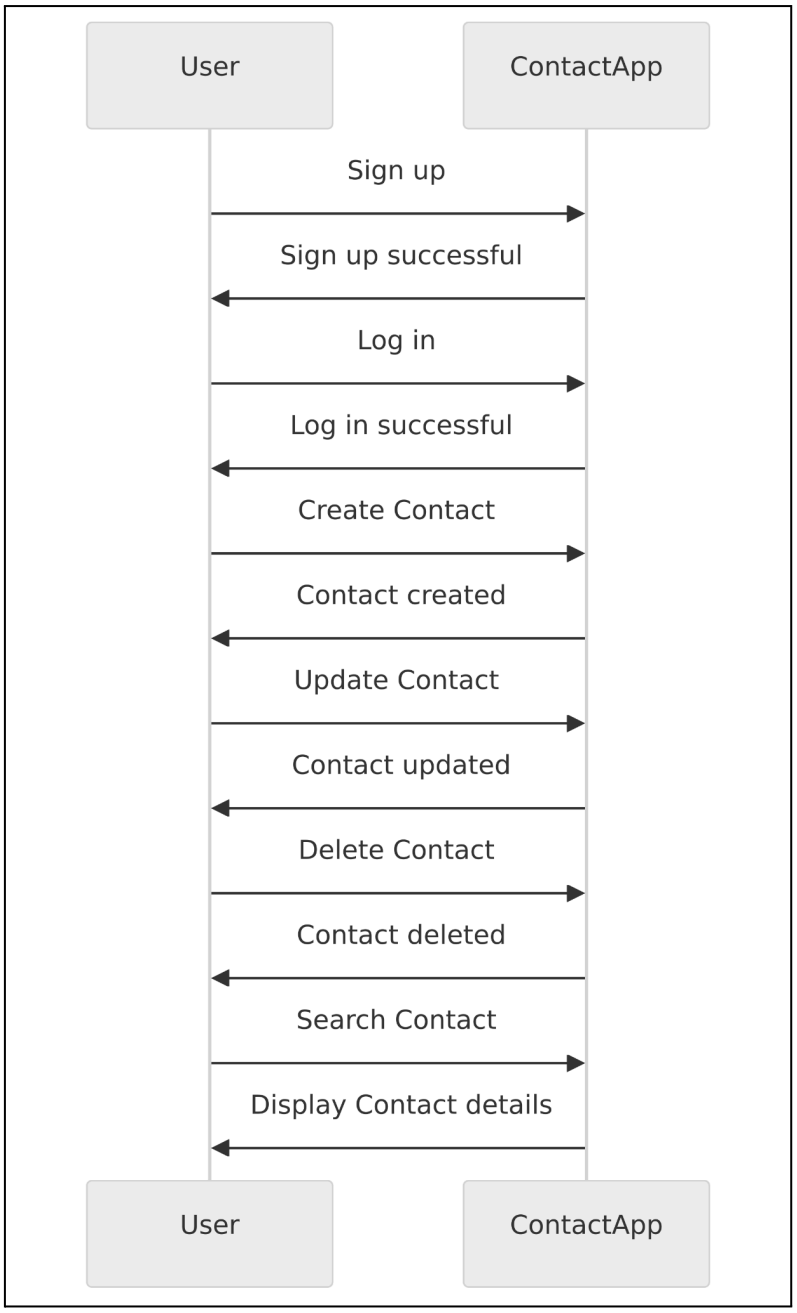


Fig 9 : Sequence Diagram for CMS App

The sequence diagram above showcases the interaction flow between the User and the ContactApp in the contact management system. As the User engages with the application, actions like signing up, logging in, creating, updating, or deleting contacts are initiated. Subsequently, the ContactApp provides feedback, indicating successful execution of these actions or delivering pertinent information, such as

contact creation or deletion confirmation.

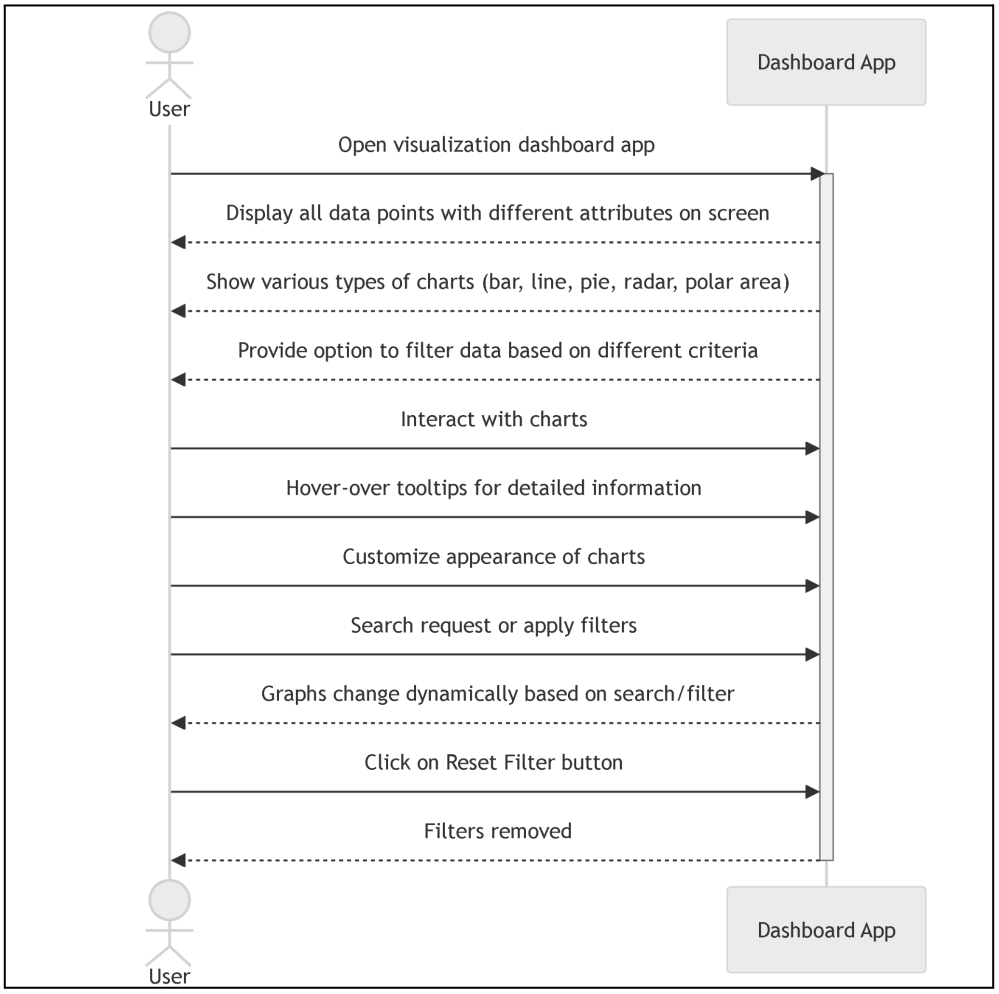


Fig 10 : Sequence Diagram for Dashboard App

The above sequence diagram showcases how user interaction with filters on the frontend triggers a series of requests and responses between the frontend and backend, ultimately resulting in the retrieval of filtered data from the database and its presentation in a visual format on the user's dashboard.

4.3. CONSTRAINTS, ALTERNATIVES AND TRADEOFFS

Here's an overview of the Constraints, Alternatives, and Trade-offs for each of the applications:

Weather Application

- ❖ Constraints:

  - Relies on external weather APIs, which may have usage limits or require

payment for higher usage levels.

- Data accuracy and availability are dependent on the external API's reliability.

❖ Alternatives:

- Use multiple weather APIs for redundancy and to mitigate downtime.

- Implement caching strategies to reduce the number of API calls and improve performance.

❖ Trade-offs:

- Using multiple APIs increases complexity and maintenance overhead.

- Caching may lead to slightly outdated information but improves performance.

Food Recipe App

❖ Constraints:

- Relies on external recipe databases or APIs, which may have usage limits or require payment for higher usage levels.

- Need to ensure the recipes are accurate and reliable.

❖ Alternatives:

- Allow user-generated content to expand the recipe database.

- Partner with a recipe website or service to access their database through an API.

❖ Trade-offs:

- User-generated content requires moderation and validation.

- Partnering with a service may incur costs or limit the app's independence.

Contact Management System

❖ Constraints:

- Requires MongoDB for data storage, which may require additional configuration and maintenance compared to traditional SQL databases.

- Requires backend server hosting for API access and data management.

❖ Alternatives:

- Use a different database technology that might better suit the application's needs.

- Consider serverless architecture to reduce hosting and maintenance costs.

❖ Trade-offs:

- Switching databases requires data migration and potentially rewriting parts of the application.

- Serverless architecture may limit scalability or introduce latency.

Dashboard Visualization Application

❖ Constraints:

- Requires complex data visualization libraries and frameworks, which may increase frontend complexity and maintenance.

- Need to ensure data security and privacy, especially when dealing with sensitive business data.

❖ Alternatives:

- Use simpler visualization techniques or libraries to reduce complexity.

- Implement strict access control and encryption for data security.

❖ Trade-offs:

- Simpler visualizations may limit the depth of analysis or user interactivity.

- Strict security measures may add complexity and impact performance.

These considerations should help analyze the strengths and weaknesses of each project, guiding decision-making processes and ensuring that it is easier to address potential challenges effectively.

5. SCHEDULE, TASKS AND MILESTONES

5.1. GANTT CHART

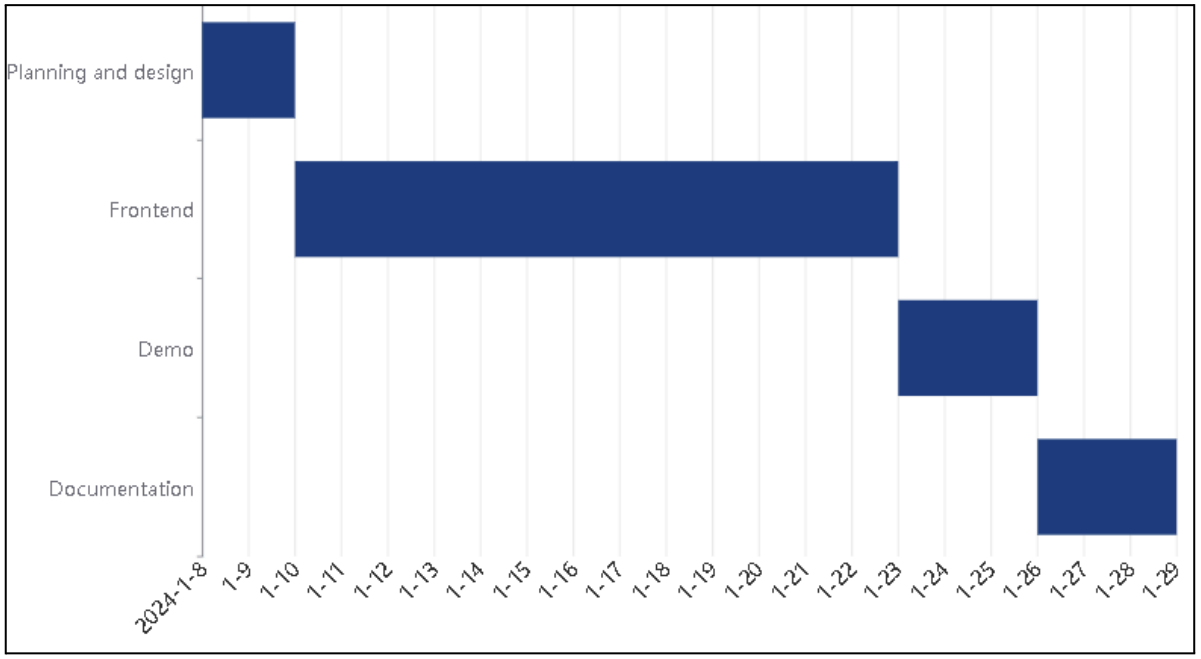


Fig 11 : Weather App Gantt Chart

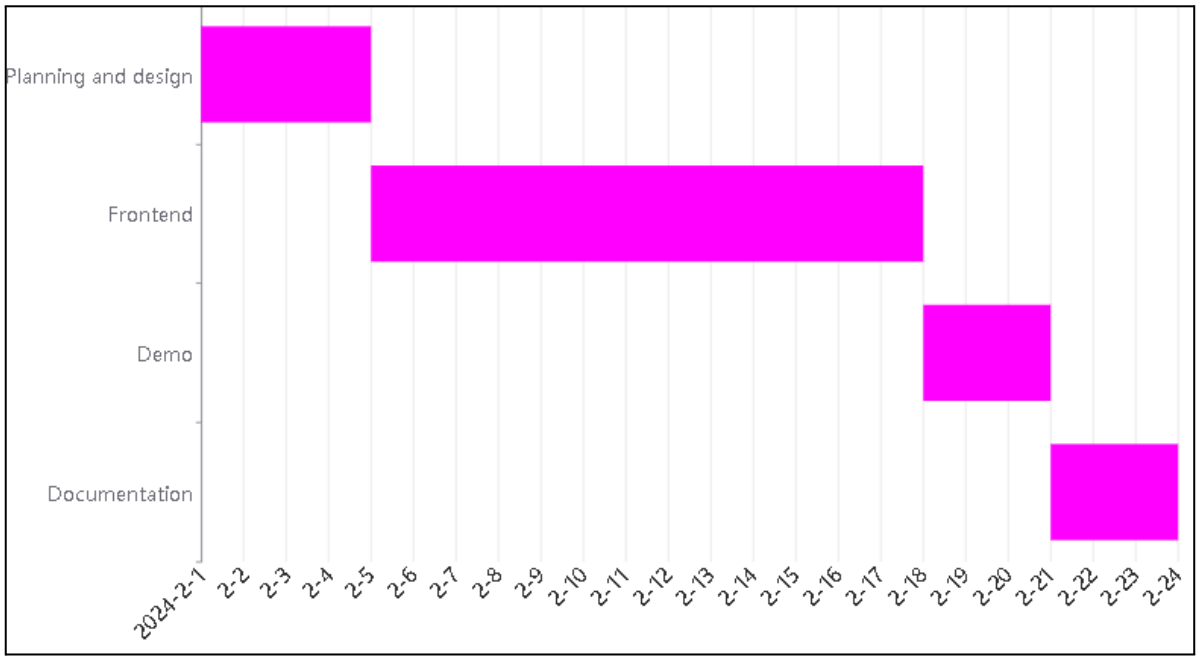


Fig 12 : Food Recipe App Gantt Chart

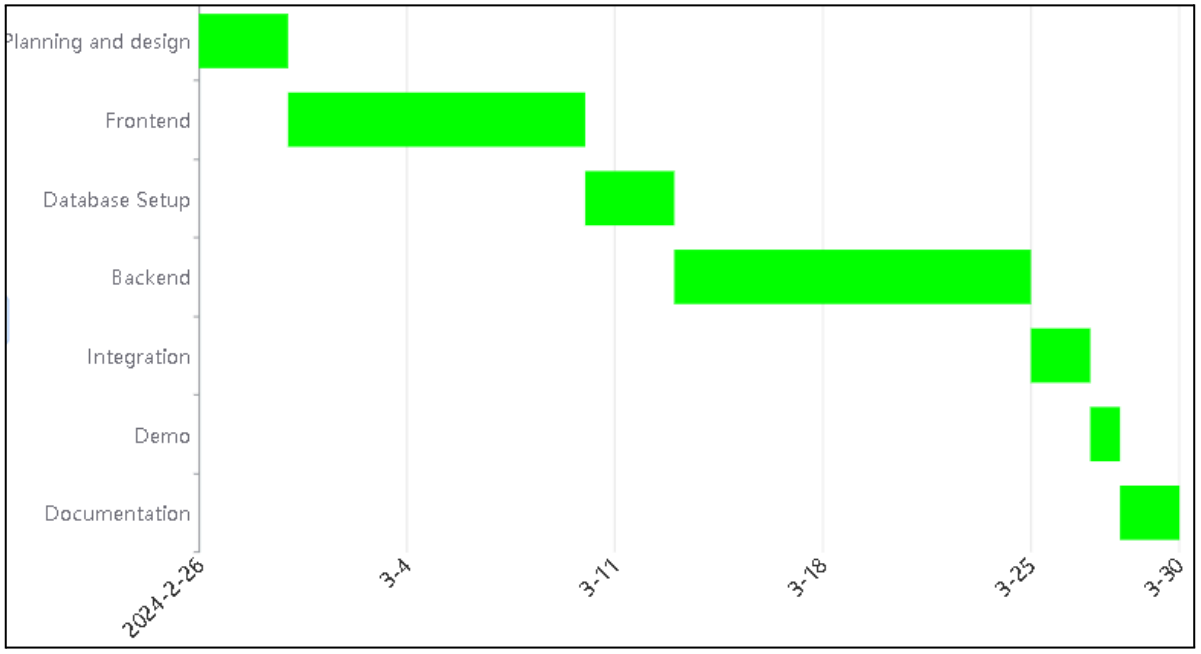


Fig 13 : CMS App Gantt Chart

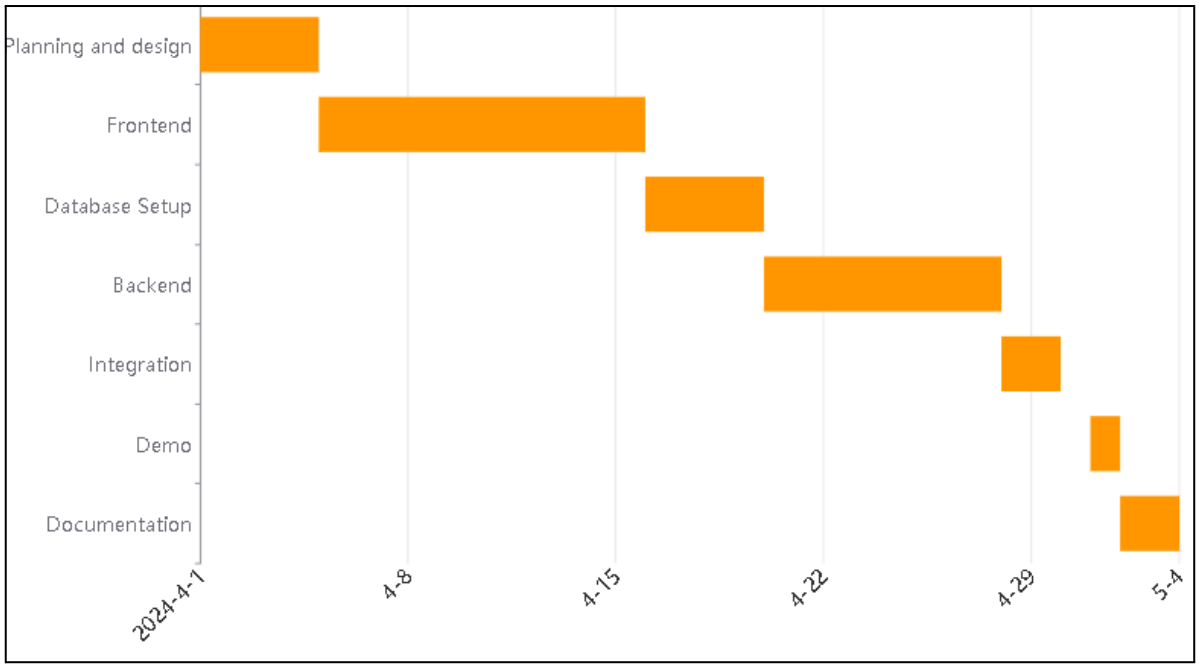


Fig 14 : Visualization Dashboard App Gantt Chart

5.2. MODULE DESCRIPTION

5.2.1 CMS APP MODULES

5.2.1.1 MODULE - 1 User Authentication Module

Description: This module handles user authentication and authorization processes. It includes functionalities such as user sign-up, login, and logout.

Features:

• Sign-up: Allows users to create a new account by providing necessary details like name, email, and password.

• Login: Authenticates users based on their credentials (email and password) to access the application.

• Logout: Logs users out of their accounts securely.

5.2.1.2 MODULE - 2 Contact Management Module

Description: This module manages all aspects related to contacts, including creation, retrieval, updating, and deletion of contacts.

Features:

• Create Contact: Enables users to add new contacts by entering details such as name, address, email, and contact information.

• Read Contact: Retrieves contact details based on specified criteria, allowing users to view existing contacts.

• Update Contact: Allows users to modify existing contact information, such as updating an email address or phone number.

• Delete Contact: Removes contacts from the database permanently, based on user request.

5.2.1.3 MODULE - 3 Database Interaction Module

Description: This module interacts with the database to perform CRUD (Create, Read, Update, Delete) operations on contact data.

Features:

• Data Persistence: Ensures that contact information is stored securely and efficiently in the database.

• Database Connectivity: Establishes connections with the database management system (e.g., MongoDB) to execute database operations.

• Query Execution: Executes queries to retrieve, insert, update, or delete contact records based on user actions.

5.2.1.4 MODULE - 4 Authentication Management Module

Description: Manages the authentication process by validating user credentials and ensuring secure access to the application.

Features:

• Authentication Validation: Verifies user credentials during login to authenticate users securely.

• Token Generation: Generates authentication JSON Web Token upon successful login for subsequent authorized requests.

5.2.1.5 MODULE - 5 Search Functionality Module

Description: Implements advanced search functionality to allow users to find contacts efficiently based on specific criteria.

Features:

• Advanced Filters: Enables users to apply filters such as name to narrow down search results.

• Real-time Search: Provides instant search results as users type, enhancing user experience and productivity.

5.2.2 VISUALIZATION DASHBOARD APP MODULES

5.2.2.1 MODULE - 1 User Interface Module

Description: The User Interface Module is responsible for presenting the data visualization dashboard to users in an intuitive and user-friendly manner. It includes functionalities for creating interactive charts, handling user interactions, and providing a seamless experience.

Features:

• Interactive Charts: Renders interactive charts and visualizations using libraries like Chart.js, allowing users to explore data dynamically.

• User Interaction: Enables users to interact with charts by selecting data points, hovering or applying filters for deeper analysis.

• Dashboard Navigation: Facilitates easy navigation between different sections of the dashboard, providing a cohesive user experience.

• Data Presentation: Presents data in a visually appealing manner with clear labels, legends, and tooltips for better understanding.

5.2.2.2 MODULE - 2 Data Visualization Module

Description: The Data Visualization Module is responsible for rendering various charts and visualizations based on the dataset. It includes functionalities for generating bar charts, pie charts, line charts, radar charts, polar area charts, and doughnut charts.

Features:

• Bar Chart: Displays data using vertical bars to represent numerical values.

• Pie Chart: Represents data in a circular graph, divided into slices to illustrate numerical proportions.

• Line Chart: Connects data points with straight lines to show trends or changes over time.

• Radar Chart: Visualizes multivariate data on a two-dimensional plane with multiple axes radiating from the center.

• Polar Area Chart: Presents data in a circular graph with each segment's area proportional to its value.

5.2.2.3 MODULE - 3 Backend API Module

Description: This module handles backend API requests and interacts with the database to fetch, filter, and manipulate data. It includes functionalities for retrieving all data, filtering data by various parameters like year, topic, or sector

Features:

• Fetch All Data: Retrieves all data from the database and returns it to the client.

• Filter Data: Allows users to filter data based on parameters such as year, topic, sector, or region.

6. PROJECT DEMONSTRATION

Weather App

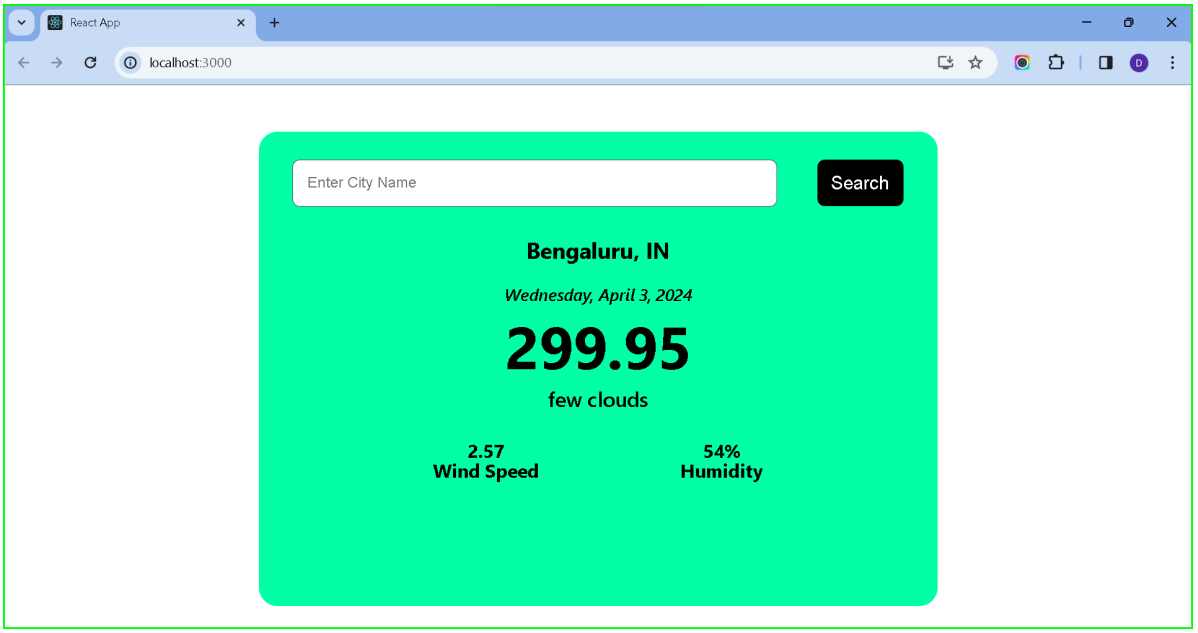


Fig 15 : Weather App - Home Page

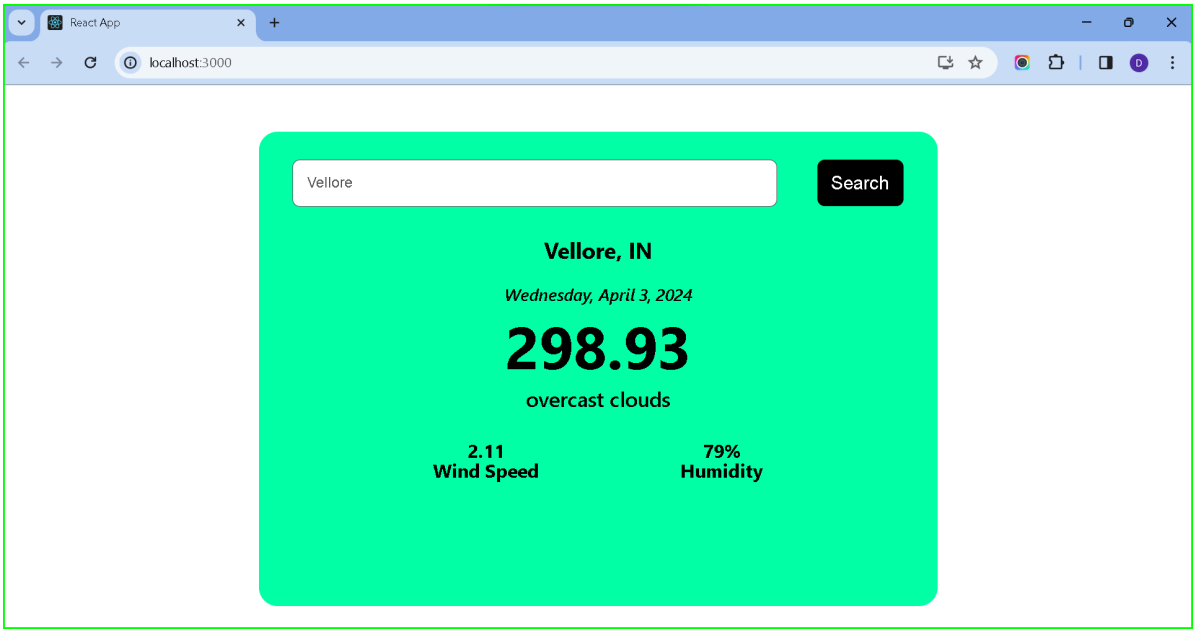


Fig 16: Searching weather for a different city - Vellore

Food Recipe App

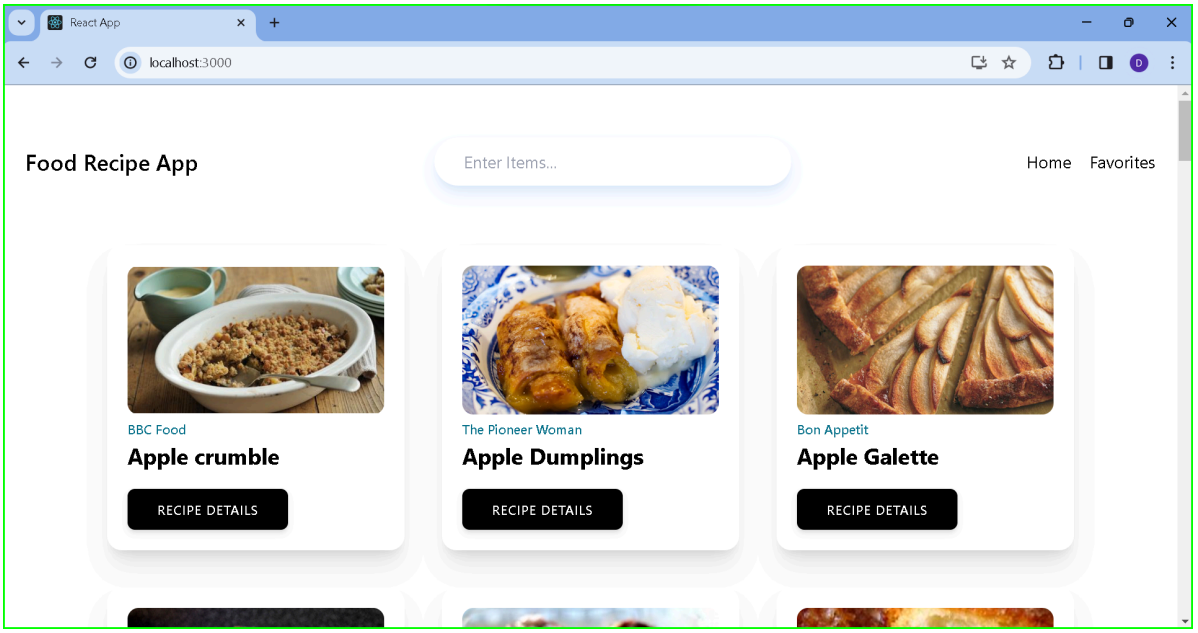


Fig 17 : Recipe App - Home Page

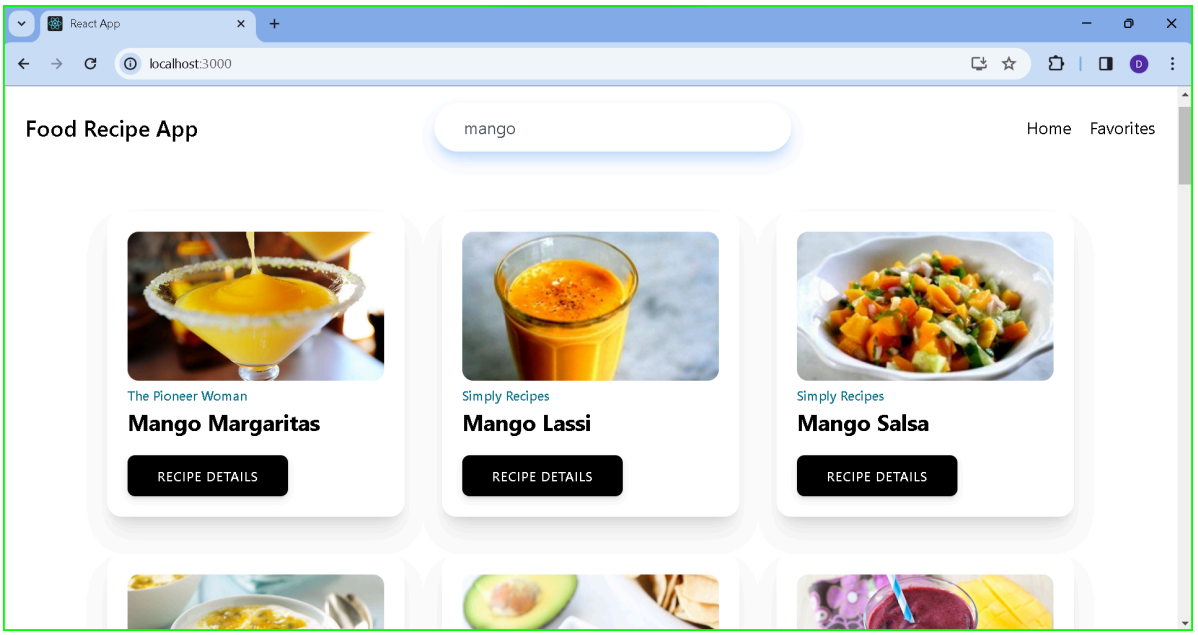


Fig 18 : Searching for a particular food item

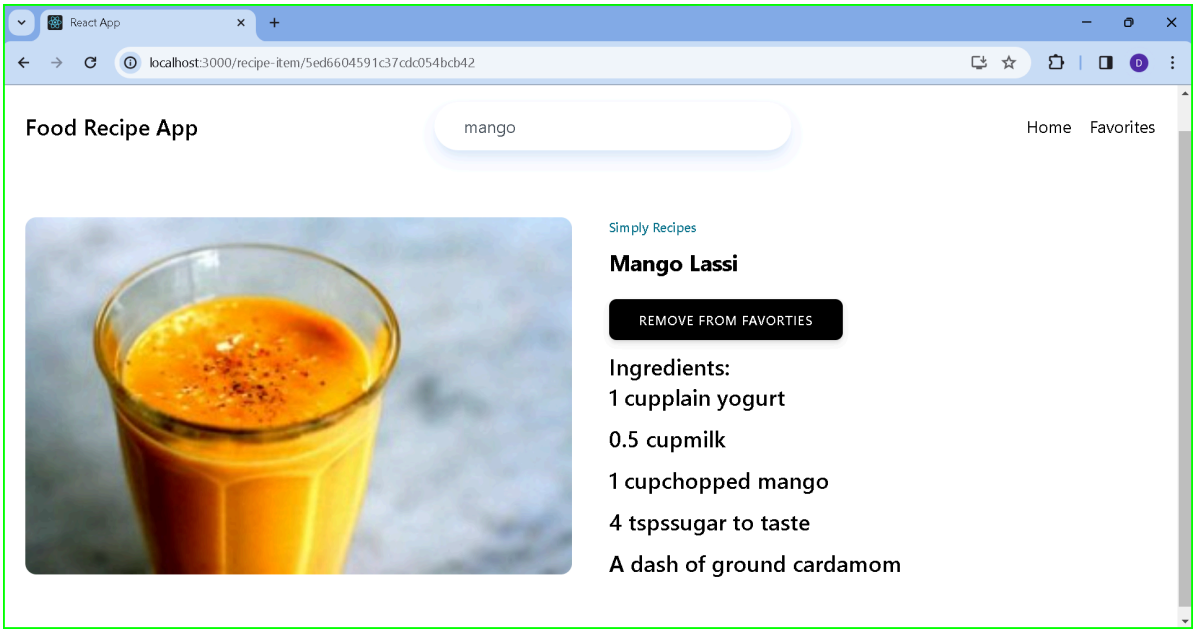


Fig 19 : Fetching recipe details

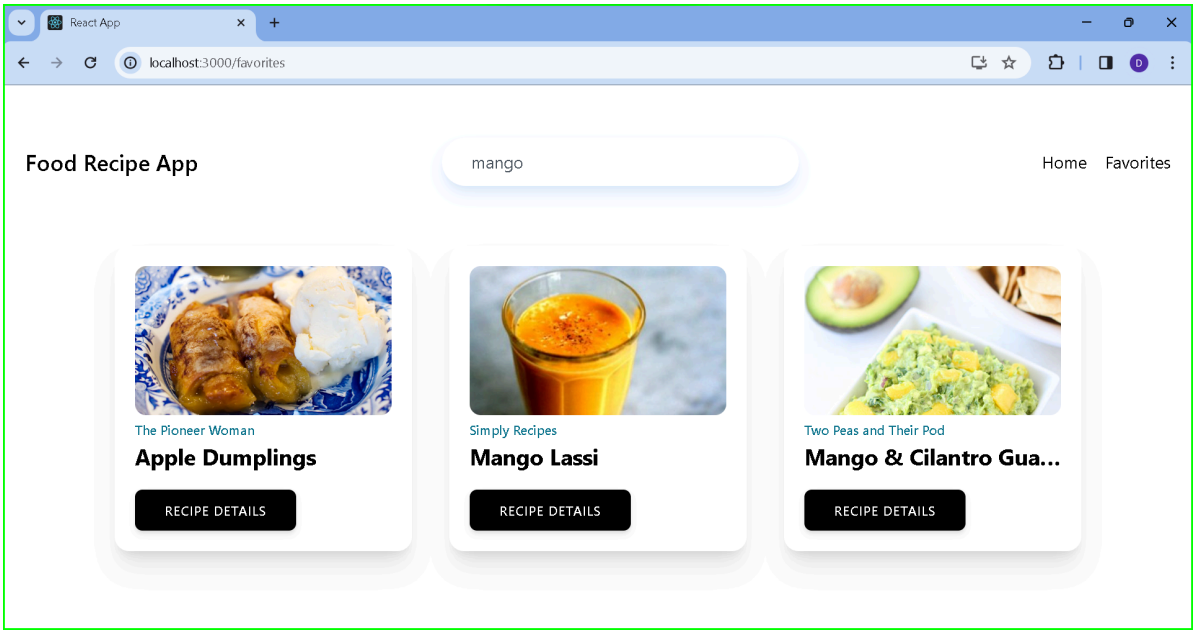


Fig 20 : Favorites page

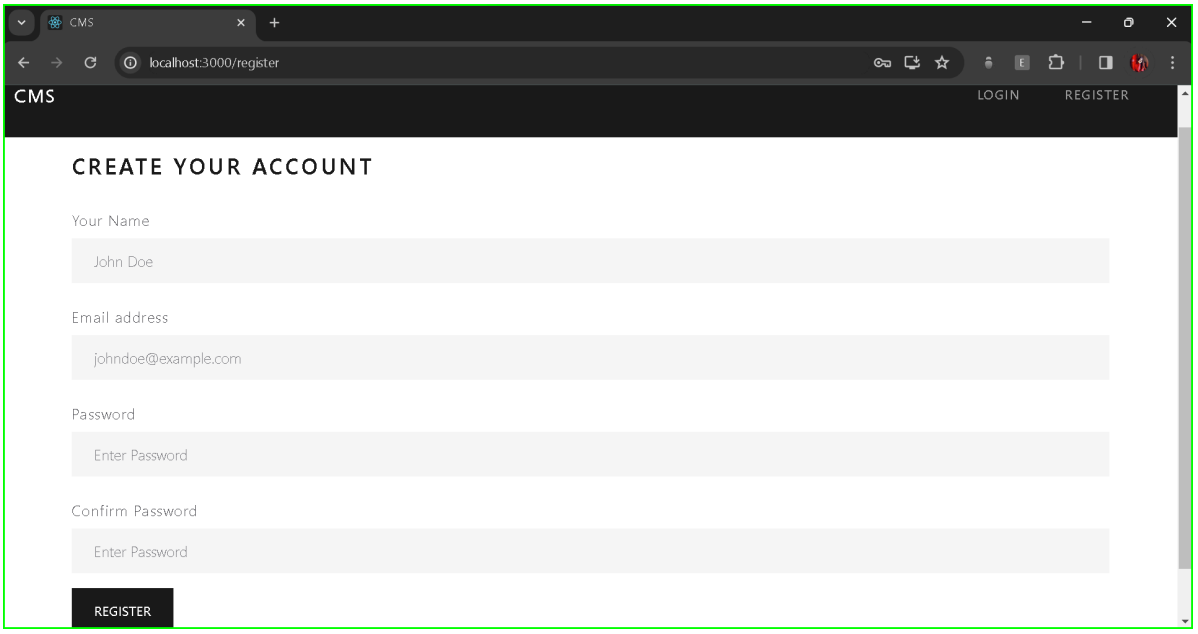


Fig 21 : Registering a new user

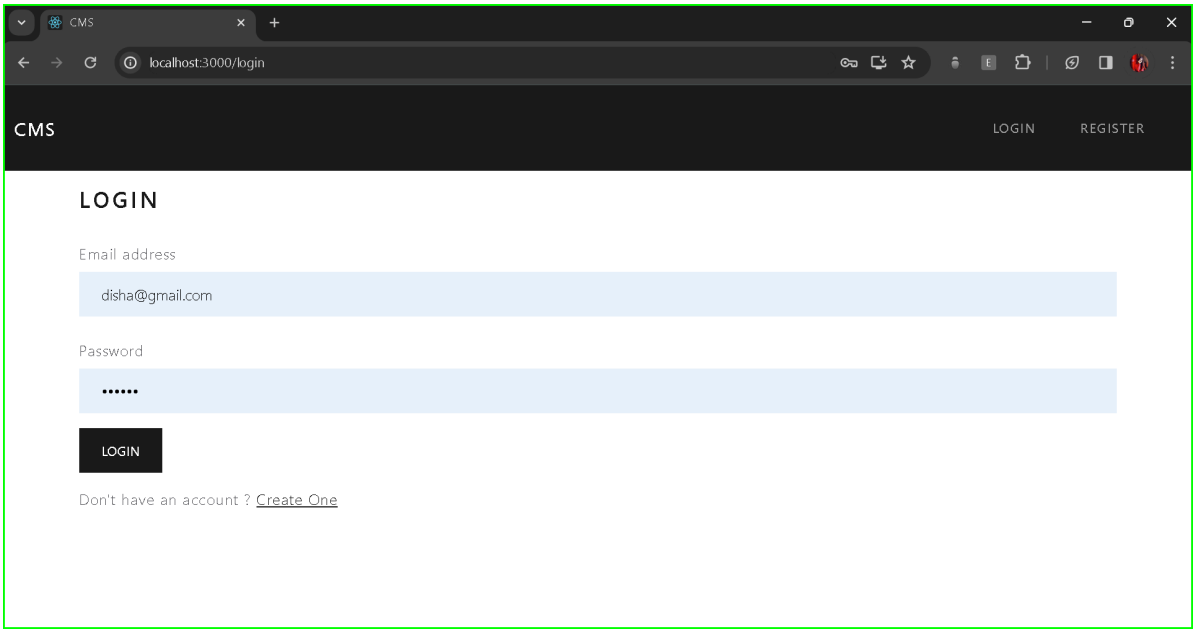


Fig 22 : Sign-in page for existing user

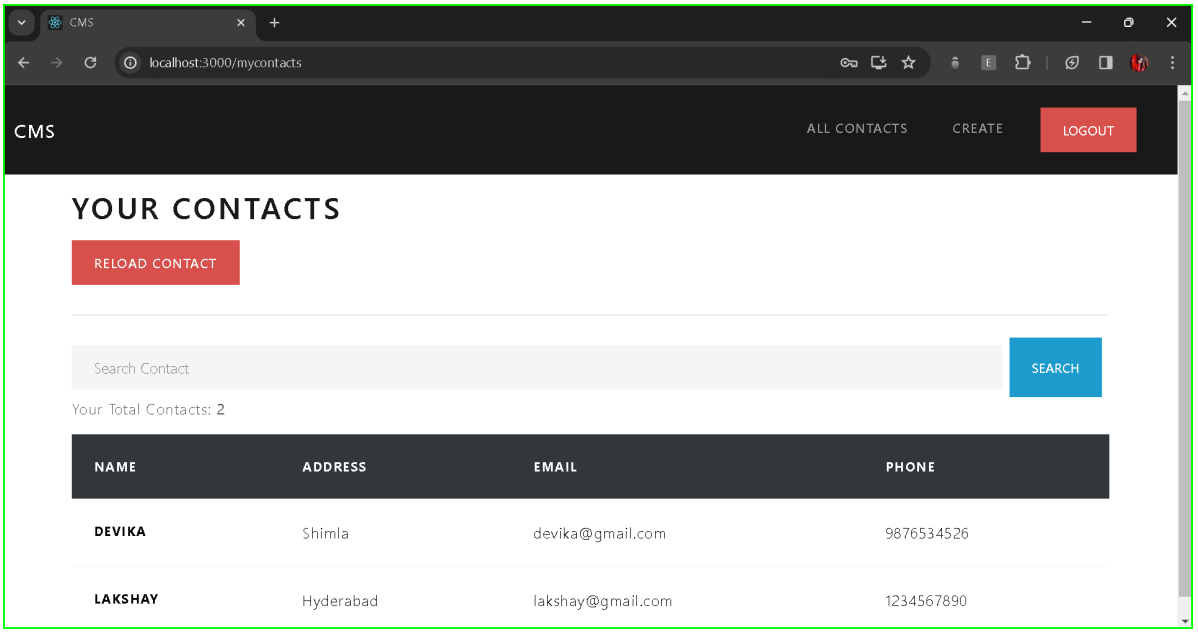


Fig 23 : CMS App Home page

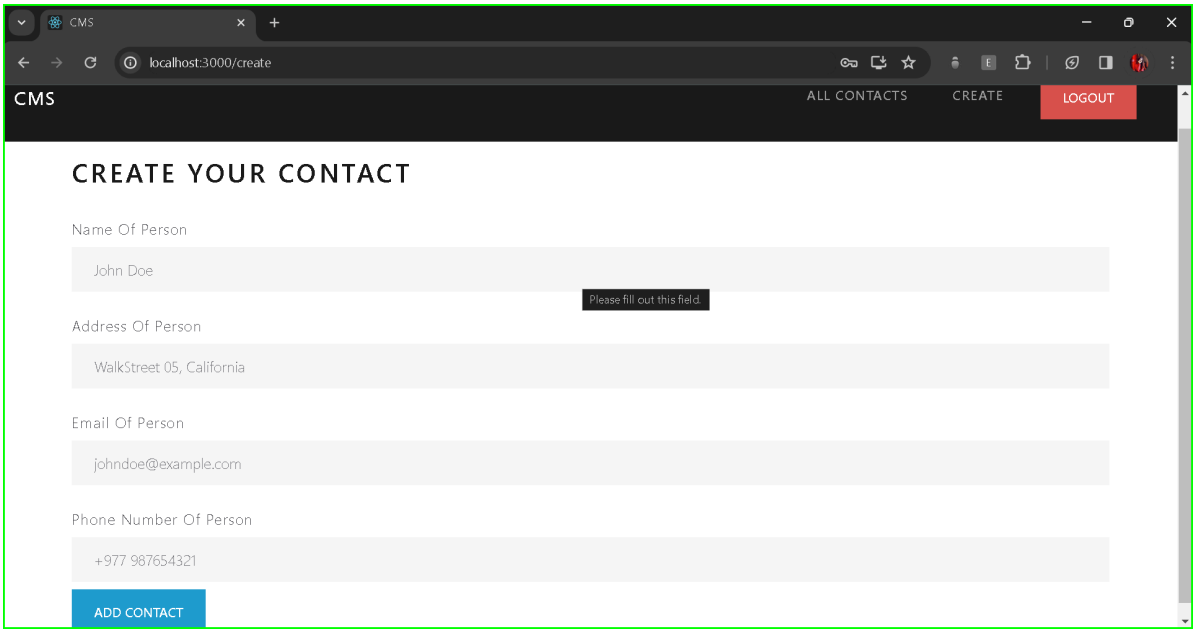


Fig 24 : Create new contact page

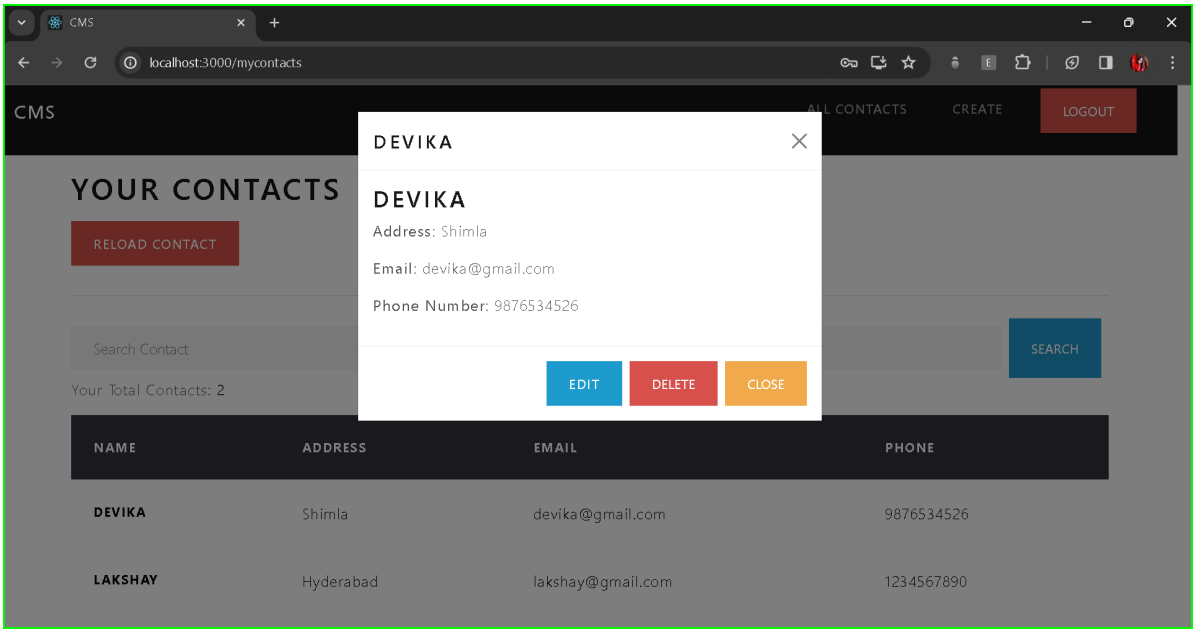


Fig 25 : Edit/Delete existing contact

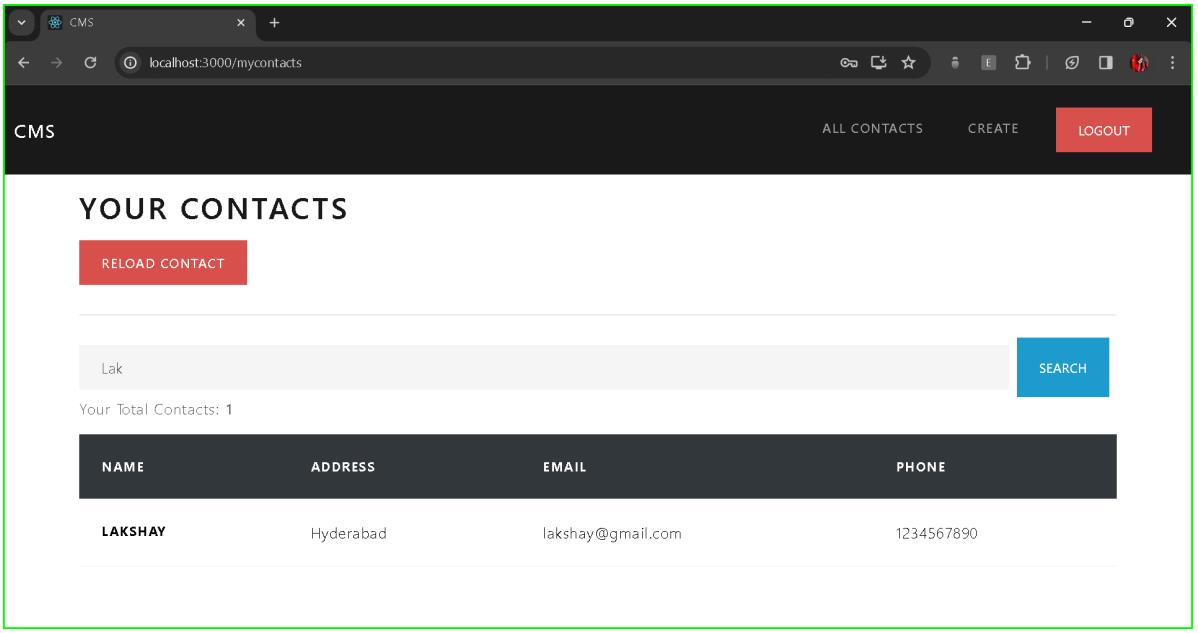


Fig 26 : Search for an existing contact

Visualization Dashboard App

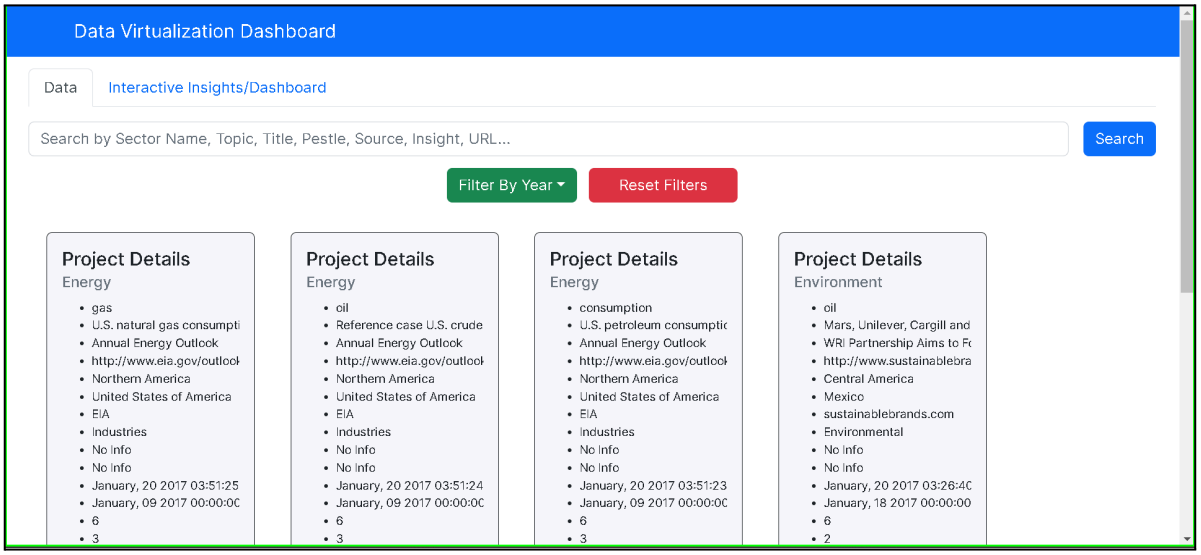


Fig 27 : Dashboard App - Home Page

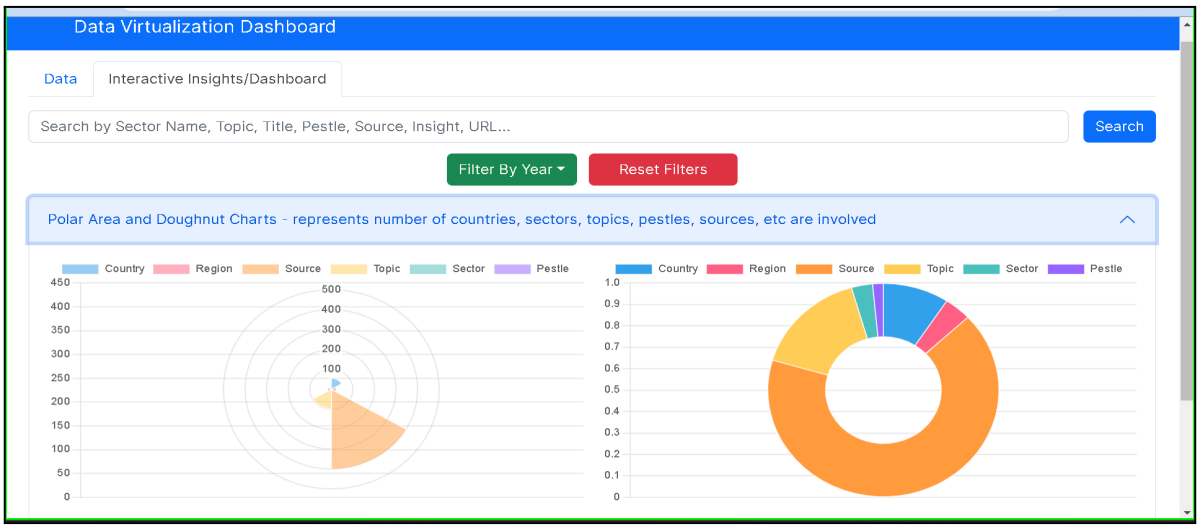


Fig 28 : Polar Area and Doughnut Charts

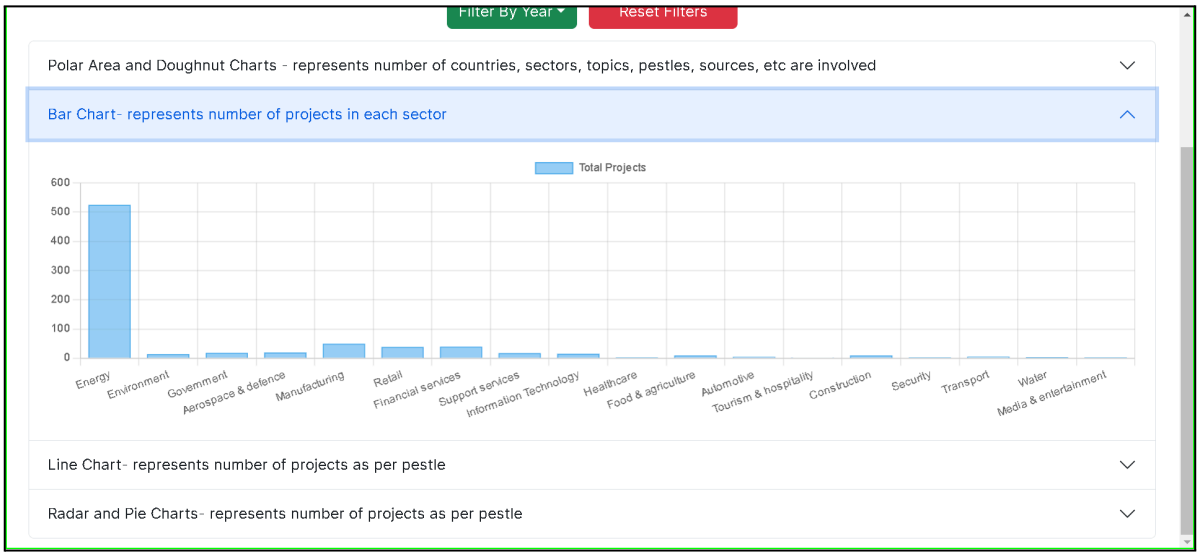


Fig 29 : Bar Chart

Fig 30 : Line Chart

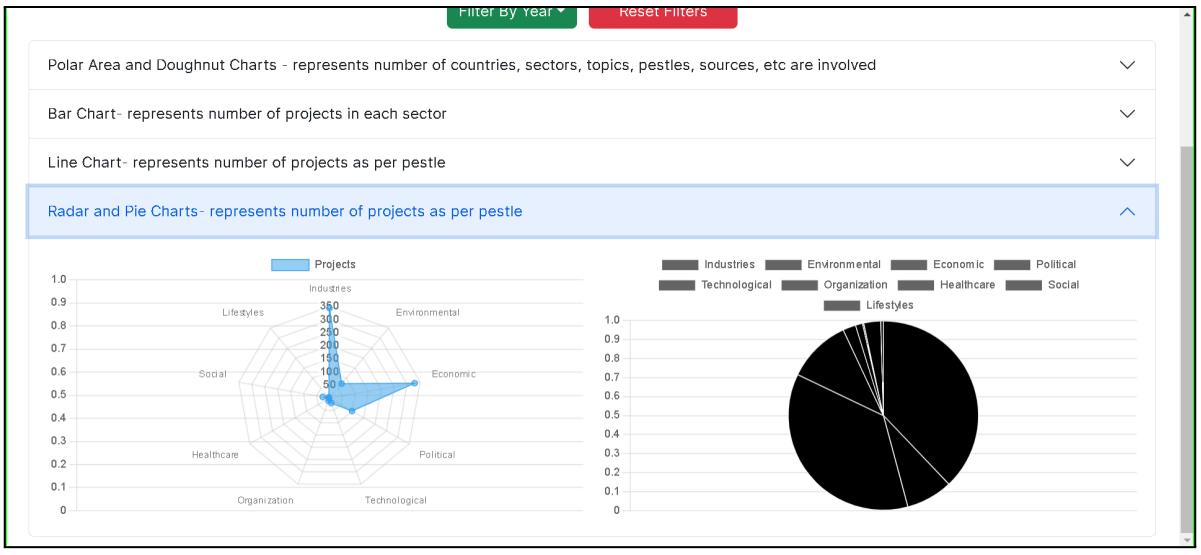


Fig 31 : Radar and Pie Charts

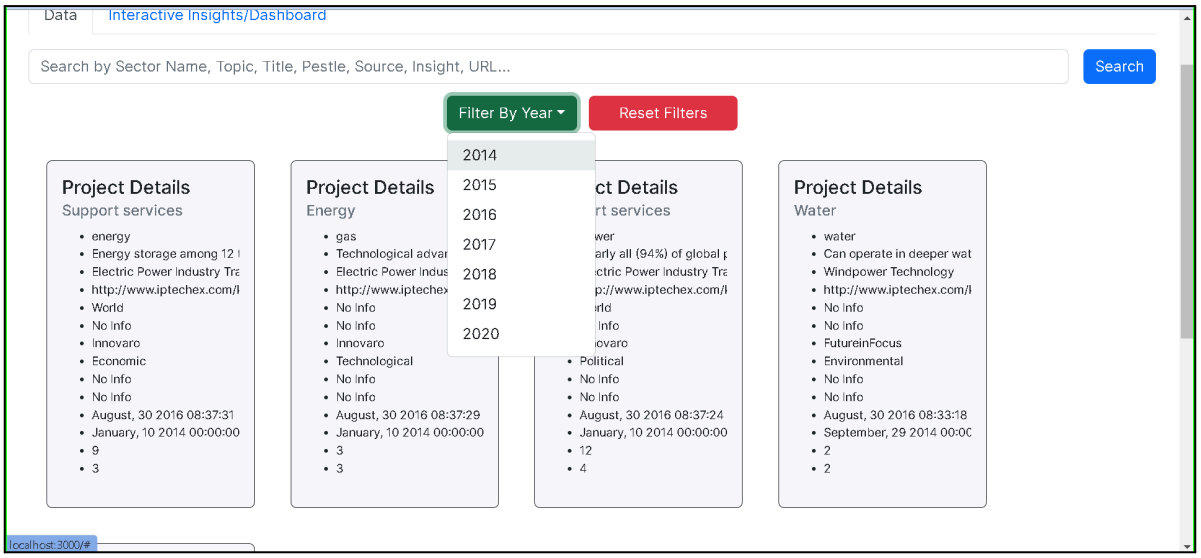


Fig 32 : Data Filtered by Year

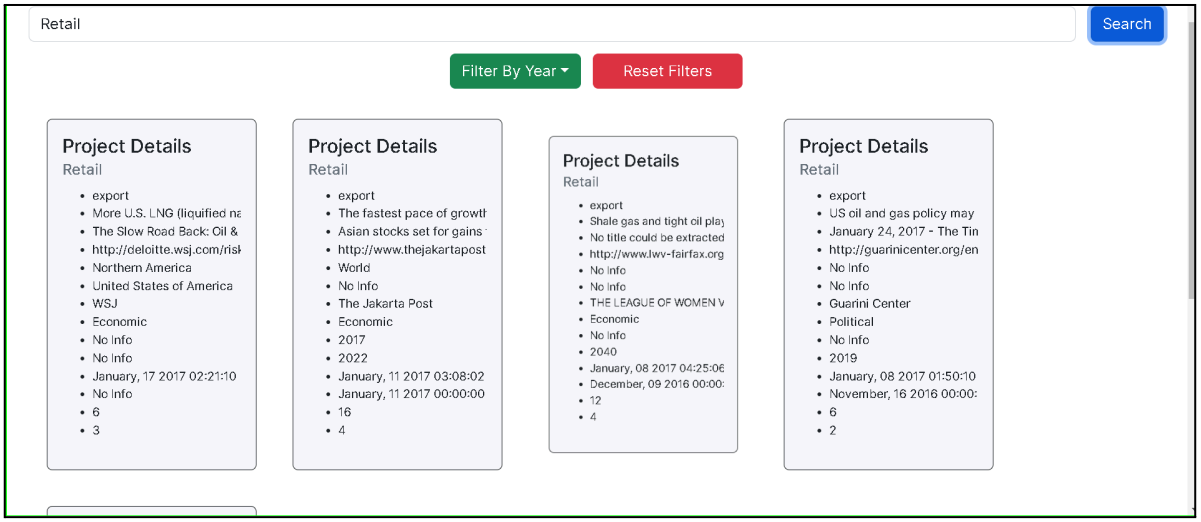


Fig 33 : Data Filtered by Sector Name

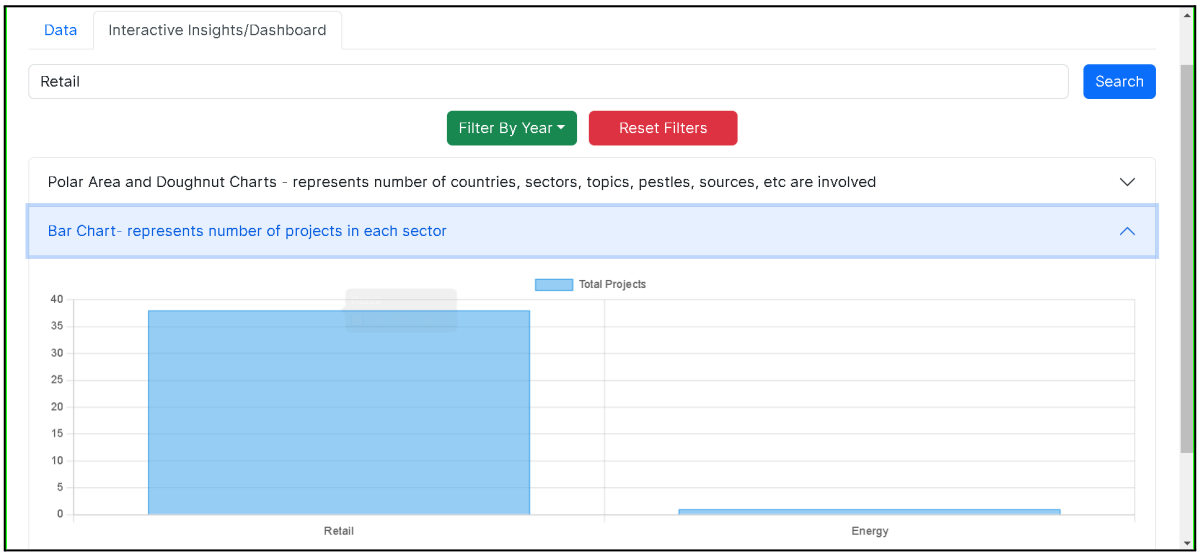


Fig 34 : Graph changes dynamically by searching the Sector Name

7. RESULT AND DISCUSSION

The completion of multiple projects spanning various domains showcases a diverse skill set and a comprehensive understanding of web development principles and technologies. Each project exhibits a meticulous approach to design, functionality, and user experience.

Starting with the Weather App, its intuitive user interface and dynamic data fetching capabilities provide users with real-time weather updates for their specified locations. Leveraging the OpenWeatherMap API ensures reliable data retrieval, functional, potential improvements include adding detailed forecasts and optimizing for scalability and maintainability and could explore adding more detailed forecasts and geolocation functionality for enhanced usability. Nonetheless, the project serves as a valuable learning experience in frontend development.

The Food Recipe App extends the user experience by offering a seamless recipe browsing experience. Implemented functionalities like fetching recipes from the Forkify API and global state management using React Context API ensure efficient data handling and consistent performance. While the app meets current requirements admirably, potential future enhancements could involve user authentication, personalized recommendations, and social media integration for a more interactive experience.

Transitioning to the Contact Management Application, the project demonstrates the effectiveness of the MERN stack in facilitating web application development. While basic CRUD and search functionality is present, future iterations could focus on adding features like contact categorization and advanced search filters to enhance usability. Ensuring robust security measures and optimizing performance for large datasets would be crucial for maintaining a responsive and secure user experience.

Finally, the Data Visualization Dashboard project provides users with a powerful tool for exploring and analyzing data trends. While the current implementation offers flexibility in data filtering and visualization, future improvements could involve enhancing filtering capabilities and implementing user authentication and authorization features for improved security. Additionally, incorporating feedback mechanisms and analytics tools would enable

continuous refinement based on user usage patterns. Nevertheless, the project lays a solid foundation for further customization and development to meet the evolving needs of users in data analysis and decision-making.

Overall, these projects demonstrate the value of practical training in frontend and backend development, offering opportunities for technical learning and creative problem-solving. Through iterative refinement and continuous improvement, each project has the potential to evolve into valuable resources for users in their respective domains. Future scope includes exploring advanced functionalities, improving security measures, optimizing performance, and incorporating user feedback to ensure that each application continues to meet the evolving needs of its users effectively. Through iterative development and continuous improvement, these projects have the potential to evolve into even more robust and feature-rich applications, further showcasing expertise and creativity in web development.

8. SUMMARY

This project report encapsulates the successful execution of four distinct web development apps, each tailored to address specific user requirements and technological challenges.

The Weather App provides real-time weather updates for designated cities, featuring an intuitive interface and dynamic data fetching capabilities via the OpenWeatherMap API.
The Food Recipe App offers users a comprehensive platform to explore, save, and search for recipes effortlessly. Key functionalities include recipe fetching and global state management, ensuring efficient data handling and a consistent user experience.

The Contact Management Application utilizes the MERN stack architecture to provide a flexible solution for contact management. It effectively handles basic CRUD and search operations, offering users a seamless experience in managing their contacts. And lastly, the Data Visualization Dashboard project empowers users to analyze data trends through interactive charts and graphs. Its flexible data filtering and visualization capabilities provide users with a powerful tool for data analysis.

Therefore, each project demonstrates a meticulous approach to design and functionality, addressing specific user needs within their respective domains of weather information, recipe browsing, contact management, and data visualization.

9. REFERENCES

Journals:

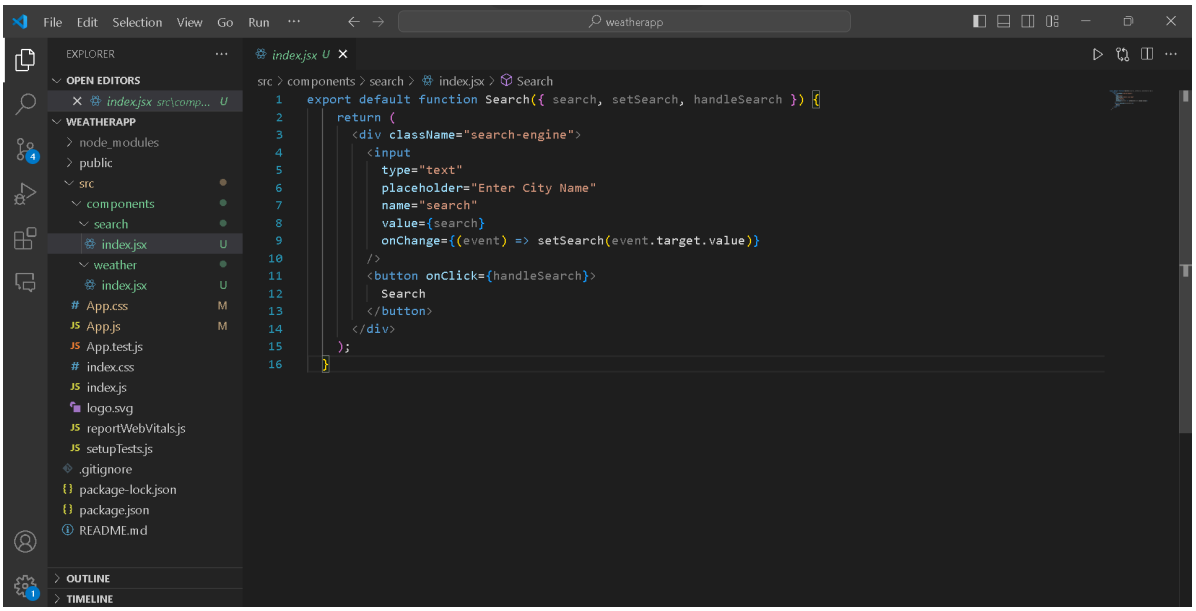
1. Avinash Mishra, Arshita Gupta, “React JS – A Frontend Javascript Library”, International Research Journal of Modernization in Engineering Technology and Science (IRJMETS), November 2022.
2. Yogesh Baiskar, Priyas Paulzagade, Krutik Koradia, Pramod Ingole, Dhiraj Shirbhate, “MERN: A Full-Stack Development”, International Journal for Research in Applied Science and Engineering Technology (IJRASET), January 2022.
3. Yogesh Kadam, Akhil Goplani, Shubit Mattoo, Shashank Kumar Gupta, Darshan Amrutkar, Jyoti Dhanke “Introduction to MERN Stack & Comparison with Previous Technologies”, European Chemical Bulletin , June 2023.
4. Ciprian-Octavian Truică, Alexandru Boicea, Ionuț Trifan, “CRUD Operations in MongoDB”, International Conference on Advanced Computer Science and Electronics Information (ICACSEI), July 2013.
5. Isha Bankar, Anjali Chiwande, Namita Ghadse, Trupti Shastrakar, “Contact Management System”, Journal of Emerging Technologies and Innovative Research (JETIR), June 2022.
6. Kaarshnee Dewan, Aadith Lasar, Bhushan Bhokse, “ FinanceVue - A MERN Stack Finance Dashboard Application”, International Journal of Novel Research and Development (IJNRD), April 2024.

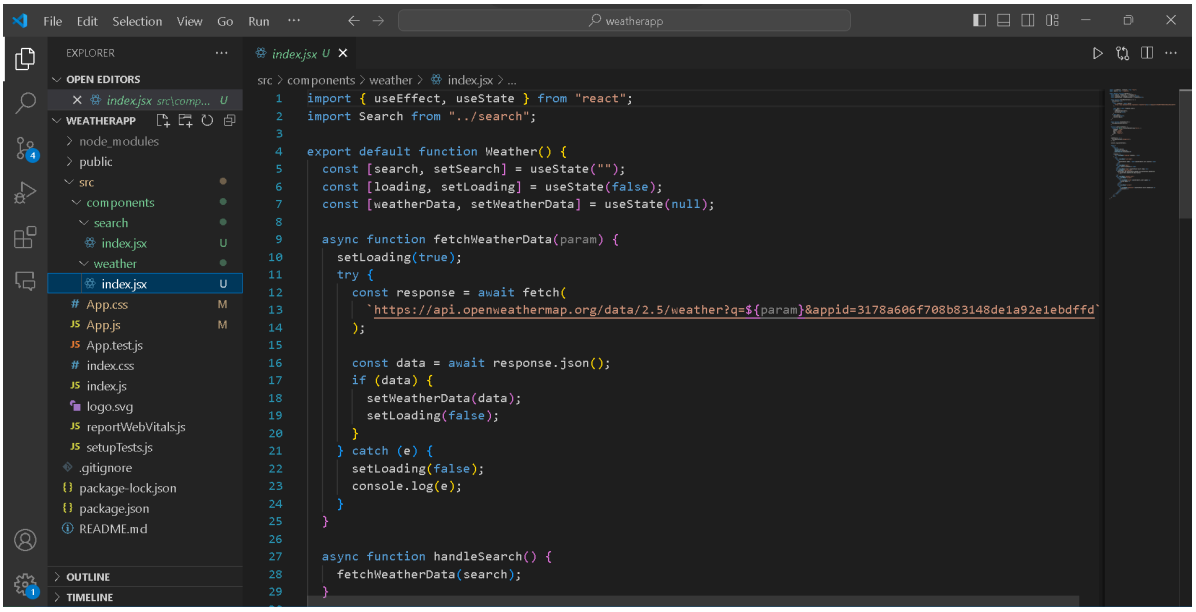
Weblinks:

1. Stackoverflow**.** https://stackoverflow.com/
2. React. https://react.dev/
3. Oracle. https://www.oracle.com/in/database/mern-stack/
4. MongoDB. https://www.mongodb.com/mern-stack
5. Express JS. https://expressjs.com/
6. Node JS. https://nodejs.org/en
7. Javatpoint. https://www.javatpoint.com/mern-stack
8. GFG. https://www.geeksforgeeks.org/mern-stack-development-roadmap/

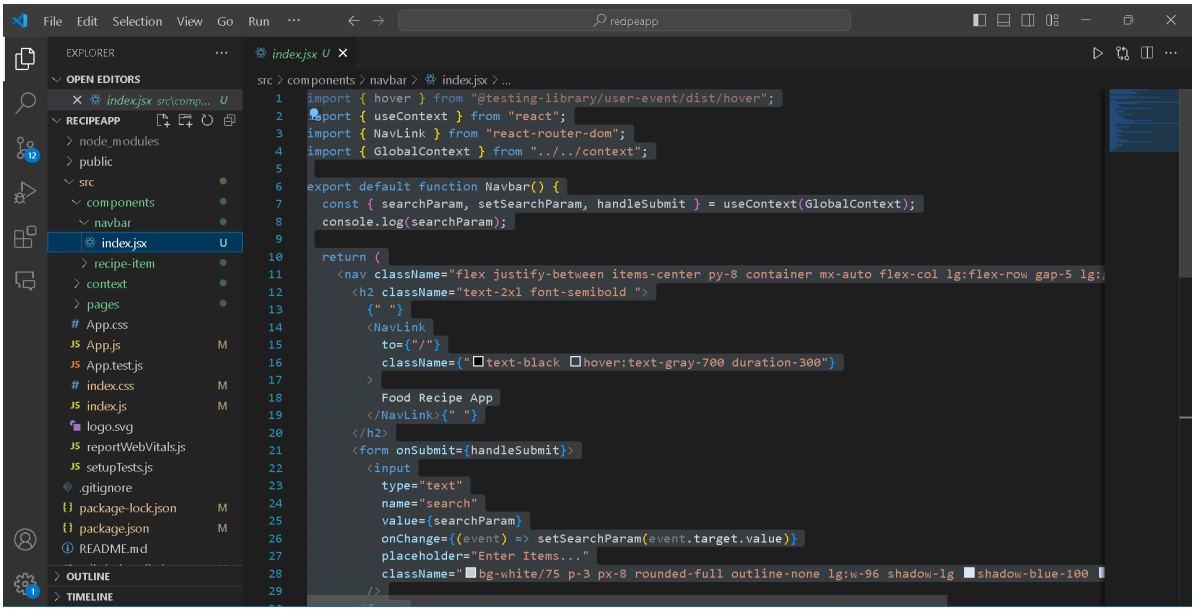
APPENDIX A – SAMPLE CODE

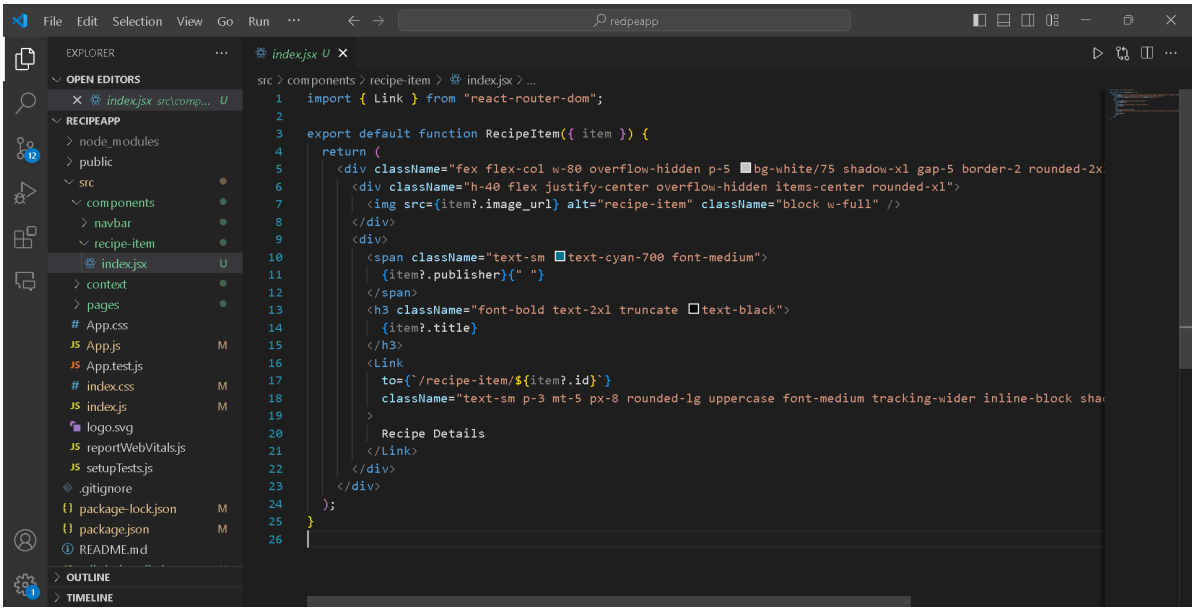
Weather App Code:

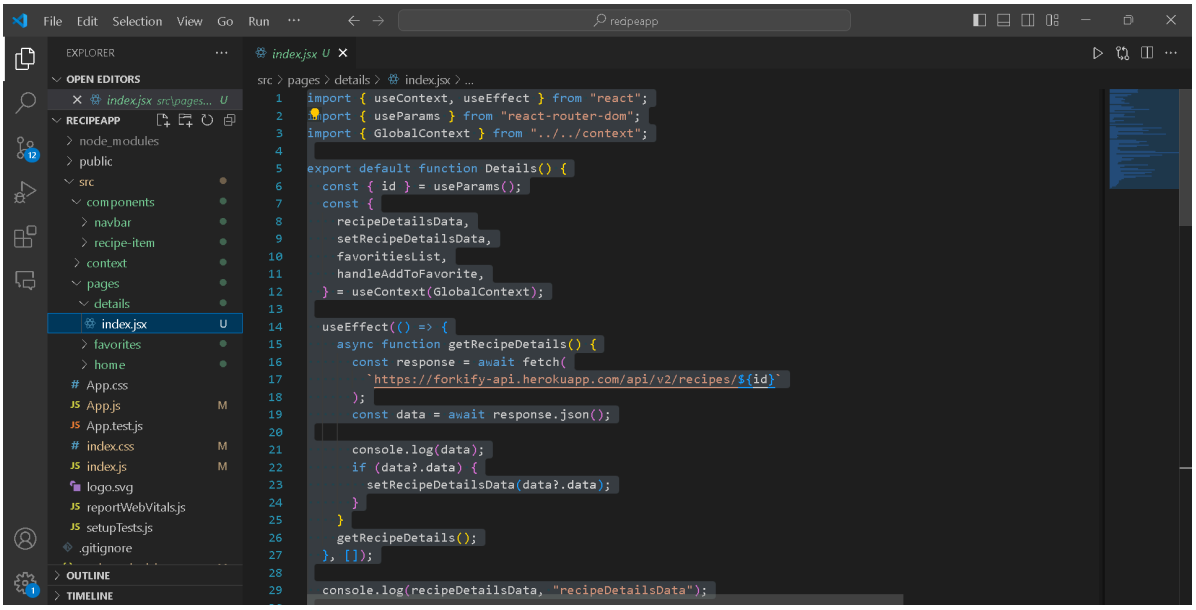




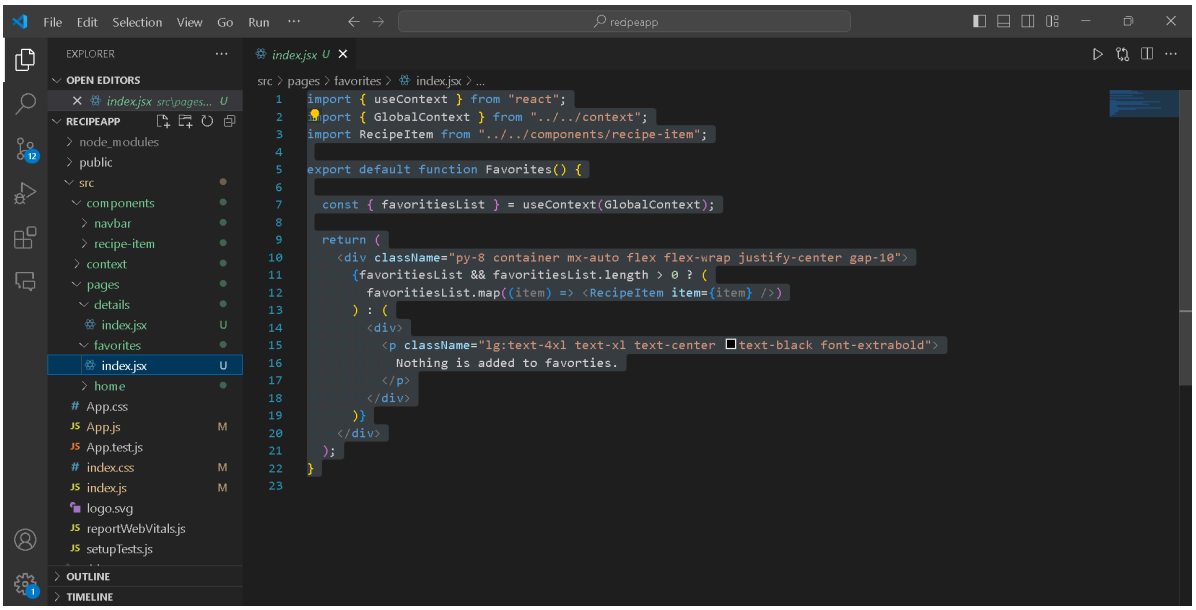
Recipe App Code:



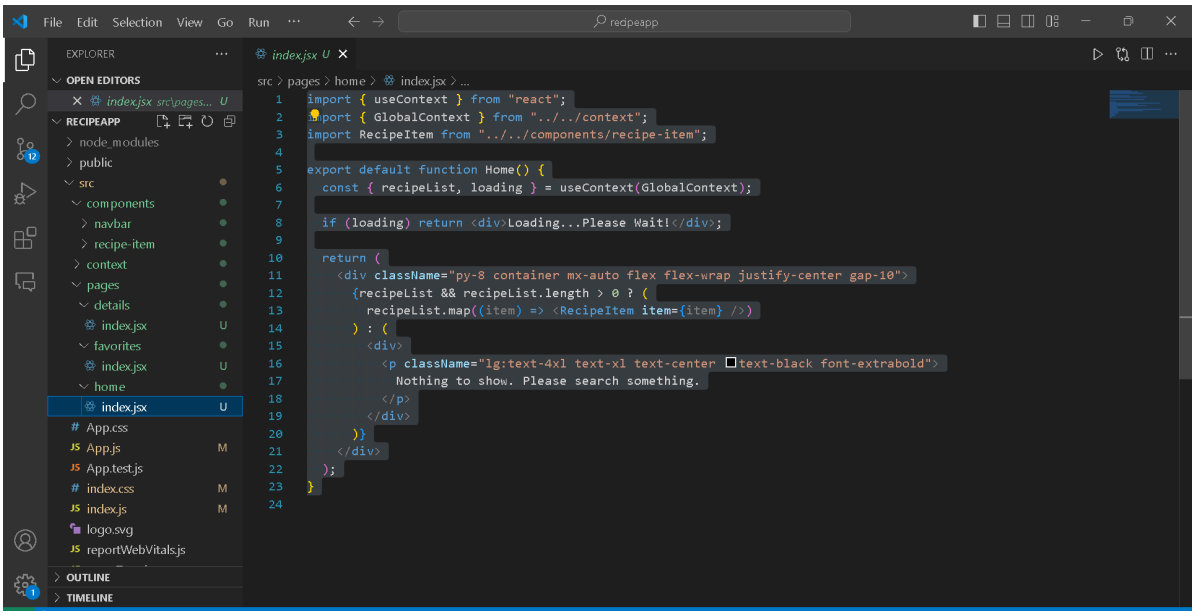




```
import { useContext, useEffect } from "react";
import { useParams } from "react-router-dom";
import { GlobalContext } from "../../context";

export default function Details() {
  const { id } = useParams();
  const {
    recipeDetailsData,
    setRecipeDetailsData,
    favoritiesList,
    handleAddToFavorite,
  } = useContext(GlobalContext);

  useEffect(() => {
    async function getRecipeDetails() {
      const response = await fetch(
        `https://forkify-api.herokuapp.com/api/v2/recipes/${id}`
      );
      const data = await response.json();

      console.log(data);
      if (data?.data) {
        setRecipeDetailsData(data?.data);
      }
    }
    getRecipeDetails();
  }, []);

  console.log(recipeDetailsData, "recipeDetailsData");
```



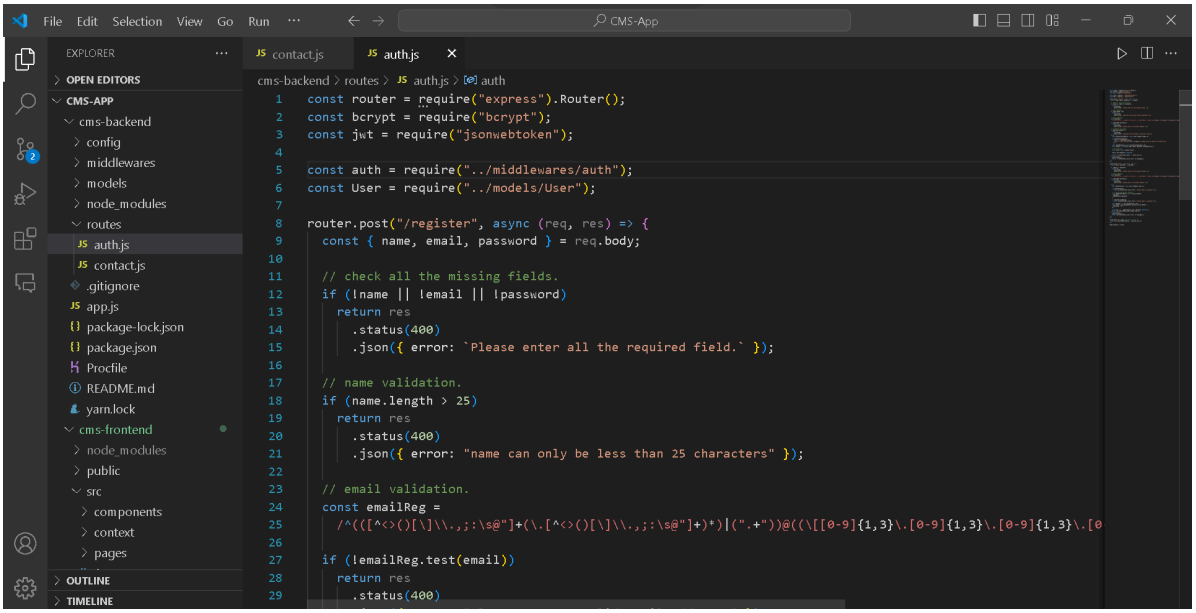
```
import { useContext } from "react";
import { GlobalContext } from "../../context";
import RecipeItem from "../../components/recipe-item";

export default function Favorites() {

  const { favoritiesList } = useContext(GlobalContext);

  return (
    <div className="py-8 container mx-auto flex flex-wrap justify-center gap-10">
      {favoritiesList && favoritiesList.length > 0 ? (
        favoritiesList.map((item) => <RecipeItem item={item} />)
      ) : (
        <div>
          <p className="lg:text-4xl text-xl text-center text-black font-extrabold">
            Nothing is added to favorties.
          </p>
        </div>
      )}
    </div>
  );
}
```

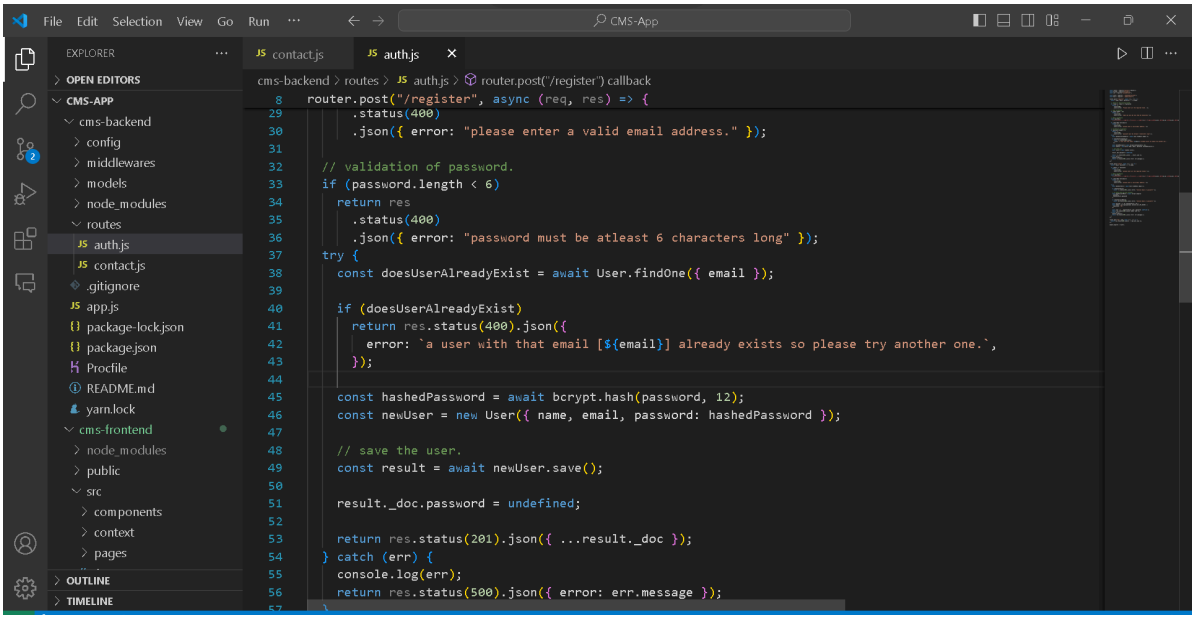


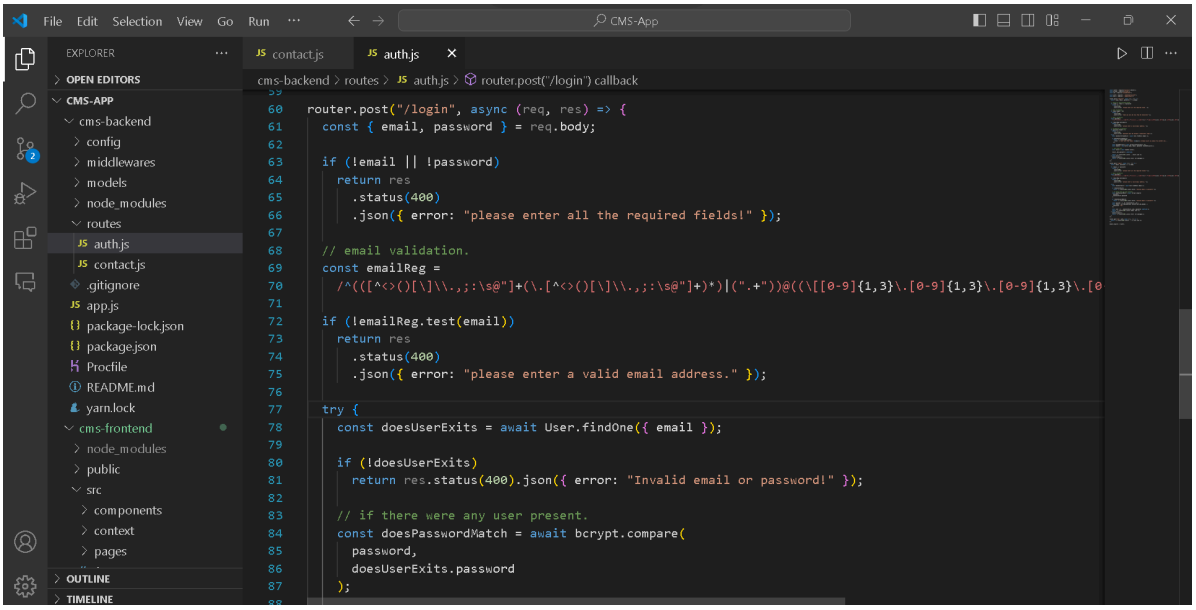
```
import { useContext } from "react";
import { GlobalContext } from "../../context";
import RecipeItem from "../../components/recipe-item";

export default function Home() {
  const { recipeList, loading } = useContext(GlobalContext);

  if (loading) return <div>Loading...Please Wait!</div>;

  return (
    <div className="py-8 container mx-auto flex flex-wrap justify-center gap-10">
      {recipeList && recipeList.length > 0 ? (
        recipeList.map((item) => <RecipeItem item={item} />)
      ) : (
        <div>
          <p className="lg:text-4xl text-xl text-center text-black font-extrabold">
            Nothing to show. Please search something.
          </p>
        </div>
      )}
    </div>
  );
}
```

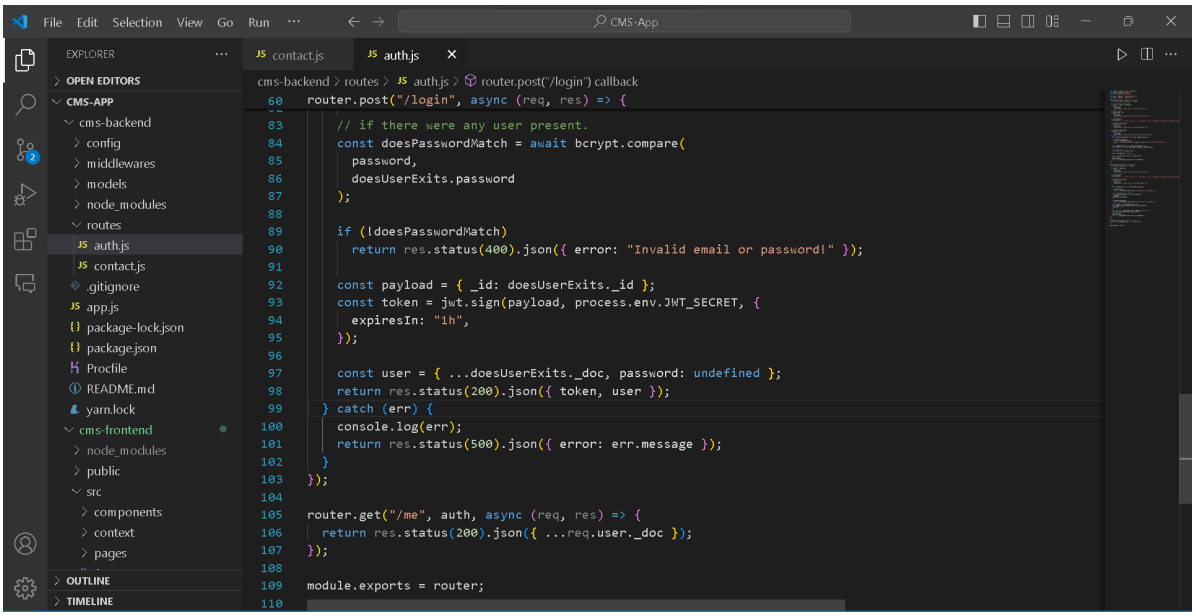
CMS App Code Link:

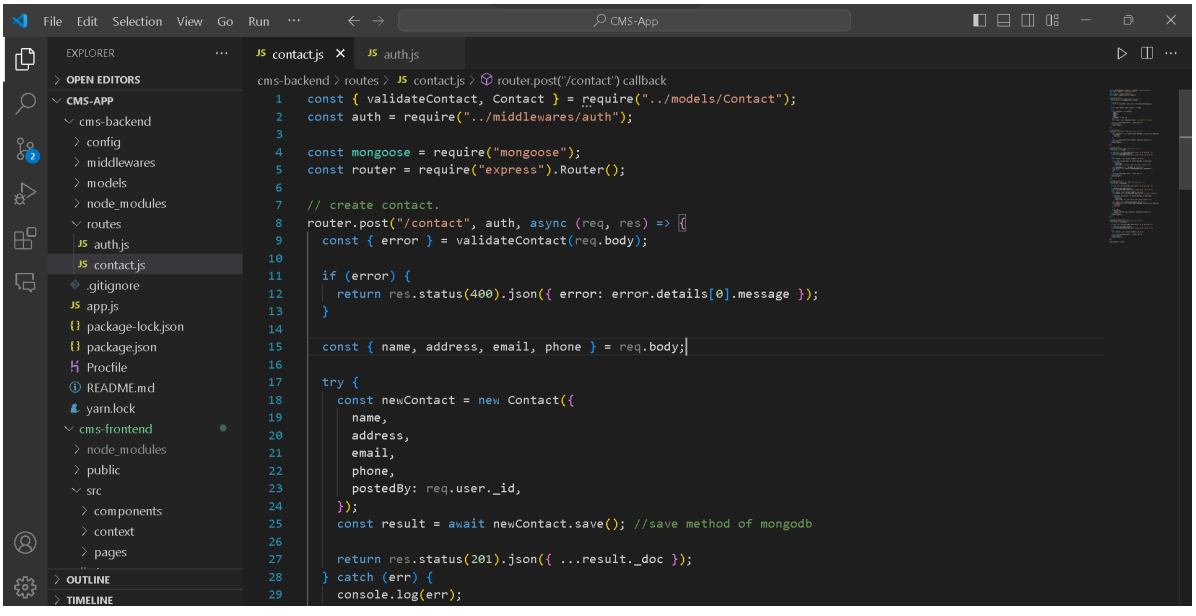
https://drive.google.com/file/d/1ibKNgpBLuGCDQjBE-AjNNErxINyCl4l9/view?usp=sharing

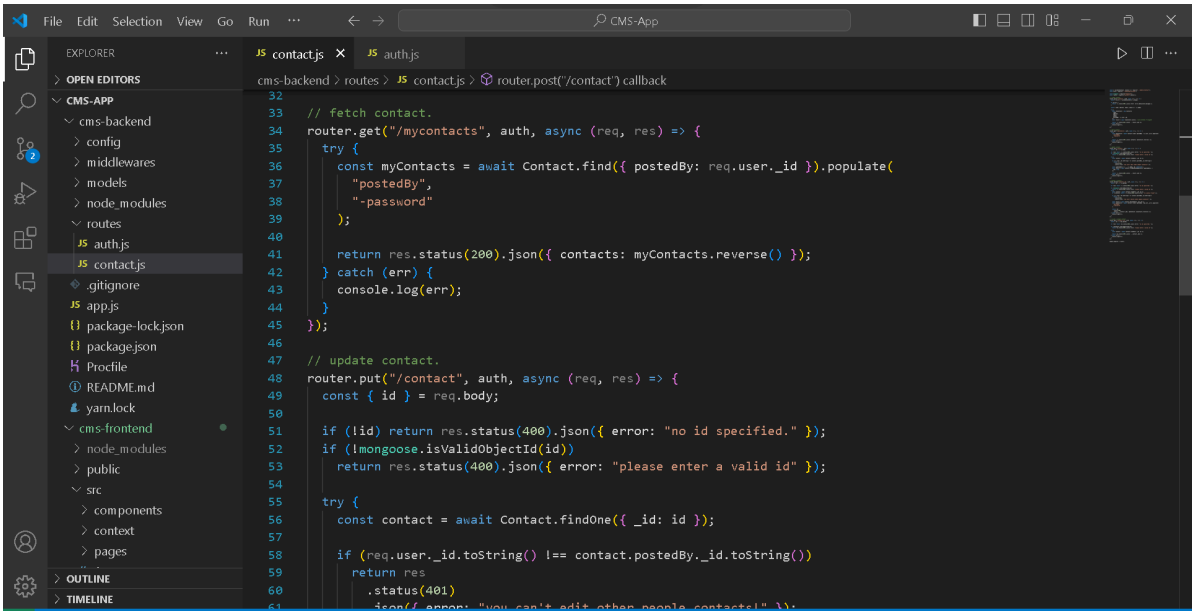


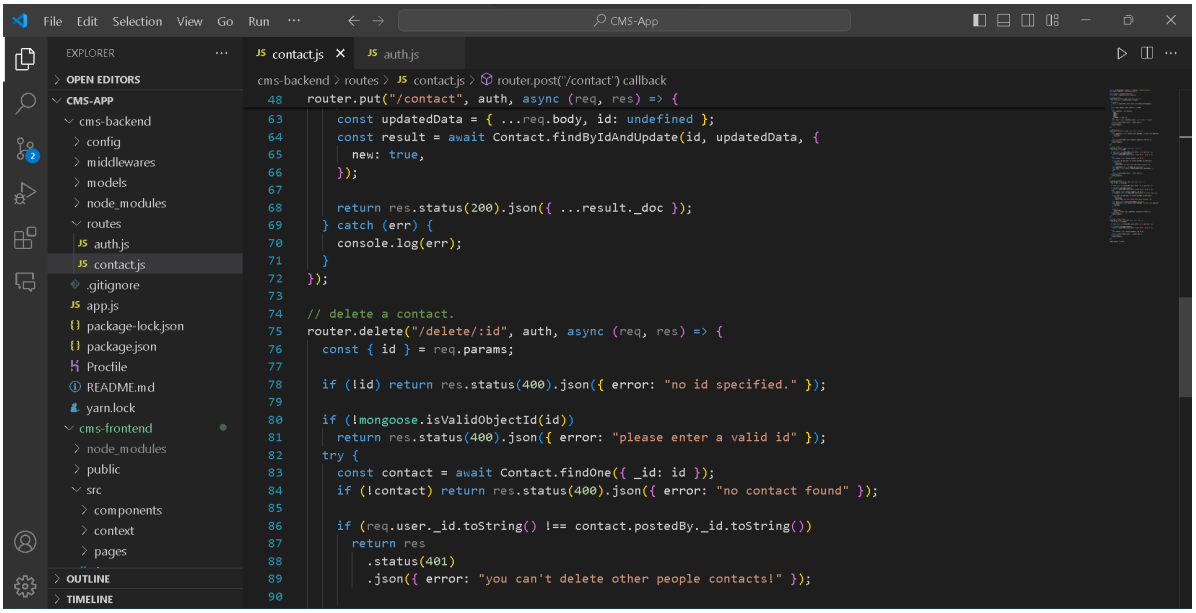


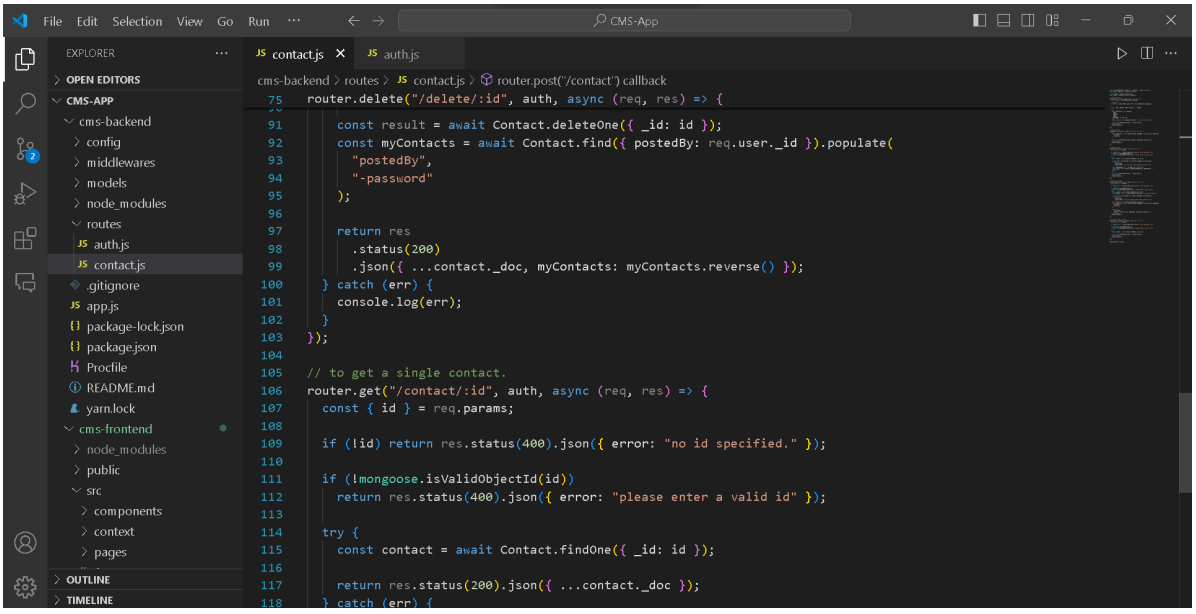

File Edit Selection View Go Run
CMS-App
EXPLORER
contact.js
auth.js
cms-backend > routes > auth.js > router.post("/login") callback
OPEN EDITORS
CMS-APP
cms-backend
config
middlewares
models
node_modules
routes
auth.js
contact.js
.gitignore
app.js
package-lock.json
package.json
Procfile
README.md
yarn.lock
cms-frontend
node_modules
public
src
components
context
pages
OUTLINE
TIMELINE
router.post("/login", async (req, res) => {
const { email, password } = req.body;
if (!email || !password)
return res
.status(400)
.json({ error: "please enter all the required fields!" });
// email validation.
const emailReg =
/^(([^<>()[\]\\.,;:\s@"]+(\.[^<>()[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0
if (!emailReg.test(email))
return res
.status(400)
.json({ error: "please enter a valid email address." });
try {
const doesUserExits = await User.findOne({ email });
if (!doesUserExits)
return res.status(400).json({ error: "Invalid email or password!" });
// if there were any user present.
const doesPasswordMatch = await bcrypt.compare(
password,
doesUserExits.password
);


File Edit Selection View Go Run
CMS-App
EXPLORER
contact.js
auth.js
cms-backend > routes > auth.js > router.post("/login") callback
router.post("/login", async (req, res) => {
// if there were any user present.
const doesPasswordMatch = await bcrypt.compare(
password,
doesUserExits.password
);
if (!doesPasswordMatch)
return res.status(400).json({ error: "Invalid email or password!" });
const payload = { _id: doesUserExits._id };
const token = jwt.sign(payload, process.env.JWT_SECRET, {
expiresIn: "1h",
});
const user = { ...doesUserExits._doc, password: undefined };
return res.status(200).json({ token, user });
} catch (err) {
console.log(err);
return res.status(500).json({ error: err.message });
}
});
router.get("/me", auth, async (req, res) => {
return res.status(200).json({ ...req.user._doc });
});
module.exports = router;


File Edit Selection View Go Run
CMS-App
EXPLORER
contact.js
auth.js
cms-backend > routes > contact.js > router.post("/contact") callback
const { validateContact, Contact } = require("../models/Contact");
const auth = require("../middlewares/auth");
const mongoose = require("mongoose");
const router = require("express").Router();
// create contact.
router.post("/contact", auth, async (req, res) => {
const { error } = validateContact(req.body);
if (error) {
return res.status(400).json({ error: error.details[0].message });
}
const { name, address, email, phone } = req.body;
try {
const newContact = new Contact({
name,
address,
email,
phone,
postedBy: req.user._id,
});
const result = await newContact.save(); //save method of mongodb
return res.status(201).json({ ...result._doc });
} catch (err) {
console.log(err);

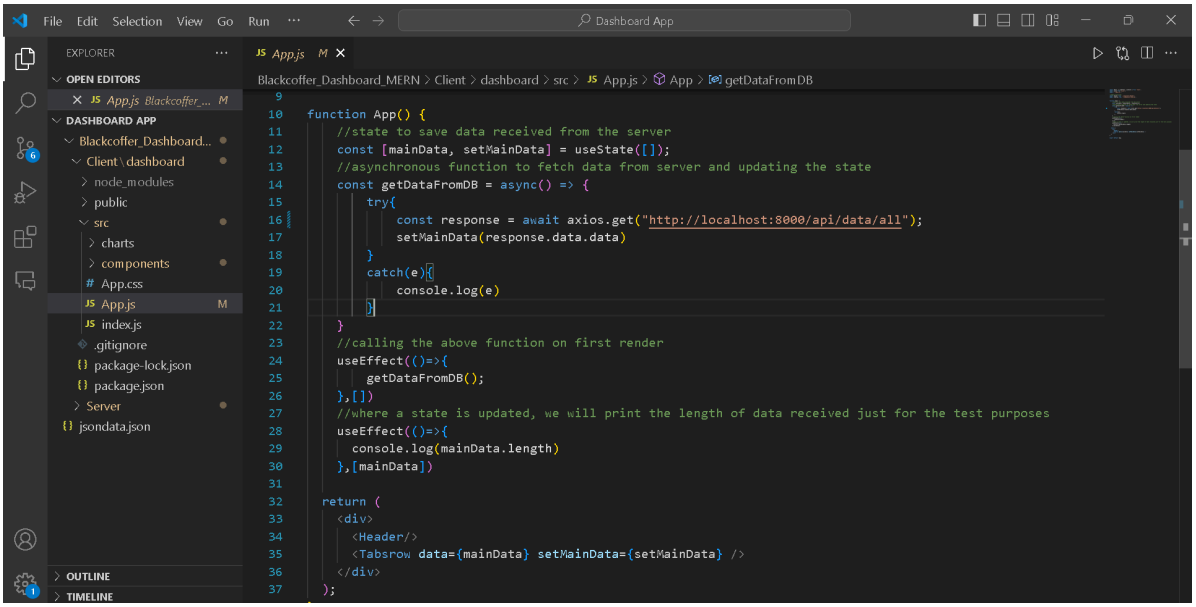

File Edit Selection View Go Run ···
CMS-App
EXPLORER
contact.js
auth.js
cms-backend > routes > contact.js > router.post("/contact") callback
OPEN EDITORS
CMS-APP
cms-backend
config
middlewares
models
node_modules
routes
auth.js
contact.js
.gitignore
app.js
package-lock.json
package.json
Procfile
README.md
yarn.lock
cms-frontend
node_modules
public
src
components
context
pages
OUTLINE
TIMELINE
// fetch contact.
router.get("/mycontacts", auth, async (req, res) => {
try {
const myContacts = await Contact.find({ postedBy: req.user._id }).populate(
"postedBy",
"-password"
);
return res.status(200).json({ contacts: myContacts.reverse() });
} catch (err) {
console.log(err);
}
});
// update contact.
router.put("/contact", auth, async (req, res) => {
const { id } = req.body;
if (!id) return res.status(400).json({ error: "no id specified." });
if (!mongoose.isValidObjectId(id))
return res.status(400).json({ error: "please enter a valid id" });
try {
const contact = await Contact.findOne({ _id: id });
if (req.user._id.toString() !== contact.postedBy._id.toString())
return res
.status(401)

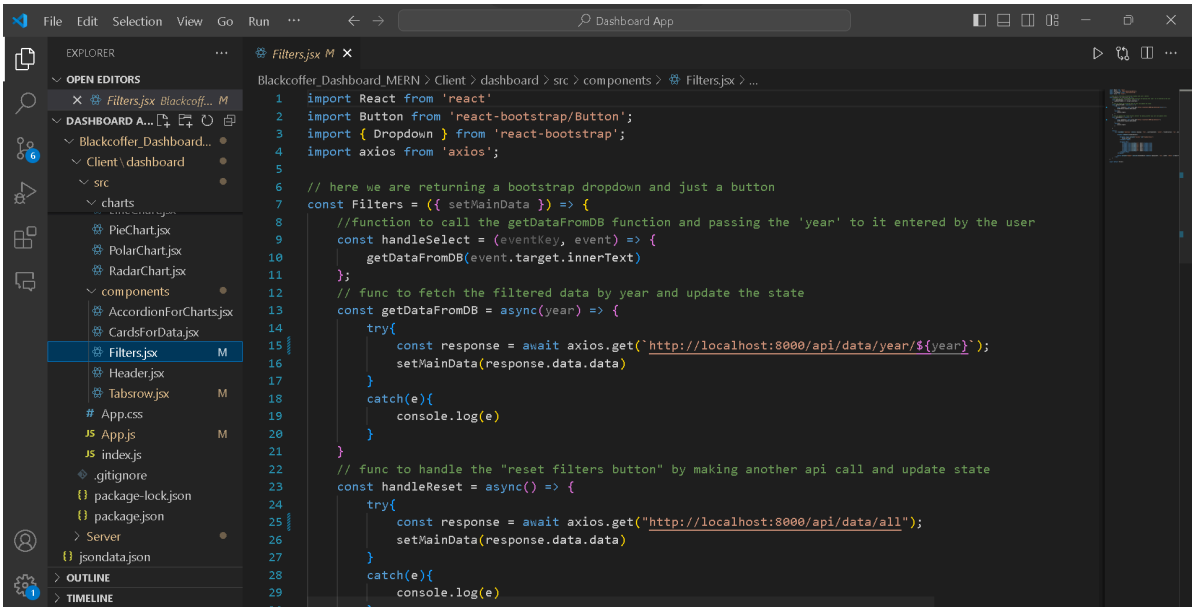

File Edit Selection View Go Run ···
CMS-App
EXPLORER
contact.js
auth.js
cms-backend > routes > contact.js > router.post("/contact") callback
router.put("/contact", auth, async (req, res) => {
const updatedData = { ...req.body, id: undefined };
const result = await Contact.findByIdAndUpdate(id, updatedData, {
new: true,
});
return res.status(200).json({ ...result._doc });
} catch (err) {
console.log(err);
}
});
// delete a contact.
router.delete("/delete/:id", auth, async (req, res) => {
const { id } = req.params;
if (!id) return res.status(400).json({ error: "no id specified." });
if (!mongoose.isValidObjectId(id))
return res.status(400).json({ error: "please enter a valid id" });
try {
const contact = await Contact.findOne({ _id: id });
if (!contact) return res.status(400).json({ error: "no contact found" });
if (req.user._id.toString() !== contact.postedBy._id.toString())
return res
.status(401)
.json({ error: "you can't delete other people contacts!" });

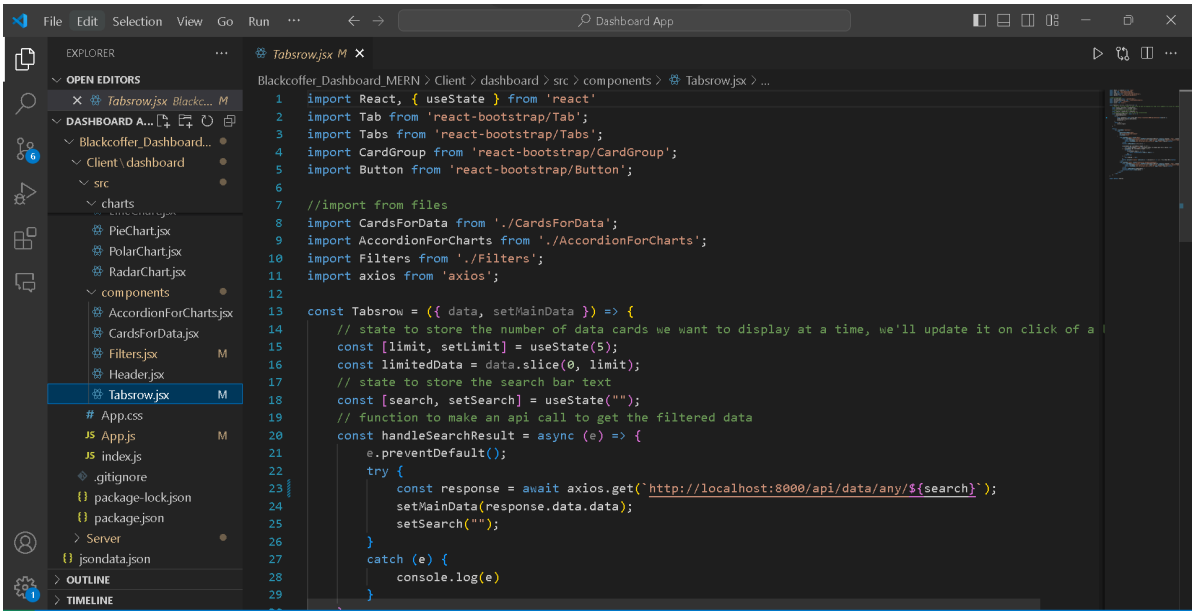

File Edit Selection View Go Run ···
CMS-App
EXPLORER
contact.js
auth.js
cms-backend > routes > contact.js > router.post("/contact") callback
router.delete("/delete/:id", auth, async (req, res) => {
const result = await Contact.deleteOne({ _id: id });
const myContacts = await Contact.find({ postedBy: req.user._id }).populate(
"postedBy",
"-password"
);
return res
.status(200)
.json({ ...contact._doc, myContacts: myContacts.reverse() });
} catch (err) {
console.log(err);
}
});
// to get a single contact.
router.get("/contact/:id", auth, async (req, res) => {
const { id } = req.params;
if (!id) return res.status(400).json({ error: "no id specified." });
if (!mongoose.isValidObjectId(id))
return res.status(400).json({ error: "please enter a valid id" });
try {
const contact = await Contact.findOne({ _id: id });
return res.status(200).json({ ...contact._doc });

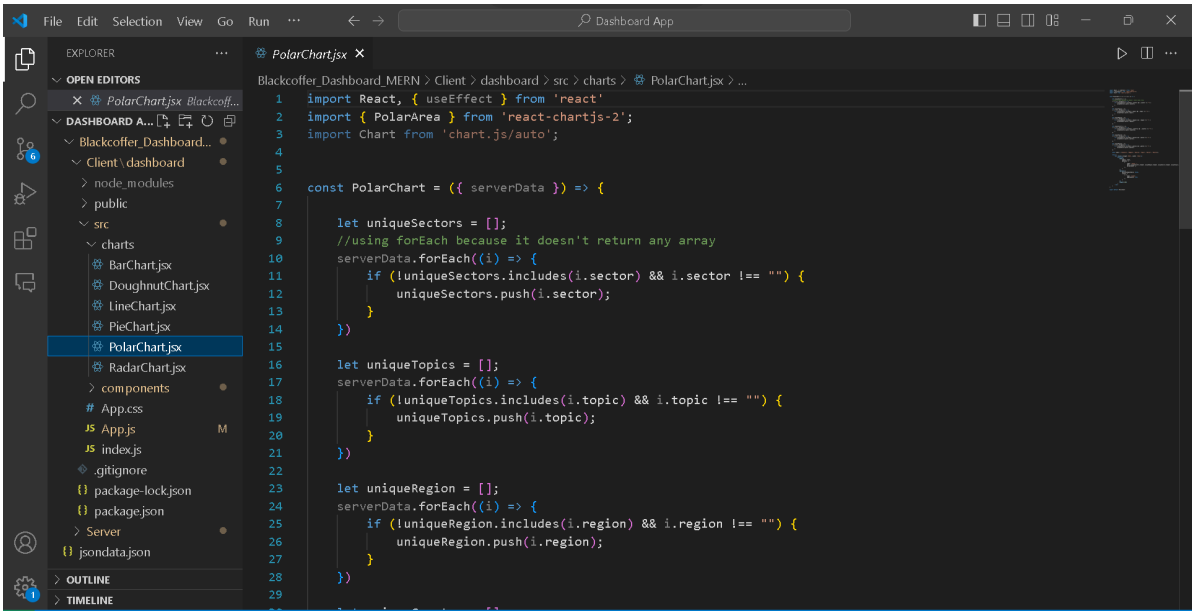
Dashboard App Code Link:

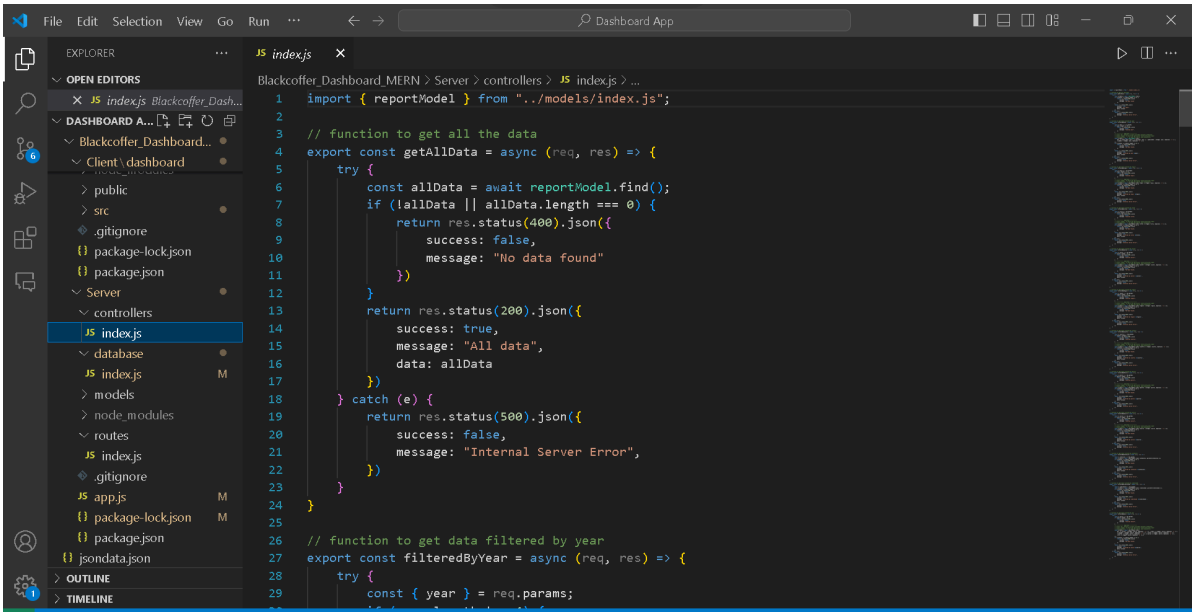
https://drive.google.com/file/d/1bqBarRbGht-Ufu9s-OGd97DnH0cHQzcv/view?usp=sharing

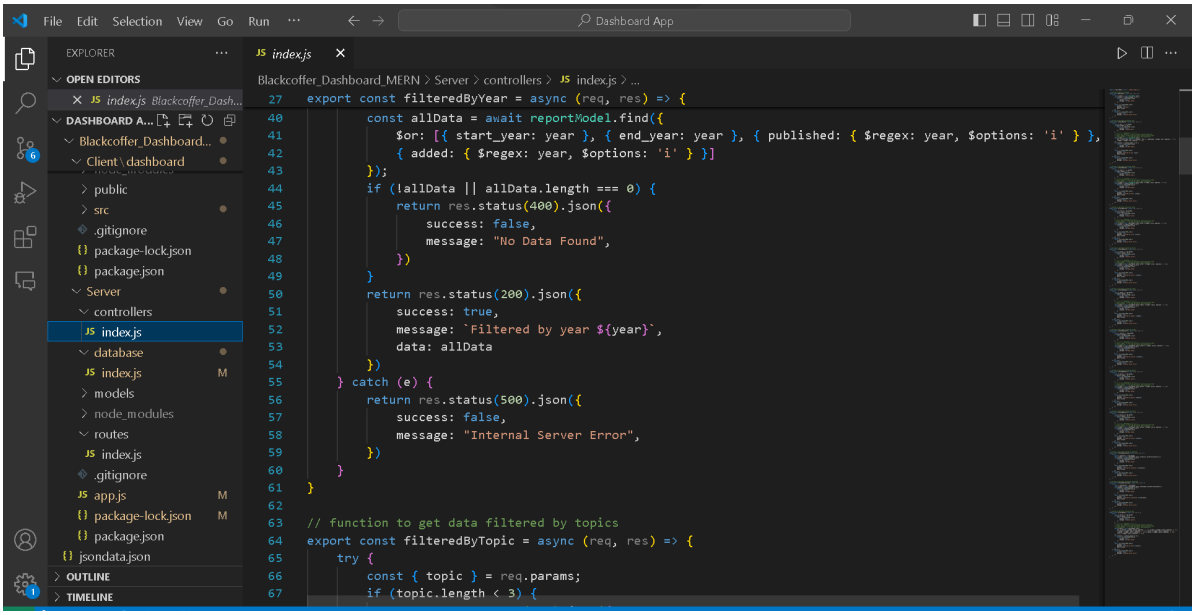


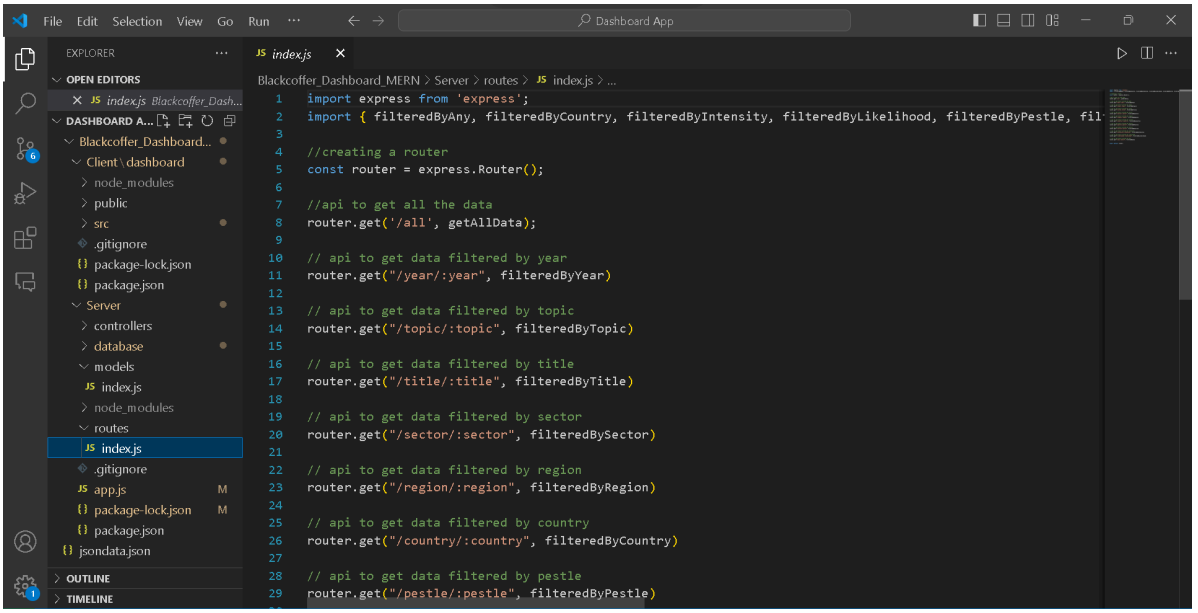



File Edit Selection View Go Run
Dashboard App
EXPLORER
Tabsrow.jsx M
Blackcoffer_Dashboard_MERN > Client > dashboard > src > components > Tabsrow.jsx > ...
import React, { useState } from 'react'
import Tab from 'react-bootstrap/Tab';
import Tabs from 'react-bootstrap/Tabs';
import CardGroup from 'react-bootstrap/CardGroup';
import Button from 'react-bootstrap/Button';
//import from files
import CardsForData from './CardsForData';
import AccordionForCharts from './AccordionForCharts';
import Filters from './Filters';
import axios from 'axios';
const Tabsrow = ({ data, setMainData }) => {
// state to store the number of data cards we want to display at a time, we'll update it on click of a
const [limit, setLimit] = useState(5);
const limitedData = data.slice(0, limit);
// state to store the search bar text
const [search, setSearch] = useState("");
// function to make an api call to get the filtered data
const handleSearchResult = async (e) => {
e.preventDefault();
try {
const response = await axios.get(`http://localhost:8000/api/data/any/${search}`);
setMainData(response.data.data);
setSearch("");
}
catch (e) {
console.log(e)
}
OPEN EDITORS
DASHBOARD A...
Blackcoffer_Dashboard...
Client \ dashboard
src
charts
PieChart.jsx
PolarChart.jsx
RadarChart.jsx
components
AccordionForCharts.jsx
CardsForData.jsx
Filters.jsx M
Header.jsx
Tabsrow.jsx M
App.css
App.js M
index.js
.gitignore
package-lock.json
package.json
Server
jsondata.json
OUTLINE
TIMELINE


File Edit Selection View Go Run
Dashboard App
EXPLORER
PolarChart.jsx
Blackcoffer_Dashboard_MERN > Client > dashboard > src > charts > PolarChart.jsx > ...
import React, { useEffect } from 'react'
import { PolarArea } from 'react-chartjs-2';
import Chart from 'chart.js/auto';
const PolarChart = ({ serverData }) => {
let uniqueSectors = [];
//using forEach because it doesn't return any array
serverData.forEach((i) => {
if (!uniqueSectors.includes(i.sector) && i.sector !== "") {
uniqueSectors.push(i.sector);
}
})
let uniqueTopics = [];
serverData.forEach((i) => {
if (!uniqueTopics.includes(i.topic) && i.topic !== "") {
uniqueTopics.push(i.topic);
}
})
let uniqueRegion = [];
serverData.forEach((i) => {
if (!uniqueRegion.includes(i.region) && i.region !== "") {
uniqueRegion.push(i.region);
}
})
OPEN EDITORS
DASHBOARD A...
Blackcoffer_Dashboard...
Client \ dashboard
node_modules
public
src
charts
BarChart.jsx
DoughnutChart.jsx
LineChart.jsx
PieChart.jsx
PolarChart.jsx
RadarChart.jsx
components
App.css
App.js M
index.js
.gitignore
package-lock.json
package.json
Server
jsondata.json
OUTLINE
TIMELINE


File Edit Selection View Go Run
Dashboard App
EXPLORER
index.js
Blackcoffer_Dashboard_MERN > Server > controllers > index.js > ...
import { reportModel } from "../models/index.js";
// function to get all the data
export const getAllData = async (req, res) => {
try {
const allData = await reportModel.find();
if (!allData || allData.length === 0) {
return res.status(400).json({
success: false,
message: "No data found"
})
}
return res.status(200).json({
success: true,
message: "All data",
data: allData
})
} catch (e) {
return res.status(500).json({
success: false,
message: "Internal Server Error",
})
}
}
// function to get data filtered by year
export const filteredByYear = async (req, res) => {
try {
const { year } = req.params;
OPEN EDITORS
DASHBOARD A...
Blackcoffer_Dashboard...
Client \ dashboard
public
src
.gitignore
package-lock.json
package.json
Server
controllers
index.js
database
index.js M
models
node_modules
routes
index.js
.gitignore
app.js M
package-lock.json M
package.json
jsondata.json
OUTLINE
TIMELINE


File Edit Selection View Go Run
Dashboard App
EXPLORER
index.js
OPEN EDITORS
DASHBOARD A...
Blackcoffer_Dashboard...
Client \ dashboard
public
src
.gitignore
package-lock.json
package.json
Server
controllers
index.js
database
index.js
models
node_modules
routes
index.js
.gitignore
app.js
package-lock.json
package.json
jsondata.json
OUTLINE
TIMELINE
Blackcoffer_Dashboard_MERN > Server > controllers > index.js > ...
export const filteredByYear = async (req, res) => {
const allData = await reportModel.find({
$or: [{ start_year: year }, { end_year: year }, { published: { $regex: year, $options: 'i' } },
{ added: { $regex: year, $options: 'i' } }]
});
if (!allData || allData.length === 0) {
return res.status(400).json({
success: false,
message: "No Data Found",
})
}
return res.status(200).json({
success: true,
message: `Filtered by year ${year}`,
data: allData
})
} catch (e) {
return res.status(500).json({
success: false,
message: "Internal Server Error",
})
}
}
// function to get data filtered by topics
export const filteredByTopic = async (req, res) => {
try {
const { topic } = req.params;
if (topic.length < 3) {


File Edit Selection View Go Run
Dashboard App
EXPLORER
index.js
OPEN EDITORS
DASHBOARD A...
Blackcoffer_Dashboard...
Client \ dashboard
node_modules
public
src
.gitignore
package-lock.json
package.json
Server
controllers
database
models
index.js
node_modules
routes
index.js
.gitignore
app.js
package-lock.json
package.json
jsondata.json
OUTLINE
TIMELINE
Blackcoffer_Dashboard_MERN > Server > routes > index.js > ...
import express from 'express';
import { filteredByAny, filteredByCountry, filteredByIntensity, filteredByLikelihood, filteredByPestle, fil
//creating a router
const router = express.Router();
//api to get all the data
router.get('/all', getAllData);
// api to get data filtered by year
router.get("/year/:year", filteredByYear)
// api to get data filtered by topic
router.get("/topic/:topic", filteredByTopic)
// api to get data filtered by title
router.get("/title/:title", filteredByTitle)
// api to get data filtered by sector
router.get("/sector/:sector", filteredBySector)
// api to get data filtered by region
router.get("/region/:region", filteredByRegion)
// api to get data filtered by country
router.get("/country/:country", filteredByCountry)
// api to get data filtered by pestle
router.get("/pestle/:pestle", filteredByPestle)