

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.

CONTENTS		Page No.
	Acknowledgement	i.
	Executive Summary	ii.
	Table of Contents	iii.
	List of Figures	v.
	List of Tables	vi.
	List of Abbreviations	vi.
1	INTRODUCTION	1
	1.1 Objectives	1
	1.2 Motivation	2
	1.3 Background	2
2	PROJECT DESCRIPTION AND GOALS	4
	2.1 Survey on Existing System	4
	2.2 Research Gap	5
	2.3 Problem Statement	7
3	TECHNICAL SPECIFICATION	9
	3.1 Requirements	9
	3.1.1 Functional	9
	3.1.2 Non-Functional	10
	3.2 Feasibility Study	11
	3.2.1 Technical Feasibility	11
	3.2.2 Economic Feasibility	12
	3.2.3 Social Feasibility	13
	3.3 System Specification	14
	3.3.1 Hardware Specification	14
	3.3.2 Software Specification	15
	3.3.3 Standards and Policies	16
4	DESIGN APPROACH AND DETAILS	18

	4.1 System Architecture	18
	4.2 Design	20
	4.2.1 Data Flow Diagram	20
	4.2.2 Use Case Diagram	21
	4.2.3 Class Diagram	23
	4.2.4 Sequence Diagram	25
	4.3 Constraints, Alternatives and Tradeoffs	26
5	SCHEDEULE, TASKS AND MILESTONES	29
	5.1 Gantt Chart	29
	5.2 Module Description	30
	5.2.1 CMS App Modules	30
	5.2.1.1 Module 1 : User Authentication	31
	5.2.1.2 Module 2 : Contact Management	31
	5.2.1.3 Module 3 : Database Interaction	31
	5.2.1.4 Module 4 : Authentication Management	32
	5.2.1.5 Module 5 : Search Functionality	32
	5.2.2 Visualization Dashboard App Modules	32
	5.2.2.1 Module 1 : User Interface	32
	5.2.2.2 Module 2 : Data Visualization	33
	5.2.2.3 Module 3 : Backend API	33
6	PROJECT DEMONSTRATION	34
7	RESULT & DISCUSSION	43
8	SUMMARY	45
9	REFERENCES	46
	APPENDIX A – SAMPLE CODE	47

LIST OF FIGURES

Figure No.	Title	Page No.
1	System Architecture of CMS App	18
2	System Architecture of Dashboard App	19
3	Data Flow Diagram for CMS App	20
4	Data Flow Diagram for Dashboard App	21
5	Use Case Diagram for CMS App	22
6	Use Case Diagram for Dashboard App	23
7	Class Diagram for CMS App	23
8	Class Diagram for Dashboard App	24
9	Sequence Diagram for CMS App	25
10	Sequence Diagram for Dashboard App	26
11	Weather App Gantt Chart	29
12	Food Recipe App Gantt Chart	29
13	CMS App Gantt Chart	30
14	Dashboard App Gantt Chart	30
15	Weather App - Home Page	34
16	Searching weather for a different city - Vellore	34
17	Recipe App - Home Page	35
18	Searching for a particular food item	35
19	Fetching recipe details	36
20	Favorites page	36
21	Registering a new user	37
22	Sign-in page for existing user	37
23	CMS App Home page	38
24	Create new contact page	38
25	Edit/Delete existing contact	39
26	Search for an existing contact	39
27	Dashboard App - Home Page	40
28	Polar Area and Doughnut Charts	40

29	Bar Chart	41
30	Line Chart	41
31	Radar and Pie Charts	41
32	Data Filtered by Year	42
33	Data Filtered by Sector Name	42
34	Graph changes dynamically by searching the Sector Name	42

LIST OF TABLES

Table No.	Title	Page No.
1	Gaps Identified in existing system/technology	5

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	<ol style="list-style-type: none">React only deals with the View of MVC, so other tools are required for backend development.Some developers find JSX programming difficult to learn.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	<ol style="list-style-type: none">MERN can introduce complexity into your project, especially if you're not familiar with all of the technologies involved.While MERN can be a great choice for small and medium-sized applications, it may not scale well for very large or complex applications.Security is a major concern for any web application, and MERN stacks can be vulnerable to security attacks if not properly configured.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	<ol style="list-style-type: none"> 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	<ol style="list-style-type: none"> 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	<ol style="list-style-type: none"> 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	<ol style="list-style-type: none"> 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.

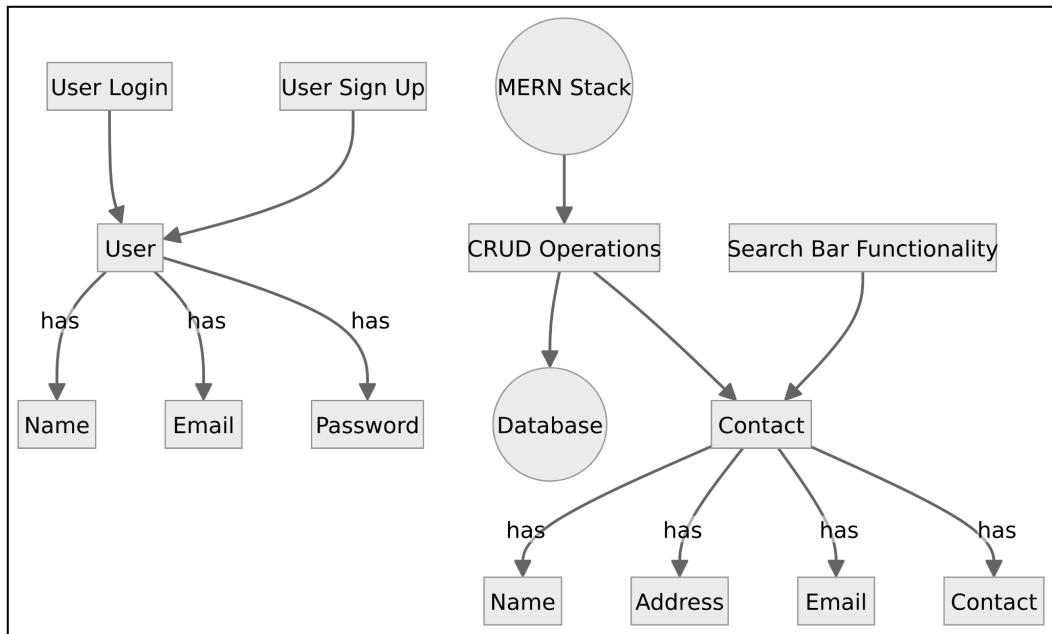


Fig 1 : System Architecture of the CMS App

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.

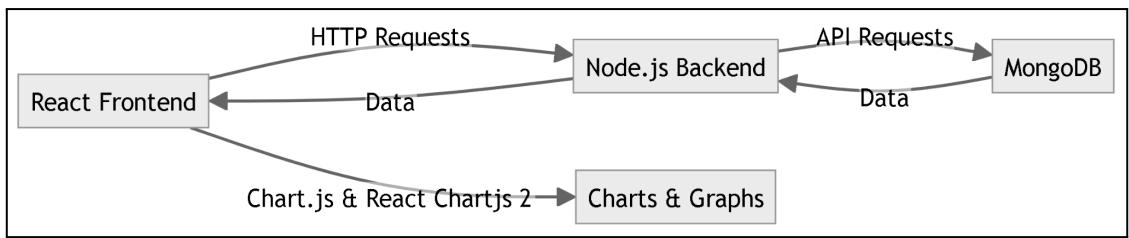


Fig 2 : System Architecture of the Dashboard App

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

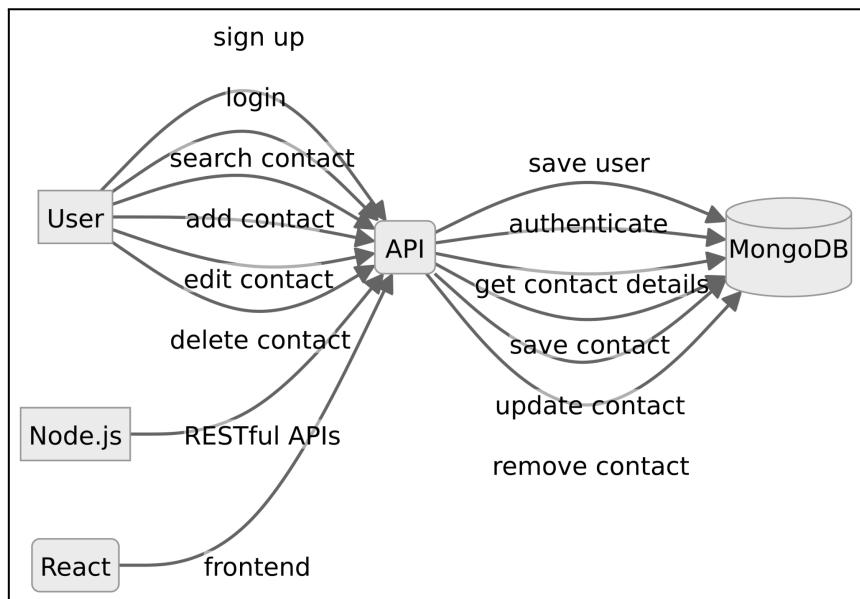


Fig 3 : Data Flow Diagram for CMS App

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.

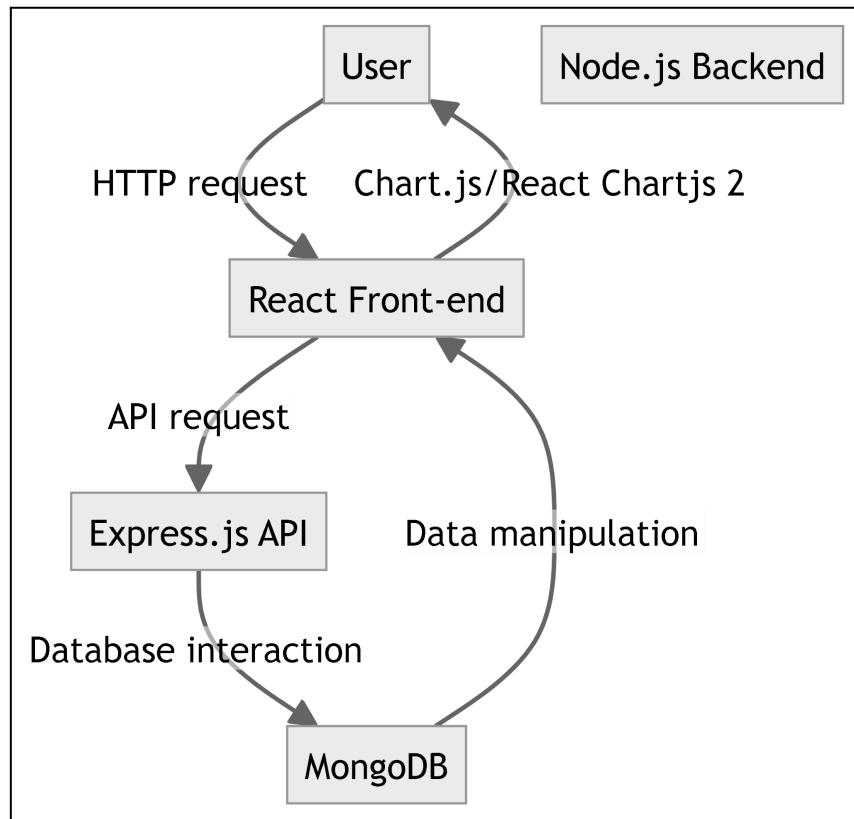


Fig 4 : Data Flow Diagram for Dashboard App

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

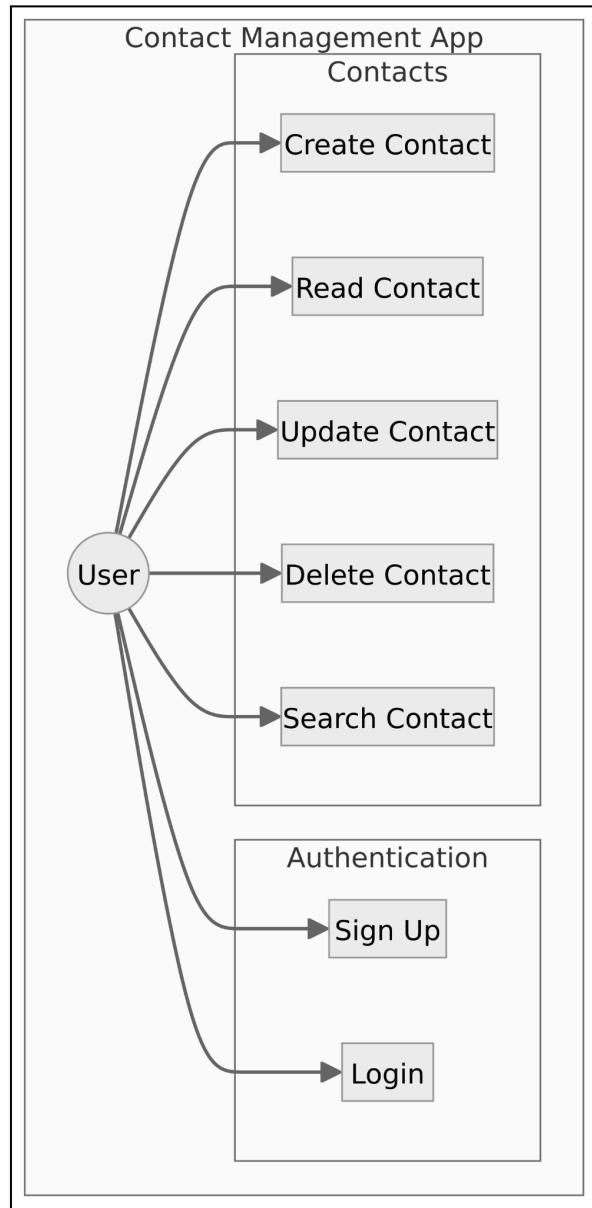


Fig 5 : Use Case Diagram for CMS App

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.

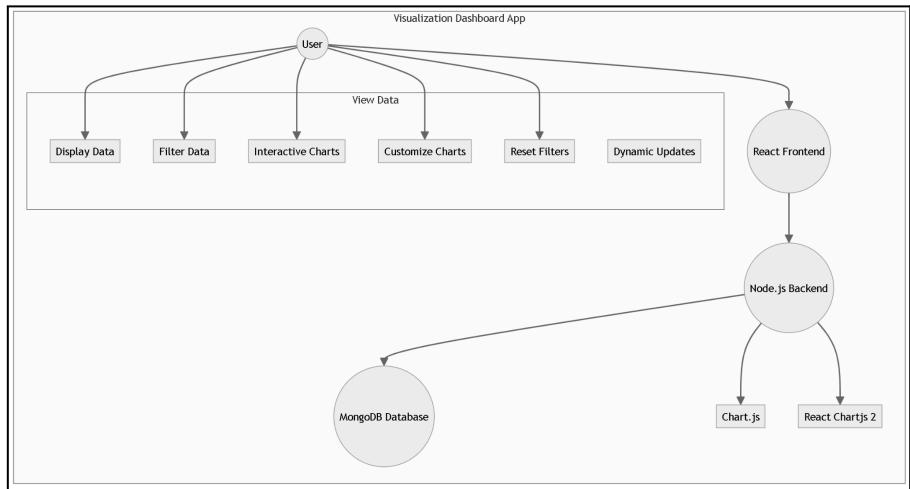


Fig 6 : Use Case Diagram for Dashboard App

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

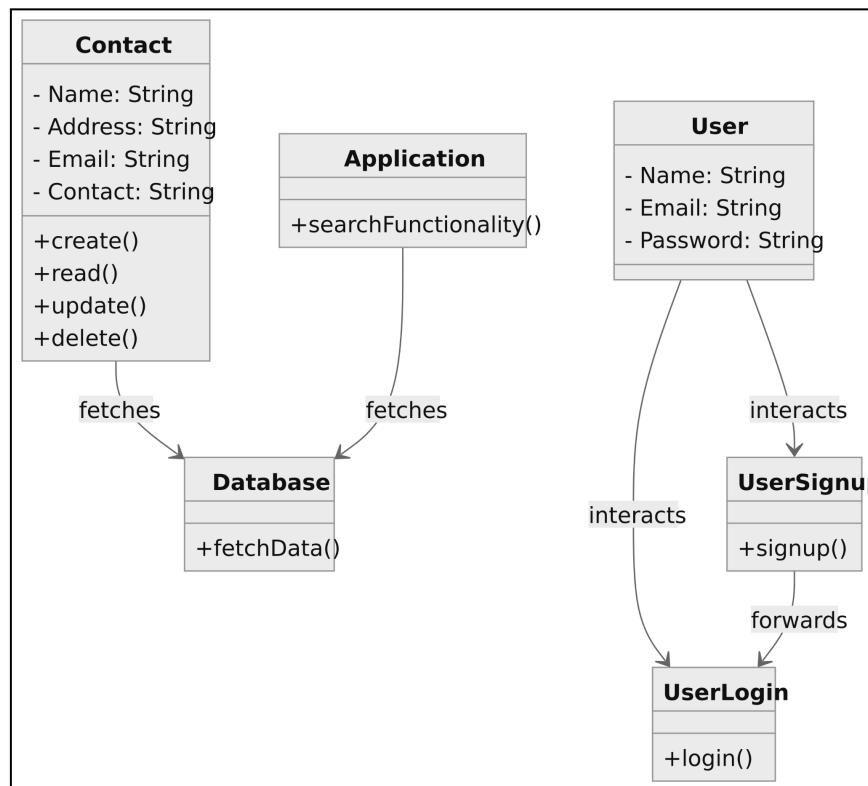


Fig 7 : Class Diagram for CMS App

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.

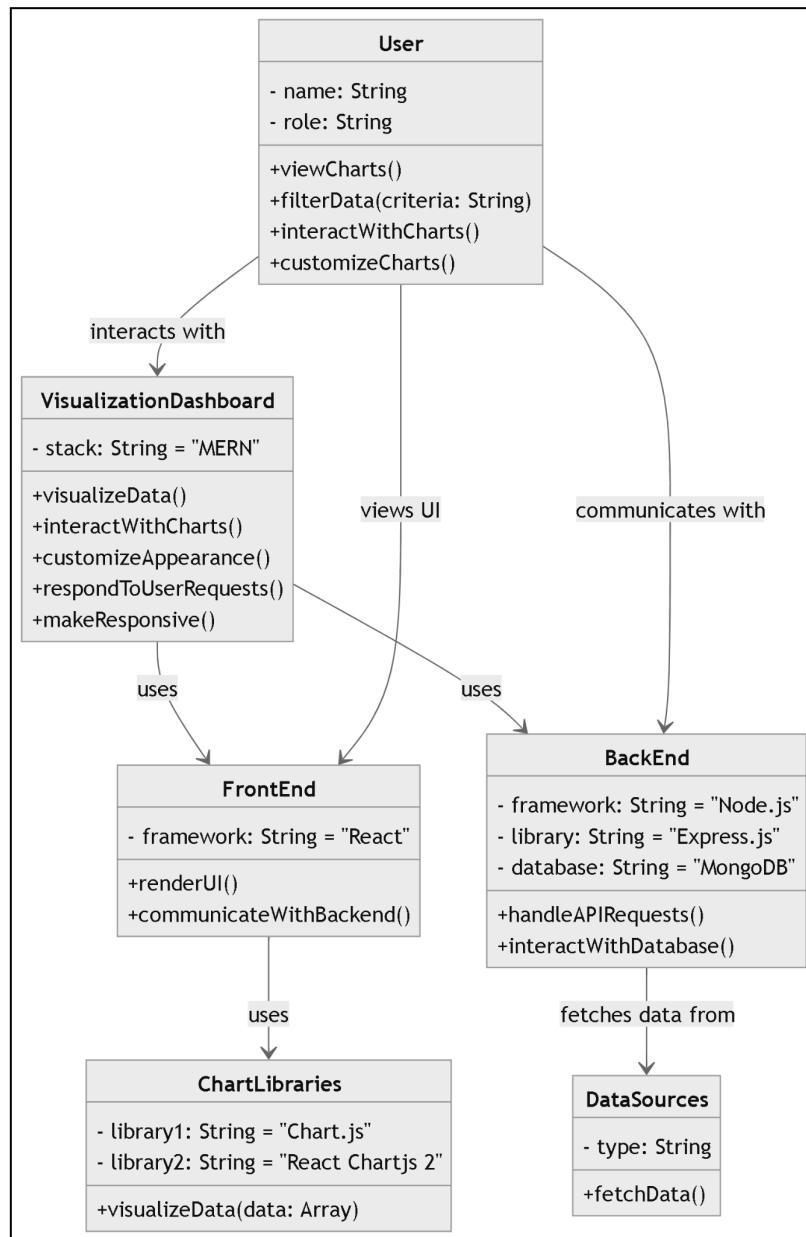


Fig 8 : Class Diagram for Dashboard App

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

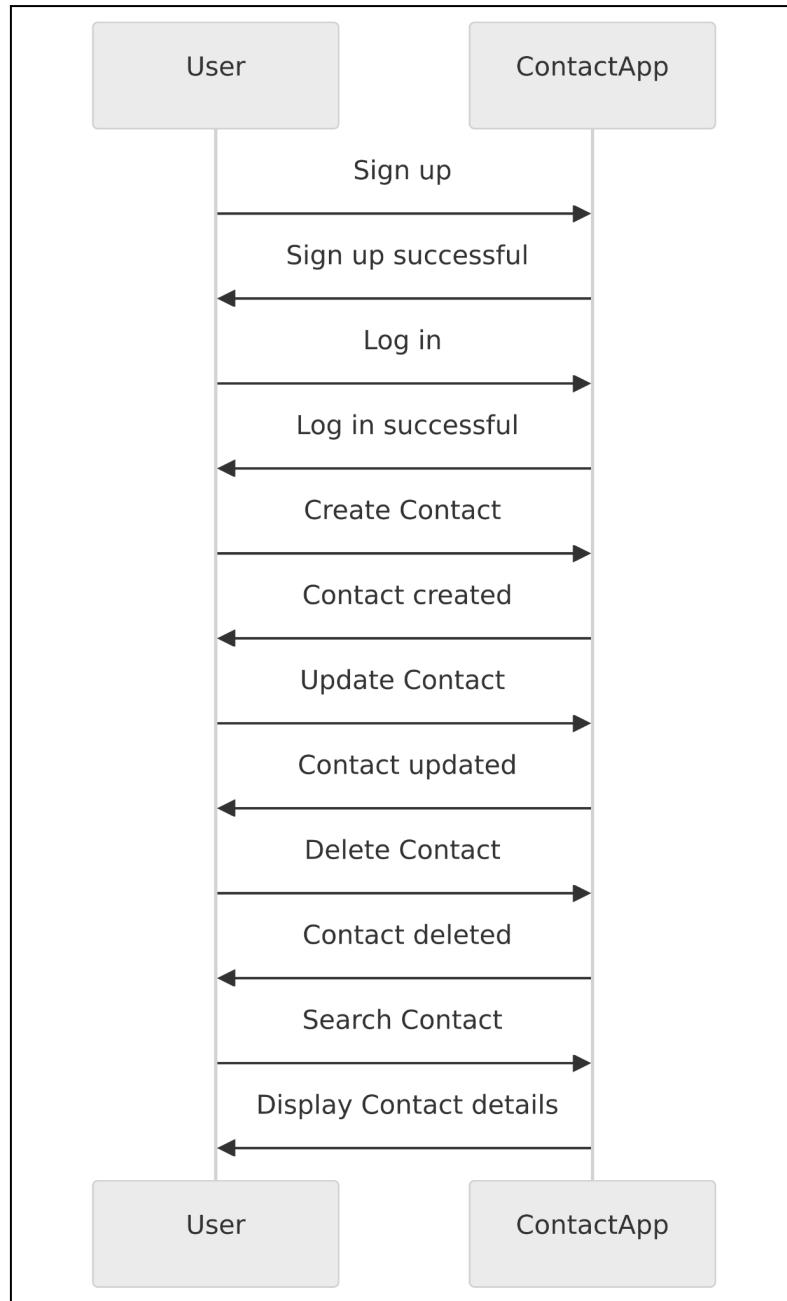


Fig 9 : Sequence Diagram for CMS App

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.

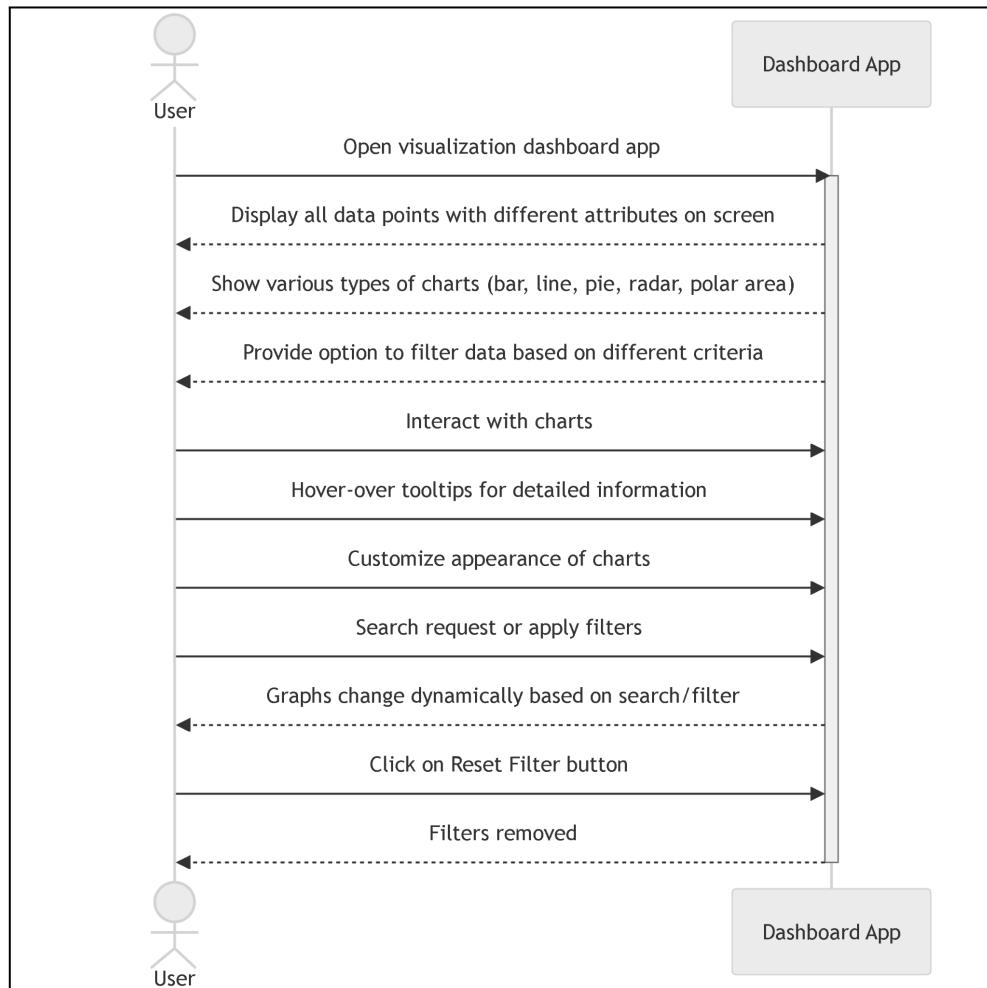


Fig 10 : Sequence Diagram for Dashboard App

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

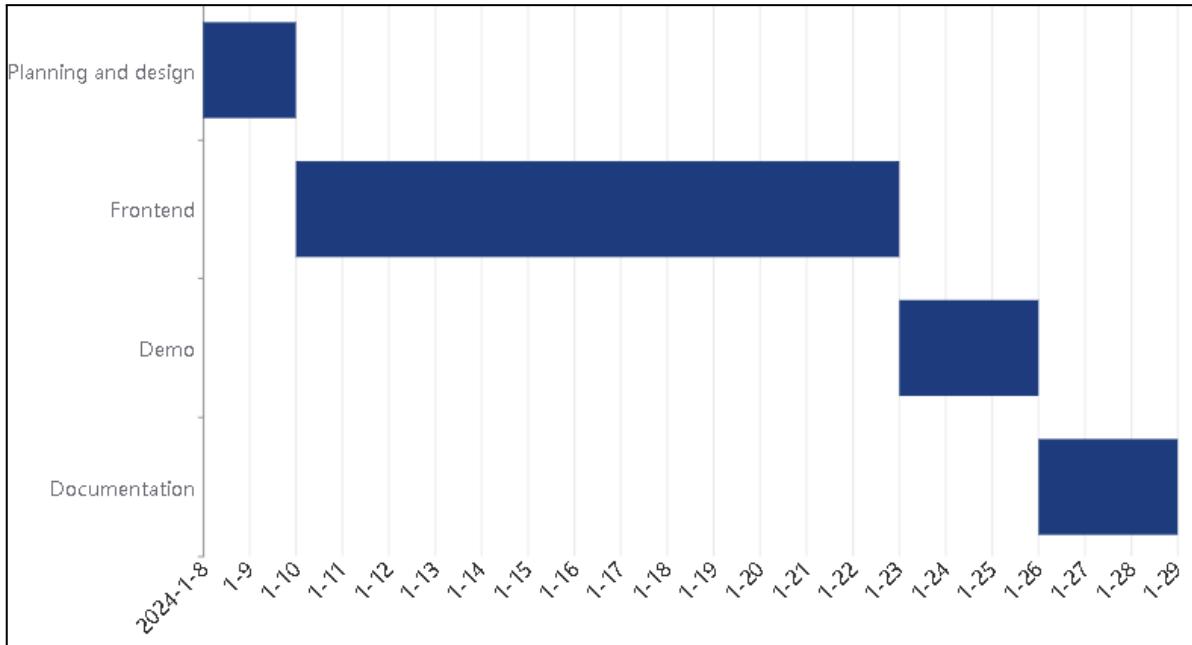


Fig 11 : Weather App Gantt Chart

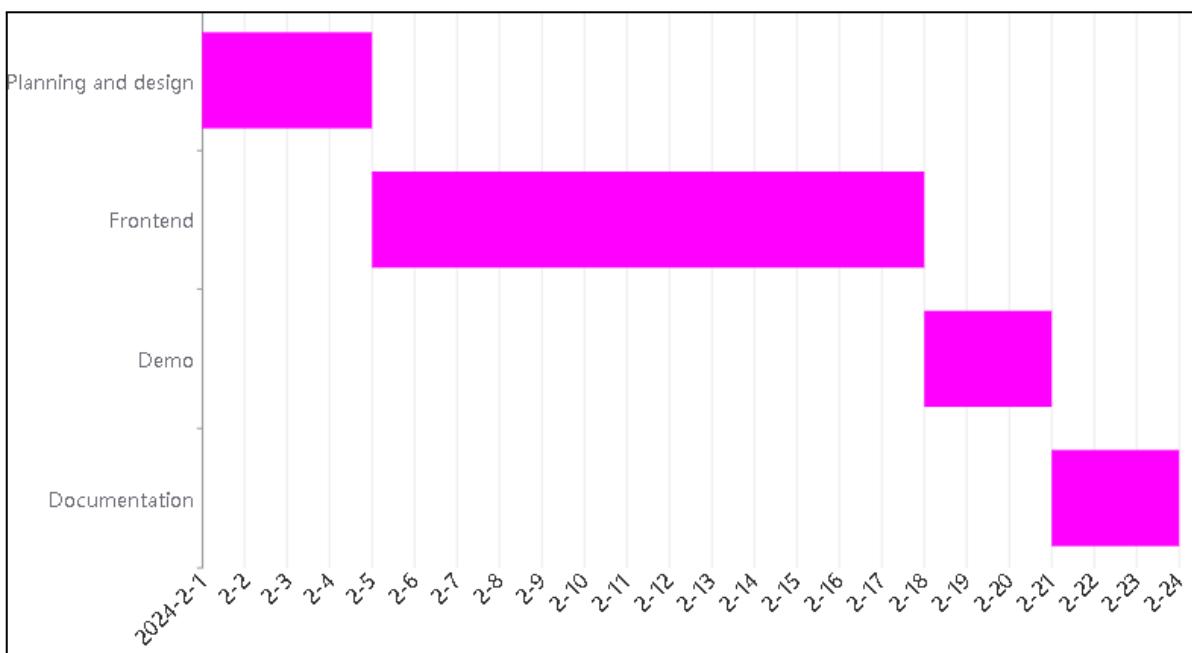


Fig 12 : Food Recipe App Gantt Chart

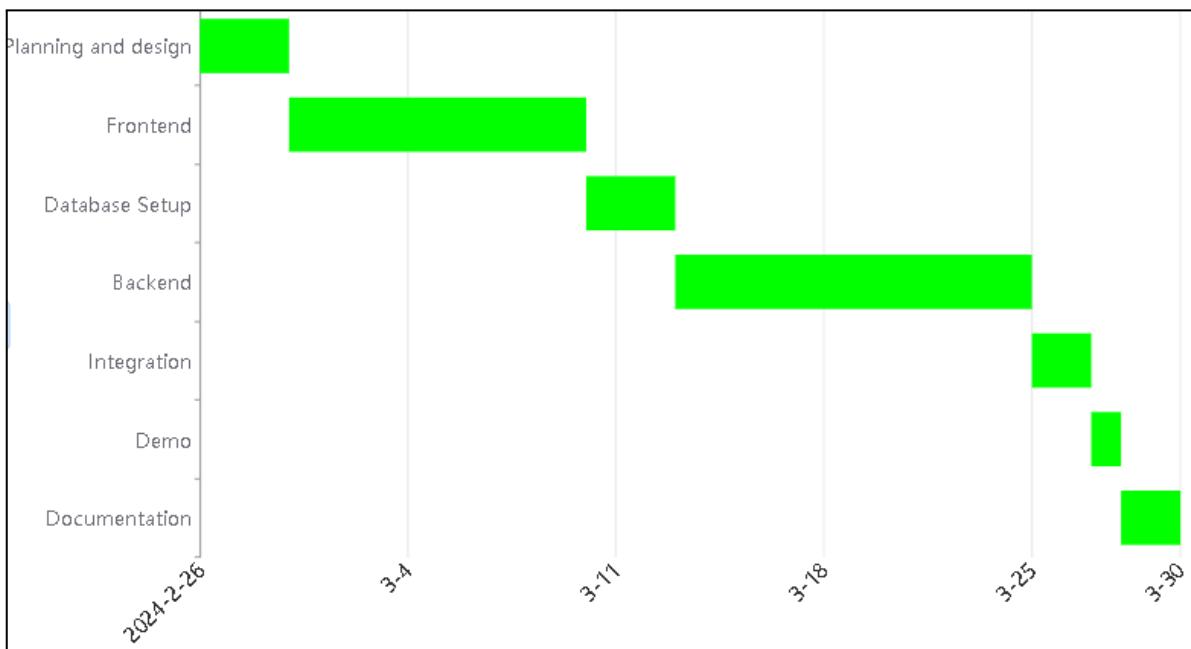


Fig 13 : CMS App Gantt Chart

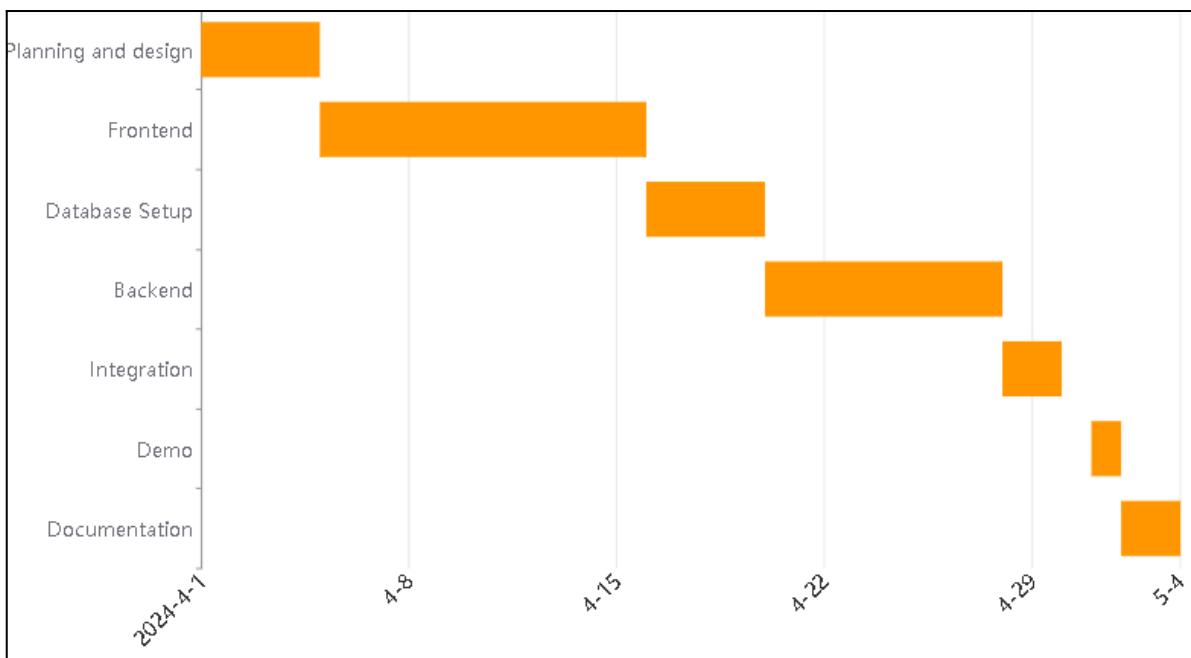


Fig 14 : Visualization Dashboard App Gantt Chart

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

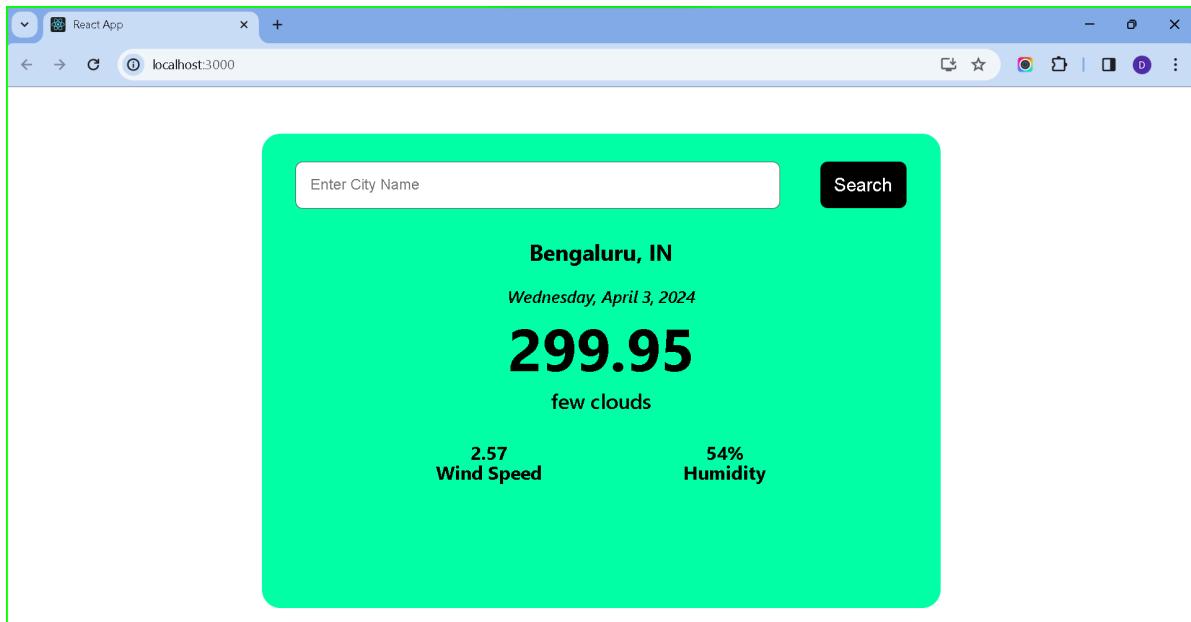


Fig 15 : Weather App - Home Page

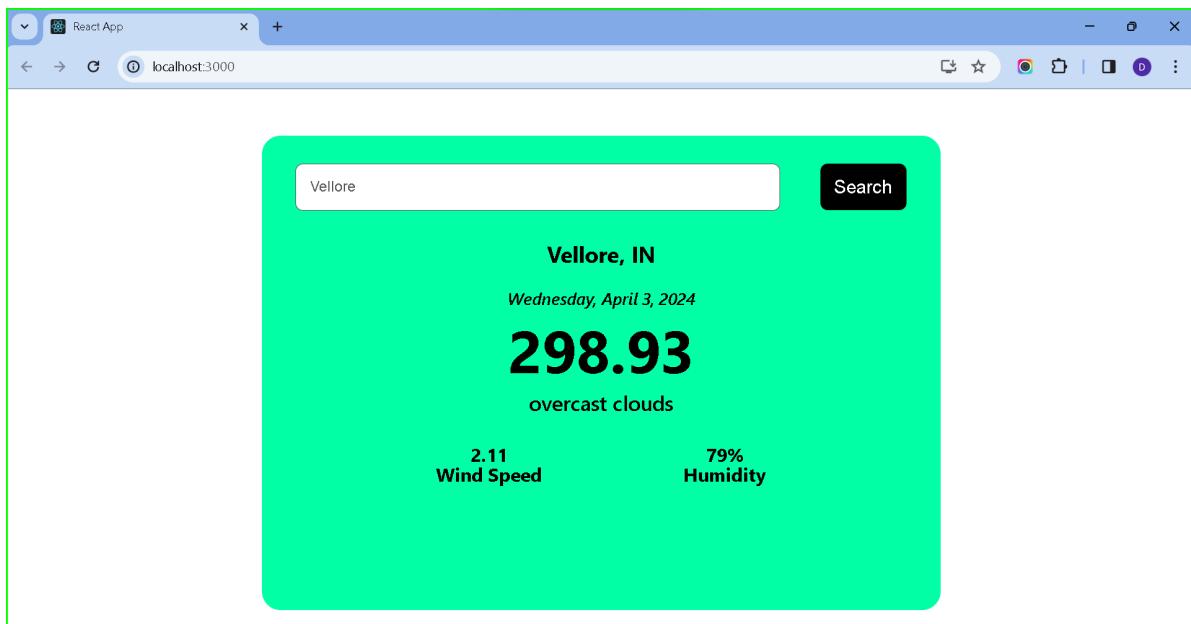


Fig 16: Searching weather for a different city - Vellore

Food Recipe App

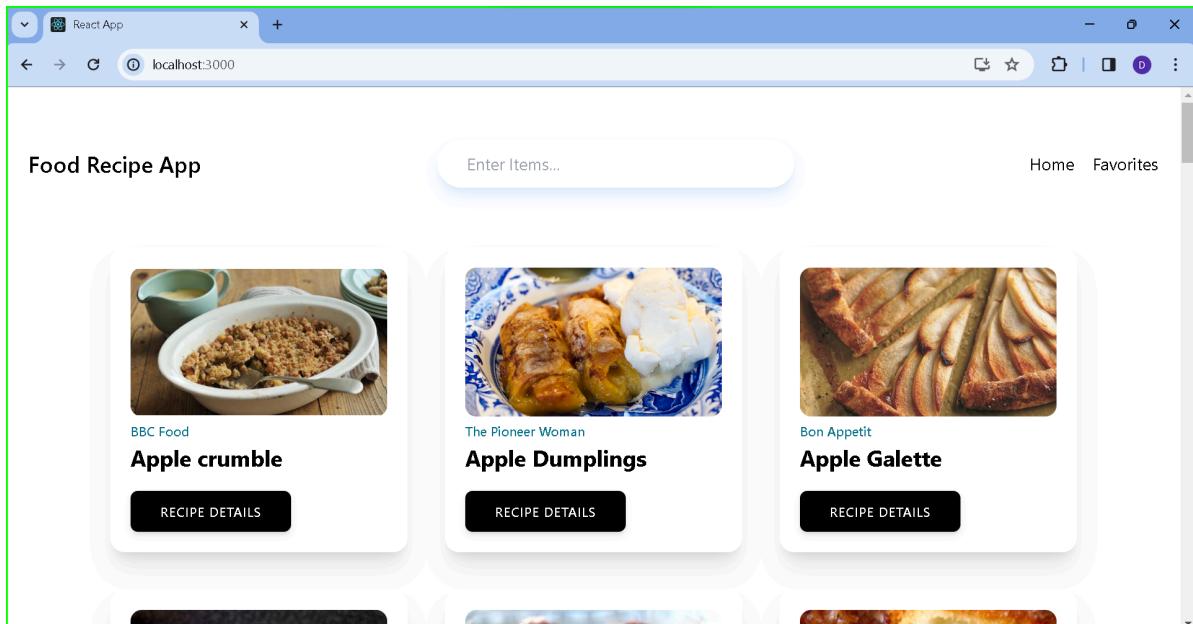


Fig 17 : Recipe App - Home Page

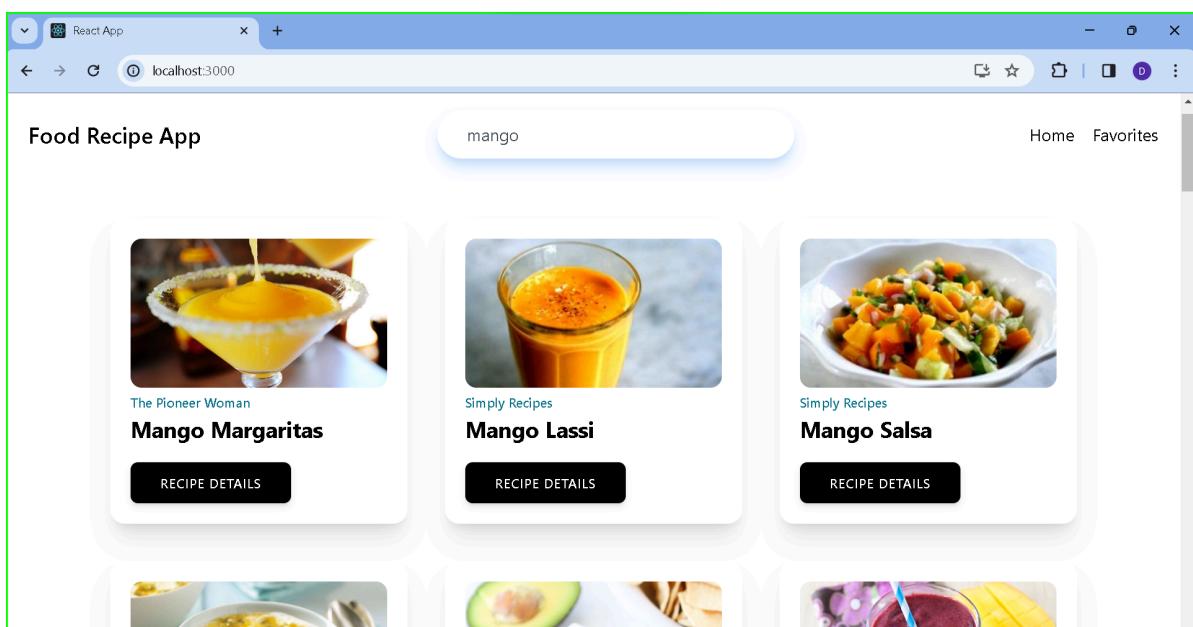


Fig 18 : Searching for a particular food item

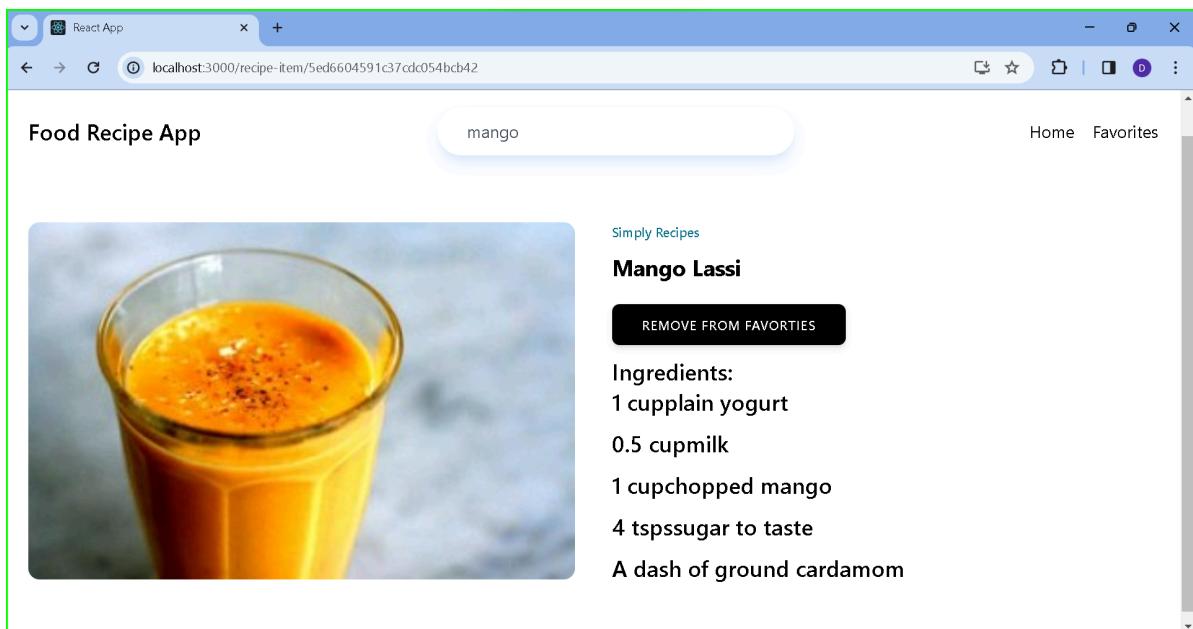


Fig 19 : Fetching recipe details

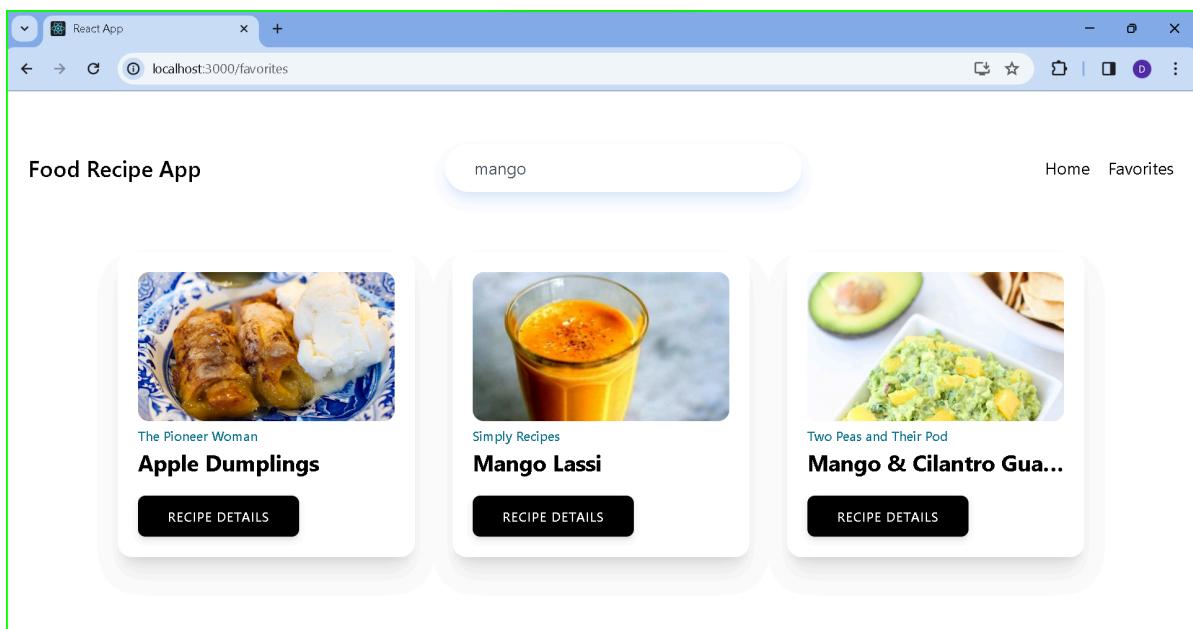


Fig 20 : Favorites page

Contact Management System

The screenshot shows a web browser window titled "CMS" with the URL "localhost:3000/register". The page has a dark header with "LOGIN" and "REGISTER" links. The main content area is titled "CREATE YOUR ACCOUNT" and contains four input fields: "Your Name" (with "John Doe" entered), "Email address" (with "johndoe@example.com" entered), "Password" (with "Enter Password" placeholder), and "Confirm Password" (with "Enter Password" placeholder). A black "REGISTER" button is at the bottom.

Fig 21 : Registering a new user

The screenshot shows a web browser window titled "CMS" with the URL "localhost:3000/login". The page has a dark header with "LOGIN" and "REGISTER" links. The main content area is titled "LOGIN" and contains two input fields: "Email address" (with "disha@gmail.com" entered) and "Password" (with "....." placeholder). A black "LOGIN" button is at the bottom. Below the button, text says "Don't have an account? [Create One](#)".

Fig 22 : Sign-in page for existing user

YOUR CONTACTS			
RELOAD CONTACT			
<input type="text" value="Search Contact"/> SEARCH			
Your Total Contacts: 2			
NAME	ADDRESS	EMAIL	PHONE
DEVIKA	Shimla	devika@gmail.com	9876534526
LAKSHAY	Hyderabad	lakshay@gmail.com	1234567890

Fig 23 : CMS App Home page

CREATE YOUR CONTACT			
Name Of Person	<input type="text" value="John Doe"/> <div style="margin-left: 10px;">Please fill out this field</div>		
Address Of Person	<input type="text" value="WalkStreet 05, California"/>		
Email Of Person	<input type="text" value="johndoe@example.com"/>		
Phone Number Of Person	<input type="text" value="+977 987654321"/>		
ADD CONTACT			

Fig 24 : Create new contact page

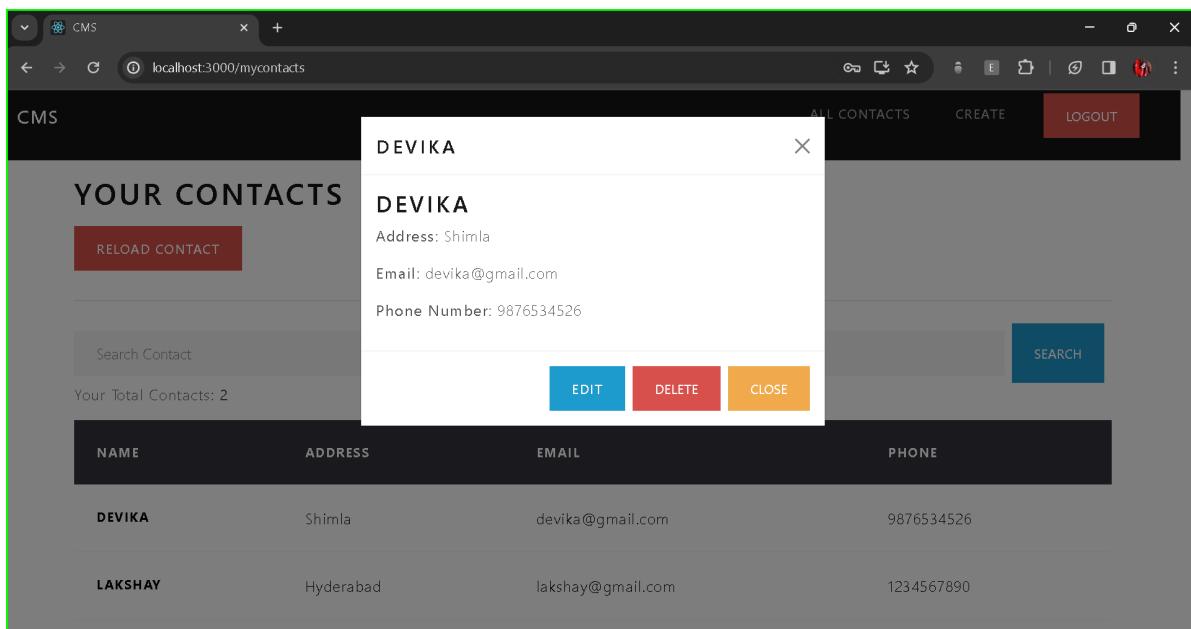


Fig 25 : Edit/Delete existing contact

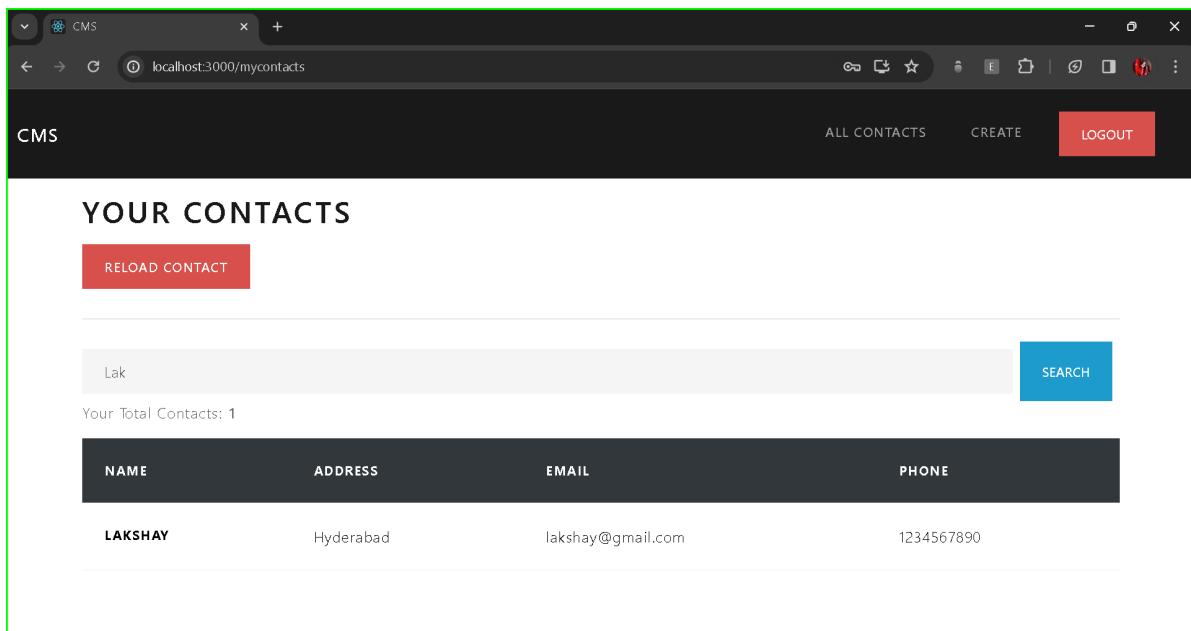


Fig 26 : Search for an existing contact

Visualization Dashboard App

The screenshot shows the 'Data Virtualization Dashboard' home page. At the top, there are two tabs: 'Data' (selected) and 'Interactive Insights/Dashboard'. Below the tabs is a search bar with placeholder text 'Search by Sector Name, Topic, Title, Pestle, Source, Insight, URL...'. To the right of the search bar are three buttons: 'Filter By Year' (green), 'Reset Filters' (red), and a blue 'Search' button. The main area contains four cards, each titled 'Project Details' and showing a list of items under a specific category:

- Project Details Energy**
 - gas
 - U.S. natural gas consumption
 - Annual Energy Outlook
 - <http://www.eia.gov/outlook>
 - Northern America
 - United States of America
 - EIA
 - Industries
 - No info
 - No info
 - January, 20 2017 03:51:25
 - January, 09 2017 00:00:0C
 - 6
 - 3
- Project Details Energy**
 - oil
 - Reference case U.S. crude
 - Annual Energy Outlook
 - <http://www.eia.gov/outlook>
 - Northern America
 - United States of America
 - EIA
 - Industries
 - No info
 - No info
 - January, 20 2017 03:51:24
 - January, 09 2017 00:00:0C
 - 6
 - 3
- Project Details Energy**
 - consumption
 - U.S. petroleum consumption
 - Annual Energy Outlook
 - <http://www.eia.gov/outlook>
 - Northern America
 - United States of America
 - EIA
 - Industries
 - No info
 - No info
 - January, 20 2017 03:51:23
 - January, 09 2017 00:00:0C
 - 6
 - 3
- Project Details Environment**
 - oil
 - Mars, Unilever, Cargill and WRI Partnership Aims to Re
 - <http://www.sustainablebrands.com>
 - Central America
 - Mexico
 - sustainablebrands.com
 - Environmental
 - No info
 - No info
 - January, 20 2017 03:26:4C
 - January, 18 2017 00:00:00
 - 6
 - 2

Fig 27 : Dashboard App - Home Page

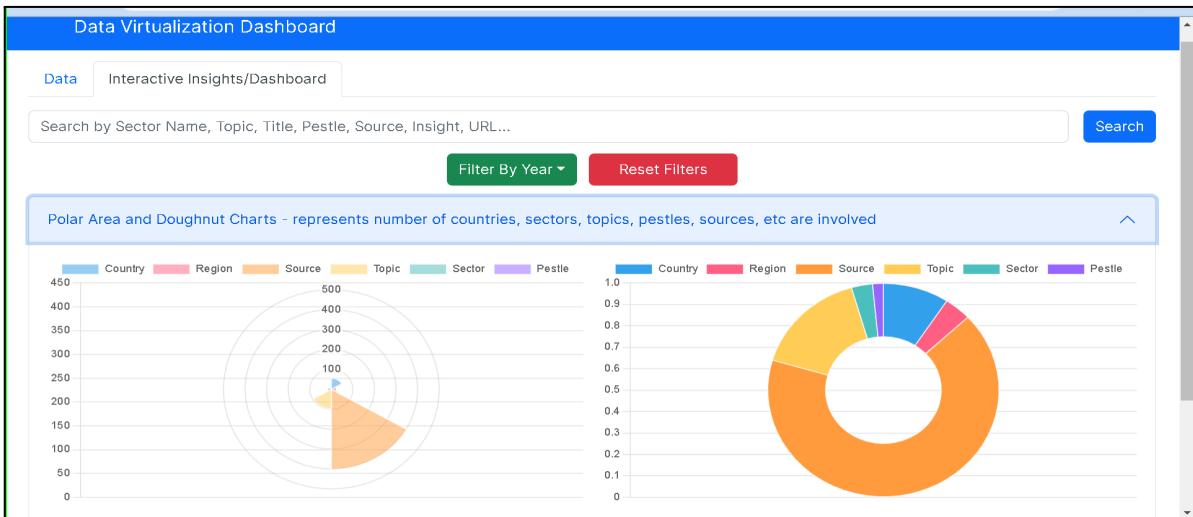


Fig 28 : Polar Area and Doughnut Charts



Fig 29 : Bar Chart

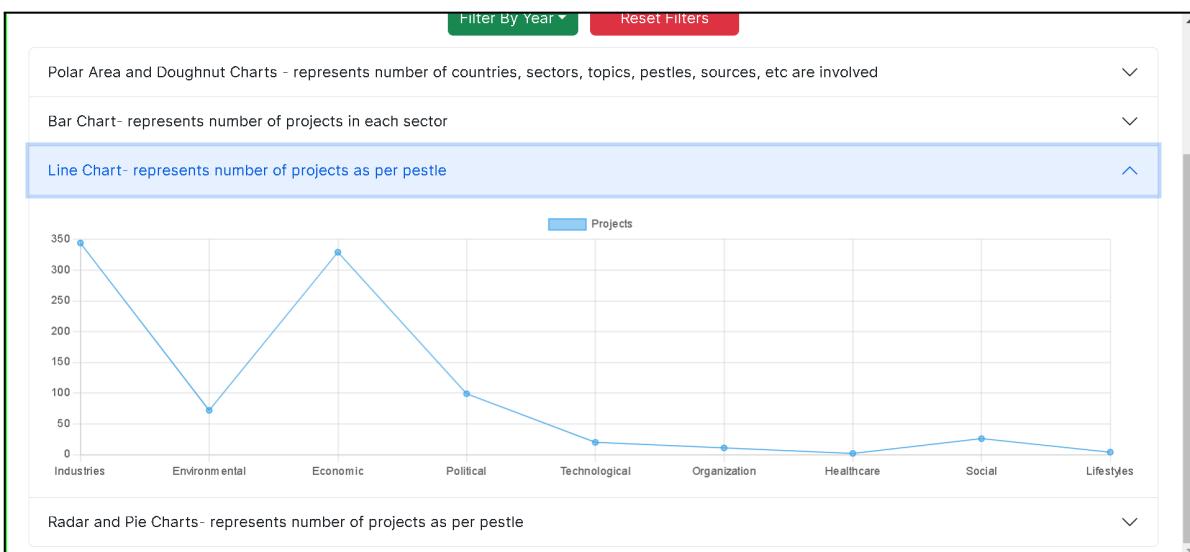


Fig 30 : Line Chart

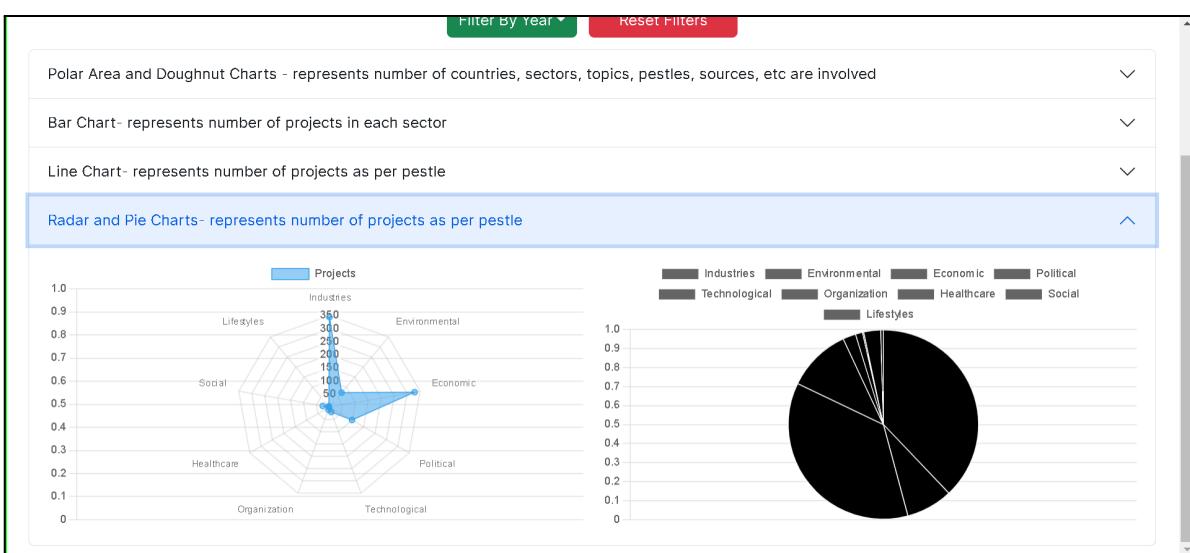


Fig 31 : Radar and Pie Charts

Data Interactive Insights/Dashboard

Search by Sector Name, Topic, Title, Pestle, Source, Insight, URL...

Filter By Year ▾ Reset Filters

Year	Project Details	Project Details	Project Details	Project Details
2014	Support services	Energy	Water	
2015	gas	Technological	water	
2016	• Energy storage among 12 !	• Electric Power Indus	early all (94%) of global	
2017	• Electric Power Indus	• http://www.iptechex.com/l	Electric Power Industry Tra	
2018	• World	• World	• http://www.iptechex.com/l	
2019	• No info	• Innovaro	• Windpower Technology	
2020	• No info	• Technological	• http://www.iptechex.com/l	
	• Innovaro	• Political	• No info	
	• Technological	• No Info	• No info	
	• No Info	• No Info	• August, 30 2016 08:37:24	
	• August, 30 2016 08:37:29	• August, 30 2016 08:37:24	• August, 30 2016 08:33:18	
	• January, 10 2014 00:00:00	• January, 10 2014 00:00:00	• September, 29 2014 00:00:	
	• 9	• 3	• 2	
	• 3	• 3	• 2	

localhost:3000/#

Fig 32 : Data Filtered by Year

Retail

Filter By Year ▾ Reset Filters

Sector	Project Details	Project Details	Project Details	Project Details
Retail	Retail	Retail	Retail	Retail
	• export	• export	• export	• export
	• More U.S. LNG (liquified na	• The fastest pace of growth	• Shale gas and tight oil play	• US oil and gas policy may
	• The Slow Road Back: Oil &	• Asian stocks set for gains	• No title could be extracted	• January 24, 2017 - The Tin
	• http://deloitte.wsj.com/risk	• http://www.thejakartapost	• No info	• No info
	• Northern America	• World	• No info	• No info
	• United States of America	• No info	• THE LEAGUE OF WOMEN V	• Guarini Center
	• WSJ	• The Jakarta Post	• Economic	• Political
	• Economic	• Economic	• No info	• No info
	• No info	• 2017	• 2040	• 2019
	• No info	• 2022	• January, 08 2017 04:25:06	• January, 08 2017 01:50:10
	• January, 17 2017 02:21:10	• January, 11 2017 03:08:02	• December, 09 2016 00:00:	• November, 16 2016 00:00:
	• No info	• January, 11 2017 00:00:00	• 12	• 6
	• 6	• 16	• 4	• 2
	• 3	• 4		

Fig 33 : Data Filtered by Sector Name

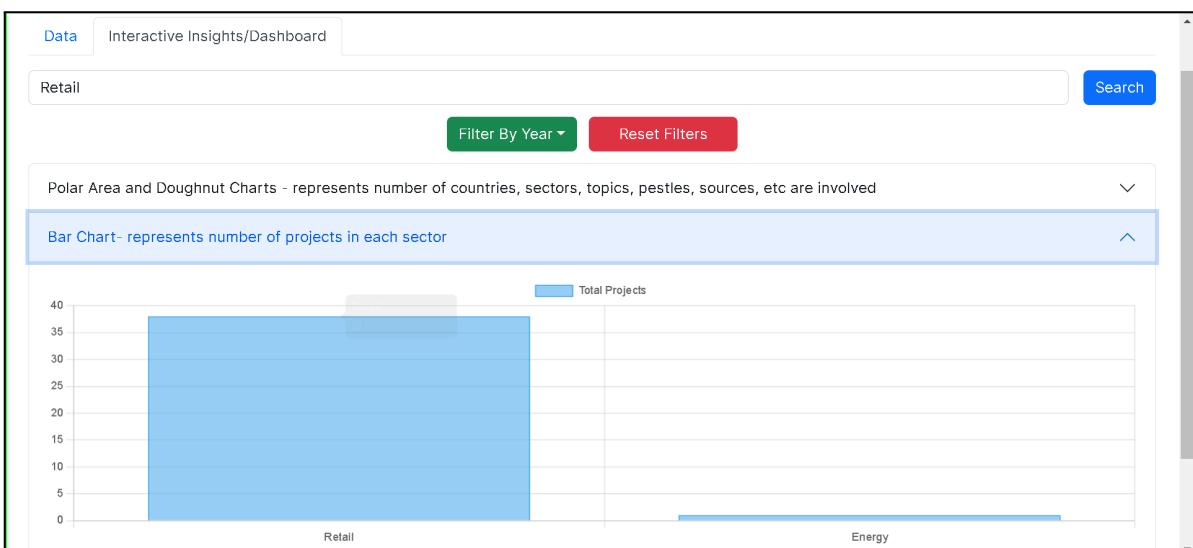


Fig 34 : Graph changes dynamically by searching the Sector Name

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:

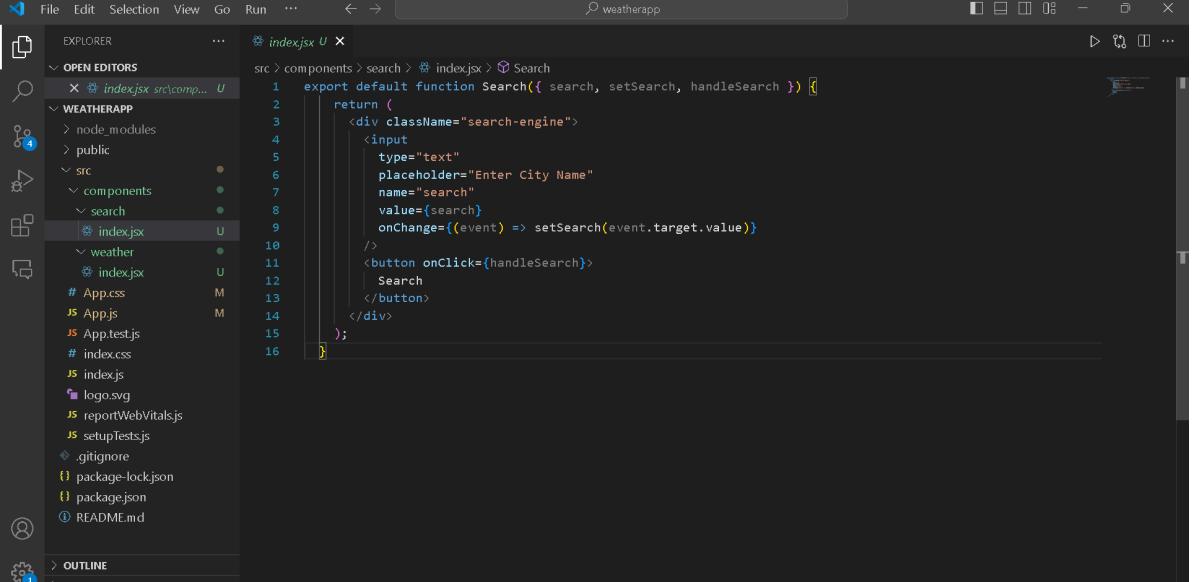
1. Avinash Mishra, Arshita Gupta, “React JS – A Frontend Javascript Library”, International Research Journal of Modernization in Engineering Technology and Science (IRJMETS), November 2022.
2. Yogesh Baiskar, Priyas Paulzagade, Krutik Koradia, Pramod Ingole, Dhiraj Shirbhate, “MERN: A Full-Stack Development”, 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/>

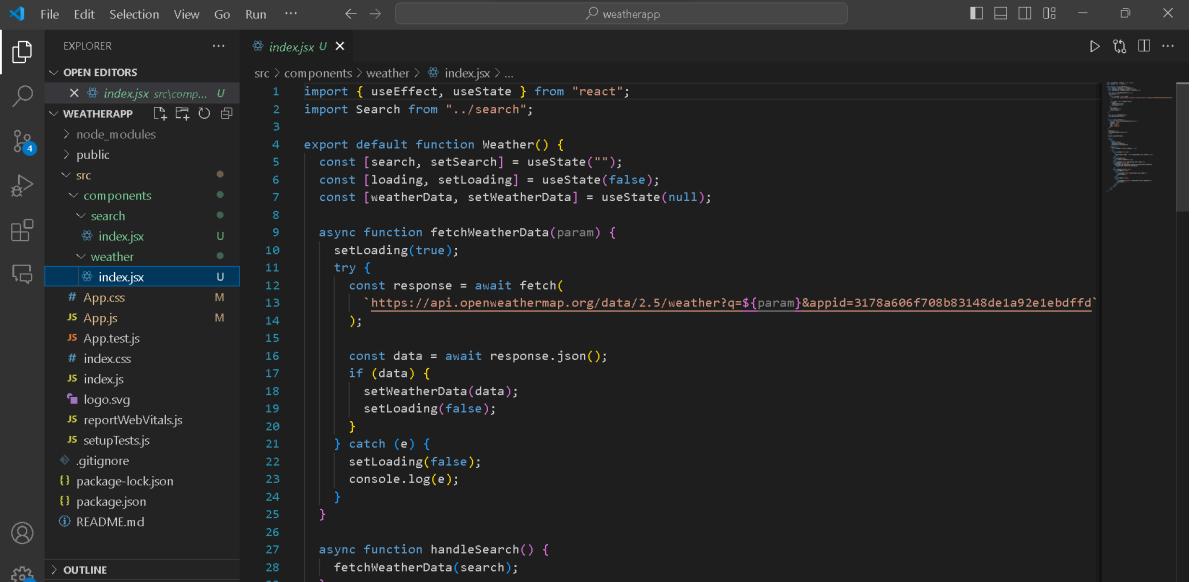
APPENDIX A – SAMPLE CODE

Weather App Code:



This screenshot shows the initial state of the Weather App code in VS Code. The file `index.jsx` is open in the editor, displaying a simple search component. The code uses a functional component structure with class-based styling (`Search`) and a stateless functional component (`Search`). The component takes a `search` prop and a `handleSearch` callback. It contains a search input and a button labeled "Search".

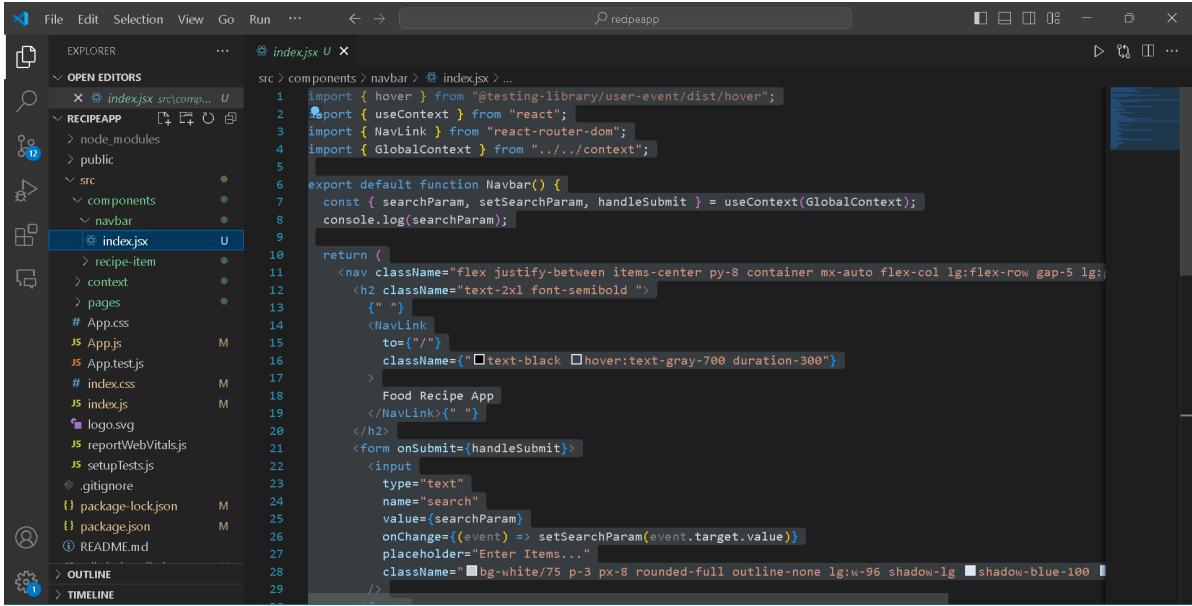
```
1  export default function Search({ search, setSearch, handleSearch }) {
2    return (
3      <div className="search-engine">
4        <input
5          type="text"
6          placeholder="Enter City Name"
7          name="search"
8          value={search}
9          onChange={(event) => setSearch(event.target.value)}
10     />
11     <button onClick={handleSearch}>
12       | Search
13     </button>
14   );
15 }
16 }
```



This screenshot shows the completed state of the Weather App code in VS Code. The file `index.jsx` now includes logic for fetching weather data. It imports `useEffect` and `useState` from React and the `Search` component from the `search` folder. The component is named `Weather`. It uses `useState` to manage the search term, loading status, and weather data. The `fetchWeatherData` function sends a GET request to the OpenWeatherMap API to fetch weather data for the specified city. The `handleSearch` function triggers this fetch operation.

```
1  import { useEffect, useState } from "react";
2  import Search from "../search";
3
4  export default function Weather() {
5    const [search, setSearch] = useState("");
6    const [loading, setLoading] = useState(false);
7    const [weatherData, setWeatherData] = useState(null);
8
9    async function fetchWeatherData(param) {
10      setLoading(true);
11      try {
12        const response = await fetch(
13          `https://api.openweathermap.org/data/2.5/weather?q=${param}&appid=3178a606f708b83148de1a92e1ebdff`
14        );
15
16        const data = await response.json();
17        if (data) {
18          setWeatherData(data);
19          setLoading(false);
20        }
21      } catch (e) {
22        setLoading(false);
23        console.log(e);
24      }
25    }
26
27    async function handleSearch() {
28      fetchWeatherData(search);
29    }
30 }
```

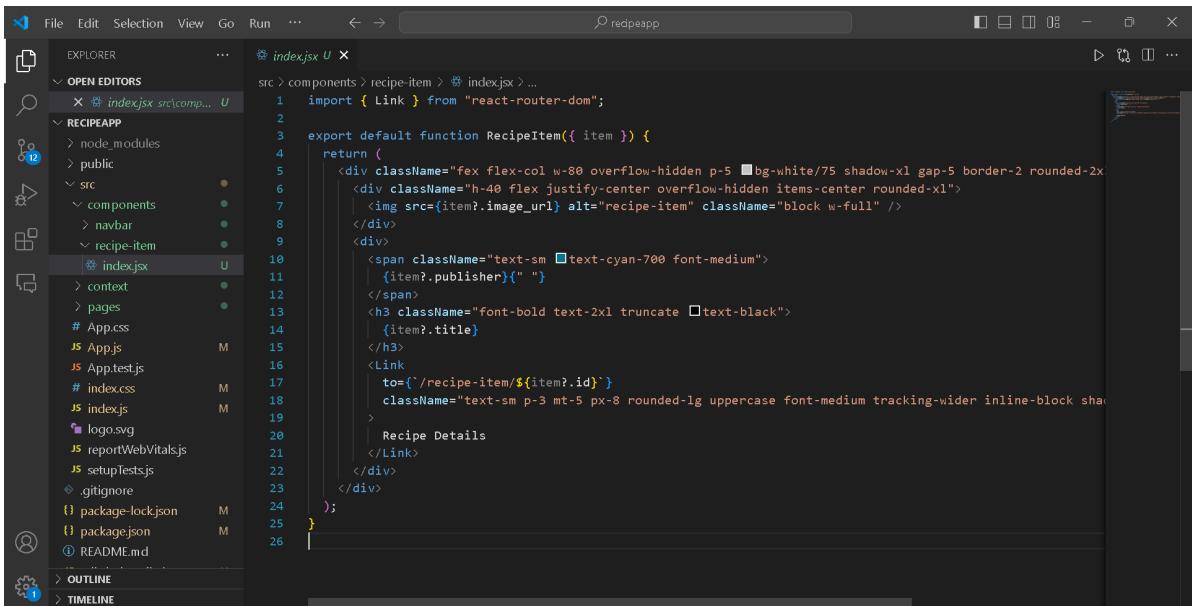
Recipe App Code:



VS Code interface showing the Navbar component code. The file path is src > components > navbar > index.jsx. The code imports hover, useContext, NavLink, and GlobalContext from their respective packages. It defines a Navbar function that logs the searchParam to the console and returns a navigation bar with a search input.

```
src > components > navbar > index.jsx ...
1 import { hover } from '@testing-library/user-event/dist/hover';
2 import { useContext } from "react";
3 import { NavLink } from "react-router-dom";
4 import { GlobalContext } from "../../context";
5
6 export default function Navbar() {
7   const { searchParam, setSearchParam, handleSubmit } = useContext(GlobalContext);
8   console.log(searchParam);
9
10  return (
11    <nav className="flex justify-between items-center py-8 container mx-auto flex-col lg:flex-row gap-5 lg:gap-0" ...
12      <h2 className="text-2xl font-semibold" ...
13        ...
14        <NavLink
15          to="/"
16          className="text-black hover:text-gray-700 duration-300" ...
17        >
18          Food Recipe App
19        </NavLink> ...
20      </h2>
21      <form onSubmit={handleSubmit}>
22        <input
23          type="text"
24          name="search"
25          value={searchParam}
26          onChange={(event) => setSearchParam(event.target.value)}
27          placeholder="Enter Items..." ...
28          className="bg-white/75 p-3 px-8 rounded-full outline-none lg:w-96 shadow-lg shadow-blue-100" ...
29        />

```



VS Code interface showing the RecipeItem component code. The file path is src > components > recipe-item > index.jsx. The code imports Link from react-router-dom. It defines a RecipeItem function that takes an item as a prop and returns a card with a title, publisher, and a link to the item's details.

```
src > components > recipe-item > index.jsx ...
1 import { Link } from "react-router-dom";
2
3 export default function RecipeItem({ item }) {
4   return (
5     <div className="flex flex-col w-80 overflow-hidden p-5 bg-white/75 shadow-xl gap-5 border-2 rounded-2xl" ...
6       <div className="h-40 flex flex-grow justify-center overflow-hidden items-center rounded-xl" ...
7         <img src={item?.image_url} alt="recipe-item" className="block w-full" />
8       </div>
9       <div>
10         <span className="text-sm text-cyan-700 font-medium">
11           {item?.publisher} ...
12         </span>
13         <h3 className="font-bold text-2xl truncate text-black" ...
14           | {item?.title}
15         </h3>
16         <Link
17           to={`/recipe-item/${item?.id}`}
18           className="text-sm p-3 mt-5 px-8 rounded-lg uppercase font-medium tracking-wider inline-block shadow-blue-100" ...
19         >
20           Recipe Details
21         </Link>
22       </div>
23     </div>
24   );
25 }
```

The screenshot shows the VS Code interface with the file `index.jsx` open in the editor. The code implements a `Details` function that uses the `useContext` and `useEffect` hooks. It fetches recipe details from an API and updates state. The `setRecipeDetailsData` function is used to update the state with fetched data.

```
src > pages > details > index.jsx > ...
1 import { useContext, useEffect } from "react";
2 import { useParams } from "react-router-dom";
3 import { GlobalContext } from "../../context";
4
5 export default function Details() {
6   const { id } = useParams();
7   const [
8     recipeDetailsData,
9     setRecipeDetailsData,
10    favoritesList,
11    handleAddToFavorite,
12  } = useContext(GlobalContext);
13
14   useEffect(() => {
15     async function getRecipeDetails() {
16       const response = await fetch(
17         `https://forkify-api.herokuapp.com/api/v2/recipes/${id}`
18       );
19       const data = await response.json();
20
21       console.log(data);
22       if (data?.data) {
23         setRecipeDetailsData(data?.data);
24       }
25     }
26     getRecipeDetails();
27   }, []);
28
29   console.log(recipeDetailsData, "recipeDetailsData");
}
```

The screenshot shows the VS Code interface with the file `index.jsx` open in the editor. The code implements a `Favorites` function that uses the `useContext` hook to access the `GlobalContext`. It maps over the `favoritesList` array to render `RecipeItem` components. If no items are present, it displays a message indicating that nothing is added to favorites.

```
src > pages > favorites > index.jsx > ...
1 import { useContext } from "react";
2 import { GlobalContext } from "../../context";
3 import RecipeItem from "../../components/recipe-item";
4
5 export default function Favorites() {
6   const { favoritesList } = useContext(GlobalContext);
6
7   return (
8     <div className="py-8 container mx-auto flex flex-wrap justify-center gap-10">
9       {favoritesList && favoritesList.length > 0 ? (
10         favoritesList.map((item) => <RecipeItem item={item} />)
11       ) : (
12         <div>
13           <p className="lg:text-4xl text-xl text-center text-black font-extrabold">
14             Nothing is added to favorites.
15           </p>
16         </div>
17       )
18     </div>
19   );
20 }
```

The screenshot shows the VS Code interface with the file `index.jsx` open in the editor. The code implements a `Home` function that uses the `useContext` hook to access the `GlobalContext`. It checks if `loading` is true and returns a loading message. Otherwise, it maps over the `recipeList` array to render `RecipeItem` components. If no items are present, it displays a message indicating that nothing to show and prompting the user to search.

```
src > pages > home > index.jsx > ...
1 import { useContext } from "react";
2 import { GlobalContext } from "../../context";
3 import RecipeItem from "../../components/recipe-item";
4
5 export default function Home() {
6   const { recipeList, loading } = useContext(GlobalContext);
6
7   if (loading) return <div>Loading...Please Wait!</div>;
7
8   return (
9     <div className="py-8 container mx-auto flex flex-wrap justify-center gap-10">
10       {recipeList && recipeList.length > 0 ? (
11         recipeList.map((item) => <RecipeItem item={item} />)
12       ) : (
13         <div>
14           <p className="lg:text-4xl text-xl text-center text-black font-extrabold">
15             Nothing to show. Please search something.
16           </p>
17         </div>
18       )
19     </div>
20   );
21 }
```

CMS App Code Link:

[https://drive.google.com/file/d/1ibKNgpBLuGCDQjBE-AjNNExI NyCl4l9/view
?usp=sharing](https://drive.google.com/file/d/1ibKNgpBLuGCDQjBE-AjNNExI NyCl4l9/view?usp=sharing)

This screenshot shows a code editor interface with two tabs open: 'auth.js' and 'contact.js'. The 'auth.js' tab is active, displaying the following code:

```
cms-backend > routes > JS auth.js x
const router = require("express").Router();
const bcrypt = require("bcrypt");
const jwt = require("jsonwebtoken");
const auth = require("../middlewares/auth");
const User = require("../models/User");

router.post("/register", async (req, res) => {
  const { name, email, password } = req.body;

  // check all the missing fields.
  if (!name || !email || !password)
    return res
      .status(400)
      .json({ error: `Please enter all the required field.` });

  // name validation.
  if (name.length > 25)
    return res
      .status(400)
      .json({ error: "name can only be less than 25 characters" });

  // email validation.
  const emailReg =
    `/^([a-zA-Z0-9]+@[a-zA-Z0-9]+\.[a-zA-Z0-9]+)+([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3})$/`;
  if (!emailReg.test(email))
    return res
      .status(400)
```

This screenshot shows the continuation of the 'auth.js' file from the previous screenshot, specifically the logic for the '/register' route's callback.

```
router.post("/register", async (req, res) => {
  const router = require("express").Router();
  const bcrypt = require("bcrypt");
  const jwt = require("jsonwebtoken");
  const auth = require("../middlewares/auth");
  const User = require("../models/User");

  router.post("/register", async (req, res) => {
    .status(400)
    .json({ error: "please enter a valid email address." });

    // validation of password.
    if (password.length < 6)
      return res
        .status(400)
        .json({ error: "password must be atleast 6 characters long" });
    try {
      const doesUserAlreadyExist = await User.findOne({ email });

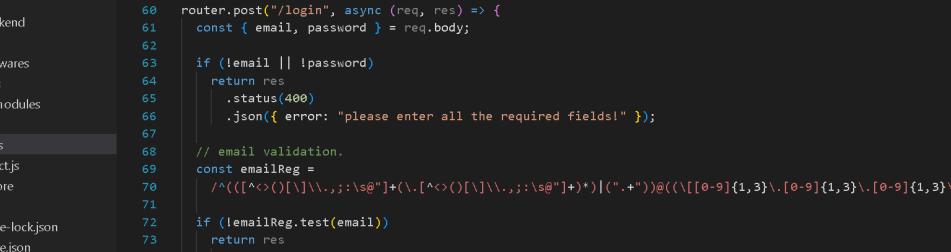
      if (doesUserAlreadyExist)
        return res.status(400).json({
          error: `a user with that email [${email}] already exists so please try another one.`
        });

      const hashedPassword = await bcrypt.hash(password, 12);
      const newUser = new User({ name, email, password: hashedPassword });

      // save the user.
      const result = await newUser.save();

      result._doc.password = undefined;

      return res.status(201).json({ ...result._doc });
    } catch (err) {
      console.log(err);
      return res.status(500).json({ error: err.message });
    }
  });
});
```



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** On the left, it shows the project structure under "CMS-APP". The "auth.js" file is currently selected.
- Code Editor:** The main area displays the "auth.js" file content. The code handles a POST request to "/login", validating email and password, and checking if the user exists using bcrypt.
- Status Bar:** At the top right, there are icons for minimizing, maximizing, and closing the window.

```
File Edit Selection View Go Run ... ← → 🔍 CMS-App
```

```
EXPLORER contact.js auth.js x
cms-backend > routes > auth.js > router.post("/login") callback
>
60 router.post("/login", async (req, res) => {
61   const { email, password } = req.body;
62
63   if (!email || !password)
64     return res
65       .status(400)
66       .json({ error: "please enter all the required fields!" });
67
68 // email validation.
69 const emailReg =
70   /((([a-z]([a-z\d]*[a-z])?([a-z\d]*[a-z])?)|([a-z\d]*[a-z]))\.\w{2,3})@((\w{1,3}\.\w{1,3}\.\w{1,3})|(\w{1,3}\.\w{1,3}))/
71
72 if (!emailReg.test(email))
73   return res
74     .status(400)
75     .json({ error: "please enter a valid email address." });
76
77 try {
78   const doesUserExists = await User.findOne({ email });
79
80   if (!doesUserExists)
81     return res.status(400).json({ error: "Invalid email or password!" });
82
83   // if there were any user present.
84   const doesPasswordMatch = await bcrypt.compare(
85     password,
86     doesUserExists.password
87   );
88 }
```

The screenshot shows the VS Code interface with the following details:

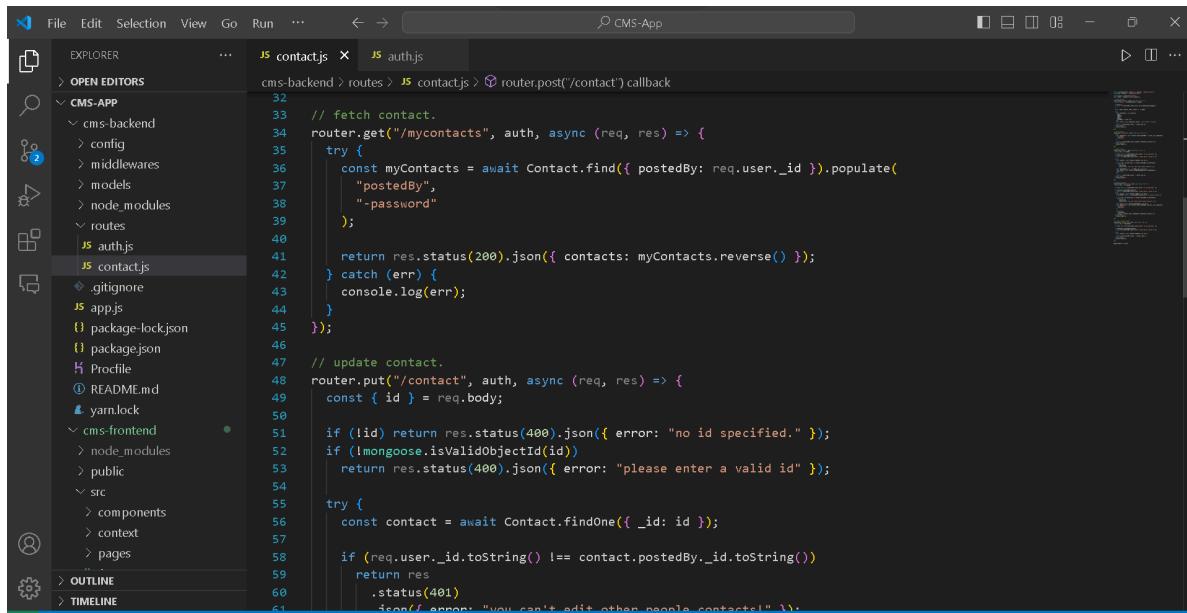
- File Explorer:** On the left, it shows the project structure under "cms-backend":
 - cms-backend
 - routes
 - auth.js
 - contact.js
 - gitignore
 - app.js
 - package-lock.json
 - package.json
 - Profile
 - README.md
 - yarn.lock
 - cms-frontend
 - public
 - src
 - components
 - context
 - pages
- Editor:** The main editor area displays the content of the "auth.js" file:

```
cms-backend > routes > auth.js > router.post("/login") callback
60   router.post("/login", async (req, res) => {
61     // if there were any user present.
62     const doesPasswordMatch = await bcrypt.compare(
63       password,
64       doesUserExits.password
65     );
66
67     if (!doesPasswordMatch)
68       return res.status(400).json({ error: "Invalid email or password!" });
69
70     const payload = { _id: doesUserExits._id };
71     const token = jwt.sign(payload, process.env.JWT_SECRET, {
72       expiresIn: "1h",
73     });
74
75     const user = { ...doesUserExits._doc, password: undefined };
76     return res.status(200).json({ token, user });
77   } catch (err) {
78     console.log(err);
79     return res.status(500).json({ error: err.message });
80   }
81 });
82
83 router.get("/me", auth, async (req, res) => {
84   return res.status(200).json({ ...req.user._doc });
85 });
86
87 });
88
89 module.exports = router;
```
- Search Bar:** At the top center, it says "CMS-App".
- Control Bar:** At the top right, there are standard window control buttons (minimize, maximize, close).

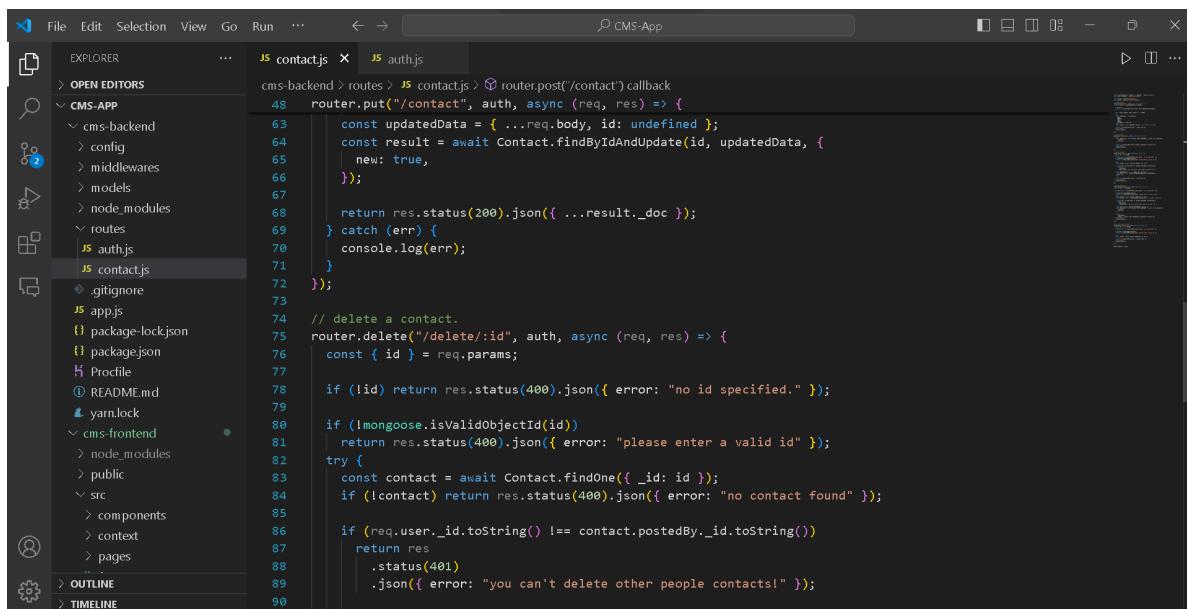
The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure under "CMS-APP". The "contact.js" file is currently selected.
- Code Editor (Right):** Displays the content of the "contact.js" file, which contains code for handling contact submissions using Express, Mongoose, and authentication middleware.
- Status Bar:** Shows "CMS-App" and various status icons.

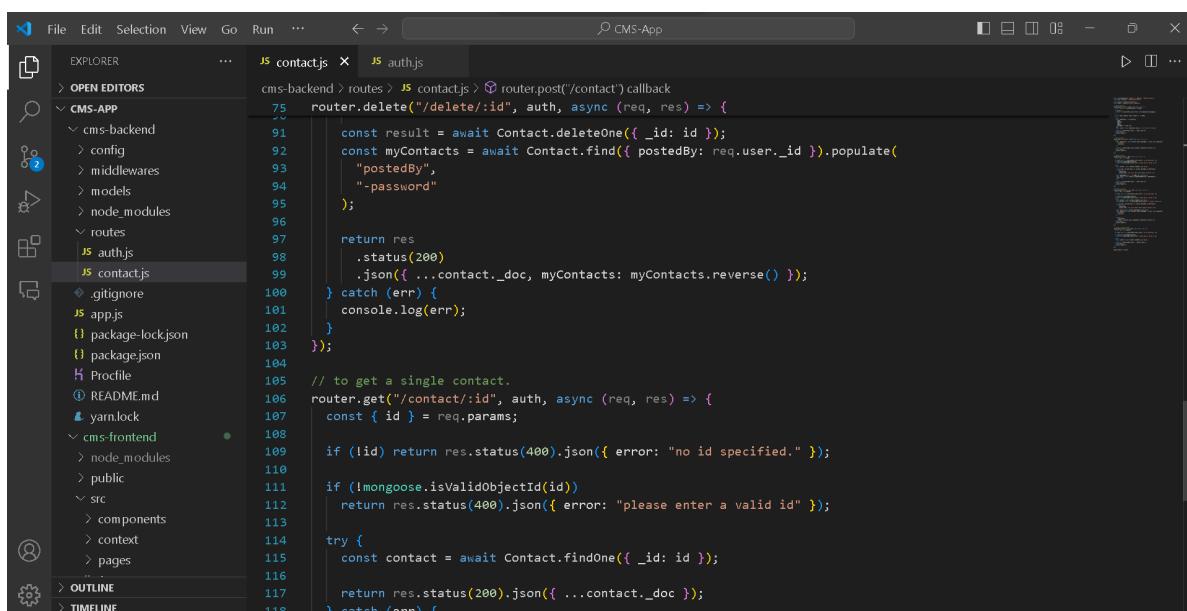
```
cms-backend > routes > contact.js > router.post("/contact") callback
1 const { validateContact, Contact } = require("../models/Contact");
2 const auth = require("../middlewares/auth");
3
4 const mongoose = require("mongoose");
5 const router = require("express").Router();
6
7 // create contact.
8 router.post("/contact", auth, async (req, res) => [
9   const { error } = validateContact(req.body);
10
11   if (error) {
12     return res.status(400).json({ error: error.details[0].message });
13   }
14
15   const { name, address, email, phone } = req.body;
16
17   try {
18     const newContact = new Contact({
19       name,
20       address,
21       email,
22       phone,
23       postedBy: req.user._id,
24     });
25     const result = await newContact.save(); //save method of mongodb
26
27     return res.status(201).json({ ...result._doc });
28   } catch (err) {
29     console.log(err);
30   }
31 ])
```



```
cms-backend > routes > JS contact.js > router.post('/contact') callback
32
33 // fetch contact.
34 router.get("/mycontacts", auth, async (req, res) => {
35   try {
36     const myContacts = await Contact.find({ postedBy: req.user._id }).populate(
37       "postedBy",
38       "-password"
39     );
40
41     return res.status(200).json({ contacts: myContacts.reverse() });
42   } catch (err) {
43     console.log(err);
44   }
45 });
46
47 // update contact.
48 router.put("/contact", auth, async (req, res) => {
49   const { id } = req.body;
50
51   if (!id) return res.status(400).json({ error: "no id specified." });
52   if (!mongoose.isValidObjectId(id))
53     return res.status(400).json({ error: "please enter a valid id" });
54
55   try {
56     const contact = await Contact.findOne({ _id: id });
57
58     if (req.user._id.toString() !== contact.postedBy._id.toString())
59       return res
60         .status(401)
61         .json({ error: "you can't edit other people contacts!" });
62   }
63 }
```



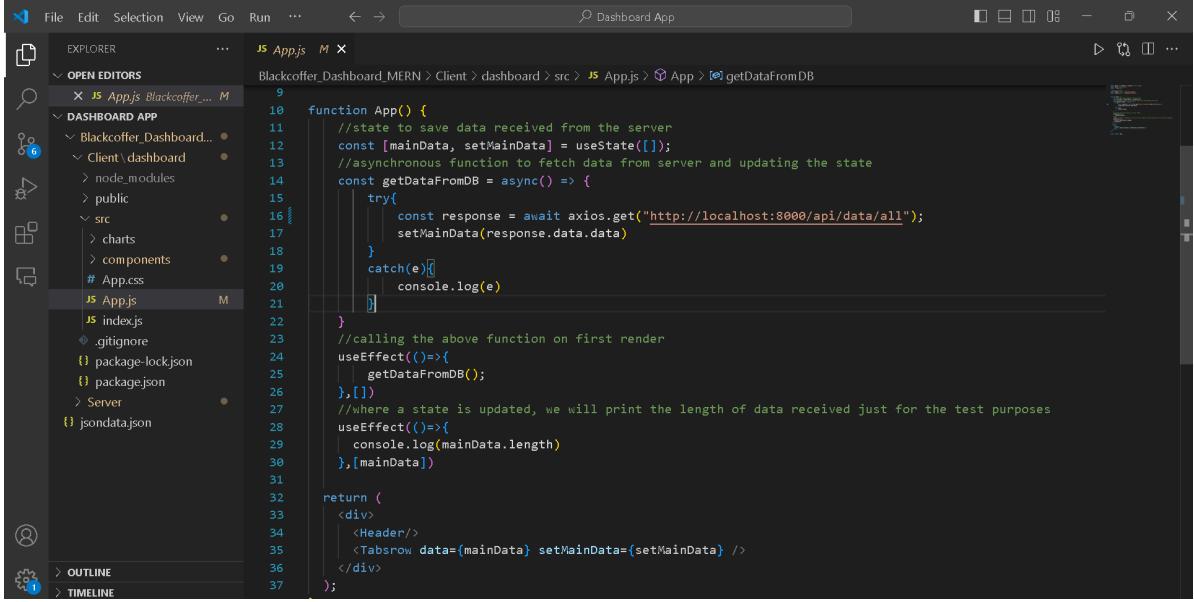
```
cms-backend > routes > JS contact.js > router.put('/contact') callback
48   router.put("/contact", auth, async (req, res) => {
49     const updatedData = { ...req.body, id: undefined };
50     const result = await Contact.findByIdAndUpdate(id, updatedData, {
51       new: true,
52     });
53
54     return res.status(200).json({ ...result._doc });
55   } catch (err) {
56     console.log(err);
57   }
58 }
59
60 // delete a contact.
61 router.delete("/delete/:id", auth, async (req, res) => {
62   const { id } = req.params;
63
64   if (!id) return res.status(400).json({ error: "no id specified." });
65
66   if (!mongoose.isValidObjectId(id))
67     return res.status(400).json({ error: "please enter a valid id" });
68   try {
69     const contact = await Contact.findOne({ _id: id });
70
71     if (!contact) return res.status(400).json({ error: "no contact found" });
72
73     if (req.user._id.toString() !== contact.postedBy._id.toString())
74       return res
75         .status(401)
76         .json({ error: "you can't delete other people contacts!" });
77   }
78 }
```



```
cms-backend > routes > JS contact.js > router.delete('/delete/:id') callback
75   router.delete("/delete/:id", auth, async (req, res) => {
76
77     const result = await Contact.deleteOne({ _id: id });
78     const myContacts = await Contact.find({ postedBy: req.user._id }).populate(
79       "postedBy",
80       "-password"
81     );
82
83     return res
84       .status(200)
85       .json({ ...contact._doc, myContacts: myContacts.reverse() });
86   } catch (err) {
87     console.log(err);
88   }
89 }
90
91
92 // to get a single contact.
93 router.get("/contact/:id", auth, async (req, res) => {
94   const { id } = req.params;
95
96   if (!id) return res.status(400).json({ error: "no id specified." });
97
98   if (!mongoose.isValidObjectId(id))
99     return res.status(400).json({ error: "please enter a valid id" });
100
101   try {
102     const contact = await Contact.findOne({ _id: id });
103
104     return res
105       .status(200)
106       .json({ ...contact._doc });
107   } catch (err) {
108     console.log(err);
109   }
110 }
```

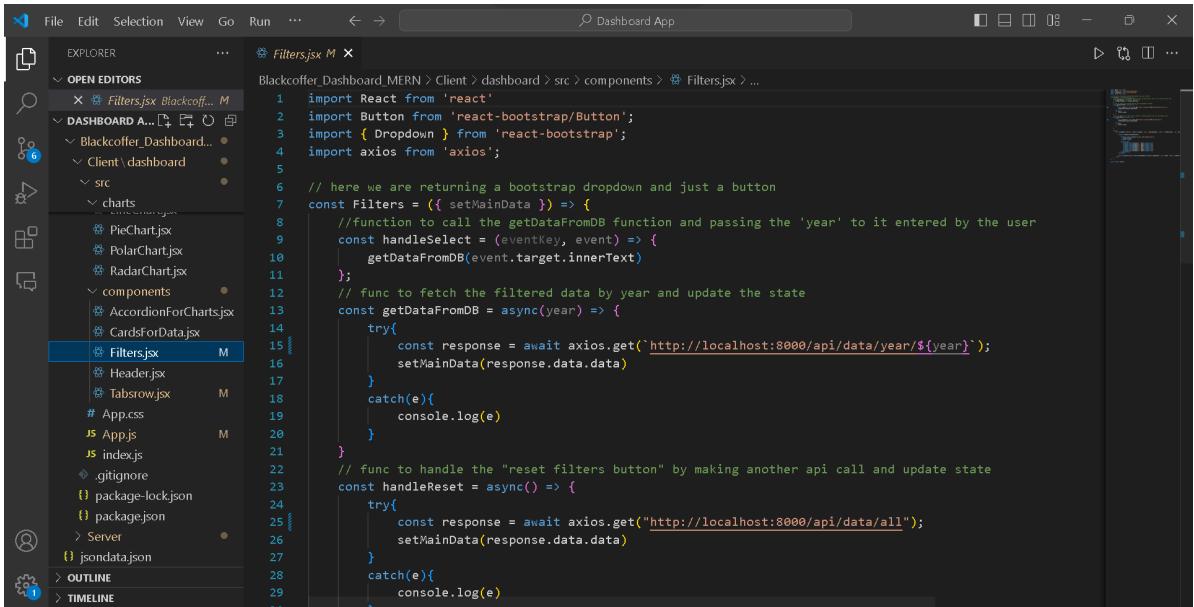
Dashboard App Code Link:

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



```
File Edit Selection View Go Run ... Dashboard App

EXPLORER OPEN EDITORS JS App.js M X
Blackcoffer_Dashboard_MERN > Client > dashboard > src > JS App.js > App > getDataFromDB
9
10 function App() {
11     //state to save data received from the server
12     const [mainData, setMainData] = useState([]);
13     //asynchronous function to fetch data from server and updating the state
14     const getDataFromDB = async() => {
15         try{
16             const response = await axios.get("http://localhost:8000/api/data/all");
17             setMainData(response.data.data)
18         }
19         catch(e){
20             console.log(e)
21         }
22     }
23     //calling the above function on first render
24     useEffect(()=>{
25         getDataFromDB();
26     },[])
27     //where a state is updated, we will print the length of data received just for the test purposes
28     useEffect(()=>{
29         console.log(mainData.length)
30     },[mainData])
31
32     return (
33         <div>
34             <Header/>
35             <Tabsrow data={mainData} setMainData={setMainData} />
36         </div>
37     );
}
```



```
File Edit Selection View Go Run ... Dashboard App

EXPLORER OPEN EDITORS Filters.jsx M X
Blackcoffer_Dashboard_MERN > Client > dashboard > src > components > Filters.jsx ...
1 import React from 'react'
2 import Button from 'react-bootstrap/Button';
3 import { Dropdown } from 'react-bootstrap';
4 import axios from 'axios';
5
6 // here we are returning a bootstrap dropdown and just a button
7 const Filters = ({ setMainData }) => {
8     //function to call the getDataFromDB function and passing the 'year' to it entered by the user
9     const handleSelect = (eventKey, event) => {
10         const year = eventKey;
11         const handleSelect = (eventKey, event) => {
12             const year = eventKey;
13             const handleSelect = (eventKey, event) => {
14                 try{
15                     const response = await axios.get(`http://localhost:8000/api/data/year/${year}`);
16                     setMainData(response.data.data)
17                 }
18                 catch(e){
19                     console.log(e)
20                 }
21             }
22             // func to fetch the filtered data by year and update the state
23             const getDataFromDB = async(year) => {
24                 try{
25                     const response = await axios.get(`http://localhost:8000/api/data/year/${year}`);
26                     setMainData(response.data.data)
27                 }
28                 catch(e){
29                     console.log(e)
30                 }
31         }
32         // func to handle the "reset filters button" by making another api call and update state
33         const handleReset = async() => {
34             try{
35                 const response = await axios.get("http://localhost:8000/api/data/all");
36                 setMainData(response.data.data)
37             }
38             catch(e){
39                 console.log(e)
40             }
41         }
42     }
43 }
```

```

1 import React, { useState } from 'react';
2 import Tab from 'react-bootstrap/Tab';
3 import Tabs from 'react-bootstrap/Tabs';
4 import CardGroup from 'react-bootstrap/CardGroup';
5 import Button from 'react-bootstrap/Button';
6
7 //import from files
8 import CardsForData from './CardsForData';
9 import AccordionForCharts from './AccordionForCharts';
10 import Filters from './Filters';
11 import axios from 'axios';
12
13 const Tabsrow = ({ data, setMainData }) => {
14   // state to store the number of data cards we want to display at a time, we'll update it on click of a tab
15   const [limit, setLimit] = useState(5);
16   const limitedData = data.slice(0, limit);
17   // state to store the search bar text
18   const [search, setSearch] = useState("");
19   // function to make an api call to get the filtered data
20   const handleSearchResult = async (e) => {
21     e.preventDefault();
22     try {
23       const response = await axios.get(`http://localhost:8000/api/data/any/${search}`);
24       setMainData(response.data.data);
25       setSearch("");
26     }
27     catch (e) {
28       console.log(e)
29     }
29

```

```

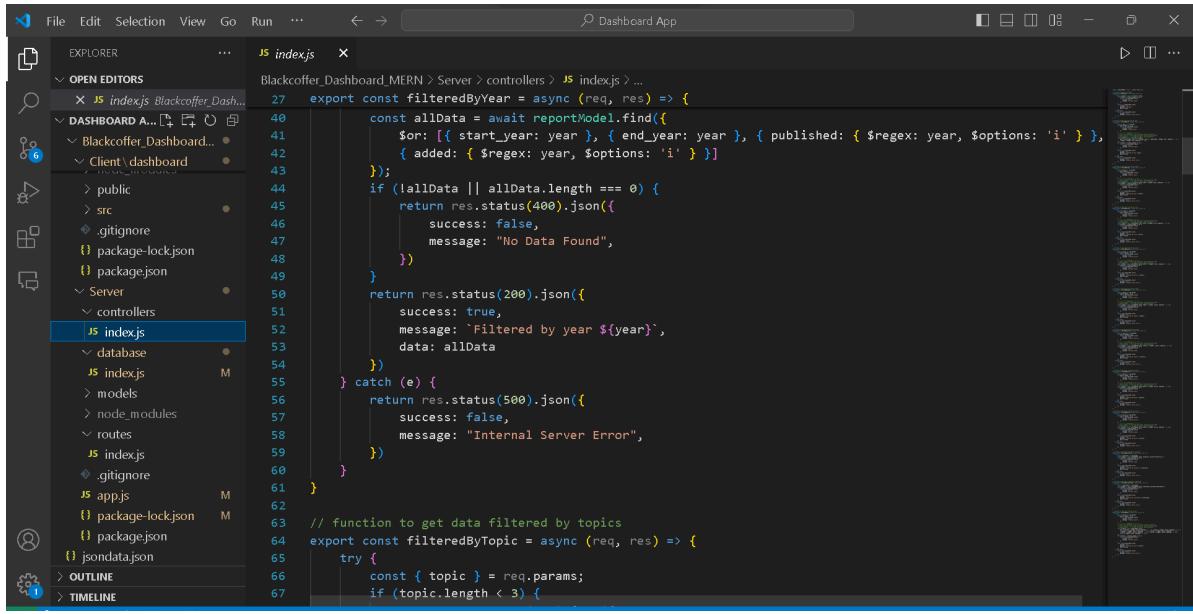
1 import React, { useEffect } from 'react';
2 import { PolarArea } from 'react-chartjs-2';
3 import Chart from 'chart.js/auto';
4
5 const PolarChart = ({ serverData }) => {
6
7   let uniqueSectors = [];
8   //using forEach because it doesn't return any array
9   serverData.forEach((i) => {
10     if (!uniqueSectors.includes(i.sector) && i.sector !== "") {
11       uniqueSectors.push(i.sector);
12     }
13   })
14
15   let uniqueTopics = [];
16   serverData.forEach((i) => {
17     if (!uniqueTopics.includes(i.topic) && i.topic !== "") {
18       uniqueTopics.push(i.topic);
19     }
20   })
21
22   let uniqueRegion = [];
23   serverData.forEach((i) => {
24     if (!uniqueRegion.includes(i.region) && i.region !== "") {
25       uniqueRegion.push(i.region);
26     }
27   })
28
29

```

```

1 import { reportModel } from "../models/index.js";
2
3 // function to get all the data
4 export const getAllData = async (req, res) => {
5   try {
6     const allData = await reportModel.find();
7     if (!allData || allData.length === 0) {
8       return res.status(400).json({
9         success: false,
10         message: "No data found"
11       })
12     }
13     return res.status(200).json({
14       success: true,
15       message: "All data",
16       data: allData
17     })
18   } catch (e) {
19     return res.status(500).json({
20       success: false,
21       message: "Internal Server Error",
22     })
23   }
24
25 // Function to get data filtered by year
26 export const filteredByYear = async (req, res) => {
27   try {
28     const { year } = req.params;
29     const filteredData = await reportModel.find({ year });
30     return res.status(200).json({
31       success: true,
32       message: "Data filtered by year",
33       data: filteredData
34     })
35   } catch (e) {
36     return res.status(500).json({
37       success: false,
38       message: "Internal Server Error",
39     })
40   }
41
42

```



```
Blackcoffer_Dashboard_MERN > Server > controllers > index.js > ...
27  export const filteredByYear = async (req, res) => {
40      const allData = await reportModel.find({
41          $or: [{ start_year: year }, { end_year: year }, { published: { $regex: year, $options: 'i' } },
42              { added: { $regex: year, $options: 'i' } }]
43      });
44      if (!allData || allData.length === 0) {
45          return res.status(400).json({
46              success: false,
47              message: "No Data Found",
48          })
49      }
50      return res.status(200).json({
51          success: true,
52          message: `Filtered by year ${year}`,
53          data: allData
54      })
55  } catch (e) {
56      return res.status(500).json({
57          success: false,
58          message: "Internal Server Error",
59      })
60  }
61
62 // function to get data filtered by topics
63 export const filteredByTopic = async (req, res) => {
64     try {
65         const { topic } = req.params;
66         if (topic.length < 3) {
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
159
160
161
162
163
164
165
166
167
168
169
169
170
171
172
173
174
175
176
177
178
179
179
180
181
182
183
184
185
186
187
188
189
189
190
191
192
193
194
195
196
197
198
199
199
200
201
202
203
204
205
206
207
208
209
209
210
211
212
213
214
215
216
217
217
218
219
219
220
221
222
223
224
225
226
227
227
228
229
229
230
231
232
233
234
235
235
236
237
237
238
239
239
240
241
242
243
244
244
245
246
246
247
247
248
248
249
249
250
251
251
252
252
253
253
254
254
255
255
256
256
257
257
258
258
259
259
260
260
261
261
262
262
263
263
264
264
265
265
266
266
267
267
268
268
269
269
270
270
271
271
272
272
273
273
274
274
275
275
276
276
277
277
278
278
279
279
280
280
281
281
282
282
283
283
284
284
285
285
286
286
287
287
288
288
289
289
290
290
291
291
292
292
293
293
294
294
295
295
296
296
297
297
298
298
299
299
300
300
301
301
302
302
303
303
304
304
305
305
306
306
307
307
308
308
309
309
310
310
311
311
312
312
313
313
314
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
139
```