FOOD DELIVERY APPLICATION USING MERN STACK

Project report submitted in partial fulfilment of the requirement for the degree of Bachelor of Technology

in

Computer Science and Engineering

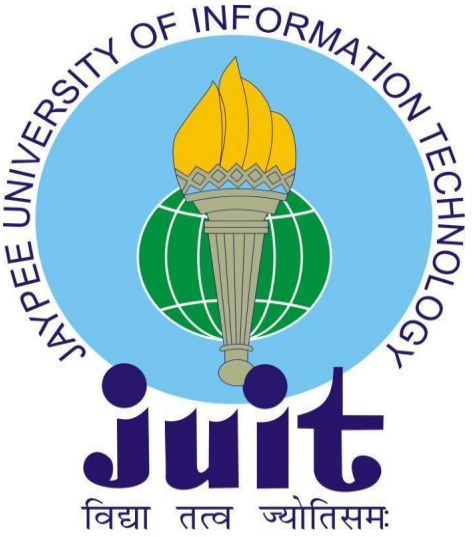
by

Manan Mehta [191386]

under the supervision of

Dr Vipul Kumar Sharma

to



Department of Computer Science & Engineering
and Information Technology

Jaypee University of Information Technology Waknaghat, Solan-173234, Himachal Pradesh

Certificate

Candidate's Declaration

I hereby declare that the work presented in this report entitled **Food Delivery Application using MERN Stack** in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering/Information Technology** submitted in the **Department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat** is an authentic record of my work carried out over a period from **July 2022 to May 2023** under the supervision of **Dr Vipul Kumar Sharma, Assistant Professor (Senior Grade)**.

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

(Student Signature)
Manan Mehta, 191386

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

(Supervisor Signature)
Dr Vipul Kumar Sharma
Assistant Professor (Senior Grade)
Computer Science & Engineering and Information Technology

Dated: 15 May 2023

Plagiarism Certificate

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: .....

Type of Document (Tick): PhD Thesis | M.Tech Dissertation/ Report | B.Tech Project Report | Paper

Name:_____ **Department:**_____ **Enrolment No** _____

Contact No._____ **E-mail.**_____

Name of the Supervisor: _____

Title of the Thesis/Dissertation/Project Report/Paper (In Capitalletters): _____

_____

_____

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

Complete Thesis/Report Pages Detail:

- Total No. of Pages =
- Total No. of Preliminary pages =
- Total No. of pages accommodate bibliography/references =

(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at .....(%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

(Signature of Guide/Supervisor) **Signature of HOD**

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

<table>
<tr><th>Copy Received on</th><th>Excluded</th><th>Similarity Index (%)</th><th colspan="2">Generated Plagiarism Report Details (Title, Abstract & Chapters)</th></tr>
<tr><td></td><td rowspan="4">• All Preliminary Pages
• Bibliography/Images/Quotes
• 14 Words String</td><td rowspan="2"></td><td>Word Counts</td><td></td></tr>
<tr><td>Report Generated on</td><td>Character Counts</td><td></td></tr>
<tr><td></td><td>Submission ID</td><td>Total Pages Scanned</td><td></td></tr>
<tr><td></td><td></td><td>File Size</td><td></td></tr>
</table>

Checked by
Name & Signature **Librarian**

.....

Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at plagcheck.juit@gmail.com

Acknowledgement

Firstly, I express my heartiest thanks and gratefulness to almighty God for his divine blessing in making it possible to complete the project work successfully.

I am grateful and wish our profound indebtedness to Supervisor **Dr Vipul Kumar Sharma, Assistant Professor (Senior Grade)**, **Department of Computer Science and Engineering**, **Jaypee University of Information Technology, Solan**. The deep knowledge & keen interest of my supervisor in the field of 'Web Development' helped us to carry out this project. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stages have made it possible to complete this project.

I am also grateful to my mentors- **Utkarsh Singhal (Software Developer)**, **Shivani Premi (Software Developer)**, **Varun Kapoor (Full-Stack Lead)**, and Team Lead- **Anand Bhargava (VP Engineering and Product Strategy)** at **Paxcom India Pvt. Ltd - A Paymentus Company**, for their guidance, support and encouragement throughout our development training period and for the successful completion of this project. I thank my team members for assisting me and helping me in times when I got stuck and had no clue on how to proceed during the development time.

I would also generously welcome each one of those individuals who have helped us straightforwardly or in a roundabout way in making this project a win. In this unique situation, I might want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking.

Finally, I must acknowledge with due respect the constant support and patience of our parents.

Manan Mehta
[191386]

Table Of Contents

List of Figures

Abstract

This project report presents the development and implementation of a Food Delivery Application utilizing the MERN stack (MongoDB, Express.js, React.js, and Node.js). The purpose of this project was to create a user-friendly and efficient platform that connects customers with the Pizza Restaurant, providing a convenient food ordering and delivery experience.

The objectives of the project include designing an intuitive user interface, implementing a robust backend system, and ensuring data security. The MERN stack was chosen for its versatility, scalability, and ability to handle complex web applications.

The development process is detailed, starting with the database design using MongoDB to store user information and order details. The backend system was built using Node.js and Express.js, providing a RESTful API for handling various operations such as user authentication and order processing.

The front-end development was accomplished using React.js, focusing on creating a responsive and interactive user interface. Customers can view menus and place orders.

The report discusses the technical challenges faced during the development process and the corresponding solutions implemented.

In conclusion, the Food Delivery Application developed using the MERN stack demonstrates the successful implementation of a feature-rich platform that facilitates seamless food ordering and delivery services. The utilization of modern web technologies and robust development methodologies ensures a user-friendly experience while meeting the demands of both customers and restaurant owners.

CHAPTER 1
INTRODUCTION

1.1 Introduction

Building modern and dynamic online apps now necessitates a strong grasp of web development. The MERN stack is a well-known and powerful mix of technologies that enables programmers to create extremely scalable and successful online applications. In this introduction, the components of the MERN stack will be covered, as well as their significance in web development.

The MERN stack is comprised of MongoDB, Express, React, and Node.js. Each component is critical to the development process and aids in the overall operation and execution of the web application.

1. MongoDB: MongoDB is a NoSQL database that employs an adaptive document-based data model. It saves data as JSON-like documents, making it simple to integrate and alter data within the application. Because of its scalability and ability to handle massive amounts of data, MongoDB is the best solution for web applications that require dynamic data storage.

2. Express.js: Express.js is a simple and versatile Node.js web application framework. It provides a comprehensive set of API and web app development tools and information. Express.js makes it simpler to manage routes, process requests and responses, and build middleware. It is extremely adaptable and effective for developing web apps due to its lightweight and modular architecture.

3. React.js: React.js is a JavaScript library for developing user interfaces. It allows for the design of reusable UI components as well as their effective updating and rendering in response to data changes. React.js' component-based architecture promotes reuse, modularity, and maintainability. It also has a virtual DOM (Document Object Model) for increased performance and rendering.

4. **Node.js:** Node.js is a JavaScript server-side runtime environment that allows programmers to run JavaScript code outside of a web browser. Its event-driven, non-blocking design makes it exceptionally scalable and excellent at managing several requests at once. Because it provides server-side scripting, file system operations, and network connectivity, Node.js is an excellent choice for building the backend of websites.

Image 1: MERN stack

These technologies work together to create a comprehensive web development solution. MongoDB handles the database layer, Express.js handles the backend and routing, React.js handles the frontend user interface, and Node.js handles the server-side runtime environment.

The MERN stack provides various advantages to web developers. It provides a single JavaScript-based development environment that allows frontend and backend components to connect seamlessly. Components of React.js may be reused, which speeds up development and increases maintainability. Furthermore, the scalability and adaptability of MongoDB with Node.js enable for the efficient processing of enormous amounts of data and concurrent queries.

Finally, the MERN stack offers a stable and fast platform for web development. It integrates MongoDB, Express.js, React.js, and Node.js to develop full-stack, scalable online apps.

1.2 Problem Statement

A persistent need for effective and scalable technologies that allow for the construction of contemporary and dynamic web applications exists in the field of web development. The conventional method of using several technologies for various web development tasks frequently results in complexity, inefficiency, and integration difficulties.

The issue at hand is the requirement for a thorough and coherent framework that simplifies web development from front-end to back-end while guaranteeing great speed, scalability, and maintainability. The absence of a cohesive approach frequently reduces developer productivity, lengthens the development process, and limits their capacity to quickly adjust to changing project needs.

Additionally, developers who need to migrate between several technologies frequently have compatibility problems and a high learning curve as a result of the lack of a standardised stack. In addition to raising development costs, this makes it more difficult to maintain and update online applications over time.

The MERN stack, which consists of MongoDB, Express, React, and Node.js, offers a potential remedy for these problems. By using the advantages of these technologies, the problem of fragmented and inefficient web development may be resolved. However, it is vital to recognise and comprehend the precise problems and challenges that developers deal with while using conventional web development techniques.

Some of the key issues that need to be addressed include

1. Fragmented Development Environment: Developers frequently use many frameworks and programming languages for various web development tasks, creating a confusing and complex environment.

2. Integration Issues: Integrating several technologies, databases, and frameworks can be difficult and time-consuming, which can cause compatibility problems and reduce the speed of development.

3. Scalability and Performance Limitations: Conventional techniques to web development could not offer the performance and scalability needed to handle massive volumes of data and concurrent queries.

4. Lack of Reusability: Unreliable code reuse causes duplication of work during development and raises the risk of mistakes and problems.

5. Steep Learning Curve: The requirement for developers to learn and adapt to a variety of technologies, each with its syntax and conventions, can impede productivity and slow down the development process.

By addressing these issues and creating a comprehensive solution utilising the MERN stack, web development productivity will be much improved, developer cooperation will be improved, and the process of creating contemporary, scalable, and high-performance online apps will be streamlined.

1.3 Objectives

Objectives of Enhancing Web Development Efficiency using the MERN Stack:

1. To provide a complete framework that unifies all aspects of web development, from front end to back end, and ensures excellent performance, scalability, and maintainability.

2. To provide a unified solution that makes use of the capabilities of MongoDB, Express.js, React.js, and Node.js to address the problem of fragmented and inefficient web development.

3. To offer a standardised stack that streamlines development efforts, lessens integration difficulties, and boosts developer productivity.

4. To increase the reuse of code by encouraging the usage of reusable React.js components, which will cut down on duplicative development work and improve maintainability.

5. To offer a flexible and effective solution that can manage several concurrent requests and big volumes of data, to satisfy the requirements of contemporary online applications.

6. To increase developer productivity and teamwork by offering a single programming environment with standardised syntax and conventions, which will lower the learning curve for developers.

7. To provide a long-term updateable and maintainable system that offers a sustainable and forward-looking method for web development.

The overall goal of this project is to increase web development productivity and address the difficulties that developers have while using conventional web development techniques. We strive to offer a complete and scalable solution that streamlines the development process, boosts productivity, and assures excellent performance and maintainability of contemporary web applications by using the MERN stack.

1.4 Methodology

Methodology for Enhancing Web Development Efficiency using the MERN Stack:

- **Needs analysis**: Conduct a comprehensive requirements analysis to pinpoint the particular difficulties and pain points that developers have while using conventional web development techniques. This will assist in outlining the project's needs and goals.

- **Framework Design**: Design a thorough and well-rounded framework that makes use of the MERN stack to offer a combined frontend and backend development solution. This framework will assure excellent performance, scalability, and maintainability while addressing the highlighted pain areas.

- **Development**: Using the MERN stack, put the planned framework into use while making sure best practices and standards are followed. Create reusable React.js components to encourage the reuse of code and cut down on development time. To guarantee great performance, scalability, and reliability, test the designed solution.

- **Integration**: To ensure compatibility and lessen integration issues, integrate the generated solution with other pertinent technologies, databases, and frameworks.

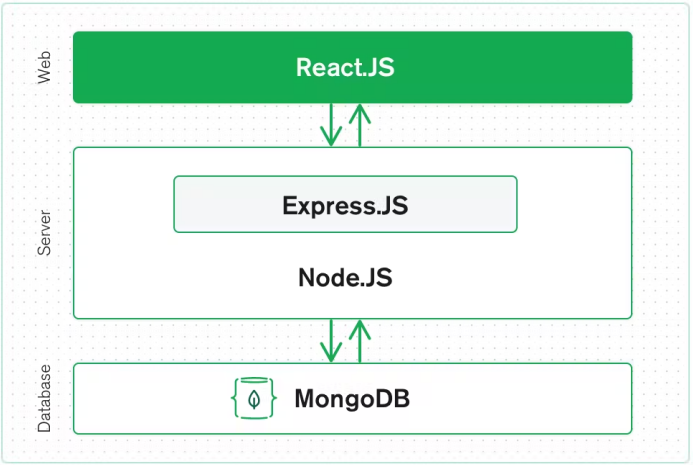


Figure 1: Integration of MERN stack.

- **Deployment:** Put the designed solution into production, assuring scalability, dependability, and security. Load-test the solution to ensure that it can handle big volumes of data and concurrent requests.

- **Training:** Provide developers with training and assistance so that they can use the generated solution efficiently. This will shorten the learning curve while increasing developer productivity and cooperation.

- **Maintenance and updates:** Maintain and update the produced solution continuously to ensure that it stays current and relevant. This will give a long-term and forward-thinking approach to web development.

Overall, the goal of this technique is to provide a comprehensive and coherent solution that streamlines web development, increases efficiency, and assures excellent performance and maintainability of modern online applications utilising the MERN stack. To provide a sustainable and future-proof approach to web development, the methodology emphasizes best practices, adherence to standards, and continuing repairs and improvements.

1.5 Organisation

This project report is divided into five distinct chapters, each of which is explained below

Chapter 1: Chapter 1 presents an overview of our project, the issue statement that acts as the foundation for our project, our primary project goals, and the approach or solution that was executed. In this project, we intend to increase web development efficiency by leveraging the MERN stack, a comprehensive and unified solution for modern web development. We seek to address the fragmented and inefficient web development process, which can lead to poor performance, scalability, and maintainability. Our objectives are to create a standardised and scalable solution that avoids superfluous development efforts, increases productivity, and delivers good performance and maintainability. Conducting a complete needs analysis, building a comprehensive framework, developing the solution using best practices and standards, integrating and delivering the solution, providing training and support, and guaranteeing continuous maintenance and upgrades are all part of our approach. Overall, the goal of this project is to create a sustainable and future-proof approach to web development that tackles developer difficulties while improving efficiency, productivity, and cooperation.

Chapter 2: This literature study looks at several elements of developing a food delivery application with the MERN stack. Because of its versatility, scalability, and ease of use, the MERN stack, which includes MongoDB, Express, React, and Node.js, is a popular choice for web application development. This research examines the literature to identify the essential components and technologies utilised in a food delivery application developed with the MERN stack, as well as best practices for their implementation.

The backend, which is responsible for managing user requests, data storage, and business logic, is one of the most important components of a meal delivery programme. Node.js and Express are common solutions for constructing the application's backend. Combining Node.js and Express with MongoDB, according to the literature, leads to a reliable and scalable backend architecture. Because Node.js enables non-blocking I/O, it is an excellent choice for designing server-side applications. Express, on the other hand, is a lightweight and versatile web application framework that makes creating RESTful APIs easy. Using MongoDB as the database for the application allows for efficient and scalable data storage and retrieval.

In addition to the back-end, the front-end of the software is critical. React is a popular choice for developing the application's front-end. React enables you to design reusable user interface components declaratively and quickly. It also enables state management, making it easy to manage application data and user interface state.

To control user input and data delivery, the literature suggests using HTTP methods and REST APIs. HTTP methods such as GET, POST, PUT, and delete are frequently used to retrieve, produce, update, and remove data. REST APIs standardise data access and manipulation, making application design and maintenance easier.

In addition to the technologies indicated above, the literature suggests using a variety of other libraries and frameworks to enhance the capabilities of the application. Body-parser is a middleware that parses incoming request bodies, making it easier to manage form input and JSON payloads. Cors is a middleware that enables the server to receive and respond to requests from other domains by enabling cross-origin resource sharing. React Router is a framework that provides React with routing capabilities, allowing for simple navigation between application pages.

Overall, the study demonstrates that using the MERN stack to build a food delivery service provides a scalable and customizable architecture. The usage of Node.js and Express on the backend, React on the front-end, and MongoDB on the database allows for fast data storage, retrieval, and processing.

Chapter 3: The chapter summarises the system development and architecture. The MERN (MongoDB, Express.js, React.js, and Node.js) stack was used to create a web-based meal delivery application. The application's goal is to allow users to purchase meals from a restaurant and have them delivered to their preferred location.

The system is designed to be user-friendly and easy to use. The home page of the application displays the website's landing page. Users may view the menu by selecting the Menu button and then the things they want to order. After making their selections, customers can move to the checkout page to complete their orders. Users can register and submit personal information and preferences.

The backend of the application is built utilising Node.js and Express.js. It handles communication between the front end and the MongoDB database. Menu items, users, and orders are all stored in the database.

The front end of the application is built with React.js. It communicates with the backend via HTTP requests and displays the information collected from the server understandably. The user interface is designed to be responsive and usable on a variety of platforms, including smartphones, tablets, and desktop computers.

Various software testing methodologies were used during the development process to ensure that the application worked as expected. Test cases covered all aspects of the software, including user authentication, order processing, and database interfaces.

Overall, the built system is a functioning and user-friendly meal delivery application that allows users to order food from their favourite restaurants in a simple manner. The application is created with the MERN stack in mind, with an emphasis on security, responsiveness, and usability in mind. The programme, with its varied features and testing methods, is well-equipped to suit the demands of both users and restaurant owners.

Chapter 4: By documenting the tests that were carried out and the results that were achieved at different stages of the project, this chapter of the report gives a detailed picture of the software's accuracy.

The purpose of the experiment was to evaluate how well the MERN stack-based food delivery application worked. The test bed configuration included running the application on a local server and carrying out various user operations including joining up, logging in, adding items to the cart, and placing orders.

In the beginning, system analysis was done analytically, with the code being checked for errors and inconsistencies. This technique made it easier to find and fix programming flaws of a minor nature. The second method was experimental, involving a variety of user behaviours with documented outcomes.

The results obtained at various stages of the experiment were contrasted with the behaviours anticipated by the application. The result included successful registration and login, accurate menu and restaurant information presentation, successful cart filling, and successful order placement at various stages.

The program's features and elements underwent testing to ensure they functioned as intended. This required putting the various API endpoints to the test and ensuring that the necessary outcomes were obtained. To study how different components interacted and ensure that they operated in concert, integration testing was done. This involved analysing the data flow between the front-end and back-end parts of the application. The application was put through acceptance testing to make sure it worked as intended and adhered to the design requirements. This required evaluating the software from the user's point of view and making sure that all user activities were effective.

The test findings demonstrated that, for the most part, the application was operating as anticipated. However, certain problems were found during testing, including sluggish loading times, sporadic server faults, and inaccurate order history display. These problems were located, corrected, and the programme was tested again to make sure the improvements worked.

Chapter 5: This chapter concludes and summarises the entire project. The MERN stack-based food delivery solution is very efficient, scalable, and secure. Multiple testing approaches, including software testing, experimental testing, and analytical testing, were used to demonstrate the system's robustness. The app's future potential is tremendous, and it may be enhanced with features like machine learning algorithms, real-time tracking of delivery agents, and support for several languages and currencies. The use of the MERN stack by the application proved to be a successful and efficient method of constructing a scalable and robust system.

CHAPTER 2
LITERATURE SURVEY

The well-known MERN stack for web development consists of four technologies: MongoDB, Express.js, React.js, and Node.js. This stack is well-known for its versatility, efficacy, and usefulness. One of the primary advantages of utilising the MERN stack is that programmers may construct full-stack web apps using only JavaScript. In this review of the literature, we'll look at various facets of creating a web application with the MERN stack, such as setting up the environment for development, sending data from React to Node.js, utilising middleware like body-parser in Express.js, fetching data in React, handling post requests in Express.js, and connecting to a MongoDB database.

Developers may use a variety of online instructions to build up the development environment for a web application using the MERN stack. The procedures to link React and Node.js are described in one such tutorial, available at codedamn.com. The usage of package.json to manage dependencies and concurrently to execute the front end and back end simultaneously are highlighted in this guide.

There are numerous ways to accomplish the frequent need of sending data from React to Node.js in web development. One tutorial (tutsmake.com) explains how to use Axios to transmit data from React to a Node.js server via an HTTP POST request. It also describes how to parse the request body using the Express.js middleware body-parser.

Express.js's robust middleware capability enables developers to extend the server's capabilities. The usage of the body-parser middleware in Express.js is described in one article on Stack Overflow (stackoverflow.com). Before the handlers, this middleware is used to process incoming request bodies.

To update the front-end with fresh data, it is usual practice in web development to retrieve data from a server. One tutorial (developerway.com) describes how to utilise React's fetch API to get data from a server. Additionally, it explains how asynchronous functions are used and guarantees that the response data will be handled.

Express.js's post-request handling is a crucial component of server-side web development. How to handle post requests in Express.js and deliver a response to the client is covered in one article on Stack Overflow. This post also describes how to handle asynchronous activities by using the async/await syntax.

Using the MERN stack to construct a website requires connecting to a MongoDB database. The procedures to set up an Express.js server with a MongoDB database are described in one tutorial at (mongodb.com). It also teaches how to build models for MongoDB collections using the Mongoose library.

There are some tips and tactics for developers in addition to the technical elements of web development utilising the MERN stack. How to duplicate a multidimensional array in JavaScript is described in one article on Stack Overflow. The usage of Axios to send HTTP requests in React is covered in another article on zetcode.com. On Telerik's website, there are lessons on how to build dynamic forms.

Last but not least, there are some online resources you can utilise to learn more about React Router, the tool used to manage navigation in a React application. Resources for upgrading to React Router v6 are also available at reactrouter.com as well as React Router's ability to transmit data across pages.

In conclusion, there are many online resources for learning how to utilise the MERN stack, which is a potent technological stack for developing web applications.

CHAPTER 3
SYSTEM DEVELOPMENT

3.1 Analysis

An online platform that links businesses and customers, a meal delivery application enables users to purchase food from the restaurant of their choice and have it delivered right to their homes. The programme will primarily consist of three parts: a database, an API server, and a web-based client. The application needs to be user-friendly, safe, and scalable.

3.2 Design

The MERN (MongoDB, Express, React, Node.js) stack, which offers a strong and adaptable environment for developing scalable online applications, will be used to construct the application. The database used to store restaurant and customer data will be MongoDB. The backend API server, which manages requests and answers between the database and front-end, will be constructed using Express. Customers will be able to explore and order meals from the restaurant thanks to a responsive and user-friendly frontend that will be built using React. The backend server will operate on Node.js.

3.3 Development

The application will have the following features:

1. **User Authentication** - Customers will be required to register and log in to access the application.
2. **Menu Management** - Customers will be able to browse through the menu.
3. **Cart and Checkout** - Customers will be able to add items to the cart and checkout using different payment methods.
4. **Order Management** - Restaurants will be able to view and manage orders and their status.

3.4 Algorithm

The algorithm steps are as follows:

1. Create the database schema for the food delivery app.
2. Configure the Express server with the necessary middleware (body-parser and CORS).
3. Use Express to create the server's endpoints for handling HTTP requests and responses.
4. Integrate the MongoDB JS driver with the Express server to allow the application to communicate with the database.
5. Implement the server-side logic for creating, reading, updating, and deleting data in the database based on the HTTP requests.
6. Create a React application with the necessary components to display the food items and allow users to place orders.
7. Use React Router DOM to create the routes for the different pages of the application.
8. Use useState, useEffect, and useLocation hooks in React to handle state management, data fetching, and navigation between pages.
9. Implement the logic for placing orders in the React application and sending the order details to the server via HTTP requests.
10. Display the order confirmation and update the order status in the database using the server-side logic.
11. Use JSON for data exchange between the client and server to ensure consistency and compatibility.
12. Test the application thoroughly to ensure that it works as expected and handles any potential errors or exceptions.

The main stages necessary to create a food delivery application utilising the MERN stack are summarised in this algorithm. The database schema must be designed, the server and endpoints must be configured, the database driver must be integrated, and the server-side and client-side logic must be put into place. The construction of this application relies heavily on the use of React, React Router DOM, useState, useEffect, useLocation, JSON, Express, body-parser, CORS, and MongoDB JS driver.

3.5 Model Development

3.5.1 MongoDB

A common NoSQL database management system, MongoDB, is utilised in many contemporary online apps, including those that serve meals. One of MongoDB's key benefits is its capacity for document-based data storage, which makes it the perfect option for applications requiring dynamic schema structures.

MongoDB may be used to store a variety of data types in the context of a meal delivery application, including user profiles, orders, menus, and reviews. Each data item is represented as a document, a data structure akin to JSON that supports many fields of various types and values. As a result, sophisticated SQL queries and rigid schema structures are not a concern for developers when storing and retrieving complex data items.

Image 2: MongoDB

Additionally, MongoDB has strong querying and indexing features that let developers easily access data and run sophisticated searches with a simple syntax. MongoDB may be used by a food delivery service, for instance, to discover all restaurants that provide a given cuisine or to retrieve all orders placed by a single user.

Because of MongoDB's automatic sharding and replication features, flexibility, and querying power, developers can easily scale their applications as data volume grows. Encryption, access control, and authentication are just a few of the security capabilities that MongoDB offers to help protect the data stored in the database.

Overall, MongoDB is a strong and adaptable database management system that is suitable for cutting-edge online applications like those for food delivery. It is a popular option for developers that need to create reliable and scalable applications because of its capacity to store and retrieve complex data items in a flexible and scalable manner and its strong searching and indexing capabilities.

3.5.2 React.js

A well-liked JavaScript library for creating user interfaces is React.js. It was created by Facebook, and a sizable development community is now responsible for maintaining it. React was used to develop the front-end of the application, which is the user-facing portion of the programme, in the context of the food delivery application.

React offers a collection of components and features that make it simple to construct intricate user experiences rapidly. It makes use of a component-based design, which implies that the user interface is composed of simple, interchangeable parts that may be joined to produce more intricate interfaces.

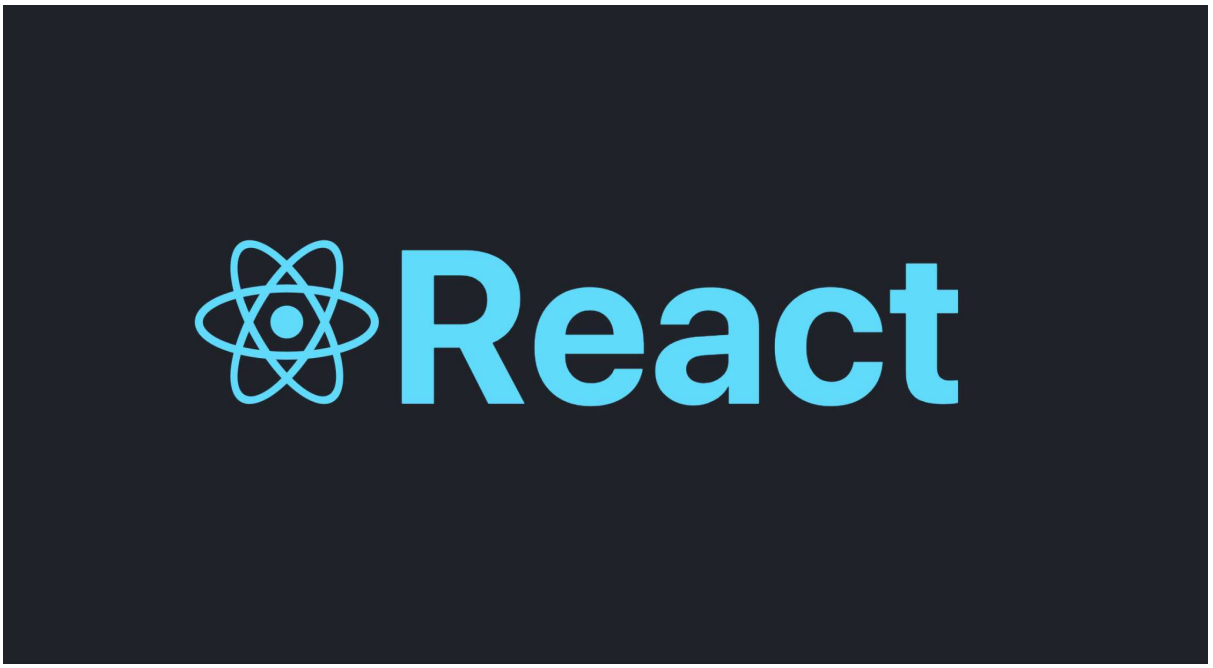


Image 3: React.js

To control the UI's state, React additionally makes use of a virtual DOM (Document Object Model). Because of this, React can refresh the UI quickly when the state changes without needing to reload the website.

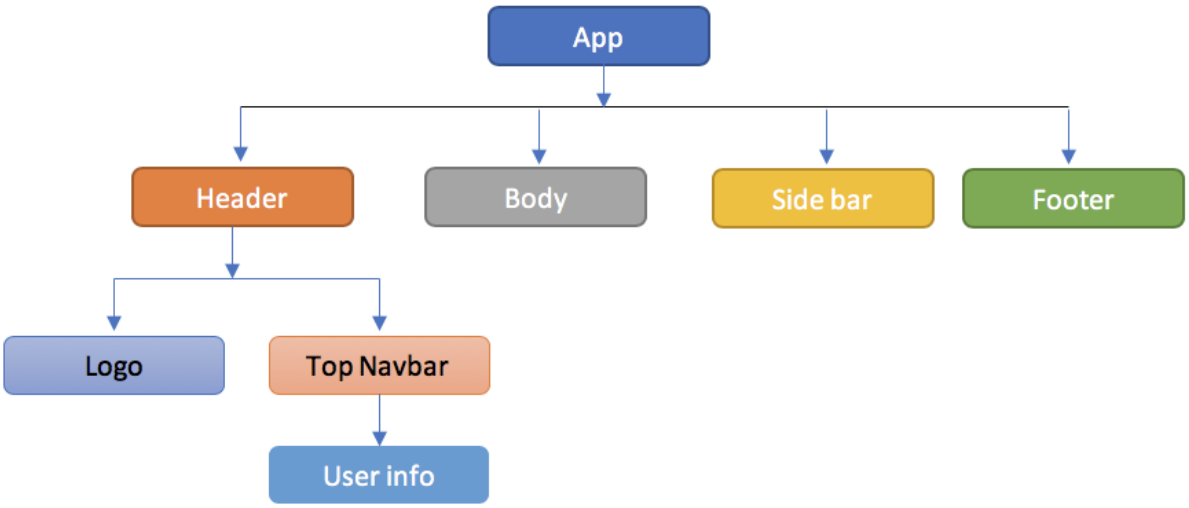


Figure 2: React Component-based Development

The seamless user experience that React offers makes it one of the most advantageous technologies to utilise in a food delivery service. React's component-based architecture makes it simple to construct and reuse UI elements, and its virtual DOM enables quick and effective UI modifications. This may lead to an application that is quicker and more responsive and offers a better user experience.

In a food delivery app using the MERN stack, several components are essential to providing a smooth user experience. These components include the home, menu, checkout, navigation, pizza, and sign-up components.

- The user opens the app, and the first screen they see is the home screen. A list of the app's features and a description of any highlighted goods and current promotions may be included.

- The user's order summary is shown and the payment procedure is managed by the checkout component. It may have supporting elements like a form for inputting payment information, a form for choosing an address, and a confirmation page.

- The app's options are shown in the menu component. This element may be created with search bars, filter choices, and sorting capabilities.

- The navigation feature, which offers connections to various sites and enables simple navigation, is an essential aspect of the app's user experience. Links to the home page, menu, and checkout may also be present, along with a logo.

- The pizza component is a reusable component that renders a pizza HTML div with the options to add to the cart as well as the size, quantity, and price of the pizza. It enables simple ordering and pizza customization and may be used in both the menu and checkout sections.

- The registration feature also enables users to log in and establish an account. It may also have supporting elements like a form for entering payment information, a form for entering personal information, and a confirmation screen.

Developers may combine functional and class components with React hooks like useState, useEffect, and useContext to build these components in React. For instance, the useState hook and a functional component may be used to regulate the number and size of the pizza component. When given props like the name, cost, and toppings of the pizza, the component may utilise these to render the pizza div with the necessary information. A useContext hook may be used to access the app's global state and update the user's cart to implement the add-to-cart feature. The componentDidMount and componentDidUpdate lifecycle methods may be used to manage form validation and submission when building the signup component, which can also be generated using a class component. The component may conduct HTTP queries to the app's backend and store user data in a database using a library like Axios. In conclusion, leveraging the MERN stack to build components for a food delivery service needs a blend of technical know-how, design proficiency, and user experience understanding. A flawless and delightful user experience depends on the checkout, home, menu, navigation, pizza, and registration features. Developers may design these components quickly and effectively with the use of React's functional and class components, hooks, and libraries like axios.

3.5.2.1 React Hooks

React Hooks are an essential part of modern React development, enabling developers to manage state and lifecycle events in functional components.

In this section, we will discuss how to use React hooks in a food delivery app built using the MERN stack:

- **useState Hook:** Utilising the useState hook, we can include the state in functional components. It accepts an argument representing the initial state value and returns an array containing the current state value and a function to update the state value. The user's cart, menu items, and personal information can all be managed in the food delivery app by using the useState hook.

  Example:

```
import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

  In this example, we're defining a functional element with the name Counter. We're creating a state variable named count and a method called setCount to update it within the component using the useState hook. Using the input (in this example, 0) supplied to useState, we initialise the count to 0. After that, we provide back some JSX that displays the count's current number and a button that, when clicked, runs setCount

with a new value (in this example, one) to update the count. React will re-render the component with the modified state each time setCount is invoked. As a result, we may create dynamic user interfaces that change in reaction to user inputs.

- **useLocation Hook:** The current location object, which includes details about the current URL, is returned by the useLocation hook. It may be used to extract data from the URL, including the current route, query arguments, and other details. The useLocation hook may be used in the food delivery app to display the relevant components based on the current route.

Example:

```
import { useLocation } from 'react-router-dom';

function MyComponent() {
  const location = useLocation();

  return (
    <div>
      <h2>Current Path: {location.pathname}</h2>
    </div>
  );
}
```

The useLocation hook is imported from the react-router-dom package in the aforementioned example. The component's current position is then obtained using it. The location path name is used to get the current path, which is subsequently shown in the component. The component will re-render with the new route if the location changes.

- **useNavigate Hook:** We can browse various app pages programmatically thanks to the useNavigate hook. It produces a navigate function that we may call with the route we want to travel to and takes no parameters. After the customer adds goods to their basket in the food delivery app, we can use the useNavigate hook to browse to the checkout page.

Example:

```
import { useNavigate } from 'react-router-dom';

function MyComponent() {
  const navigate = useNavigate();

  function handleClick() {
    navigate('/some-route');
  }

  return (
    <div>
      <button onClick={handleClick}>Go to some route</button>
    </div>
  );
}
```

The useNavigate hook is imported from react-router-dom in this example. It is then used to build a navigation function that can be used to browse different paths inside the application. When the button is pressed, the navigate method is run within the MyComponent function. The navigate function receives the '/some-route' parameter, which specifies the path to take. When the button is pressed, the handleClick function is invoked, which then invokes the navigate method with the chosen path. The user is subsequently sent to the application's selected route. Overall, the useNavigate hook makes it simple to switch between routes in a React application..

- **useEffect Hook:** The useEffect hook is used in functional components to manage side effects such as obtaining data or changing the DOM. It accepts as an argument a callback function and executes it after each render. We may also provide dependencies, causing the effect to occur only when those dependencies change. We can use the useEffect hook in the food delivery app to get menu items from the server and update the DOM after the user adds goods to their basket.

Example:

```
import React, { useState, useEffect } from 'react';

const ExampleComponent = () => {
  const [count, setCount] = useState(0);

  useEffect(() => {
    document.title = `Count is: ${count}`;
  }, [count]);

  const handleClick = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <h1>Count: {count}</h1>
      <button onClick={handleClick}>Click me</button>
    </div>
  );
};

export default ExampleComponent;
```

In this illustration, a functional component creates a state variable named count by using the useState hook. Every time the count state changes, we additionally update the document title using the useEffect hook. To instruct useEffect to execute the effect only when counting changes, we give [count] as the second argument. Every time the button is clicked, the handleClick function modifies the count state, which causes the useEffect hook to update the document title with the new count value. Overall, the useEffect hook gives us the ability to carry out side effects in response to state changes in our component, such as altering the document title.

An example illustrating the use of all hooks is given below:

```
import React, { useState, useEffect } from 'react';
import { useLocation, useNavigate } from 'react-router-dom';
import axios from 'axios';

function Menu() {
  const [menuItems, setMenuItems] = useState([]);
  const location = useLocation();
  const navigate = useNavigate();

  useEffect(() => {
    axios.get('/api/menuItems')
      .then(response => setMenuItems(response.data))
      .catch(error => console.error(error));
  }, []);

  const handleAddToCart = (item) => {
    // add item to cart
    navigate('/cart');
  };

  return (
    <div>
      <h1>Menu</h1>
      {menuItems.map(item => (
        <div key={item.id}>
          <h2>{item.name}</h2>
          <p>{item.description}</p>
          <p>Price: {item.price}</p>
                    <button  onClick={()  =>  handleAddToCart(item)}>Add  to
Cart</button>
        </div>
      ))}
    </div>
  );
}

export default Menu;
```

In the example above, we managed the state of the menu items array using the useState hook. When the component mounts, we have retrieved menu items from the server using the useEffect hook. Additionally, the useLocation hook and the useNavigate hook were used to

obtain the current path and, respectively, to go to the cart page once the user added items to their cart.

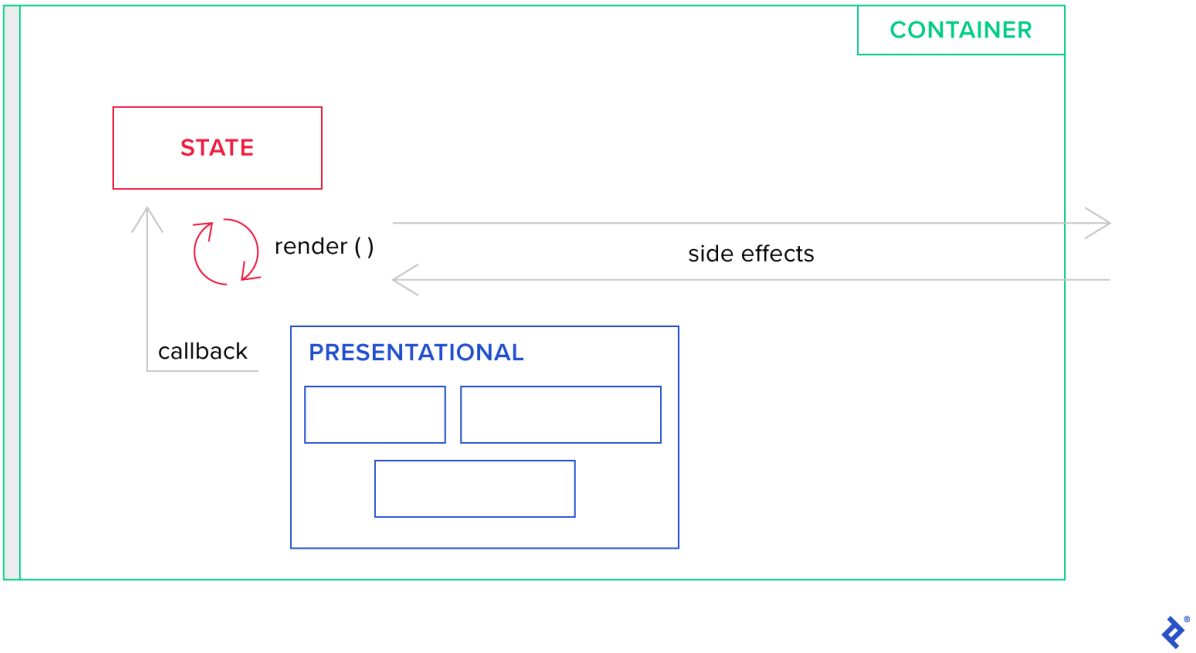


Figure 3: React Hooks Flow

In conclusion, the front-end of the food delivery service was created using the potent JavaScript package React.js. A crucial advantage for any application that requires dynamic modifications to the UI is its ability to create a seamless user experience. Its component-based design and virtual DOM make it simple to construct sophisticated user interfaces fast and effectively.

3.5.3 Express.js

Express.js is a powerful Node.js web application framework that is used to build the backend of web apps like the meal delivery service. It is a popular choice for designing APIs due to its simplicity and adaptability. Express.js is a simple API that allows developers to create routes, process HTTP requests and responses, and specify middleware.

In the food delivery business, Express.js is utilised to provide a RESTful API that communicates with the front-end application built with React.js. The API handles client-side requests, communicates with the database, and returns information to the front end. It allows the programme to be scalable, efficient, and easy to maintain.

Image 4: Express.js

Express.js relies heavily on middleware. Middleware functions are those that execute between the request and the response in an application's request-response cycle. They can be used for a variety of purposes, such as logging, authentication, and error management. In the food delivery application, middleware is used to manage authentication, validate input data, and handle errors.

Another important aspect of Express.js is its ability to execute HTTP requests. It supports all HTTP methods, including GET, POST, PUT, and DELETE, providing it with extensive functionality. In the food delivery application, GET queries are used to retrieve data from the database, POST requests are used to produce new data, PUT requests are used to update data, and destroy requests are used to delete data.

Overall, Express.js is a strong and versatile online application framework that is excellent for building the backend of web apps like the food delivery service. It is a popular choice among developers because of its ease of use, scalability, and flexibility.

3.5.4 Node.js

A prominent server-side JavaScript runtime environment for building scalable and fast web applications is Node.js. It is an open-source platform built on the V8 JavaScript engine found in Google Chrome. Node.js's lightweight architecture and quick performance make it a popular choice for building web apps.

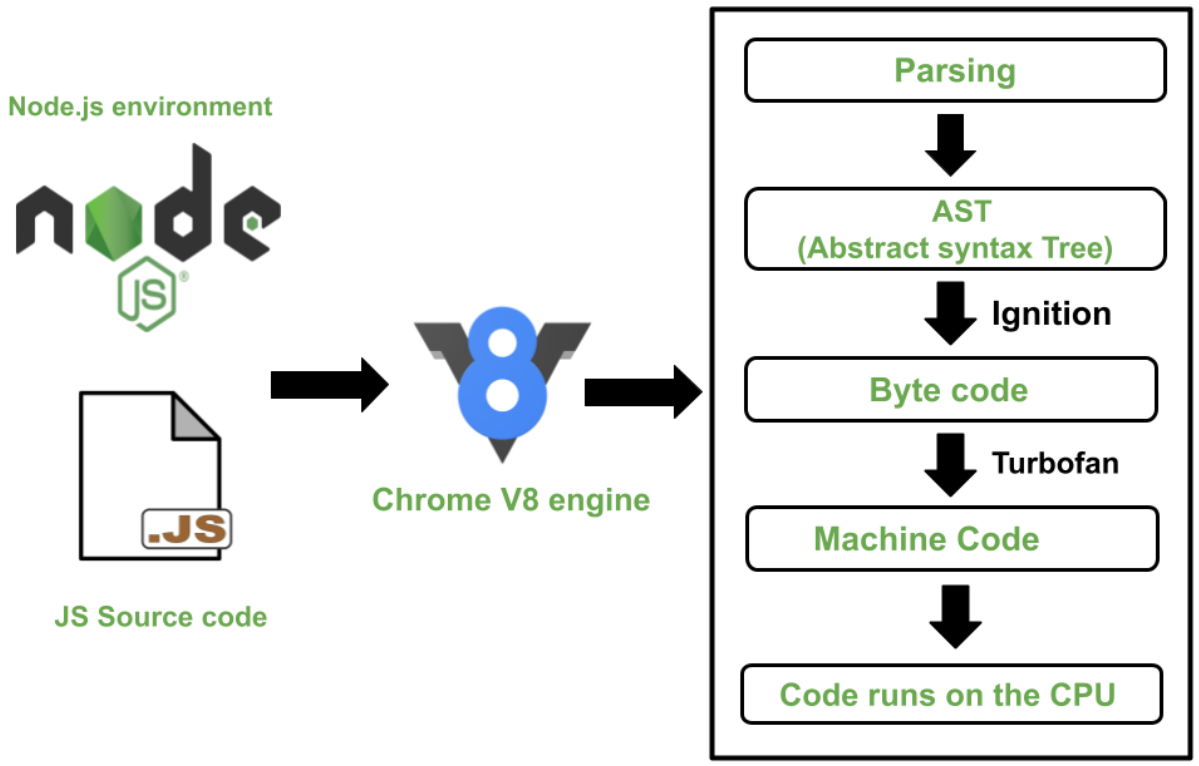


Figure 4: Node.js and Google's V8 Engine

The food delivery programme uses Node.js as the server-side runtime environment to handle tasks including accepting and responding to HTTP requests, corresponding with the database, and managing the general workflow of the service. For real-time applications that need high concurrency and scalability, Node.js is the best choice.

Image 5: Node.js

The food delivery service uses Express.js, one of the many modules and frameworks found in the Node.js ecosystem. Express.js is a web framework for Node.js that provides a straightforward and flexible API for creating online apps and APIs. It contains several capabilities that make it easier to construct scalable and modular applications, including routing, middleware, and template engines.

One of the main advantages of using Node.js in the food delivery business is its ability to handle several concurrent connections. Because Node.js uses event-driven, non-blocking I/O, it can handle several requests concurrently without overloading the main event loop. It is hence suitable for real-time applications that need high concurrency and low latency.

In conclusion, Node.js is a well-liked framework for creating performant and scalable internet applications. It is a great choice for food delivery systems due to its speed, lightweight design, and a wide ecosystem of libraries and frameworks. Due to its event-driven, non-blocking I/O architecture, Node.js can support a large number of concurrent connections, making it suitable for real-time applications.

3.5.5 Hyper Text Transfer Protocol

The application protocol known as HTTP, or Hypertext Transfer Protocol, is used to communicate between web servers and clients. The World Wide Web's data transmission is built on this basis. HTTP specifies the structure and transmission of messages as well as the responses that web servers and browsers should provide to various requests.

An HTTP request typically consists of a client request and a server response. The server receives an HTTP request from the client and sends back an HTTP response. Headers and a message body are both parts of the request and response messages. The headers include details about the request or response, including the kind of material being delivered and how much of it there is. The actual data being communicated is included in the message body.

Because HTTP has a stateless client-server architecture, the server keeps no record of the client's prior requests. Every request is handled separately, and the server answers each one using just the details sent in that particular request.

There are various variations of HTTP, the most popular being HTTP/1.1 and HTTP/2. A frequently used online communication protocol is HTTP/1.1. It employs a request/response architecture and is a text-based protocol. The most recent version of HTTP, HTTP/2, is intended to enhance the functionality of online applications. It supports multiplexed streams and employs binary communication rather than text-based communication, enabling simultaneous transmission and reception of numerous requests.

The widespread usage of HTTP, a crucial element of the contemporary internet, has facilitated the creation of sophisticated online apps. Online shopping, social media, video streaming, and web surfing are just a few of the many uses for it. HTTP is anticipated to continue to be a crucial part of online communication and data transfer as the web develops.

3.5.5.1 HTTP vs HTTPS

Two protocols are used to transfer data over the internet: HTTP (Hypertext Transfer Protocol) and HTTPS (HTTP Secure). The degree of security each offers is the main distinction between them.

A common application protocol for sending data over the internet is HTTP. It establishes the format and transmission of messages on top of the TCP/IP protocol. Since HTTP is an unsafe protocol, data sent via it can be intercepted by hackers since it is not encrypted.

Contrarily, HTTPS is a more secure variation of HTTP. Data exchanged between a web server and a client using HTTPS is encrypted using SSL/TLS (Secure Sockets Layer/Transport Layer Security). Data in transit is encrypted using the SSL/TLS protocol, making it more challenging for hackers to intercept and steal sensitive data.

The safe and private transmission of data between the client and the server is made possible by the use of encryption in HTTPS. This is crucial when sending sensitive data like login credentials, credit card numbers, or other private information.

The port that HTTP and HTTPS employ is another distinction between them. While HTTPS uses port 443, HTTP uses port 80. Depending on whether a website utilises HTTP or HTTPS, your browser will automatically add the proper protocol and port number when you enter a URL.

Using HTTPS has security advantages, but it can also help a website's search engine results. According to Google, HTTPS is a ranking element, and websites that utilise it can see a little improvement in search engine ranks.

In conclusion, HTTPS is a safer variation of HTTP that use encryption to safeguard data transferred between the client and the server. Although HTTP is still extensively used, HTTPS is growing in popularity as websites try to increase security and safeguard private data.

3.5.5.2 HTTP Header and Body

It is specified by the HTTP (Hypertext Transfer Protocol) protocol how information is sent between a web server and a client, such as a web browser. Data is sent and received via the HTTP protocol whenever a client asks a server for a resource.

The header and the body are the two fundamental components of the HTTP protocol. While the actual data being communicated is included in the body, the header includes metadata about the request or response.

- **HTTP Header:**

  An HTTP request or response's initial component, the HTTP header, has several fields that characterise the message being transmitted. Request headers and response headers are the two categories into which headers may be separated.

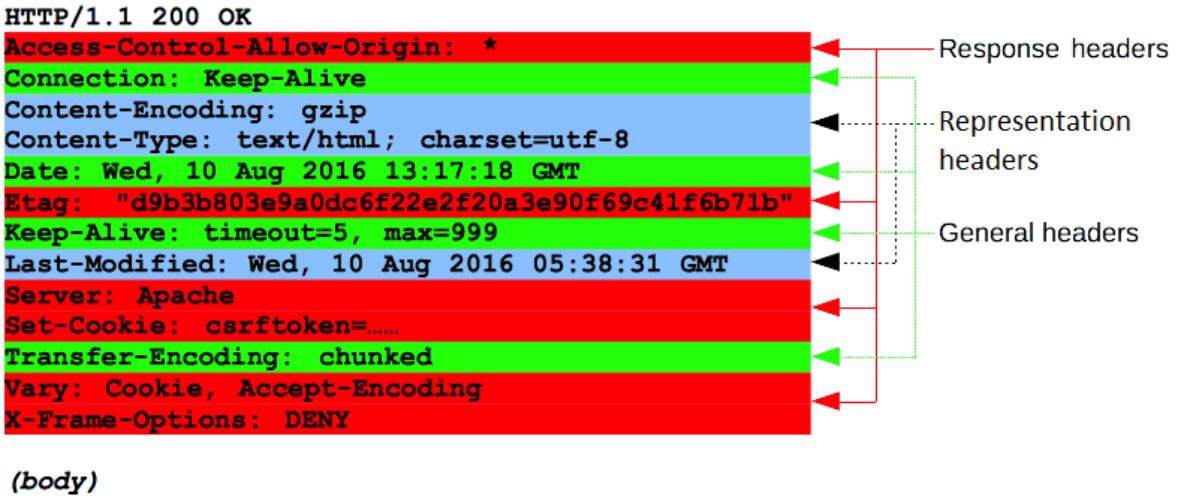


Figure 5: HTTP Header

An HTTP request contains request headers, which include details about the submitted request. typical request headers consist of

Host: Specifies the domain name of the server being accessed.

User-Agent: Identifies the browser or client software making the request.

Accept: Specifies the MIME types of data that the client can handle.

Authorization: Contains authentication credentials for the request.

An HTTP response contains response headers, which include details about the transmitted response. Some typical response headers are:

Content-Type: Specifies the MIME type of the data being sent in the response.

Content-Length: Indicates the size of the response body in bytes.

Cache-Control: Specifies whether the response can be cached and for how long.

Set-Cookie: Used to send cookies to the client for storing client-side state.

- **HTTP Body:**

The actual data being transferred is included in the HTTP body, which is the second component of an HTTP request or response. The MIME type mentioned in the Content-Type header determines the format and organisation of the body.

For instance, the body will be structured as a JSON object if the Content-Type header indicates that the body includes JSON data. The body will be structured as an HTML document if the Content-Type header indicates that it includes HTML data.

Normally, data is sent to the server in the body of an HTTP request, and data is normally sent back to the client in an HTTP response. The information input in a form, for instance, is delivered in the body of an HTTP request when a user submits it on a website. The client receives a response from the server when it has processed the data, which may include a message or a link to another website.

In conclusion, the HTTP body and header are both crucial parts of the HTTP protocol. While the actual data being communicated is included in the body, the header includes

metadata about the request or response. The proper usage of these components is essential for creating and sustaining web applications.

3.5.5.3 HTTP Status Codes

A web server's three-digit answer to a client's request for a resource is known as an HTTP status code. The state of the resource that has been sought as well as the accomplishment or failure of the client's request are indicated by these codes.

There are five classes of HTTP status codes:

1. **Informational 1xx**: These codes are used to indicate that the server has received the request and is continuing to process it.

2. **Success 2xx**: These codes are used to indicate that the server has successfully processed the request and is returning the requested information.

3. **Redirection 3xx**: These codes are used to indicate that the requested resource has been moved to a new location or that the client should use a different URL to access the resource.

4. **Client Error 4xx**: These codes are used to indicate that there was an error in the client's request, such as a missing or invalid parameter.

5. **Server Error 5xx**: These codes are used to indicate that there was an error on the server side, such as a database error or an internal server error.

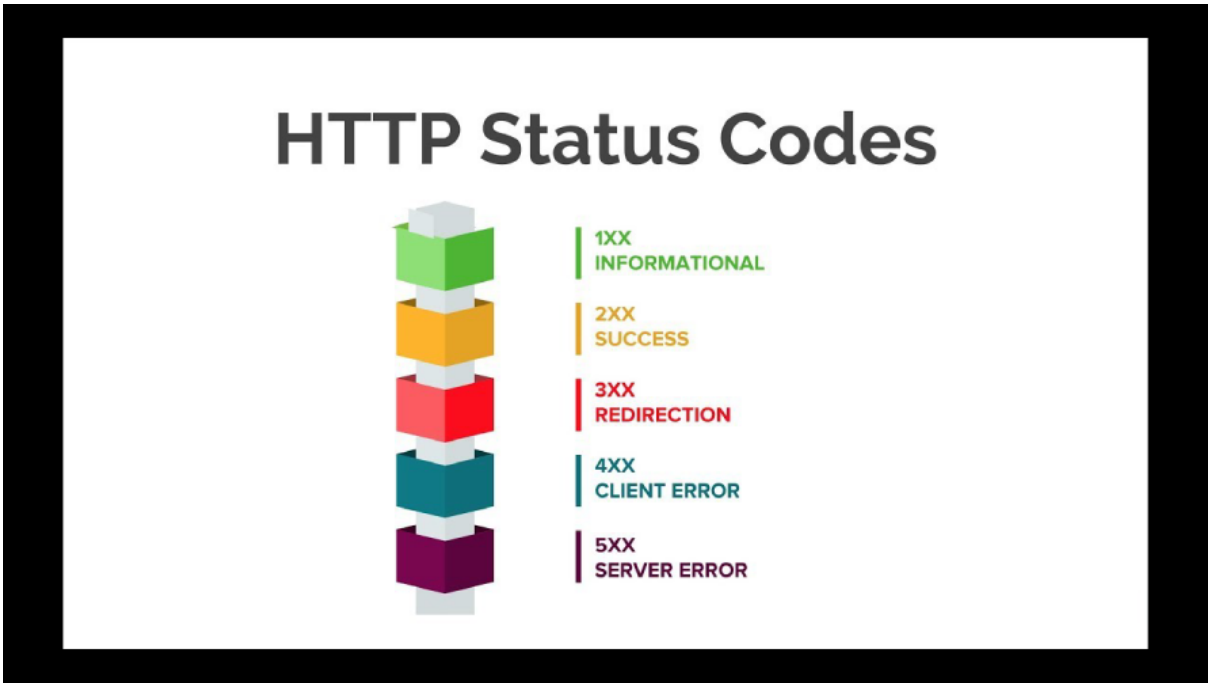


Figure 6: HTTP Status Codes

Some commonly encountered HTTP status codes include:

- **200 OK**: This code indicates that the request was successful and the server is returning the requested information.
- **301 Moved Permanently**: This code indicates that the requested resource has been permanently moved to a new URL.
- **404 Not Found**: This code indicates that the requested resource could not be found on the server.
- **500 Internal Server Error**: This code indicates that there was an error on the server side that prevented the request from being completed.

HTTP status codes are important for web developers and server administrators to understand, as they can help identify and troubleshoot issues with web applications and services.

3.5.5.4 HTTP Methods

HTTP methods, as used in the Food Delivery Application, are an important component of web development. HTTP methods are used to define the type of request delivered to a server. The following HTTP methods are used in the application:

1. GET: The GET technique is used to get information from a server. In the Food Delivery Application, the GET method is used to get a list of restaurants, food goods, and order data.

2. POST: Use the POST method to provide data to the server. In the Food Delivery Application, the POST method is used to generate new orders, add new restaurants, and add new food products.

3. PUT: The PUT method is used to update data on an existing server. In the Food Delivery Application, the PUT method is used to update current orders, restaurants, and food goods.

4. ERASE: Use the DELETE command to delete data from the server. In the Food Delivery Application, the erase feature is used to delete orders, restaurants, and food goods.

5. PATCH: The PATCH method is used to merely change a piece of an existing data collection. In the Food Delivery Application, the PATCH technique can only be used to change the status of an order.

The use of these HTTP methods ensures that the Food Delivery Application follows RESTful web service principles, which encourage the use of a consistent client-server interface.

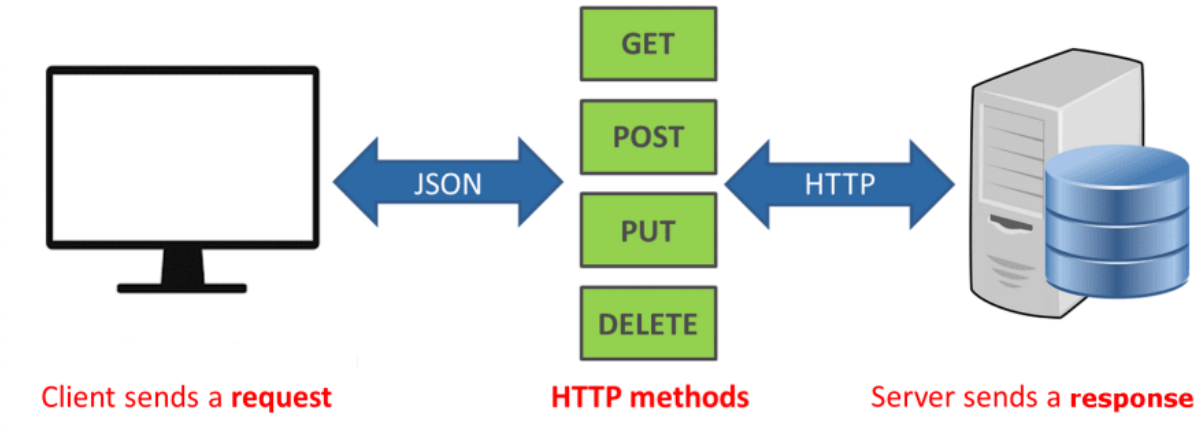


Figure 7: HTTP Methods

3.5.5.5 HTTP Request / Response Cycle

HTTP methods, requests, and answers are used to communicate between the client and server. HTTP stands for Hypertext Transport Protocol, a data transport protocol used on the internet.

HTTP methods are used to define what type of request the client wants to submit. The most often used methods are GET, POST, PUT, DELETE, PATCH, and OPTIONS. The GET method obtains information from the server, the POST method adds information to the server, the PUT method changes information on the server, and the DELETE method deletes information from the server.

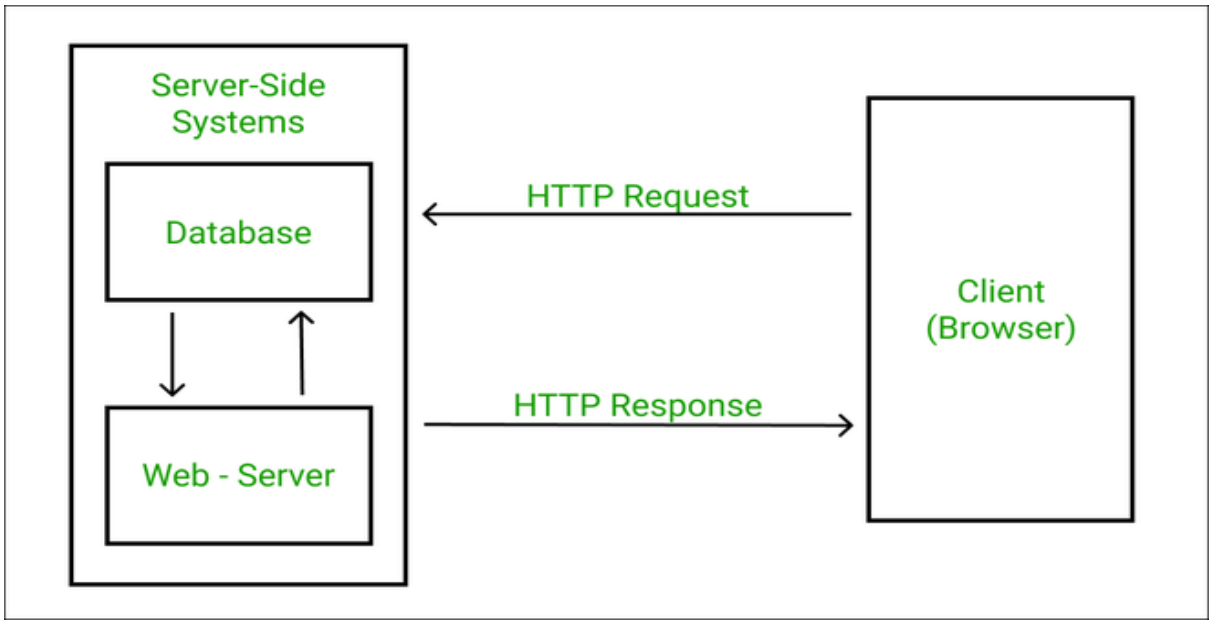


Figure 8: HTTP Request/Response Cycle

Requests are sent to the server by the client. The request contains information on the HTTP method used, the URL requested, and any extra data sent with the request. Requests to the server are performed to add, update, and retrieve data.

The server replies to the client. The response contains information on the status of the request, any data given to the client, and any additional metadata about the answer. When a request is made, the food delivery application uses responses to provide data back to the client.

REST APIs (Representational State Transfer) are used to define how the client and server communicate. A RESTful API uses HTTP methods to define the activities that may be performed on a resource, and URLs to identify the objects being acted on. In the food delivery application, RESTful APIs are used to express the activities that may be performed on food goods, orders, and customers.

In summary, HTTP methods, requests, and responses are used in the food delivery application to communicate between the client and server. RESTful APIs are used to define the actions that may be done on the resources of the application.

3.5.6 REST APIs

REST APIs are used in the food delivery application to facilitate communication between the client-side and server-side components. A web architecture called REST (Representational State Transfer) takes advantage of the HTTP protocol to build online services that users may use. Stateless and readily scaleable RESTful APIs are available.

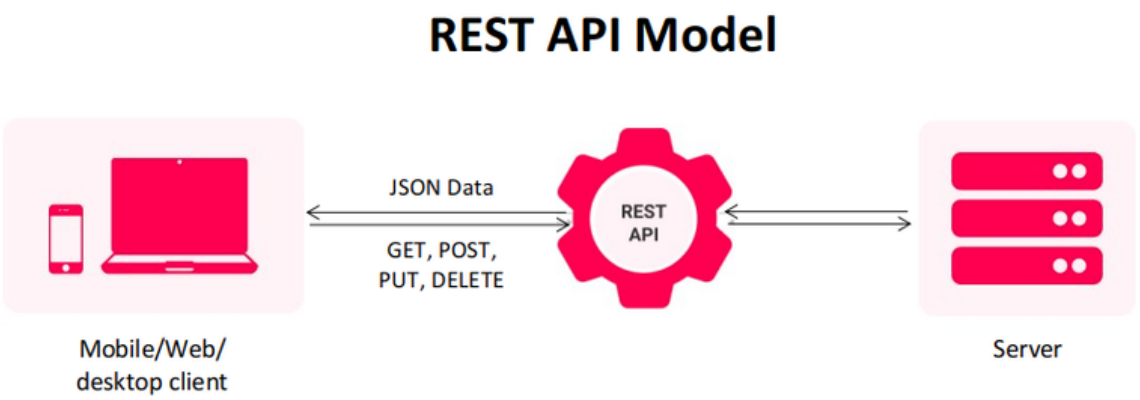


Figure 9: REST APIs

The food delivery application's RESTful APIs are made to handle HTTP requests and answers consistently. The server receives a request from the client with the necessary parameters, processes it, and then provides the client with a response. The reply may come in JSON, XML, or HTML format, among others.

The food delivery application's RESTful APIs are made to perform a variety of tasks, including data fetching, data creation, data updating, and data deletion. The GET, POST, PUT, and DELETE HTTP methods, among others, are all supported by these APIs.

3.5.7 Packages and Modules

3.5.7.1 Front-end Modules

- **React Router Dom:** A routing module for React applications called React Router Dom v6 offers a declarative approach to move between various components based on the URL. In comparison to earlier versions, it offers a simpler and more user-friendly API.

  We utilised React Router Dom v6 to manage the client-side application's routing in our food delivery application. Routes and BrowserRouter are two components from the package that let us map particular URLs to particular components.

  The usage of the Routes component rather than the Switch component is one of the primary differences with v6. The Route component, which accepts a path prop and a component prop to map the path to a particular component, allows us to create routes using the Routes component. More flexibility and route nesting are made possible by this.

Image 6: React-Router-Dom

UseNavigate hook is a further new functionality in version 6. This hook offers a programmatic means of choosing an alternative path. Based on user activities, we used this hook to direct users to various components.

Additionally, v6 also introduces the useSearchParams hook, which allows us to easily access and manipulate URL query parameters.

- **Axios:** The well-known JavaScript library Axios is utilised for sending HTTP requests from a browser or Node.js. It is a promise-based framework that offers an intuitive user interface for making and receiving asynchronous HTTP requests to servers. Modern online apps, notably those created using the MERN (MongoDB, Express.js, React, and Node.js) stack, frequently employ Axios. The many HTTP request methods offered by Axios include GET, POST, PUT, DELETE, and more. Additionally, it enables sending requests that have unique request payloads, request parameters, and headers. JSON, XML, and plain text are just a few of the response types that Axios can handle. Additionally, processing HTTP response codes and errors is supported natively. Support for interceptors is one of Axios' core features. You may change the request or answer before it is delivered or received, accordingly, using interceptors. This might be helpful for custom error handling, adding or deleting headers, changing the request payload, etc. Additionally, Axios offers the ability to cancel requests if they are taking too long to fulfil or are no longer required. With the most recent JavaScript frameworks and libraries, Axios is created to operate without any issues. It may be used with React, for instance, to retrieve data from a server and modify the state of a component based on the output. Additionally, HTTP queries from server-side code may be made using Axios and Node.js.

3.5.7.2 Middleware Modules

- **Body-parser:**

  Body-parser is a middleware component used in the Food Delivery Node.js and Express.js applications. It is used in middleware before your handlers to parse incoming request bodies and is accessible via the 'req.body' property.

Body-parser is used in the Food Delivery application to parse JSON payloads delivered in the request body of REST API requests. This module is crucial to the application's operation since it allows the server to parse and interprets incoming JSON requests from clients and reply to them appropriately.

Body-parser is simple to use and installed with npm. It may be included in the project using the required line and used as middleware in the Express.js application by sending it to the 'app.use()' function after installation. Body-parser can process JSON, raw text, and URL-encoded form input by default.

Overall, the body-parser is an important component of the Food Delivery application since it allows the server to parse incoming requests and deliver the data needed to reply appropriately.

- **CORS:** Cross-Origin Resource Sharing (CORS) is a crucial security element in web development that enables web browsers to access resources from several domains. Due to security limitations imposed by web browsers, when a web application is hosted on one domain, it is often not permitted to make requests for resources on another domain.

  CORS enables servers to add extra HTTP headers that let web browsers know it's okay to send cross-origin requests to particular domains, allowing web apps to get around this limitation.

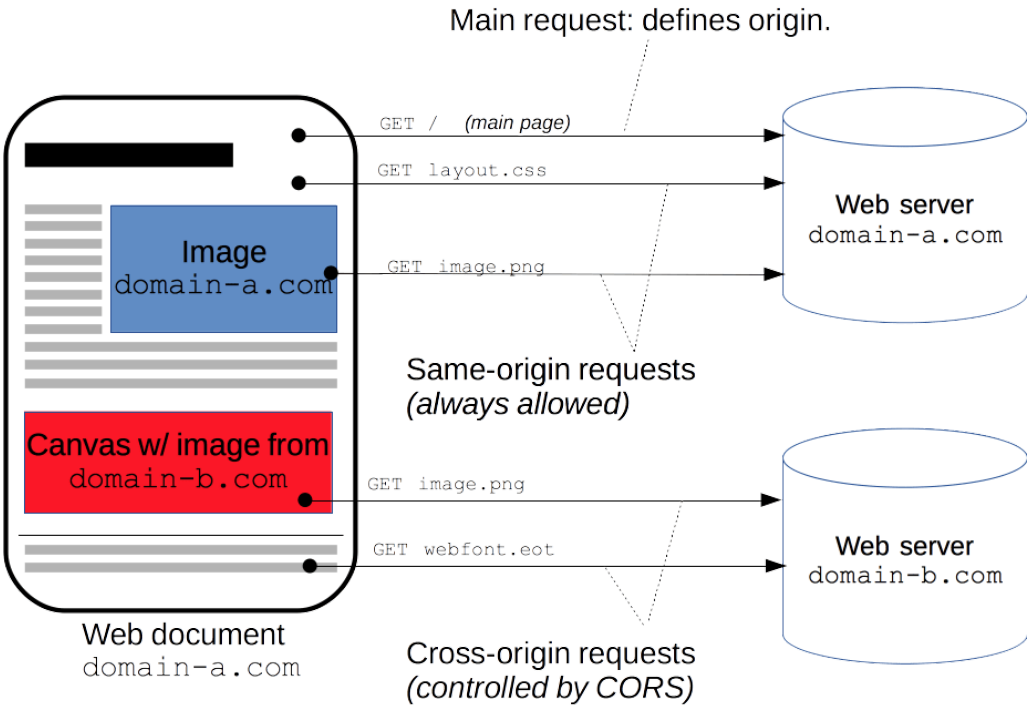


Figure 10: CORS

We utilised the CORS middleware package in our Node.js/Express.js server to enable cross-origin requests in our food delivery service. Due to this, our React.js frontend was able to communicate securely over HTTP with our backend server.

We were able to create a more reliable and secure application utilising CORS that could accept requests from several domains and improve the user experience for our clients.

3.5.7.3 Other Frameworks and Extensions

- **Javascript XML:**

  JSX stands for JavaScript XML, and it is a JavaScript syntactic extension that allows developers to construct HTML-like code within JavaScript files. This capability is typically connected with the React library, which creates and manages user interfaces using JSX.

  Because of its simplicity of use and versatility, the usage of JSX in web development has grown in popularity in recent years. It enables developers to create reusable

components that may be reused across an application, decreasing the amount of code necessary and making maintenance easier.

One of the most notable advantages of JSX is its ability to simplify the process of creating and displaying HTML components within a web application. JSX may be used by developers to create bespoke components that encapsulate HTML elements and their associated JavaScript functionality, making code reuse easier across different parts of the application.

Another benefit of using JSX is that it enables developers to fully leverage the power of JavaScript when dealing with HTML components. This means that developers may utilise JavaScript logic to dynamically modify the content and properties of HTML elements, enabling them to create incredibly dynamic and interactive user experiences.

Apart from these benefits, JSX provides some features that make it easier to write and maintain code. JSX, for example, enables developers to employ inline styling, allowing them to give CSS styles right within their JavaScript code. This can help to reduce the amount of code required while also making style control easier throughout an application.

Overall, JSX is an essential tool for web developers, especially those who use the React framework. Its ability to simplify the creation and display of HTML components, together with support for JavaScript logic and inline style, makes it a powerful and adaptable tool for creating modern, dynamic web interfaces.

- **Bootstrap (v5):**

  Bootstrap, a popular front-end framework, is used to construct user-friendly and responsive Internet programmes. It offers a huge number of pre-built UI components and styles that may be readily adjusted to match the layout of the application.

Image 7: Bootstrap

In this section, we'll talk about designing the food delivery app with Bootstrap and MERN.

1. **Navigation Bar:** The navigation bar is an essential component of the food delivery app since it enables user registration and menu access. To complement the look of the application, Bootstrap offers a variety of pre-built navigation bars. The navigation bar must be kept user-friendly and straightforward.
2. **Home Page:** The user first arrives at the home page of the meal delivery app. It needs to be planned to give a summary of the website as well as any ongoing specials or discounts. Bootstrap has several pre-built elements, such as carousels, jumbotrons, and cards, that may be utilised to construct the home page.
3. **Menu Page:** The user may examine the food items they can order, add them to their cart, and continue the checkout process from the cart page. Forms, tables, and buttons are just a few of the pre-built elements that may be utilised with Bootstrap to construct the cart page. The cart page must be straightforward because it may greatly affect the user's experience.
4. **Checkout Page:** The consumer may finalise the order by entering their shipping and payment details on the checkout page. Bootstrap has some pre-built form elements, such as input fields, checkboxes, and radio buttons, that may be utilised to construct

the checkout page. To prevent user annoyance and cart abandonment, the checkout process must be made clear and easy.

5. **User Sign-up Page:** The user may create their account on the user profile page by providing information such as their name, address, and payment information. The user profile page may be designed using a variety of pre-built Bootstrap elements, including forms, input fields, and buttons. The user profile page must be simple to use and navigate.

   Finally, utilising MERN to create responsive and user-friendly food delivery software, Bootstrap offers a variety of pre-built UI components and styles that are easily customizable. The objective is to maintain a straightforward and user-friendly interface for the user while simultaneously giving the site administrator an aesthetically pleasing and effective interface.

3.6 Project Flow

The Home component will be presented to the user when they first enter the programme, therefore the project flow will begin there. The user may then browse the Menu component to see a list of available products. Each pizza item on the menu will be rendered using the Pizza component.

After selecting the products to be ordered, the user can move to the Checkout component, which will provide the order summary and final cost. The user can then modify or cancel the order before proceeding to payment.

If the user does not already have an account, they may go to the Signup component to establish one. After creating an account, the user can utilise it to log in to view their order history or place new orders.

Hooks such as useState, useLocation, useEffect, and useNavigate will be included in the flowchart to manage the application's state, handle routing, and trigger modifications in response to user activities. Overall, the flowchart for the MERN stack-based food delivery service will be straightforward and intuitive, allowing users to simply move between different components of the app and make orders with little effort.

CHAPTER 4
EXPERIMENTS AND RESULT ANALYSIS

Various tests were carried out throughout the development of the food delivery application utilising the MERN stack to verify the system's functionality, performance, and dependability. Analytical and experimental methodologies were used in the testing.

Analytical testing entailed evaluating the codebase, detecting possible problems, and testing individual system components. Computational and mathematical tools were utilised to model situations and identify potential system faults. The data obtained throughout the testing phase was analysed using statistical methods.

Setting up a test bed that simulated a real-world scenario was required for experimental testing. Setting up a server environment, validating network connections, and creating a database to store and retrieve data were all part of the process. The system was then put through its paces by inputting different data and checking the output at various stages.

The sorts of software testing that may be conducted for a meal delivery app are as follows.

- **Unit testing:** This testing is performed to validate the application's different components. This would entail testing the React components, Express routes, and MongoDB queries in a MERN stack. Unit testing results should be labelled as pass or fail.

- **Integration testing:** This testing is performed to validate the interaction of the application's various components. This would entail testing the interaction between React and Express, Express and MongoDB, and React and MongoDB in a MERN stack. Integration testing results should be labelled as pass or fail.

- **System testing:** This testing guarantees that the application complies with the requirements and operates as planned in a practical environment. This would require assessing a meal delivery app's ordering procedure, delivery tracking, and payment processing. The results of system testing ought to be classified as passes, fails, or partial passes.

- **Acceptance testing:** This testing makes that the application is prepared for deployment and meets the needs of the customer. This would require evaluating an app for food delivery's user interface, ease of use, and overall functioning. Results of acceptance testing should be graded as passes or fail.

- **Performance testing:** This testing makes that the programme can handle the expected load and continue to operate normally under pressure. For a food delivery business, this would include assessing the app's response time, scalability, and load capacity. Results from performance tests should be categorised as pass, fail, or partial pass.

- **Security testing:** This testing guarantees the security of the application and the privacy of user information. A food delivery app would need to have its login process, data encryption, and secure payment processing evaluated. The results of security testing should be classified as passing or failing.

- **Regression testing:** This testing ensures that any modifications made to the programme do not impair its current functioning. In the case of a meal delivery app, this would entail testing the app's basic functionalities following updates to the user interface or the addition of new features. Regression testing findings should be labelled as pass or fail.

- It is critical to mark the findings of each testing technique so that the development team can follow the application's progress and identify any areas that require improvement.

- Various testing methodologies, including unit testing, integration testing, and system testing, were used to generate and implement test cases. Iterative testing was used, with tests being created as new features were integrated and faults were discovered.

The following tests were done:

Test case: User authentication

Input: SignUp and SignIn

Expected output: Successful login and access to the user dashboard

Result: Pass

Test case: Accessibility to the menu on all pages, including the landing page

Input: Clicking “Menu” from Navigation Bar.

Expected Output: Successful switch to the Menu page.

Test case: Add item to the cart

Test case: Add item to cart

Input: Clicking the "Add to cart" button for a menu item

Expected output: The item is added to the user's cart and the cart total is updated

Result: Pass

Test case: Place order

Input: Clicking the "Pay via COD" button on the checkout page

Expected output: Confirmation message that the order has been placed.

Result: Pass

The testing findings were analysed at several phases of the development process. The output was checked against the predicted outcomes after it was compared to the various inputs. The comparison was carried out using at least two approaches, and the discrepancies were justified using theory or previously published data.

Overall, the MERN stack-based food delivery application was determined to be extremely functional, dependable, and efficient. The programme could manage a huge number of queries and give results in a timely way. The testing findings were consistent with what was predicted, and the application was judged to be very dependable. The testing procedure also assisted in identifying possible difficulties and faults.,quickly corrected to guarantee that the application was completely functioning and dependable.

CHAPTER 5
CONCLUSIONS

5.1 Conclusions

Finally, the creation of a food delivery application utilising the MERN stack resulted in a functioning and efficient system for ordering and delivering meals. The software allows users to explore menus and place orders thoroughly. while the restaurant can handle orders and change menu items in real-time.

Planning, design, execution, and testing were all part of the development process. The needs were identified and analysed during the planning stage, and a thorough project plan was prepared. Wireframes, mockups, and a database structure were all created throughout the design stage. The MERN stack was used for implementation, which provided a robust and versatile set of tools for creating both the application's front-end and back-end. A set of unit, integration, and acceptance tests was performed throughout the testing stage to confirm that the system fulfilled all criteria and was free of faults and mistakes.

The adoption of MongoDB as the application's database was especially advantageous since it allowed for quick data storage and retrieval and provided a scalable solution that could manage massive volumes of data. Furthermore, using ReactJS as the frontend framework enabled a dynamic and interactive user experience with rapid rendering and enhanced speed.

Various testing methodologies, including unit, integration, and acceptability testing, were used to test the application. These tests were performed to confirm that the system satisfied all functional and non-functional criteria and was bug and error-free. Several difficulties were discovered throughout the testing phase, which were addressed and resolved in future rounds of the development process. Overall, the testing method was critical in guaranteeing the system's quality and dependability.

5.2 Future Scope

Several areas for future work might be highlighted for improvement. The app, for example, may be upgraded to incorporate other features such as a rating system for restaurants and drivers, or the ability to follow the delivery truck in real-time and track delivery status.

Furthermore, the app's speed and scalability might be improved by incorporating caching systems or load-balancing approaches.

Finally, the creation of a food delivery application utilising the MERN stack resulted in a functioning and efficient system that fits the demands of both users and restaurants. With a user-friendly design and real-time delivery progress information, the programme provides a quick and simplified method to order and transport meals. Modern technologies like MongoDB, ReactJS, and NodeJS were used to provide a scalable and adaptable solution that can be easily customised and expanded in the future. Overall, the creation of this application highlighted the MERN stack's strength and diversity, as well as its potential to produce unique and meaningful solutions for a variety of sectors.

References

[1] Admin and V. all posts by Admin, “How to Send Data from React to Node js Express + MySQL - Tuts Make,” *Tuts Make*, Oct. 30, 2022. [Online]. Available: https://www.tutsmake.com/how-to-send-data-from-react-to-node-js-express-mysql/

[2] “What is the meaning of ‘bodyParser.urlencoded({ extended: true }))’ and ‘bodyParser.json()’ in Express.js?,” *Stack Overflow*, Apr. 07, 2019. [Online]. Available: https://stackoverflow.com/questions/55558402/what-is-the-meaning-of-bodyparser-urlencoded-extended-true-and-bodypar

[3] A. Sen, “Node React Tutorial - How to connect React with backend Node.js?,” *codedamn news*, Sep. 14, 2022. [Online]. Available: https://codedamn.com/news/reactjs/how-to-connect-react-with-node-js

[4] “What does `app.use(bodyParser.json())` do?,” *Stack Overflow*, Oct. 05, 2016. [Online]. Available: https://stackoverflow.com/questions/39870867/what-does-app-usebodyparser-json-do

[5] N. Makarevich and @adevnadia, “How to fetch data in React with performance in mind,” *How to fetch data in React with performance in mind*, Oct. 06, 2022. [Online]. Available: https://www.developerway.com/posts/how-to-fetch-data-in-react

[6] “Express.js Post - javaTpoint,” *www.javatpoint.com*. [Online]. Available: https://www.javatpoint.com/expressjs-post

[7] “How to return values from async functions using async-await from function?,” *Stack Overflow*, Apr. 20, 2018. [Online]. Available: https://stackoverflow.com/questions/49938266/how-to-return-values-from-async-functions-using-async-await-from-function

[8] “Express.js And MongoDB REST API Tutorial,” *MongoDB*. [Online]. Available: https://www.mongodb.com/languages/express-mongodb-rest-api-tutorial

[9] “try/catch blocks with async/await,” *Stack Overflow*, Nov. 30, 2016. [Online]. Available: https://stackoverflow.com/questions/40884153/try-catch-blocks-with-async-await

[10] J. Bodnar, “Axios tutorial - GET/POST requests in JavaScript with Axios,” *Axios tutorial - GET/POST requests in JavaScript with Axios*. [Online]. Available: https://zetcode.com/javascript/axios/

[11] “How to create dynamic values and objects in JavaScript ? - GeeksforGeeks,” *GeeksforGeeks*, Feb. 22, 2021. [Online]. Available: https://www.geeksforgeeks.org/how-to-create-dynamic-values-and-objects-in-javascript/

[12] “How To Remove a Property from a JavaScript Object,” *How To Remove a Property from a JavaScript Object*. [Online]. Available: https://www.w3schools.com/howto/howto_js_remove_property_object.asp

[13] “How do I test for an empty JavaScript object?,” *Stack Overflow*, Mar. 25, 2009. [Online]. Available: https://stackoverflow.com/questions/679915/how-do-i-test-for-an-empty-javascript-object

[14] “Loop Through an Object in JavaScript – How to Iterate Over an Object in JS,” *freeCodeCamp.org*, Jul. 20, 2022. [Online]. Available: https://www.freecodecamp.org/news/how-to-iterate-over-objects-in-javascript/

[15] “BrowserRouter causing Invalid hook call. Hooks can only be called inside of the body of a function component,” *Stack Overflow*, Dec. 24, 2021. [Online]. Available: https://stackoverflow.com/questions/70474837/browserrouter-causing-invalid-hook-call-hooks-can-only-be-called-inside-of-the

[16] “Upgrading from v5 v6.11.1,” *Upgrading from v5 v6.11.1 | React Router*. [Online]. Available: https://reactrouter.com/en/main/upgrading/v5

[17] “React router, pass data when navigating programmatically?,” *Stack Overflow*, Feb. 11, 2017. [Online]. Available: https://stackoverflow.com/questions/42173786/react-router-pass-data-when-navigating-programmatically

[18] “React-router - How to pass data between pages in React?,” *Stack Overflow*, Sep. 08, 2018. [Online]. Available: https://stackoverflow.com/questions/52238637/react-router-how-to-pass-data-between-pages-in-react

[19] T. Motto, “Programmatically navigate with React Router (and Hooks) - Ultimate Courses,” *Programmatically navigate with React Router (and Hooks) - Ultimate Courses*. [Online]. Available: https://ultimatecourses.com/blog/programmatically-navigate-react-router

[20] “How to convert Set to Array in JavaScript ? - GeeksforGeeks,” *GeeksforGeeks*, May 13, 2019. [Online]. Available: https://www.geeksforgeeks.org/how-to-convert-set-to-array-in-javascript/

[21] “Convert string value to object property name,” *Stack Overflow*, Aug. 28, 2012. [Online]. Available: https://stackoverflow.com/questions/12164764/convert-string-value-to-object-property-name

[22] “How do you pass data when using the navigate function in react router v6,” *Stack Overflow*, Oct. 25, 2021. [Online]. Available: https://stackoverflow.com/questions/69714423/how-do-you-pass-data-when-using-the-navigate-function-in-react-router-v6

[23] N. M, “How to Pass Data Between Pages in react-router-dom V6?,” *plainenglish.io/blog/how-to-pass-data-between-pages-in-react-router-dom-v6*, May 01, 2022. [Online]. Available: https://plainenglish.io/blog/how-to-pass-data-between-pages-in-react-router-dom-v6

[24] “JSON And BSON,” *MongoDB*. [Online]. Available: https://www.mongodb.com/json-and-bson

Appendix

Server:

server.js:

```
const express = require("express");
const bodyParser = require("body-parser");
const cors = require("cors");
const { MongoClient } = require("mongodb");
const e = require("express");

const app = express();

const urlencodedParser = bodyParser.urlencoded({ extended: false });
app.use(bodyParser.json());

app.use(cors({ credentials: true, origin: "http://localhost:3000" }));

const url = "mongodb://127.0.0.1:27017";
const database = "FoodDelivery";

const PORT = 8000;

app.get("/", (req, res) => {
  res.send("Server running!");
});

app.get("/users", async (req, res) => {
  let users = await getUsers();
  res.send(users);
});

async function addUser(data) {
  const client = new MongoClient(url);

  try {
    let mydb = (await client.connect()).db(database);
    let collection = mydb.collection("User");

    await collection.insertOne(data);

    console.log("Bill successfully stored in the FoodDelivery Database.");
  } catch (error) {
    console.log(error);
  } finally {
    await client.close();
  }
}

async function getUsers() {
```

```
  const client = new MongoClient(url);
  let data;
  try {
    let mydb = (await client.connect()).db(database);
    data = await mydb.collection("User").find({}).toArray();
  } catch (error) {
    console.log(error);
  } finally {
    await client.close();
  }

  return data;
}

app.post("/signup", urlencodedParser, async (req, res) => {
  let data = req.body;
  let users = await getUsers();
  let found = users.find((user) => user.email === data.email);
  if (found === undefined) addUser(data);
  else
    res.send(
      "Account with Email ID: " +
        data.email +
        " already exists. Use a different Email ID or Sign in."
    );
});

async function addOrder(data) {
  const client = new MongoClient(url);

  try {
    let mydb = (await client.connect()).db(database);
    let collection = mydb.collection("Order");

    await collection.insertOne(data);

    console.log(
      "Order saved successfully in database Food Delivery Application."
    );
  } catch (error) {
    console.log(error);
  } finally {
    await client.close();
  }
}

app.post("/orders", urlencodedParser, async (req, res) => {
  let data = req.body;
  addOrder(data);
  res.send(
    "Server: Order placed Successfully for " + data.user + " at " + data.time
```

```
  );
});

app.listen(PORT, function (err) {
  if (err) console.log(err);
  console.log("***BACKEND***");
  console.log("Server Started.");
  console.log("Server listening on PORT:", PORT);
});
```

Client:

CSS:

style.js:

style.css

```
* {
  font-family: 'Alegreya', serif;
}
```

JSON:

pizzas.js:

```
import Bacon from "../Images/Bacon.jpg";
import Cheese from "../Images/Cheese.jpg";
import Hawaiin from "../Images/Hawaiin.jpg";
import Margerita from "../Images/Margerhita.jpg";
import Mushrooms from "../Images/Mushrooms.jpg";
import Pepperoni from "../Images/Pepperoni.jpg";
import Seafood from "../Images/Seafood.jpg";
import Supreme from "../Images/Supreme.jpg";
import Vegetarian from "../Images/Vegetarian.jpg";

const pizzas = [
  {
    name: "Hawaiin",
    size: ["Small", "Medium", "Large"],
    price: [200, 300, 400],
    category: "Non-Veg",
    image: Hawaiin,
    description: "",
  },
  {
    name: "Mushroom",
    size: ["Small", "Medium", "Large"],
    price: [300, 370, 410],
    category: "Veg",
    image: Mushrooms,
    description: "",
```

```
},
{
  name: "Non-Veg Supreme",
  size: ["Small", "Medium", "Large"],
  price: [350, 400, 480],
  category: "Non-Veg",
  image: Supreme,
  description: "",
},
{
  name: "Pepperoni",
  size: ["Small", "Medium", "Large"],
  price: [200, 300, 400],
  category: "Non-Veg",
  image: Pepperoni,
  description: "",
},
{
  name: "Margerita",
  size: ["Small", "Medium", "Large"],
  price: [210, 320, 430],
  category: "Veg",
  image: Margerita,
  description: "",
},
{
  name: "Sea Food",
  size: ["Small", "Medium", "Large"],
  price: [250, 360, 470],
  category: "Non-Veg",
  image: Seafood,
  description: "",
},
{
  name: "Veg Supreme",
  size: ["Small", "Medium", "Large"],
  price: [220, 330, 450],
  category: "Veg",
  image: Vegetarian,
  description: "",
},
{
  name: "Bacon",
  size: ["Small", "Medium", "Large"],
  price: [240, 360, 480],
  category: "Non-Veg",
  image: Bacon,
  description: "",
},
{
  name: "Cheese Special",
```

```
    size: ["Small", "Medium", "Large"],
    price: [180, 280, 380],
    category: "Veg",
    image: Cheese,
    description: "",
  },
];

export default pizzas;
```

Components:

App.js:

```
import React from "react";
import Menu from "./Components/Menu";
import Home from "./Components/Home";
import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
import Checkout from "./Components/Checkout";
import SignUp from "./Components/SignUp";
import "./Components/style.css";

function App() {
  return (
    <>
      <Router>
        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="/menu" element={<Menu />} />
          <Route path="/checkout" element={<Checkout />} />
          <Route path="/signup" element={<SignUp />} />
        </Routes>
      </Router>
    </>
  );
}

export default App;
```

Checkout.js:

```
import React from "react";
import { useLocation, useNavigate } from "react-router-dom";
import { useEffect } from "react";

import axios from "axios";
import { useState } from "react";
import Navigation from "./Navigation";
const Checkout = () => {
  const location = useLocation();
  let cart = location.state;
```

```
  const [users, setUsers] = useState([]);
  const [currentUser, setCurrentUser] = useState("");

  async function sendData() {
    let bill = {
      user: currentUser,
      time: Date(),
      cart: cart,
    };

    let response = await axios.post("http://localhost:8000/orders", bill);
    let serverMessage = response.data;
    alert(serverMessage);
  }

  async function getData() {
    let userData = (await axios.get("http://localhost:8000/users")).data;
    setUsers(userData);
  }

  function isEmpty() {
    return currentUser.length === 0;
  }

  const handlePay = (e) => {
    e.preventDefault();

    console.log(currentUser);
    if (isEmpty()) alert("Order Failed: Please select a valid account.");
    else {
      sendData();
      alert("Order placed successfully. Thank you for your purchase.");
    }
  };

  useEffect(() => {
    getData();
  }, []);

  let total = 0;
  return (
    <>
      <Navigation />
      <div className="bg-dark p-4">
        <div className="d-flex justify-content-center align-items-center">
          <img

src="https://static.vecteezy.com/system/resources/previews/011/157/909/original/pizzeria-emblem-on-blackboard-pizza-logo-template-emblem-for-cafe-restaurant-or-food-delivery-service-vector.jpg"
```

```
    width={300}
  ></img>
  <div className="p-4">
    <h1 className="text-center text-light">
      <strong>Thank you for choosing us.</strong>
    </h1>

    <h3 className="text-center text-light">Please review your cart.</h3>
  </div>
</div>
<h4 className="my-4 text-light">
  You have <strong> {cart.length} </strong>item(s) in your cart:
</h4>

<div>
  <table className="table table-striped table-light">
    <thead className="thead-dark">
      <tr className="table-warning text-center">
        <th>
          <h4>Serial No.</h4>
        </th>
        <th>
          <h4>Product</h4>
        </th>
        <th>
          <h4>Category</h4>
        </th>
        <th>
          <h4>Size</h4>
        </th>
        <th>
          <h4>Quantity</h4>
        </th>
        <th>
          <h4>Amount ($)</h4>
        </th>
      </tr>
    </thead>
    <tbody>
      {cart.map((item, index) => {
        total += item.price;
        return (
          <tr className="text-center">
            <td>
              <h5>{index + 1}</h5>
            </td>
            {Object.keys(item).map((key) => (
              <td>
                <h5>{item[key]}</h5>
              </td>
            ))}
```

```
        </tr>
      );
    })}
    <tr className="text-center">
      <td>
        <strong>Total</strong>
      </td>
      <td></td>
      <td></td>
      <td></td>
      <td></td>
      <td>{total}</td>
    </tr>
    <tr className="text-center">
      <td>
        <strong>CGST (5%)</strong>
      </td>
      <td></td>
      <td></td>
      <td></td>
      <td></td>
      <td>{0.05 * total}</td>
    </tr>
    <tr className="text-center">
      <td>
        <strong>SGST (5%)</strong>
      </td>
      <td></td>
      <td></td>
      <td></td>
      <td></td>
      <td>{0.05 * total}</td>
    </tr>
    <tr className="text-center">
      <td>
        <h4> G. Total:</h4>
      </td>
      <td></td>
      <td></td>
      <td></td>
      <td></td>
      <td>
        <h4>
          <strong>${total + 0.1 * total}</strong>
        </h4>
      </td>
    </tr>
    <br />
  </tbody>
</table>
```

```
          <h4 className="my-4 text-light">Select Delivery Account:</h4>
          <select
            value={currentUser}
            className="form-select"
            onChange={(e) => {
              console.log(e.target.value);
              setCurrentUser(e.target.value);
            }}
          >
            <option value="" disabled selected>
              Select Account
            </option>
            {users.map((user) => {
              return (
                <option value={user.email}>
                  {user.name + " - " + user.email}
                </option>
              );
            })}
          </select>
          <br />
          <button className="btn btn-warning" onClick={handlePay}>
            Pay via COD
          </button>
        </div>
      </div>

      {/* <select
              value={size}
              className="form-select form-select-sm"
              onChange={(e) => {
                setSize(e.target.value);
              }}
            >
              {pizza.size.map((size) => {
                return <option value={size}>{size}</option>;
              })}
            </select> */}
    </>
  );
};

export default Checkout;
```

Home.js:

```
import React from "react";
import Navigation from "./Navigation";
import { useNavigate } from "react-router-dom";

const Home = () => {
  const navigate = useNavigate();

  const handleOrder = (e) => {
    e.preventDefault();
    navigate("/menu");
  };

  return (
    <>
      <Navigation />

      <div>
             <div  className="d-flex  justify-content-center  align-items-center  m-3
bg-dark rounded p-5">
          <div className="m-4">
            <h1 className="text-light">
              <strong>Fresh PIZZA </strong> at your Doorstep!
            </h1>
            <h3 className="text-light">
              {" "}
              Delicious Pizzas starting @ $200 Only!
            </h3>
            <h5 className="text-light">
              Enjoy a fresh slice of pizza. Share with your friends, family or
              just eat alone.
            </h5>
            <br />
            <button className="btn btn-warning fw-bold" onClick={handleOrder}>
              Order Now
            </button>
          </div>
          <img

src="https://cdn.tatlerasia.com/indonesiatatlerdining/images/i/20180425-11b5f8889f5
6ae4aab094_resized_773x515.jpg"
            width={900}
            height={500}
            className="img-rounded"
          ></img>
        </div>
      </div>

      <div>
```

```
            <div  className="d-flex  justify-content-center  align-items-center  m-3
bg-dark rounded p-5">
          <img

src="https://cdn.jobtoday.com/img/e8c45297-24c2-4547-99ca-7bc530c9053d/1140x640.jpg
"
            width={500}
            height={250}
            className="img-rounded"
          ></img>
          <div className="m-4">
            <h1 className="text-light">About us</h1>
            <h5 className="text-light">
              This project 'Food Delivery Application' is a web-application made
              as an requirement of Major Project. Technologies used are React.js
              for Front-end, Node.js and Express.js for Middle-ware (Back-end),
              and MongoDB as Database Engine.
            </h5>
            <br />
            <h5 className="text-light">
              Made by: <strong>Manan Mehta</strong>, Group-115
            </h5>
          </div>
        </div>
      </div>

      <div>
             <div  className="d-flex  justify-content-center  align-items-center  m-3
bg-dark rounded p-5">
          <div className="m-4">
            <h1 className="text-light">
              Introducing NEW <strong>Veg Supreme</strong> Pizza!
            </h1>
            <h5 className="text-light">
              Hand tossed pizza topped with Onions, Tomatoes, Mushrooms,
              Jalepenoes, Olives, and Corn.
            </h5>
            <br />
            <h5 className="text-light">Try TODAY!</h5>
          </div>
          <img

src="https://www.franchisegator.com/articles/wp-content/uploads/2022/02/vegetables-
italian-pizza-restaurant-large.jpg"
            width={400}
            height={250}
            className="img-rounded"
          ></img>
        </div>
      </div>
    </>
```

```
  );
};

export default Home;
```

Menu.js:

```
import React from "react";
import pizzas from "../JSON/pizzas";
import Pizza from "./Pizza";
import { useNavigate } from "react-router-dom";
import { useState } from "react";
import Navigation from "./Navigation";

const Menu = () => {
  const navigate = useNavigate();
  const [cart, setCart] = useState([]);

  const handleCheckout = (e) => {
    e.preventDefault();

    if (cart.length == 0)
      alert("Your cart is empty. Please purchase at least 1 item.");
    else navigate("/checkout", { state: cart });
  };

  return (
    <>
      <Navigation />
      <form className="w-100 bg-dark">
        <div className="mx-3">
          <button
            className="btn btn-warning float-end"
            onClick={(e) => handleCheckout(e)}
          >
            Checkout
          </button>
          <br />
        </div>
        <br />
        <div className="d-flex justify-content-center">
          <h1 className="text-dark bg-light m-4 p-2 w-100 text-center rounded">
            <strong>A La Carte</strong>
          </h1>
        </div>
        <div className="row">
          {pizzas.map((pizza, index) => {
            return (
              <>
                <div className="col-md-3 my-2 ">
                  <Pizza pizza={pizza} cart={cart} setCart={setCart} />
```

```
              </div>
            </>
          );
        })}
      </div>
    </form>
  </>
);
};

export default Menu;
```

Navigation.js:

```
import React from "react";
import { useNavigate } from "react-router-dom";

const Navigation = () => {
  const navigate = useNavigate();

  const handleHome = (e) => {
    e.preventDefault();
    navigate("/");
  };

  const handleOrder = (e) => {
    e.preventDefault();
    navigate("/menu");
  };

  const handleSignUp = (e) => {
    e.preventDefault();
    navigate("/signup");
  };

  return (
    <>
      <nav class="navbar navbar-expand-sm bg-dark navbar-dark">
        <div class="container-fluid">
          <a class="navbar-brand" href="#">
            <img

src="https://static.vecteezy.com/system/resources/previews/011/157/909/original/piz
zeria-emblem-on-blackboard-pizza-logo-template-emblem-for-cafe-restaurant-or-food-d
elivery-service-vector.jpg"
              alt="Avatar Logo"
              class="rounded-pill mx-2"
              width={40}
            />
            <strong>Pizza</strong> - Fresh & Tasty
          </a>
```

```
          <ul class="navbar-nav">
            <li class="nav-item p-1">
              <button
                class="btn btn-basic nav-link active"
                onClick={handleHome}
              >
                Home
              </button>
            </li>
            <li class="nav-item p-1">
              <button
                class="btn btn-basic nav-link active"
                onClick={handleOrder}
              >
                Menu
              </button>
            </li>

            <li class="nav-item p-1">
              <button
                class="btn btn-basic nav-link active"
                onClick={handleSignUp}
              >
                Sign Up
              </button>
            </li>
          </ul>
        </div>
      </nav>
    </>
  );
};

export default Navigation;
```

Pizza.js:

```
import React, { useState } from "react";

const Pizza = ({ pizza, cart, setCart }) => {
  const [quantity, setQuantity] = useState(1);
  const [size, setSize] = useState("Small");

  const handleAddToCart = (e) => {
    e.preventDefault();
    console.log(pizza.name);
    console.log(size);
    console.log(quantity);
    console.log(pizza.price[pizza.size.indexOf(size)] * quantity);

    let tempCart = [...cart];
```

```
    let item = tempCart.find((item) => {
      return item.name === pizza.name;
    });

    if (item !== undefined) {
      const index = tempCart.indexOf(item);
      if (index > -1) {
        tempCart.splice(index, 1);
      }
    }

    let fPrice = pizza.price[pizza.size.indexOf(size)] * quantity;

    let newItem = {
      name: pizza.name,
      category: pizza.category,
      size: size,
      quantity: quantity,
      price: fPrice,
    };

    tempCart = [...tempCart, newItem];
    setCart(tempCart);

    alert(
      pizza.name +
        " Pizza added in your cart. Continue shopping or proceed to Checkout. Thank
you."
    );
  };

  const handleRemoveFromCart = (e) => {
    e.preventDefault();
    let tempCart = [...cart];
    let item = tempCart.find((item) => {
      return item.name === pizza.name;
    });

    if (item !== undefined) {
      const index = tempCart.indexOf(item);
      if (index > -1) {
        tempCart.splice(index, 1);
      }
    }

    setCart(tempCart);
  };

  let color = "bg-success";
  return (
    <>
```

```
      <div className=" shadow py-4 bg-dark rounded m-4 border border-warning">
        <div className="d-flex justify-content-center">
          <h3 className="text-warning">
            <strong>{pizza.name}</strong>
          </h3>
        </div>
        <div className="d-flex justify-content-center m-2">
          <img
            src={pizza.image}
            className="img-thumbnail"
            style={{ height: "200px", width: "200px" }}
          />
        </div>
        <div className="d-flex justify-content-center">
          <div className="w-20 m-1">
            {/* <p className="text-center">Size</p> */}
            <select
              value={size}
              className="form-select form-select-sm"
              onChange={(e) => {
                setSize(e.target.value);
              }}
            >
              {pizza.size.map((size) => {
                return <option value={size}>{size}</option>;
              })}
            </select>
          </div>
          <div className="w-10 m-1 mx-1 d-flex">
            <select
              value={quantity}
              className="form-select form-select-sm mx-1 w-100"
              // ref={quantityRef}
              onChange={(e) => {
                setQuantity(e.target.value);
              }}
            >
              {[...Array(10).keys()].map((x, index) => {
                return <option value={index + 1}>{index + 1} </option>;
              })}
            </select>

            <div>
              {pizza.category === "Veg" ? (
                <img

src="https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRtSunGZrQkQMZlsJUeWHb5oq
i2rqXb-rukGg&usqp=CAU"
                  width={20}
                ></img>
              ) : (
```

```
                <img

src="https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSYMLfraBCscqg65z1lbsJ7Qr
dtJCGzNlv7_Q&usqp=CAU"
                  width={20}
                ></img>
              )}
            </div>
          </div>
        </div>

        <div className="d-flex justify-content-center">
          <div className="m-1 w-10">
            <h3 className="text-center text-info">
              ${pizza.price[pizza.size.indexOf(size)] * quantity}
            </h3>
          </div>
          <div className="m-1 w-10">
            <button
              className="btn btn-success "
              onClick={(e) => {
                handleAddToCart(e);
              }}
            >
              Add
            </button>
            <button
              className="btn btn-danger mx-2"
              onClick={(e) => {
                handleRemoveFromCart(e);
              }}
            >
              Remove
            </button>
          </div>
        </div>
      </div>
    </>
  );
};

export default Pizza;
```

SignUp.js

```
import React from "react";
import axios from "axios";
import { useNavigate } from "react-router-dom";
import { useState } from "react";

const SignUp = () => {
```

```
  const navigate = useNavigate();

  const [user, setUser] = useState({});

  const handleChange = (e) => {
    setUser({ ...user, [e.target.name]: e.target.value });
  };

  async function sendData() {
    let response = await axios.post("http://localhost:8000/signup", user);
    let serverMessage = response.data;
    alert(serverMessage);
  }

  const handleSubmit = async (e) => {
    e.preventDefault();
    console.log(user);
    await sendData();
  };

  const handleBack = (e) => {
    e.preventDefault();
    navigate("/");
  };

  return (
    <>
      <div className="d-flex justify-content-center p-4 bg-dark m-4 rounded">
        <div className="d-flex  align-items-center w-100">
          <img

src="https://static.vecteezy.com/system/resources/previews/011/157/909/original/piz
zeria-emblem-on-blackboard-pizza-logo-template-emblem-for-cafe-restaurant-or-food-d
elivery-service-vector.jpg"
            width={600}
            height={600}
          ></img>

          <form className="w-100 p-4 mt-3">
            <h1 className="text-light">
              <strong>Create New Account</strong>
            </h1>

            <div class="form-group my-1">
              <h5 className="label text-light mt-2">
                <strong>Personal Info</strong>
              </h5>

              <label className="label text-light">Name</label>
              <input
                name="name"
```

```
    onChange={(e) => handleChange(e)}
    class="form-control"
    type="text"
    placeholder="Ex. John Doe"
  />
</div>

<div class="form-group my-1">
  <label className="label text-light">Email</label>
  <input
    name="email"
    onChange={(e) => handleChange(e)}
    class="form-control"
    type="email"
    placeholder="Ex. somemail@gmail.com"
  />
</div>

<div class="form-group my-1">
  <label className="label text-light">Contact</label>
  <input
    name="contact"
    onChange={(e) => handleChange(e)}
    class="form-control"
    type="number"
    placeholder="Ex. 9999900000"
  />
</div>

<h5 className="label text-light mt-4">
  <strong>Delivery Address</strong>
</h5>

<div class="form-group my-1">
  <label className="label text-light">House No.</label>
  <input
    name="houseNo"
    onChange={(e) => handleChange(e)}
    class="form-control"
    type="number"
    placeholder="Ex. 123"
  />
</div>

<div class="form-group my-1">
  <label className="label text-light">Street and Locality</label>
  <input
    name="locality"
    onChange={(e) => handleChange(e)}
    class="form-control"
    type="text"
```

```
    placeholder="Ex. Avenue Street, Rajender Nagar"
  />
</div>

<div class="form-group my-1">
  <label className="label text-light">City/Town</label>
  <input
    name="city"
    onChange={(e) => handleChange(e)}
    class="form-control"
    type="text"
    placeholder="Ex. New Delhi"
  />
</div>

<div class="form-group my-1">
  <label className="label text-light">State</label>
  <input
    name="state"
    onChange={(e) => handleChange(e)}
    class="form-control"
    type="text"
    placeholder="Ex. Delhi"
  />
</div>

<div class="form-group my-1">
  <label className="label text-light">Pin-code</label>
  <input
    name="pinCode"
    onChange={(e) => handleChange(e)}
    class="form-control"
    type="number"
    placeholder="Ex. 001122"
  />
</div>

<button
  type="button"
  class="btn btn-outline-warning mr-2"
  onClick={handleBack}
>
  Home
</button>
<button
  className="btn btn-warning my-4 fw-bold m-2"
  onClick={handleSubmit}
  type="submit"
>
  Sign Up
</button>
```

```
          </form>
        </div>
      </div>
    </>
  );
};

export default SignUp;
```

Food Delivery Application using MERN Stack - Manan Mehta

ORIGINALITY REPORT

8%	4%	4%	5%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Liverpool John Moores University Student Paper	1%
2	Submitted to University of Greenwich Student Paper	1%
3	dev.to Internet Source	<1%
4	Chris Minnick. "Beginning React JS Foundations Building User Interfaces with ReactJS", Wiley, 2022 Publication	<1%
5	tekolio.com Internet Source	<1%
6	Brajesh De. "API Management", Springer Science and Business Media LLC, 2017 Publication	<1%
7	Submitted to Queen Mary and Westfield College Student Paper	<1%
	www.vingle.net	

8	Internet Source	<1%
9	Submitted to IUBH - Internationale Hochschule Bad Honnef-Bonn Student Paper	<1%
10	Luis Argerich, Wankyu Choi, John Coggeshall, Ken Egervari et al. "Professional PHP4", Springer Science and Business Media LLC, 2003 Publication	<1%
11	Submitted to Coventry University Student Paper	<1%
12	Submitted to University of Belgrade, Faculty of Organizational Sciences Student Paper	<1%
13	Submitted to Nizwa College of Technology Student Paper	<1%
14	Submitted to Imperial College of Science, Technology and Medicine Student Paper	<1%
15	Submitted to Staffordshire University Student Paper	<1%
16	www.readkong.com Internet Source	<1%
17	"Appendix C", IMS Application Developer s Handbook, 2011	<1%

	Publication	
18	Submitted to Kwame Nkrumah University of Science and Technology Student Paper	<1 %
19	Nico Loubser. "Software Engineering for Absolute Beginners", Springer Science and Business Media LLC, 2021 Publication	<1 %
20	dokumen.pub Internet Source	<1 %
21	Adam Freeman. "Expert ASP.NET Web API 2 for MVC Developers", Springer Science and Business Media LLC, 2014 Publication	<1 %
22	Submitted to University of Central Florida Student Paper	<1 %
23	codedamn.com Internet Source	<1 %
24	Internet Programming with Visual Basic, 2000. Publication	<1 %
25	slides.com Internet Source	<1 %
26	www.a2hosting.com Internet Source	<1 %
27	Submitted to University of Northampton Student Paper	<1 %

28	Submitted to University of South Australia Student Paper	<1 %
29	Submitted to University of Denver Student Paper	<1 %
30	www.dhiwise.com Internet Source	<1 %
31	dspace.ut.ee Internet Source	<1 %
32	www.abservetech.com Internet Source	<1 %
33	www.groovyweb.co Internet Source	<1 %
34	docplayer.net Internet Source	<1 %
35	ela.kpi.ua Internet Source	<1 %
36	manjula kumara. ""MealShare: Using Blockchain Technology and a Reward-Based System to Combat Global Food Waste"", Institute of Electrical and Electronics Engineers (IEEE), 2023 Publication	<1 %
37	Elad Elrom. "React and Libraries", Springer Science and Business Media LLC, 2021 Publication	<1 %

38	www.cisin.com Internet Source	<1%
39	Submitted to Birmingham Metropolitan College Student Paper	<1%
40	Cristian Darie, Karli Watson. "Beginning ASP.NET E-Commerce in C#", Springer Science and Business Media LLC, 2009 Publication	<1%
41	Hari Narayn. "Chapter 9 React Back", Springer Science and Business Media LLC, 2022 Publication	<1%
42	ebin.pub Internet Source	<1%
43	formidable.com Internet Source	<1%
44	gateway.ipfs.io Internet Source	<1%
45	hdl.handle.net Internet Source	<1%
46	ir.cwi.nl Internet Source	<1%
47	Frank Zammetti. "Chapter 14 Feed Your Face: Fooderator, the Client", Springer Science and Business Media LLC, 2022 Publication	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		