VIETNAM NATIONAL UNIVERSITY OF HOCHIMINH CITY
THE INTERNATIONAL UNIVERSITY
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Web Application Development
IT093IU

PROJECT REPORT

Topic: Home Delivery Website

By Group pdm – Member List

No.	Full name – Student name	Student ID
1	Đặng Nguyễn Trường Huy	ITITIU21010
2	Nguyễn Hoàng Minh Khôi	ITITIU21229
3	Nguyễn Trần Gia Huy	ITITIU21054

Instructor: Assoc. Prof. Nguyen Van Sinh

TABLE OF CONTENTS

ABSTRACTS

The GrabFood Web Application is a comprehensive and innovative food delivery platform that empowers users to conveniently order meals from a diverse and extensive menu. This project was designed as part of a Web Application Development course, showcasing the integration of modern web technologies and best practices to provide a seamless and engaging user experience. The primary purpose of the application is to streamline the food ordering process, allowing users to effortlessly browse through various food categories, add items to their cart, and make secure payments with confidence.

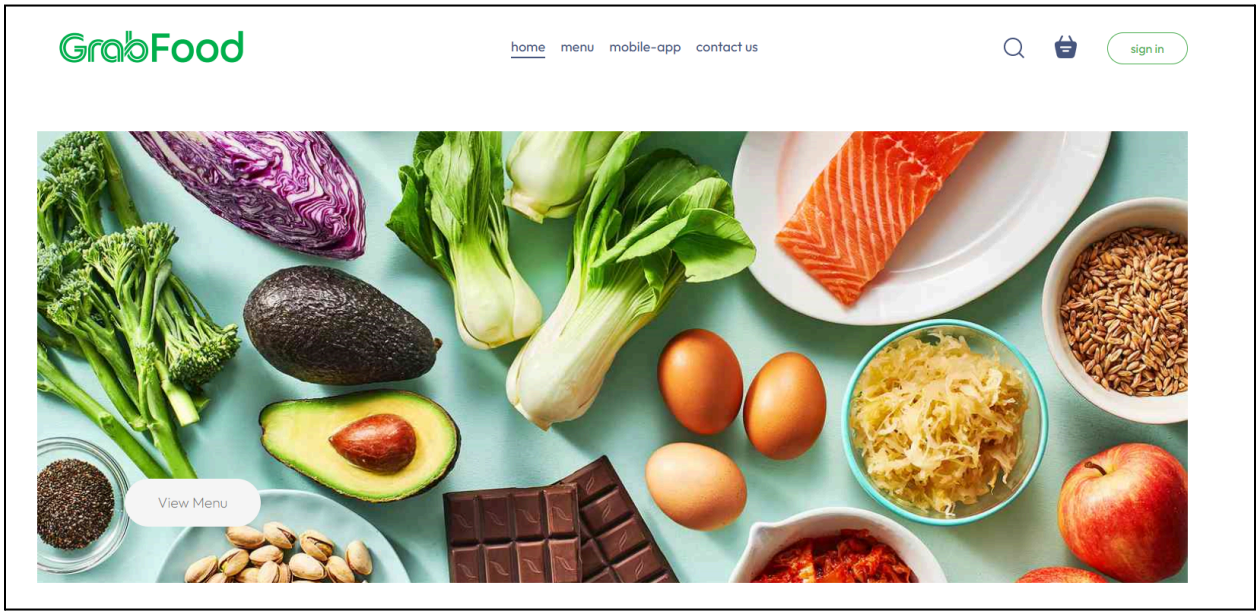
Key features of the application include an Interactive Menu Interface, where users can explore a wide range of food categories and view detailed information about each dish, including ingredients, nutritional information, and pricing. This feature enhances user engagement by providing all the necessary information to make informed choices. Additionally, the application incorporates a User Registration and Login system, supported by a secure backend that facilitates account creation and login functionality, ensuring that user data is protected and accessible only to authorized individuals.

The Cart Management feature allows users to add, edit, and remove items from their cart before finalizing their orders, providing flexibility and control over their selections. This is complemented by a Payment Integration system, which ensures smooth and secure transactions through various payment methods, enhancing user trust and satisfaction. Furthermore, the application boasts a Responsive Design, optimized for both desktop and mobile users, ensuring that the platform is accessible and user-friendly across a variety of devices.

The backend of the application is developed using robust technologies that effectively handle user authentication, database management, and transaction processing. This ensures that the application can scale efficiently as the user base grows. On the frontend, responsive frameworks are employed to create an intuitive and visually appealing interface that enhances the overall user experience.

Overall, the project successfully demonstrates the application of web development skills, offering a functional and scalable solution for food delivery that emphasizes user convenience and security. It stands as a valuable addition to the web application ecosystem, addressing the growing demand for efficient food ordering solutions. Looking ahead, future iterations of the GrabFood Web Application could include additional features such as real-time order tracking, user reviews and ratings for food items, and partnerships with local restaurants to further enhance user experience and engagement. These enhancements would not only improve the functionality of the application but also

foster a sense of community among users and local food providers, ultimately contributing to a more dynamic and interactive food delivery platform.



INTRODUCTION

In today's fast-paced world, individuals often struggle to find the time to prepare meals or visit restaurants for dining. This creates a need for a convenient and user-friendly platform that allows users to browse food options, place orders, and make payments—all from the comfort of their homes. Many existing food delivery platforms fail to provide a seamless experience, leaving room for improvement in terms of performance, accessibility, and scalability. The GrabFood Web Application is designed to address these challenges by providing an efficient and intuitive system for ordering and delivering food online. This application ensures that users can effortlessly explore menus, select their desired items, make secure payments, and receive their orders at their doorstep.

The primary goal of the GrabFood Web Application is to simplify the process of ordering and delivering food. The application focuses on providing essential features, such as menu browsing, cart management, user authentication, and secure payment integration, all while maintaining a fast and reliable backend system. Designed with scalability in mind, the project also aims to support a growing number of users and transactions over time. Additionally, ensuring the security of user data and payment information is a key priority to build trust and reliability within the system.

The scope of this project is limited to delivering the core functionalities of a web-based food ordering platform. The application is built using React.js, Node.js, JavaScript, and MongoDB, ensuring a robust and modern technology stack. While the system provides features such as browsing menus, placing orders, and making payments, advanced functionalities like real-time order tracking, integration with restaurant networks, and personalized recommendations are outside the scope of this project. Furthermore, while the web application is optimized for both desktop and mobile users, a dedicated mobile application is not part of this development phase.

The GrabFood Web Application is designed for a wide audience, including busy professionals, students, families, and food enthusiasts. It caters to anyone seeking a convenient and efficient platform for ordering and delivering food. With its emphasis on accessibility and simplicity, the application aims to enhance the food delivery experience for users from all walks of life

SYSTEM DESIGNS AND ARCHITECT

Overview:

The GrabFood Web Application is designed as a modern, full-stack web application with a clear separation between the frontend and backend components. The architecture is structured to ensure scalability, security, and a seamless user experience. The web app consists of three key sections: an index page that serves as the main interface for browsing and selecting food, a login page that handles user authentication, and an ordered food list page that facilitates payment and order review. These components are integrated using a robust client-server model, with the frontend providing a responsive and user-friendly interface and the backend handling data processing, storage, and API functionality. The system leverages RESTful APIs to enable smooth communication between the frontend, backend, and database.

Frontend Design:

The frontend of the **GrabFood Web Applicatio**n is developed using a combination of HTML, JavaScript, and the React.js framework. These technologies work together to create a responsive, dynamic, and user-friendly interface that ensures an optimal user experience.

JavaScript

JavaScript is the backbone of the frontend development and contributes the most to the application logic. It powers dynamic features such as:

- Rendering food items dynamically using React components.
- Managing user interactions like adding food to the cart, updating quantities, and processing payments.
- Handling application state using React's state management to ensure seamless updates across the UI (e.g., cart and order total updates).

CSS

CSS is used to design and style the user interface, ensuring the application is visually appealing and responsive. Key design elements include:

- Custom styles for buttons, forms, food item cards, and navigation.
- Implementation of layout designs such as grids for food menus and alignment for the cart and order summary.
- Media queries for responsive design to optimize the app for both desktop and mobile screens.

HTML

HTML provides the basic structure of the application, serving as the foundation for the React components. Although minimal, it includes essential markup such as:

- The root div element where the React app is mounted.
- Basic semantic elements like headers, sections, and input fields for forms.

Vite.js

Vite.js is utilized as the build tool for the frontend, providing a fast and efficient development environment. Its features include:

- Instant Server Start: Vite.js offers an instant server start, allowing developers to see changes in real-time without the need for lengthy build processes.
- Hot Module Replacement (HMR): Vite.js supports hot module replacement, enabling developers to make changes to the code and see the results immediately in the browser, significantly improving productivity.
- Optimized Build Process: Vite.js optimizes the build process for production, ensuring that the application is lightweight and performs well in a live environment.

React.js

React, a popular JavaScript library, is the core framework used for building the frontend. React's component-based architecture allows the web application to be modular, scalable, and easier to maintain.

- Components: The frontend is divided into reusable React components such as the header, login form, food menu, cart, and payment summary sections.
- State Management: React's state management ensures real-time updates, such as reflecting changes in the cart and displaying ordered items dynamically.
- Virtual DOM: React's virtual DOM improves performance by efficiently updating and rendering only the components that change.

The frontend ensures that users can easily navigate through the application, register or log in to their accounts, browse food items, add them to their cart, and proceed to payment. Additionally, responsive design principles are applied to optimize the application for different screen sizes, making it accessible on both desktop and mobile devices.

By using **HTML**, **JavaScrip**t, **Vite.js** and **React.js**, the frontend achieves a clean, interactive, and visually appealing design that enhances the overall user experience.

Backend Design

The backend of the GrabFood Web Application is developed using **Node.js** and **MongoDB**. This combination provides a robust, scalable, and efficient server-side architecture to handle client requests, process data, and manage storage.

Node.js

Node.js, a popular server-side JavaScript runtime, is used to handle backend operations. It allows for fast, asynchronous, and event-driven programming, which is ideal for web applications requiring real-time updates and multiple concurrent connections. Key aspects of using Node.js include:

- Handling Client Requests: Node.js manages incoming HTTP requests from the frontend, such as user registration, login, fetching food menus, updating carts, and processing orders.
- Routing: Express.js, a lightweight framework built on top of Node.js, is used to define routes for different API endpoints like user authentication, order management, and payment processing.

- Asynchronous Operations: Node.js ensures that multiple tasks like database queries and API calls are handled efficiently without blocking other processes, enhancing performance.

MongoDB

MongoDB, a NoSQL database, is used for data storage and management. It stores data in a flexible, document-based format (JSON-like), making it easier to work with dynamic and unstructured data. Key aspects of MongoDB include:

- User Data Storage: MongoDB is used to store user credentials, such as names, emails, and hashed passwords for secure authentication.
- Order Management: Orders, including food items, quantities, prices, and payment statuses, are stored and retrieved in real time.
- Menu Data: Food items with details like name, price, description, and images are stored in MongoDB collections, which can be efficiently queried and displayed to the frontend.
- Scalability: MongoDB allows for horizontal scaling, ensuring the system can manage increasing numbers of users and orders as the application grows.

Express.js Framework

Express.js, a minimal and flexible Node.js framework, is used to streamline backend development. It simplifies tasks like creating RESTful APIs, routing, and middleware integration. Key functionalities include:

- API Endpoints: Express is used to define API routes for handling user requests, such as:
    - POST /api/register – User registration
    - POST /api/login – User login
    - GET /api/food-items – Fetching the food menu
    - POST /api/cart – Managing the cart and processing orders
- Middleware: Middleware functions are used for tasks like data validation, authentication, and error handling.

Database Design

The database utilized for this project is MongoDB, a widely recognized NoSQL database that is particularly esteemed for its flexibility, scalability, and performance. Unlike traditional relational databases that rely on a fixed schema and store data in tables with rows and columns, MongoDB adopts a document-oriented format. This innovative approach allows for the storage of data in collections and documents, which can vary in structure, making it highly adaptable to the evolving needs of modern applications.

In this specific design, the primary collection is the "users" collection, which serves as a repository for documents that represent individual users of the application. Each document within this collection contains essential fields such as the user's name, email address, and password, which are crucial for user authentication and account management. This structure not only facilitates easy access to user information but also allows for the implementation of various user-related functionalities, such as account creation, login, and profile updates.

In addition to the "users" collection, there is a dedicated "food_items" collection that holds documents for a diverse array of food products available for ordering. Each document in this collection includes important attributes such as the food item's name, a detailed description, pricing information, and the category to which the item belongs. This categorization is particularly beneficial for users, as it enables them to easily navigate through different types of food offerings, enhancing their overall shopping experience.

The relationship between users and food items is established through a sophisticated cart system, which is a pivotal feature of the application. This system allows each user to have multiple food items associated with their account, effectively creating a personalized shopping experience. When a user adds food items to their cart, the application dynamically updates the user's cart data in the database, ensuring that the information is always current and accurately reflects the user's selections. This functionality not only streamlines the ordering process but also encourages users to explore various food options, ultimately leading to increased engagement and satisfaction.

Moreover, the document-oriented nature of MongoDB enables efficient querying and retrieval of data, which is essential for meeting the application's requirements for managing user accounts and food inventory seamlessly. For instance, when a user logs in, the application can quickly retrieve their associated cart items and display them in real-time, allowing for a smooth transition from browsing to purchasing. Additionally, the flexibility of MongoDB's schema allows developers to easily modify and expand the

database structure as new features are added to the application, ensuring that it can grow and adapt to changing user needs over time.

In summary, the use of MongoDB as the database for this project provides a robust foundation for managing user accounts and food inventory. Its document-oriented design, combined with the dynamic cart system, facilitates a user-friendly experience that is both efficient and scalable. As the application continues to evolve, MongoDB's inherent flexibility will support the integration of new features and enhancements, ensuring that it remains a valuable tool for users seeking a convenient and enjoyable food ordering experience.

APIs

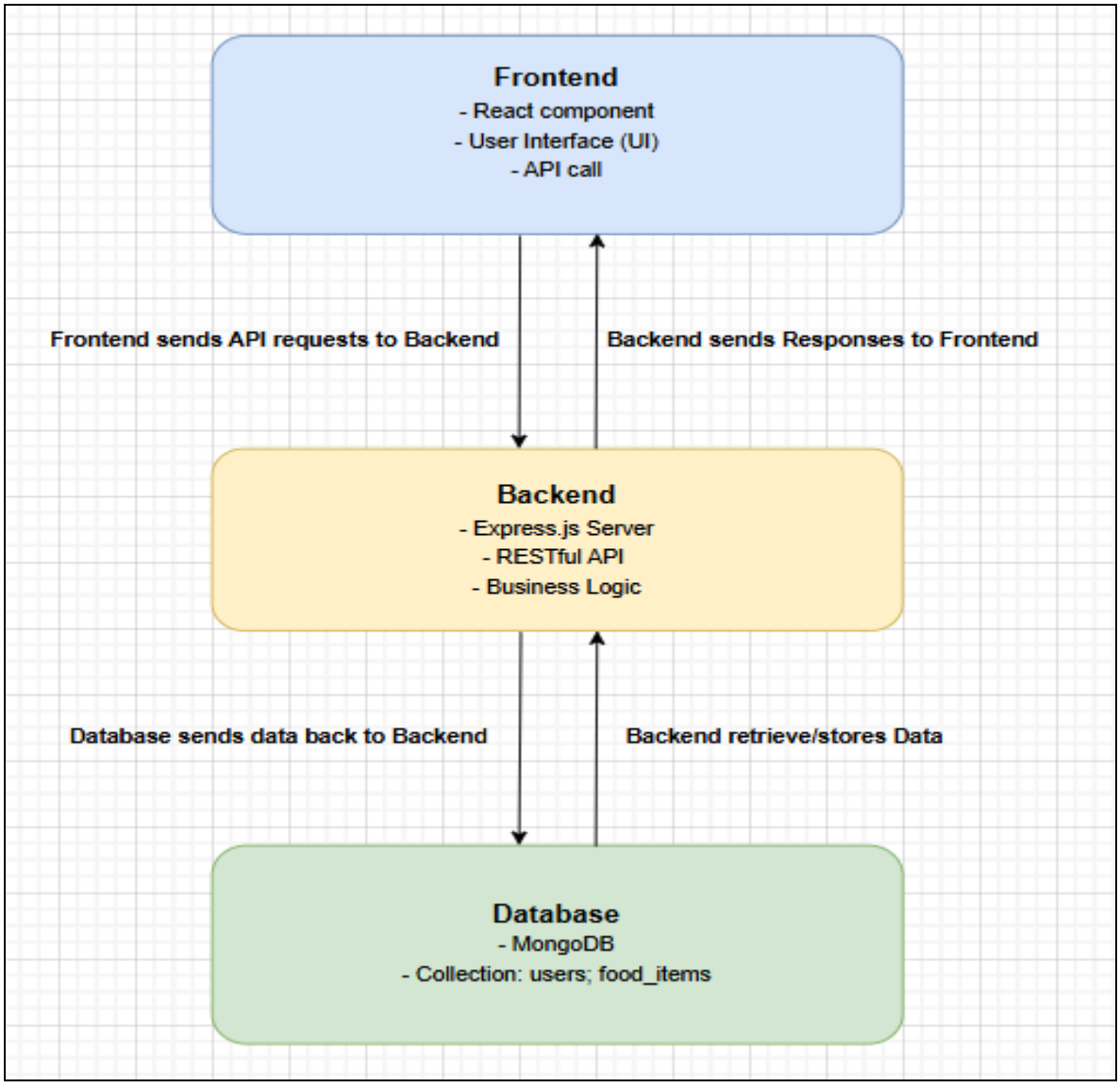
The application integrates a custom-built RESTful API for user authentication and management, utilizing Express.js on the backend. This API includes endpoints for user registration and login, allowing users to create accounts and securely authenticate using JSON Web Tokens (JWT). The API handles requests to register new users, log in existing users, and manage user sessions, ensuring a seamless experience. Additionally, the application may leverage third-party APIs for functionalities such as payment processing or location services, enhancing the overall capabilities of the platform.

User Interface (UI)

The user interface is designed with a clean and modern aesthetic, prioritizing usability and accessibility. The layout features a responsive design that adapts to various screen sizes, ensuring a consistent experience across devices. Navigation is intuitive, with a top navigation bar that includes links to

key sections such as Home, Menu, Cart, and User Profile. Users can easily browse through food categories, view detailed descriptions of items, and add them to their cart with a single click. The checkout process is streamlined, guiding users through order confirmation and payment options. Overall, the UI emphasizes a user-friendly experience, making it easy for users to interact with the application and complete their desired actions efficiently.

System Architecture Diagram



Technology

Frontend:

- HTML: The backbone of the web application, HTML is used to structure the content and layout of the user interface, ensuring that all elements are properly organized and accessible.
- CSS: Cascading Style Sheets (CSS) are employed to style the application, providing visual aesthetics such as colors, fonts, and spacing, enhancing the overall user experience.
- JavaScript: The primary programming language for adding interactivity and dynamic behavior to the web application, allowing for responsive user interactions.
- React: A JavaScript library for building user interfaces, React is utilized to create reusable components, manage the application state, and facilitate efficient rendering of UI elements.
- Vite.js: Vite.js is integrated into the frontend development process as a modern build tool that enhances the overall workflow. It provides an instant server start, allowing developers to see changes in real-time without lengthy build processes. Vite.js also supports hot module replacement (HMR), enabling developers to make changes to the code and immediately see the results in the browser, significantly improving productivity. Additionally, Vite.js optimizes the build process for production, ensuring that the application is lightweight and performs well in a live environment.

Backend:

- Node.js: A JavaScript runtime built on Chrome's V8 engine, Node.js is used for server-side development, enabling the creation of scalable and high-performance applications.
- Express.js: A web application framework for Node.js, Express simplifies the process of building robust APIs and handling HTTP requests, providing middleware support for various functionalities.

Database:

- MongoDB: A NoSQL database that stores data in a flexible, document-oriented format. MongoDB is used to manage user data and food items, allowing for efficient querying and data retrieval.

Version Control:

- Git: A distributed version control system that tracks changes in the codebase, enabling collaboration among developers and maintaining a history of project modifications.
- GitHub: A web-based platform for hosting Git repositories, GitHub is used for code collaboration, version control, and project management, allowing team members to contribute and review code efficiently.

Deployment:

- Localhost: The application is deployed on a local server during development, allowing developers to test and debug the application in a controlled environment. This setup typically involves running the backend server on a specified port (e.g., 5137) and accessing the frontend through a web browser at http://localhost:PORT.

Other Tools:

- Figma: A collaborative design tool used for creating UI/UX designs and prototypes. Figma allows the design team to visualize the application layout and user interactions before development.
- Postman: A tool for testing APIs, Postman is used to send requests to the backend endpoints, allowing developers to verify the functionality and performance of the API during development.

These technologies and tools collectively contribute to the development, deployment, and maintenance of the web application, ensuring a robust and user-friendly experience.

IMPLEMENTATION

Frontend Development:

The user interface (UI) was developed using React, which allowed for the creation of a dynamic and responsive application. The development process began with designing the layout using HTML and CSS, ensuring that the application is visually appealing and user-friendly. React components were created for various sections of the application, such as the Navbar, Home, Cart, and Food Display. Each component was designed to be reusable, promoting modularity and maintainability. State management was handled using React's built-in hooks, such as useState and useContext, to manage user authentication and cart data. The application was styled using CSS, with a focus on responsive design to ensure compatibility across different devices and screen sizes.

Navigation Bar

The navigation bar (Navbar) is a crucial component of the frontend user interface, providing users with easy access to different sections of the application. Below is a detailed breakdown of the Navbar's implementation, functionality, and design considerations.

Component Structure

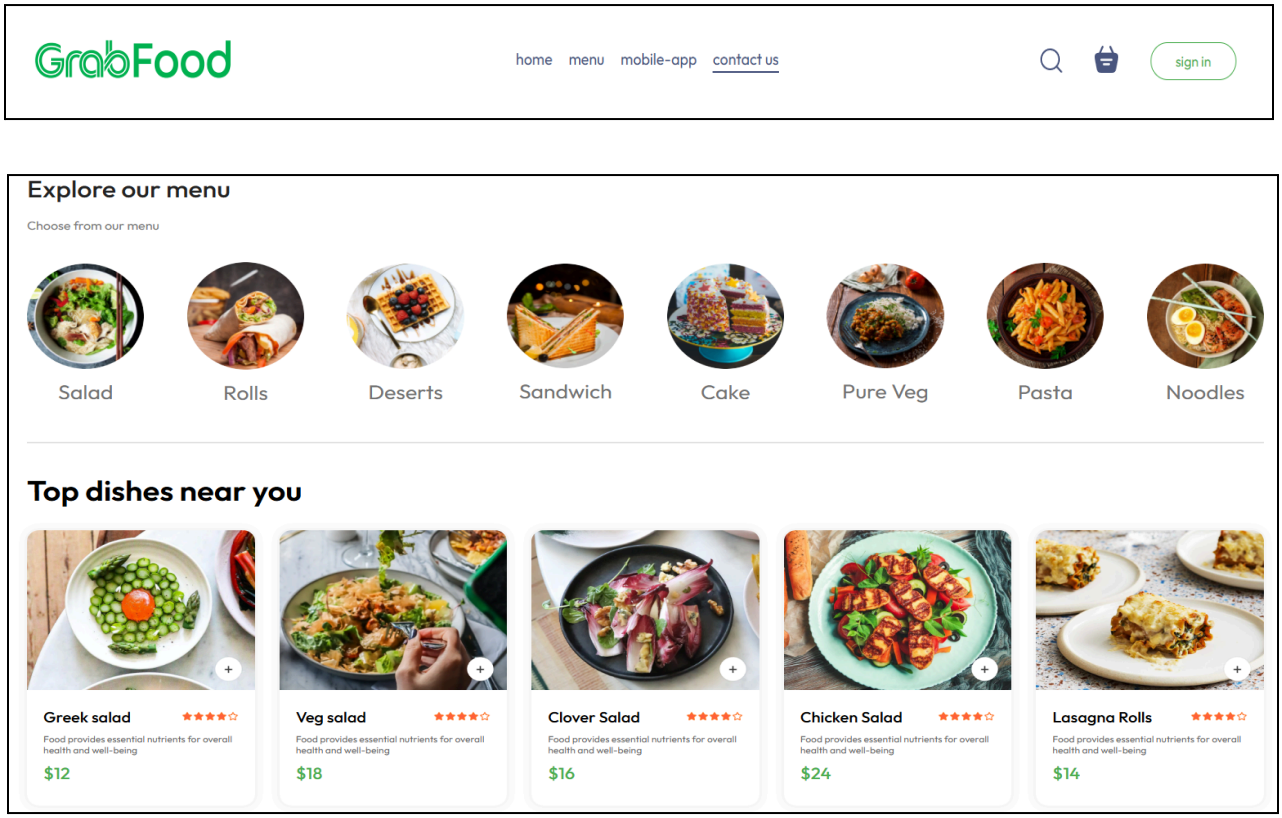
```
You, 5 hours ago | 2 authors (HuyNguyen2305 and one other)
import React, { useContext, useState } from 'react'  6.9k (gzipped: 2.7k)
import './Navbar.css'
import { assets } from '../../assets/assets'
import { Link, useNavigate } from 'react-router-dom';  224.7k (gzipped: 71k)
import { StoreContext } from '../../context/StoreContext';

const Navbar = ({setShowLogin}) => {

    const [menu,setMenu] = useState("home");

    const {getTotalCartAmount, token, setToken} = useContext(StoreContext);

    const navigate = useNavigate();

    const logout = () =>{
        localStorage.removeItem("token");
        setToken("");
        navigate("/")
    }
```

```jsx
return (
    <div className='navbar'>
        <Link to='/'><img src={assets.logo} alt="" className="logo" /></Link>
        <ul className='navbar-menu'>
            <Link to='/' onClick={()=>setMenu("home")} className={menu==="home"?"active":""}>home</Link>
            <a href="#explore-menu" onClick={()=>setMenu("menu")} className={menu==="menu"?"active":""}>menu</a>
            <a href='#app-download' onClick={()=>setMenu("mobile-app")} className={menu==="mobile-app"?"active":""}
            <a href='#footer' onClick={()=>setMenu("contact-us")} className={menu==="contact-us"?"active":""}>conta
        </ul>
        <div className='navbar-right'>
            <img src={ assets.search_icon} alt="" />
            <div className='navbar-search-icon'>
                <Link to="/cart"><img src={assets.basket_icon} alt="" /></Link>
                <div className={getTotalCartAmount()===0?"":"dot"}></div>
            </div>
            {!token?<button onClick={()=>setShowLogin(true)}>sign in</button>
            :<div className='navbar-profile'>
                <img src={assets.profile_icon} alt="" />      HuyNguyen2305, 14 hours ago • update 21:00 24121
                <ul className="nav-profile-dropdown">
                    <li><img src={assets.bag_icon} alt="" /><p>Orders</p></li>
                    <hr />
                    <li onClick={logout}><img src={assets.logout_icon} alt="" /><p>Logout</p></li>
                </ul>
            </div>}

        </div>
    </div>
)
}

export default Navbar
```

Key Features and Functionality

- Dynamic Menu State: The useState hook is used to manage the current active menu item. This allows the Navbar to highlight the active section, providing visual feedback to users about their current location within the application.
- Context API Integration: The Navbar utilizes the StoreContext to access global state variables such as token (for user authentication) and getTotalCartAmount (to display the number of items in the cart). This integration allows the Navbar to react to changes in the application state.
- Logout Functionality: The logout function removes the user's token from local storage and updates the context state, effectively logging the user out and redirecting them to the home page.
- Responsive Navigation Links: The Navbar includes links to various sections of the application, such as Home, Menu, Mobile App, and Contact Us. Each link updates the active menu state when clicked, ensuring that users can easily navigate through the application.
- User Authentication Handling: The Navbar conditionally renders either a "Sign In" button or a user profile dropdown based on the authentication state. If the user

is logged in, they can see their profile icon and access options to view orders or log out.



LoginPopup

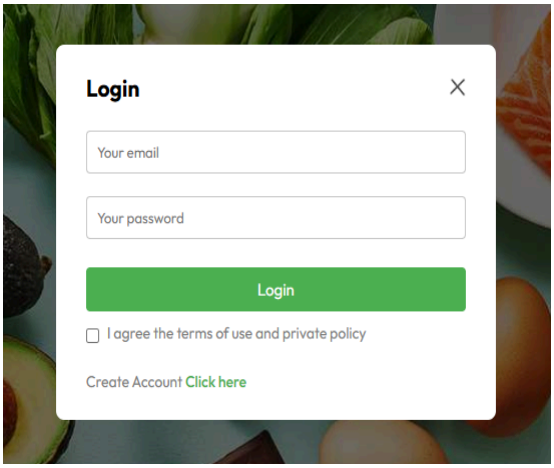
The LoginPopup is implemented as a React functional component. It utilizes hooks for state management and context for accessing global application state. Here's a simplified structure of the LoginPopup component.

Key Features and Functionality

State Management: The LoginPopup component uses the useState hook to manage the current state of the form (either "Sign up" or "Login") and the user input data (name, email, password). This allows the component to dynamically render the appropriate fields based on the user's action.

- Context API Integration: The component utilizes the StoreContext to access the API URL and the setToken function. This integration allows the component to communicate with the backend API for user authentication and to store the received token in the global state.
- Form Handling: The onChangeHandler function updates the state with user input as they type in the form fields. The onLogin function handles form submission, sending a POST request to the appropriate API endpoint based on the current state (login or registration).
- Conditional Rendering: The component conditionally renders the name input field based on whether the user is signing up or logging in. It also displays different button text and messages to guide the user through the authentication process.
- User Feedback: If the login or registration is successful, the token is stored in local storage, and the popup is closed. If there is an error (e.g., invalid credentials or user already exists), an alert is shown with the error message.

```
const LoginPopup = ({setShowLogin}) => {

    const {url, setToken} = useContext(StoreContext)

    const [currState, setCurrState] =useState("Sign up")
    const [data,setData] = useState({
        name:"",
        email:"",
        password:""
    })

    const onChangeHandler = (event) =>{
        const name = event.target.name;
        const value = event.target.value;
        setData(data=>({...data,[name]:value}))
    }
    const onLogin = async(event) =>{
        event.preventDefault()
        let newUrl = url;
        if(currState==="Login"){
            newUrl += "/api/user/login"
        }
        else{
            newUrl +="/api/user/register"
        }
        const response = await axios.post(newUrl, data);
        if(response.data.success){
            setToken(response.data.token);
            localStorage.setItem("token", response.data.token)
            setShowLogin(false)
        }
        else {
            alert(response.data.message)
        }
    }
```



Add to Cart Functionality

The "Add to Cart" functionality is a key feature of the e-commerce application, allowing users to select food items and manage their shopping cart. Below is a detailed breakdown of how this functionality is implemented, including the relevant code snippets and explanations.

Component Structure

The "Add to Cart" functionality is typically integrated within the FoodItem component, which represents each food item displayed in the application. Here's a simplified structure of the FoodItem component that includes the "Add to Cart" functionality

Key Features and Functionality

- Context API Integration: The FoodItem component uses the StoreContext to access the cartItems state and the addToCart and removeFromCart functions. This allows the component to interact with the global cart state.
- Conditional Rendering: The component conditionally renders the "Add" button or the item counter based on whether the item is already in the cart. If the item is not in the cart, the "Add" button is displayed. If it is in the cart, the counter showing the quantity of the item is displayed along with buttons to increase or decrease the quantity.
- Event Handlers: The onClick event handlers for the "Add" and "Remove" buttons call the respective functions to update the cart state. This allows users to add items to the cart or adjust the quantity of items already in the cart.

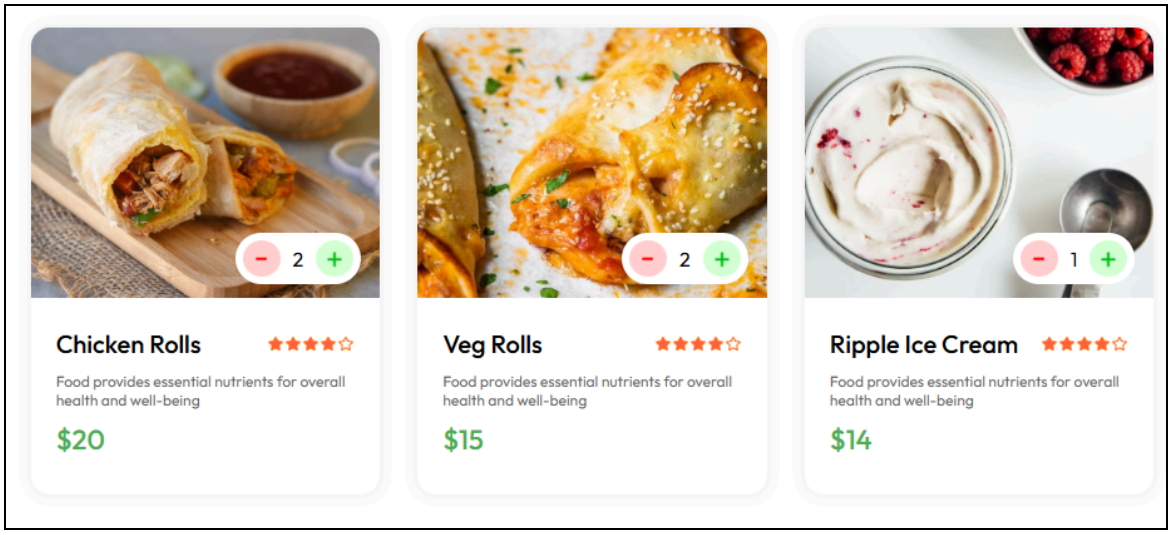
Add to Cart Function Implementation

The addToCart function is defined in the StoreContext and is responsible for updating the cart state.

Remove from Cart Function Implementation

Similarly, the removeFromCart function is responsible for decreasing the quantity of an item in the cart or removing it entirely if the quantity reaches zero.

```
const addToCart = (itemId) => {
    if (!cartItems[itemId]){
        setCartitems((prev)=>({...prev,[itemId]:1}))
    }
    else {
        setCartitems((prev)=>({...prev,[itemId]:prev[itemId]+1}))
    }
}

const removeFromCart = (itemId) => {
    setCartitems((prev)=>({...prev,[itemId]:prev[itemId]-1}))
}

const getTotalCartAmount = () => {
    let totalAmount = 0;
    for(const item in cartItems)
    {
        if (cartItems[item]>0){
            let itemInfo = food_list.find((product)=>product._id === item)
            totalAmount += itemInfo.price* cartItems[item];
        }
    }
    return totalAmount;
}
```



Food Display Functionality

The "Food Display" functionality is responsible for rendering a list of food items available for users to browse and select from. This feature is typically implemented in a component that fetches food data and displays it in a user-friendly format. Below is a detailed breakdown of how the Food Display functionality is implemented, including the relevant code snippets and explanations.

Component Structure

The FoodDisplay component is designed to display a list of food items based on the selected category. Here's a simplified structure of the FoodDisplay component

Key Features and Functionality

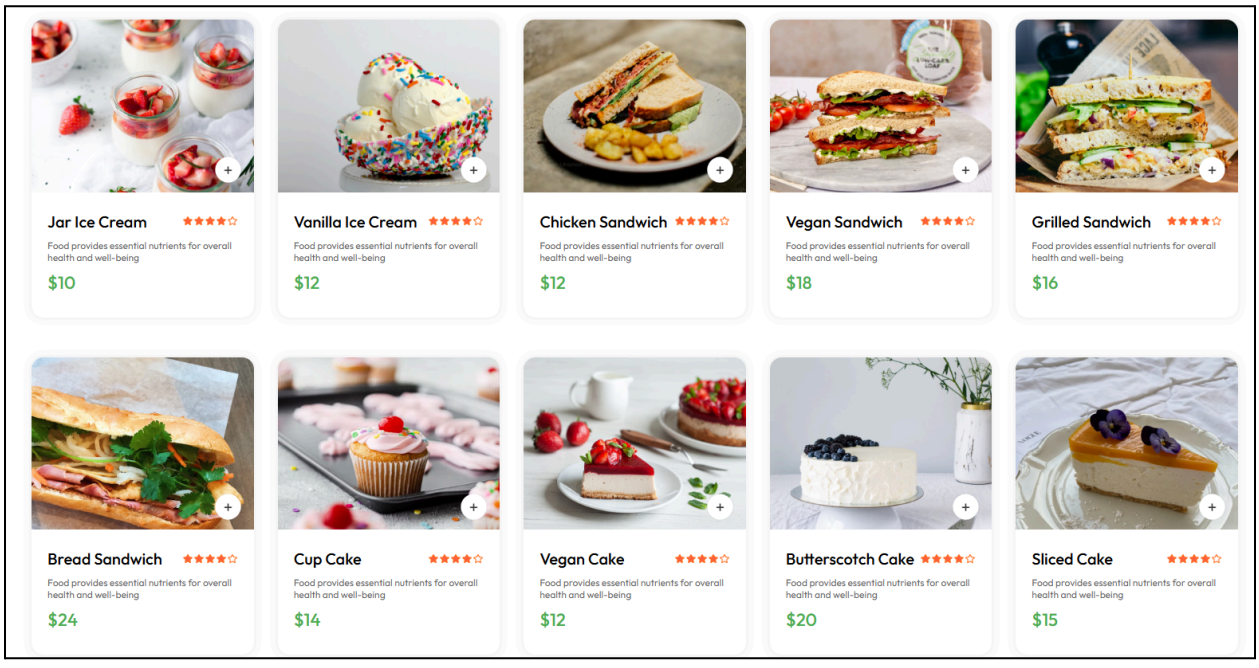
- Context API Integration: The FoodDisplay component uses the StoreContext to access the food_list, which contains all the food items available in the application. This allows the component to render the food items dynamically based on the data stored in the context.
- Category Filtering: The component accepts a category prop, which determines which food items to display. It filters the food_list based on the selected category:
- If the category is "All," it displays all food items.
- If a specific category is selected, it only displays items that belong to that category.

- Mapping Food Items: The map function is used to iterate over the food_list array. For each food item, a FoodItem component is rendered, passing relevant data such as id, name, description, price, and image as props.
- Key Prop: Each FoodItem component is assigned a unique key prop (using the index of the item in the array) to help React identify which items have changed, are added, or are removed. This improves performance and helps avoid issues with component state.

Food Item Data Structure

The food_list used in the FoodDisplay component typically contains objects with the following structure:

```
export const food_list = [
    {
        _id: "1",
        name: "Greek salad",
        image: food_1,
        price: 12,
        description: "Food provides essential nutrients for overall health and well-being",
        category: "Salad"
    },
    {
        _id: "2",
        name: "Veg salad",
        image: food_2,
        price: 18,
        description: "Food provides essential nutrients for overall health and well-being",
        category: "Salad"
    }, {
```



Backend Development

The backend of the application is built using Node.js and Express.js, providing a robust server environment for handling API requests and managing data interactions with a MongoDB database. Below is a detailed description of how the server, APIs, and database were implemented.

Server Setup

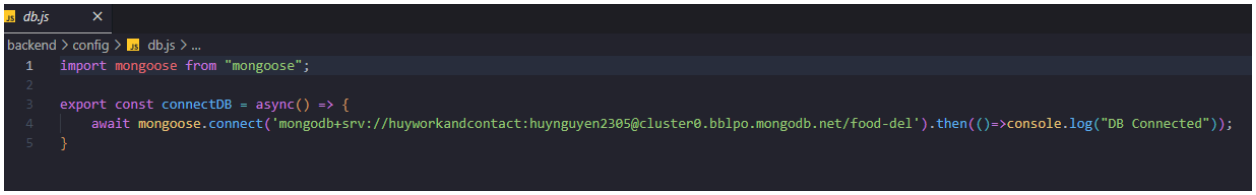
Node.js and Express.js: The backend server is created using Node.js, which allows for JavaScript to be used on the server side. Express.js is used as a web application framework to simplify the process of building the server and handling HTTP requests.
Server Configuration: The server is configured in a files (server.js) where the necessary packages are imported, middleware is set up, and API routes are defined

API Implementation

User Authentication APIs: The user authentication functionality is implemented through a dedicated router (userRoute.js). This includes endpoints for user registration and login

Controller Implementation

User Controller: The user controller (User Controller.js) contains the logic for handling user registration and login. It interacts with the database to create new users and validate existing users.



backend > models > userModel.js > ...

```
import mongoose from "mongoose";

const userSchema = new mongoose.Schema({
    name: {type:String,required:true},
    email:{type:String,required:true,unique:true},
    password:{type:String,required:true},
    cartData:{type:Object,default:{}}
},{minimize:false})

const userModel = mongoose.models.user || mongoose.model('user', userSchema);
export default userModel;
```

```
//login user
const loginUser = async (req,res) =>{
    const {email,password} = req.body;
    try {
        const user = await userModel.findOne({email})

        if (!user){
            return res.json({success:false,message:"User doesn't exist"})
        }

        const isMatch = await bcrypt.compare(password, user.password);

        if (!isMatch){
            return res.json({success:false,message:"Invalid password"})
        }

        const token = createToken(user._id);
        res.json({success:true,message:"Logged in successfully",token})

    } catch (error) {
        console.log(error);
        res.json({success:false,message:"Error logging in"})
    }
}

const createToken = (id) => {
    return jwt.sign({id},process.env.JWT_SECRET)
}
```

```js
import express from "express"
import { loginUser, registerUser } from "../controllers/UserController.js"

const userRouter = express.Router()

userRouter.post("/register", registerUser)
userRouter.post("/login", loginUser)

export default userRouter;
```

```js
//register user
const registerUser = async (req,res) => {
    const {name,password,email} = req.body;
    try {
        //checking is user exists
        const exis = await userModel.findOne({email})
        if(exis){
            return res.json({success:false, message:"User already exists"})
        }

        //validating email format and password
        if(!validator.isEmail(email)){
            return res.json({success:false, message:"Please enter valid email"})
        }

        if(password.length<5){
            return res.json({success:false,message:"Please enter a stronger password"})
        }

        //hashing user password
        const salt = await bcrypt.genSalt(10);
        const hashedPassword = await bcrypt.hash(password, salt);

        const newUser = new userModel({
            name:name,
            email:email,
            password:hashedPassword
        })

        const user = await newUser.save()
        const token = createToken(user._id)
        res.json({success:true, token});

    } catch (error) {
        console.log(error);
        res.json({success:false, message:"Error"})
    }
```

KEY FEATURES

User Authentication:

The application allows users to register and log in securely. User credentials are validated, and JSON Web Tokens (JWT) are used to manage user sessions, ensuring secure access to protected routes.

Food Item Management:

Users can browse a variety of food items categorized for easy navigation. Each food item displays relevant information, including name, description, price, and an image. Users can add items to their cart and manage their selections.

Shopping Cart Functionality:

Users can add food items to their cart, view the cart contents, and adjust quantities. The cart state is managed globally, allowing for a seamless shopping experience.

Responsive Design:

The application is designed to be responsive, ensuring that it functions well on both desktop and mobile devices. CSS media queries and flexible layouts are used to adapt the UI to different screen sizes.

Data Visualization:

The application provides a clear and organized display of food items, allowing users to easily browse and select their desired options. The use of images and descriptions enhances the user experience.

Error Handling and User Feedback:

The application includes error handling for user authentication and data submission, providing users with feedback on their actions (e.g., successful login, registration errors).

TESTING AND DEBUGGING

Testing with Postman

Postman is a powerful tool for testing APIs, allowing developers to send requests to endpoints and inspect responses. It is particularly useful for testing RESTful APIs in a web application.

1. **Creating Requests:** Postman supports various HTTP methods such as GET, POST, PUT, DELETE, etc. You can select the appropriate method based on the API endpoint you are testing.
2. **Setting Up Requests:**
    - Enter the API endpoint URL in the request field.
    - Select the HTTP method
3. **Testing User Registration API**
    - Select POST Method: Choose POST from the dropdown menu.
    - Enter URL: Enter the URL for the registration endpoint, e.g., http://localhost:5137/api/user/register.
    - Set Headers: If required, set the Content-Type header to application/json.
    - Add Request Body: In the "Body" tab, select "raw" and choose "JSON" from the dropdown.
    - Send Request: Click the "Send" button to submit the request.
    - Inspect Response: Check the response section for the status code, response body, and headers. A successful registration might return a status code of 201 and a success message.

Testing Different Scenarios:

1. **Valid and Invalid Inputs**: Test the API with both valid and invalid inputs to ensure proper validation
2. **Authentication:** For endpoints that require authentication, include the JWT token in the headers
3. **CRUD Operations:** Test all CRUD operations (Create, Read, Update, Delete) for your resources. For example:
    - GET request to retrieve food items: http://localhost:5137/api/food
    - PUT request to update a food item: http://localhost:5137/api/food/:id
    - DELETE request to remove a food item: http://localhost:5137/api/food/:id

Using Postman Collections

1. **Creating Collections:** Organize your requests into collections for better management. You can create a collection for user-related requests and another for food-related requests.
2. **Running Collections:** Use the "Runner" feature in Postman to run a series of requests in a collection. This is useful for testing workflows or sequences of API calls

Testing Results and Debugging

1. **Response Validation:**
    - Check the response status codes to ensure they match expected outcomes (e.g., 200 for success, 400 for bad requests).
    - Validate the response body to ensure it contains the expected data structure and values.
2. **Debugging:** If an API call fails, inspect the response body for error messages. This can provide insights into what went wrong (e.g., validation errors, server issues).

Challenges and Solutions

- CORS Errors: If you encounter CORS (Cross-Origin Resource Sharing) errors, ensure that your backend server is configured to allow requests from your Postman client.
- Authentication Issues: Ensure that you are sending the correct authentication tokens and that your backend is set up to validate them properly.

Resolving Issues:

- Use the console in Postman to view request and response details, which can help identify issues with headers, body formatting, or endpoint URLs.
- Using Postman for testing your API endpoints is an effective way to ensure that your application functions as expected. By systematically testing different scenarios, validating responses, and organizing requests into collections, you can streamline the testing process and quickly

DEPLOYMENT

Hosting

Currently, the application is hosted locally on your development machine using Node.js and npm. The server runs on (`localhost`) at port 5137, allowing it to test and develop the application in a local environment. This setup is ideal for development and testing purposes but is not suitable for production deployment.

Deployment Process

1. **Build the Application:** For a Node.js application, building typically involves ensuring that all dependencies are installed and that the application is ready to run: This command installs all the required packages listed in the package.json file

```
npm install
```

2. **Test the Application:** Before running the application, it is essential to test it to ensure that everything is functioning correctly. This can include running unit tests, integration tests, and manual testing using tools like Postman

```
npm test
```

3. **Run the Application:** To start the application locally, you can use the following command

```
npm run server
```

   This command typically uses a script defined in the package.json file (e.g., using nodemon to run server.js), which starts the server and listens for incoming requests on (`localhost:5137`).

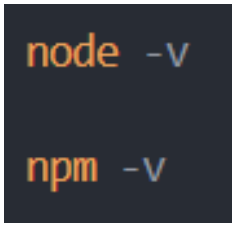
   Currently, the application is hosted locally for development and testing purposes. The deployment process involves building, testing, and running the application

using npm scripts. While there is no CI/CD pipeline in place, implementing such a system would be beneficial for future production deployments, allowing for automated testing and deployment processes to ensure a smooth workflow.

User Documentation

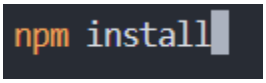
To run the project locally:

1. **Prerequisites**: Ensure you have Node.js installed on your machine. You can verify the installation by running in terminal:

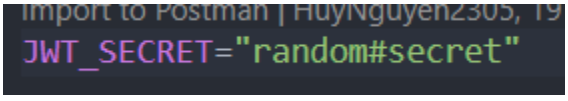


2. **Clone the Repository**: Clone the project repository from GitHub (or your version control system) to your local machine:

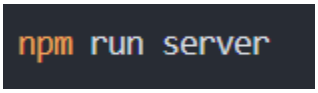
3. **Install Dependencies**: Navigate to the project directory and install the required dependencies

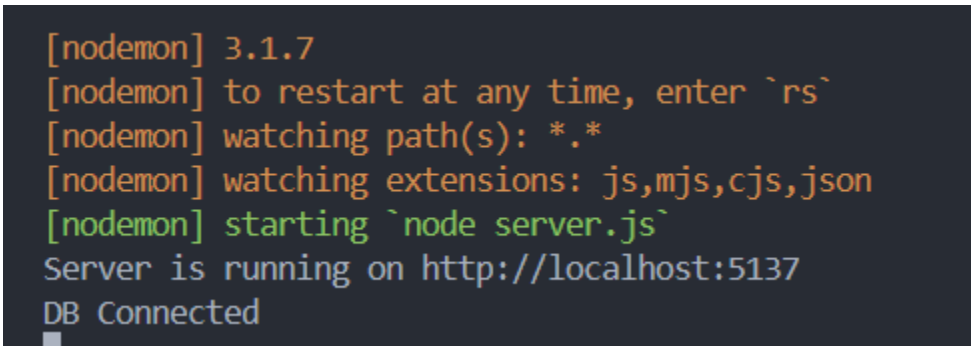


4. **Set Up Environment Variables**: Create a .env file in the root of the project directory and add any necessary environment variables



5. **Run the Application**





Usage Instructions

1. **Accessing the Application:**
   - Open your web browser and navigate to http://localhost:5137 to access the application.
2. **User Registration:**
   - To create a new account, navigate to the registration page (if applicable) and fill out the registration form with your name, email, and password. Click the "Register" button to create your account.
3. **Logging In:**
   - After registering, you can log in using your email and password. Enter your credentials in the login form and click the "Login" button.
4. **Browsing Food Items:**
   - Once logged in, you can browse the available food items displayed on the main page. You can filter items by category if applicable.
5. **Adding Items to Cart:**
   - Click on the "Add" button next to a food item to add it to your cart. You can adjust the quantity of items in your cart as needed.
6. **Viewing Cart:**
   - Navigate to the cart page to view the items you have added. You can remove items or proceed to checkout from this page.
7. **Checkout Process:**
   - Follow the prompts to enter your delivery information and complete your order.

RESULT AND DISCUSSIONS

Outcome

The primary outcome of the project was the successful development of a comprehensive web application designed specifically to facilitate the ordering of food and ensure timely delivery to users. This application effectively meets the core objectives that were established at the outset of the project, which included creating a user-friendly interface that allows customers to easily browse through a diverse selection of food items, manage their shopping cart with ease, and navigate through a streamlined checkout process. By focusing on user experience and functionality, the project has achieved its overarching goal of providing a reliable and efficient platform for food ordering, thereby enhancing the convenience of accessing food delivery services for users.

Features Implemented

Throughout the development process, several key features were successfully implemented in the application, contributing to its overall functionality and user experience. These features include:

1. **User Authentication:**
    - The application includes a robust user authentication system that allows users to create accounts, log in securely, and manage their personal information. This feature ensures that user data is protected and that only authorized individuals can access their accounts.
2. **Food Item Browsing:**
    - Users can browse a wide variety of food items that are categorized for easy navigation. This feature enhances the user experience by allowing customers to quickly find their desired food options without unnecessary hassle.
3. **Shopping Cart Functionality:**
    - The application provides a comprehensive shopping cart functionality that enables users to add food items to their cart, adjust quantities as needed, and view their cart contents at any time. This feature is crucial for ensuring that users can easily manage their selections before proceeding to checkout.

4. **Checkout Process:**
    - A streamlined checkout process has been implemented, allowing users to enter their delivery information and complete their orders efficiently. This feature is designed to minimize friction during the purchasing process, encouraging users to finalize their orders without encountering unnecessary obstacles.

Despite these successful implementations, it is important to note that the project is still missing some valuable functionalities that could significantly enhance the user experience. For instance, the application currently lacks a feature that allows users to check their order history, which would enable them to view past orders and reorder items with ease. Additionally, there is no functionality for administrators to add new food items to the database, which would limit the ability to keep the menu fresh and up-to-date. Implementing these features in future iterations of the project would greatly improve its overall utility and appeal.

Performance

In terms of performance, the application has been running smoothly during testing, with quick load times and responsive interactions. The user interface is designed to be

lightweight, which contributes to a seamless experience when navigating through the various features. Scalability is also a consideration for the application, as it has been built

with a modular architecture that allows for easy integration of additional features and functionalities in the future. As the user base grows, the application is expected to handle increased traffic without significant degradation in performance, provided that appropriate server resources are allocated. Overall, the performance metrics observed during testing indicate that the application is well-equipped to meet user demands effectively.

Future Work

As the application continues to evolve, there are several potential improvements and additional features that could significantly enhance its functionality and user experience. Some of these suggestions include:

1. **Order History Feature**:
    - Implementing an order history feature would allow users to view their past orders, making it easier for them to reorder their favorite items. This feature could include details such as order date, items purchased, and total cost, providing users with a comprehensive overview of their ordering habits.
2. **Admin Dashboard:**
    - Developing an admin dashboard would empower administrators to manage the food items in the database effectively. This dashboard could include functionalities for adding, editing, and removing food items, as well as monitoring orders and user activity. Such a feature would streamline the management process and ensure that the menu remains current and appealing.
3. **User Reviews and Ratings:**
    - Allowing users to leave reviews and ratings for food items would enhance the community aspect of the application. This feature would provide valuable feedback to both users and administrators, helping to improve food quality and service based on user experiences.
4. **Promotions and Discounts:**
    - Integrating a system for promotions and discounts could incentivize users to place orders. This could include features such as coupon codes, seasonal promotions, or loyalty rewards for frequent customers, encouraging repeat business and enhancing user engagement.
5. **Real-Time Order Tracking:**
    - Implementing real-time order tracking would provide users with updates on the status of their orders, from preparation to delivery. This feature could enhance user satisfaction by keeping them informed and reducing uncertainty during the waiting period.

Scalability

To ensure that the application can scale effectively to handle more users and data, several strategies can be employed:

1. **Cloud Hosting Solutions:**
    - Transitioning to a cloud hosting solution, such as AWS, Google Cloud, or Azure, would provide the necessary infrastructure to handle increased traffic and data storage. These platforms offer scalable resources that can be adjusted based on demand, ensuring optimal performance during peak usage times.
2. **Database Optimization:**
    - Implementing database optimization techniques, such as indexing and query optimization, would improve the efficiency of data retrieval and storage. This would be particularly important as the volume of user data and food items grows, ensuring that the application remains responsive.
3. **Load Balancing:**
    - Utilizing load balancing techniques would distribute incoming traffic across multiple servers, preventing any single server from becoming a bottleneck. This approach would enhance the application's ability to handle a larger number of concurrent users without compromising performance.
4. **Microservices Architecture:**
    - Adopting a microservices architecture would allow different components of the application to be developed, deployed, and scaled independently. This modular approach would facilitate easier updates and maintenance, as well as improve overall system resilience.

Possible Enhancements

Looking ahead, there are several exciting ideas for enhancing the application in the future:

1. **Mobile App Integration:**
    - Developing a mobile application for iOS and Android would provide users with a more convenient way to access the food ordering service. A mobile app could leverage device features such as push notifications for order updates and location services for delivery tracking.
2. **Advanced AI Features**:
    - Integrating advanced AI features could enhance the user experience significantly. For example, implementing a recommendation system that suggests food items based on user preferences and past orders could personalize the shopping experience. Additionally, AI chatbots could provide customer support, answering common queries and assisting users in real-time.
3. **Social Media Integration:**
    - Allowing users to share their favorite food items or orders on social media platforms could increase visibility and attract new users. This feature could also include referral programs that reward users for bringing in new customers.
4. **Nutritional Information:**
    - Providing detailed nutritional information for each food item could cater to health-conscious users. This feature would allow users to make informed choices based on their dietary preferences and restrictions.
5. **Multi-Language Support:**
    - Implementing multi-language support would make the application accessible to a broader audience, accommodating users from different linguistic backgrounds and enhancing inclusivity.

By considering these suggestions for improvement, scalability strategies, and possible enhancements, the application can continue to grow and adapt to meet the evolving needs of its users, ensuring long-term success and user satisfaction.

REFERENCES

https://github.com/HuyNguyen2305/food-del.git

https://www.w3schools.com

https://youtu.be/V2AE0YS8WYs?si=NGHh68kwdELTEfjD

https://youtu.be/9jRTo7ILxQc?si=MSyLyw3RmILfni9N

https://youtu.be/vS8UA8n6-ic?si=DZRdG2_m88FZwiPn

https://www.mongodb.com/?fbclid=IwZXh0bgNhZW0CMTEAAR3GqB3GvySruXQMBy9EUgilCWJKG3x9rD3hGVZCKh1R0Es8DHGs0XsIPrE_aem_5IHj0MrDK4Bfg1LY-Lr8QA

https://www.postman.com/category/e-commerce-apis?fbclid=IwZXh0bgNhZW0CMTEAAR2066Nie3QIIYOHH99EK_63tcGKpkDIoZrm1AvYSwfxyrlXwzrxVt3SXnk_aem_wzZT7FEtEKiz3ZEdPmZM9g