



# **NAAN MUDHALVAN FULL STACK MERN DEVELOPMENT- “GROCERY WEBAPP”**

## **A PROJECT REPORT**

*Submitted by*

**AKSHAYA SHREE.B (310821104007)**

**AROKIA ANCY.B (310821104012)**

**JOTHIKA.K(310821104042)**

**MONOLISA.P (310821104060)**

*in partial fulfillment for the award of the degree*

*of*

**BACHELOR OF ENGINEERING**

*in*

**COMPUTER SCIENCE AND ENGINEERING**

**JEPPIAAR ENGINEERING COLLEGE, CHENNAI**

ANNA UNIVERSITY: CHENNAI 600 025

NOVEMBER 2024

**ANNA UNIVERSITY: CHENNAI 600 025**

**BONAFIDE CERTIFICATE**

Certified that this project report **NAAN MUDHALVAN FULL STACK MERN DEVELOPMENT GROCERY WEBAPP** is the Bonafide work of **AKSHAYA SHREE.B, AROKIA ANCY.B, JOTHIKA.K, MONOLISA.P** who carried out the project work under my supervision.

**SIGNATURE**

DR. J. Anitha Gnanaselvi  
**HEAD OF THE DEPARTMENT**

CSE DEPARTMENT

JEPPIAAR ENGINEERING COLLEGE,  
CHENNAI- 600 119

**SIGNATURE**

Ms. D. Jeevitha  
**SUPERVISOR**

Assistant Professor

CSE DEPARTMENT

JEPPIAAR ENGINEERING COLLEGE,  
CHENNAI- 600 119

## TABLE OF CONTENTS

1. Introduction .....	5
2. Project Overview .....	5
3. Features .....	5
4. Architecture .....	6
5. ER Diagram .....	7
6. Setup Instructions .....	7
7. Folder Structure .....	9
8. Running the Application .....	10
9. API Documentation .....	12
10. Authentication .....	12
11. User Interface .....	12
12. Testing .....	27
13. Technical Issues .....	28
14. Future Improvements for Grocery Web App .....	29
15. Appendices .....	31

# GROCERY WEBAPP USING MERN STACK

## 1)Introduction:

our basic grocery-web app! Our app is designed to provide a seamless online shopping experience for customers, making it convenient for them to explore and purchase a wide range of products. Whether you are a tech enthusiast, a fashionista, or a homemaker looking for everyday essentials, our app has something for everyone.

With user-friendly navigation and intuitive design, our grocery-webapp app allows customers to browse through various categories, view product details, add items to their cart, and securely complete the checkout process. We prioritize user satisfaction and aim to provide a smooth and hassle-free shopping experience.

For sellers and administrators, our app offers robust backend functionalities. Sellers can easily manage their product listings, inventory, and orders, while administrators can efficiently handle customer inquiries, process payments, and monitor overall app performance.

## 2)Project Overview:

To develop a user-friendly web application for grocery shopping where users can:

- Browse products
- Add items to a cart
- Proceed to checkout
- Manage their profiles (optional)
- View their order history

## 3)Features:

### *User Features:*

1. **Homepage:**
  - Display popular categories or featured products.
  - Search bar to find specific products.
2. **Product Listing:**
  - Browse a list of products with filtering options (e.g., category, price range).
3. **Product Details:**
  - View detailed information about a product (e.g., name, price, description, image, category).
4. **Cart Management:**
  - Add/remove products to/from the cart.
  - View cart summary with total price.
5. **Checkout:**
  - Place an order after adding items to the cart.
  - Enter shipping details.
6. **User Authentication:**
  - Sign-up, login, and logout functionality using **JWT**.
  - Protected routes for logged-in users.
7. **Order History:**
  - View past orders and their details.

### *Admin Role :*

Responsibilities: The admin role has full control and administrative privileges over the system.

Permissions:

- Manage: Admins can add, edit, and delete shop information along with products.
- Manage product bookings: Admins can view and manage all product bookings made by users and agents, including canceling or modifying product bookings.
- Manage users: Admins can create, edit, and delete user accounts, as well as manage their roles and permissions.
- Generate reports: Admins have access to generate reports and analytics on product booking details, booking counts, and sales reports.

*User Role :*

Responsibilities: Users are the customers of the online shopping web application who can search for products, and make product bookings.

Permissions:

- View products: Users can search for products, based on interest.
- Product bookings: Users can select products that are available and complete the order process.
- Manage product booking: Users can view their own product order bookings, modify booking details, track booking details, and cancel their bookings
- Manage cart: Users can view their cart details and modify them if needed.

## 4)Architecture:

The grocery webapp has a simple architecture which shows a perfect flow of the application.

1. Frontend:

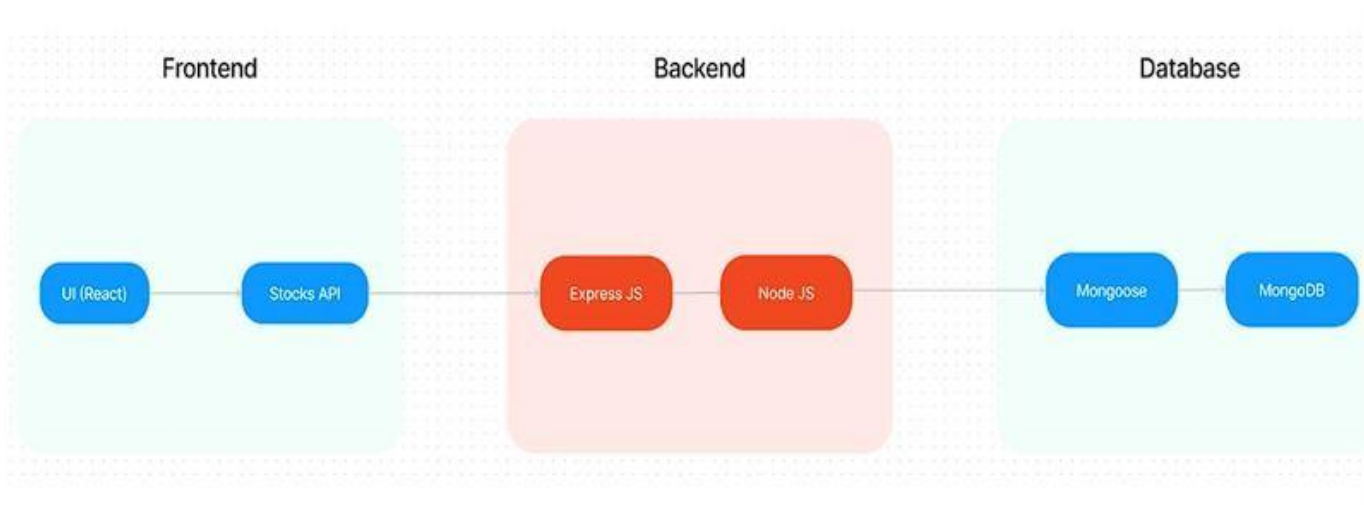
The frontend is developed using React.js, ensuring a dynamic and responsive user interface with efficient rendering of components and stock Api.

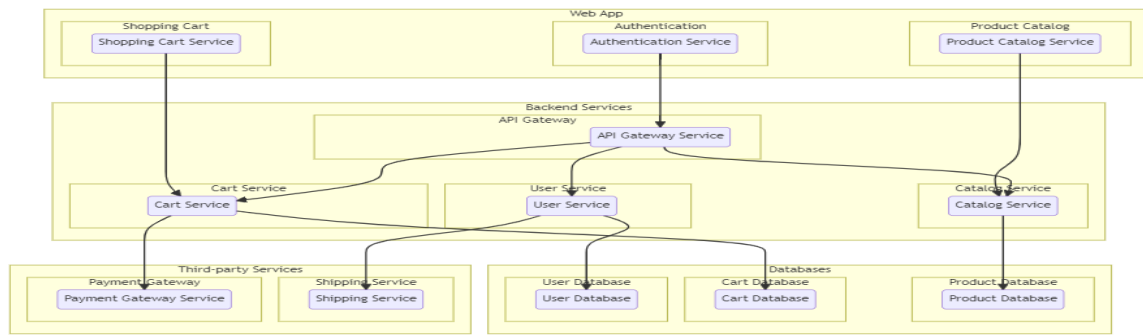
2. Backend:

The backend is built using Node.js and Express.js to provide RESTful APIs for various features such as user authentication, order processing, and payment handling.

3. Database:

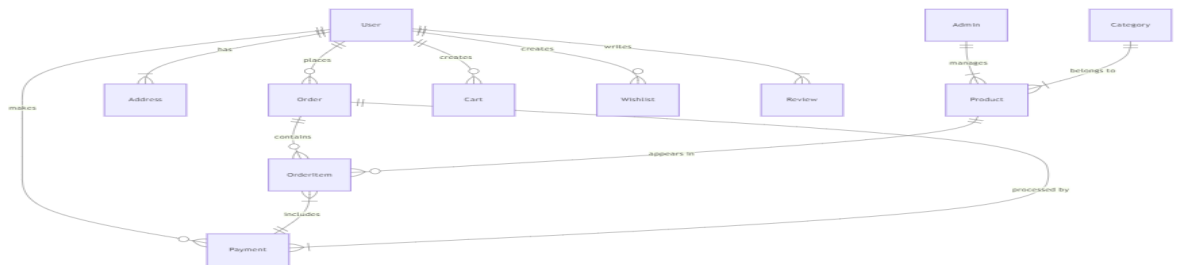
The database used for the system uses MongoDB to store and manage user data, menu items, orders, and payment information of the app.





## 5)ER Diagram:

The Entity-Relationship (ER) diagram for an flower and gift delivery app visually represents the relationships between different entities involved in the system, such as users, products, orders, and reviews. It illustrates how these entities are related to each other and helps in understanding the overall database structure and data flow within the application.



## 6)Setup Instructions:

### Pre-requisites

To develop a full-stack Grocery web app using AngularJS, Node.js, and MongoDB, there are several prerequisites you should consider. Here are the key prerequisites for developing such an application:

**Node.js and npm:** Install Node.js, which includes npm (Node Package Manager), on your development machine. Node.js is required to run JavaScript on the server side.

- Download: <https://nodejs.org/en/download/>
- Installation instructions: <https://nodejs.org/en/download/package-manager/>

**MongoDB:** Set up a MongoDB database to store hotel and booking information. Install MongoDB locally or use a cloud-based MongoDB service.

- Download: <https://www.mongodb.com/try/download/community>
- Installation instructions: <https://docs.mongodb.com/manual/installation/>

**Express.js:** Express.js is a web application framework for Node.js. Install Express.js to handle server-side routing, middleware, and API development.

- Installation: Open your command prompt or terminal and run the following command: `npm install express`

Angular: Angular is a JavaScript framework for building client-side applications. Install Angular CLI (Command Line Interface) globally to create and manage your Angular project.

Install Angular CLI:

- Angular provides a command-line interface (CLI) tool that helps with project setup and development.
- Install the Angular CLI globally by running the following command:  
`npm install -g @angular/client`

Verify the Angular CLI installation:

- Run the following command to verify that the Angular CLI is installed correctly: `ng version`

You will see the version of the Angular CLI printed in the terminal if the installation was successful.

Create a new Angular project:

- Choose or create a directory where you want to set up your Angular project.
- Open your terminal or command prompt.
- Navigate to the selected directory using the `cd` command.
- Create a new Angular project by running the following command: `ng new client` Wait for the project to be created:
- The Angular CLI will generate the basic project structure and install the necessary dependencies

Navigate into the project directory:

- After the project creation is complete, navigate into the project directory by running the following command: `cd client`

Start the development server:

- To launch the development server and see your Angular app in the browser, run the following command: `ng serve / npm start`
- The Angular CLI will compile your app and start the development server.
- Open your web browser and navigate to `http://localhost:4200` to see your Angular app running.

You have successfully set up Angular on your machine and created a new Angular project. You can now start building your app by modifying the generated project files in the `src` directory.

Please note that these instructions provide a basic setup for Angular. You can explore more advanced configurations and features by referring to the official Angular documentation: <https://angular.io>

HTML, CSS, and JavaScript: Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.

Database Connectivity: Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongoose to connect your Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations.

Front-end Framework: Utilize Angular to build the user-facing part of the application, including products listings, booking forms, and user interfaces for the admin dashboard.

Version Control: Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

- Git: Download and installation instructions can be found at: <https://git-scm.com/downloads>

Development Environment: Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

- Visual Studio Code: Download from <https://code.visualstudio.com/download>
- Sublime Text: Download from <https://www.sublimetext.com/download>
- WebStorm: Download from <https://www.jetbrains.com/webstorm/download>

To Connect the Database with Node JS go through the below provided link:

- Link: <https://www.section.io/engineering-education/nodejs-mongoosejs-mongodb/>

## Git Repository Cloning

**To run the grocery-web app project downloaded from github:**

**Follow below steps:**

### 1. Clone the Repository:

- Open your terminal or command prompt.
- Navigate to the directory where you want to store the grocery-webapp app.
- Execute the following command to clone the repository:

**git clone** [https://github.com/Akshayashree-ai/naanmudhalvan\\_grocerywebapp\\_mern](https://github.com/Akshayashree-ai/naanmudhalvan_grocerywebapp_mern)

### 2. Install Dependencies:

- Navigate into the cloned repository directory:

**cd** grocery\_project

- Install the required dependencies by running the following command:

**npm install**

### 3. Start the Development Server:

- To start the development server, execute the following command:

**npm run dev** or **npm run start**

- The e-commerce app will be accessible at <http://localhost:3000> by default. You can change the port configuration in the .env file if needed.

### 4. Access the App:

- Open your web browser and navigate to <http://localhost:3000>.
- You should see the grocery-webapp app's homepage, indicating that the installation and setup were successful.

**Project Repository Link:** [https://github.com/Akshayashree-ai/naanmudhalvan\\_grocerywebapp\\_mern](https://github.com/Akshayashree-ai/naanmudhalvan_grocerywebapp_mern)



## 7) Folder Structure:

```
grocery_project/
├── backend/
│   ├── node_modules/ # Backend dependencies
│   ├── .env # Environment variables for backend
│   ├── index.js # Main entry point for backend (Express server)
│   ├── package-lock.json # Lock file for backend dependencies
│   └── package.json # Backend dependencies and scripts
├── frontend/
│   ├── node_modules/ # Frontend dependencies
│   ├── public/
│   │   ├── favicon.ico # Website favicon
│   │   ├── index.html # Main HTML file
│   │   ├── logo192.png # Logo for PWA
│   │   ├── logo512.png # Larger logo for PWA
│   │   ├── manifest.json # Manifest file for PWA configuration
│   │   └── robots.txt # Rules for web crawlers
│   └── src/
│       ├── assets/ # Static assets
│       │   ├── empty.gif # Placeholder image
│       │   ├── login-animation.gif # Login animation GIF
│       │   └── logo.png # App logo
│       ├── components/ # Reusable UI components
│       │   ├── AllProduct.js # Component for displaying all products
│       │   ├── CardFeature.js # Component for product features
│       │   ├── CartProduct.js # Component for cart products
│       │   ├── FilterProduct.js # Component for filtering products
│       │   ├── Header.js # Header component
│       │   └── HomeCard.js # Home page card component
│       ├── pages/ # Pages for routing
│       │   ├── About.js # About page
│       │   ├── Cancel.js # Cancel page (payment or order)
│       │   ├── Cart.js # Cart page
│       │   ├── Contact.js # Contact page
│       │   ├── Home.js # Home page
│       │   ├── Login.js # Login page
│       │   ├── Menu.js # Product menu page
│       │   ├── NewProduct.js # Add new product page
│       │   ├── Signup.js # Signup page
│       │   └── Success.js # Success page (e.g., order confirmation)
│       ├── redux/ # Redux store and slices
│       │   ├── index.js # Combines Redux slices
│       │   ├── productSlide.js # Redux slice for product state
│       │   └── userSlice.js # Redux slice for user state
│       ├── utility/ # Utility scripts
│       │   └── ImageToBase64.js # Helper to convert images to Base64
│       ├── App.css # Global styles for the app
│       ├── App.js # Main React component
│       ├── App.test.js # Test file for App.js
│       ├── index.css # Global styles
│       ├── index.js # React DOM entry point
│       ├── logo.svg # SVG logo
│       ├── reportWebVitals.js # Reports app performance metrics
│       └── setupTests.js # Test setup file
├── .env # Environment variables for frontend
├── package-lock.json # Lock file for frontend dependencies
├── package.json # Frontend dependencies and scripts
├── README.md # Documentation for the project
├── tailwind.config.js # TailwindCSS configuration
├── .gitattributes # Git attributes configuration
└── README.md # Project overview (duplicated)
```

## 8) Running the Application

To set up and run the **Grocery Web App** locally, follow these steps:

### *Prerequisites:*

- Ensure that **Node.js** and **npm** (Node Package Manager) are installed on your system.
- Confirm that **MongoDB** is installed and running locally, or use **MongoDB Atlas** for a cloud database.
- Clone the project repository to your local machine.

### **Step 1: Set Up the Frontend**

#### **1. Navigate to the Frontend Directory:**

- Open a terminal window.
- Change your current directory to the frontend folder:  
“cd grocery\_project/frontend”

#### **2. Install Frontend Dependencies:**

Run the following command to install all required npm packages for the frontend: “npm install”

#### **3. Start the Frontend Development Server:**

Once dependencies are installed, start the frontend server by running: “npm start”

#### **4. Verify the Frontend:**

- Check that the UI loads correctly, and all components render without errors.
- Ensure there are no errors in the browser console.

### **Step 2: Set Up the Backend**

#### **1. Navigate to the Server Directory:**

- Open a new terminal window.
- Change your current directory to the server folder:

“cd grocery\_project/backend”

#### **2. Install Backend Dependencies:**

Run the following command to install all necessary npm packages for the backend: “npm install”

#### **3. Configure Environment Variables:**

If your application requires environment variables, create a `.env` file in the `Server` directory with necessary configurations like MongoDB URI, JWT secret, and other API keys. Example:

```
“PORT=3000
MONGODB_URI=mongodb://localhost:3000/groceryproject
JWT_SECRET=your_jwt_secret_key”
```

#### **4. Start the Backend Server:**

Start the backend server by running:  
“npm start”

#### **5. Verify the Backend:**

- Check that the backend server starts without errors.
- You should see a message in the terminal indicating that the server is running and connected to the database.

### **Step 3: Access the Application**

- Frontend (React Application):

Open <http://localhost:3000> (on another if specified) in your web browser to interact with the frontend UI.

- Backend (API Server):

Ensure that the backend server is running and can be accessed via <http://localhost:5000> (or another port if specified).

#### Step 4: Testing End-to-End Functionality

1. **Register or Log In:**
  - Test the authentication by registering a new user or logging in with existing credentials.
2. **Browse Products:**
  - Browse the list of grocery items and add products to the cart.
3. **Place an Order:**
  - Test placing an order and verifying that the order is correctly submitted.
4. **Admin Features (if applicable):**
  - Use the admin account (if available) to manage products and orders.

#### Troubleshooting:

- **Port Conflicts:**  
If you encounter port conflicts, change the port in the respective configuration files. For example:
  - Change the frontend port in `package.json` (if necessary).
  - Change the backend port in the `index.js` or `.env` file.
- **Database Connection Issues:**  
Ensure that MongoDB is running and accessible. If using a cloud database (e.g., MongoDB Atlas), verify the URI in the `.env` file.

## 9)API Documentation:

Endpoints Overview:

```
`POST /api/users/register` User registration
`POST /api/users/login` User login
`GET /api/menu` Get menu items
`POST /api/orders` Place an order
`GET /api/orders/: userId` Retrieve user orders
```

Example

Response:

```
json
{
  "success": true,
  "message": "Order placed
  successfully", "orderId": "12345"
}
```

## 10)Authentication:

- The app uses JWT (JSON Web Tokens) for user authentication and authorization.
- Tokens are generated upon login and stored in the client's local storage for subsequent API calls.

## 11)User Interface:

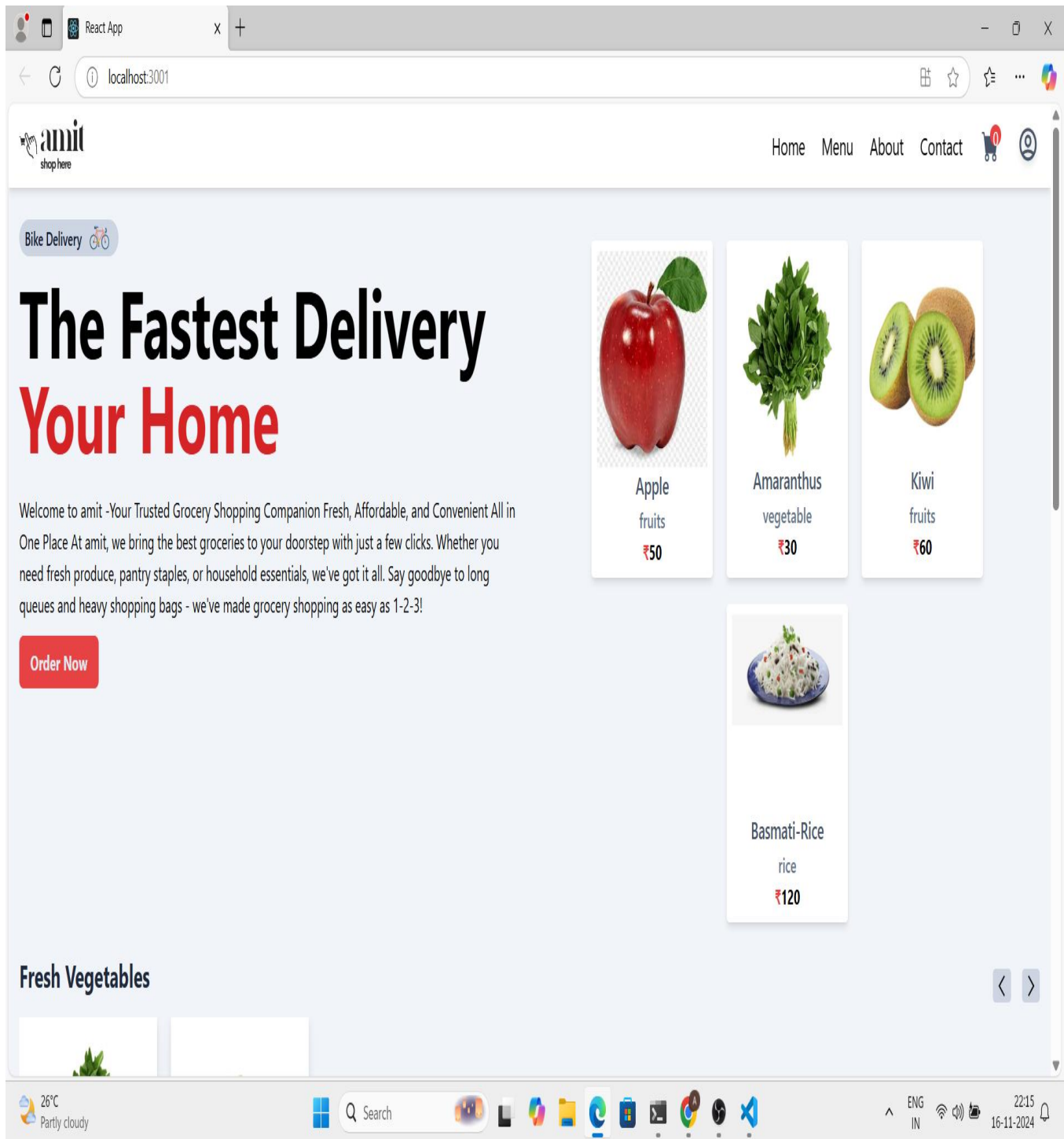
The User Interface or, UI, is the layout or window of the application. The UI of the app consists of:

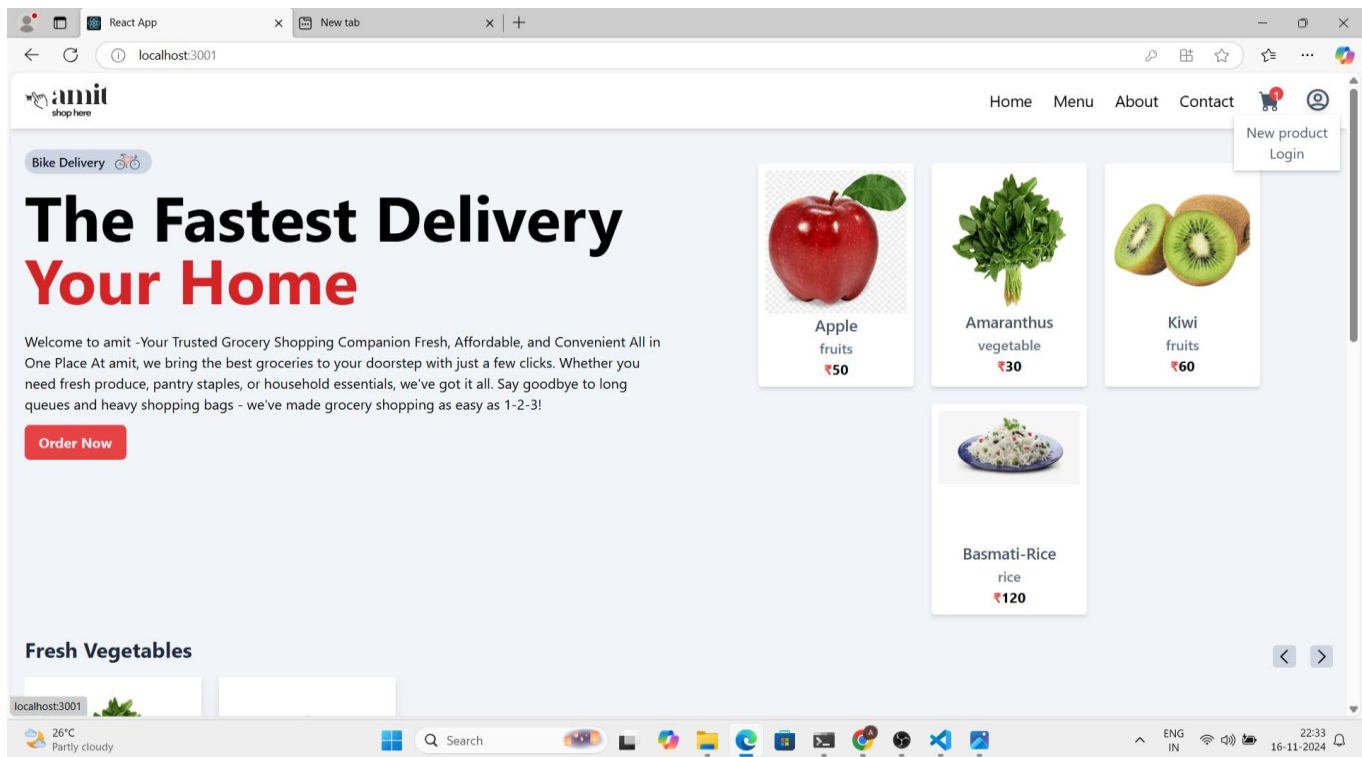
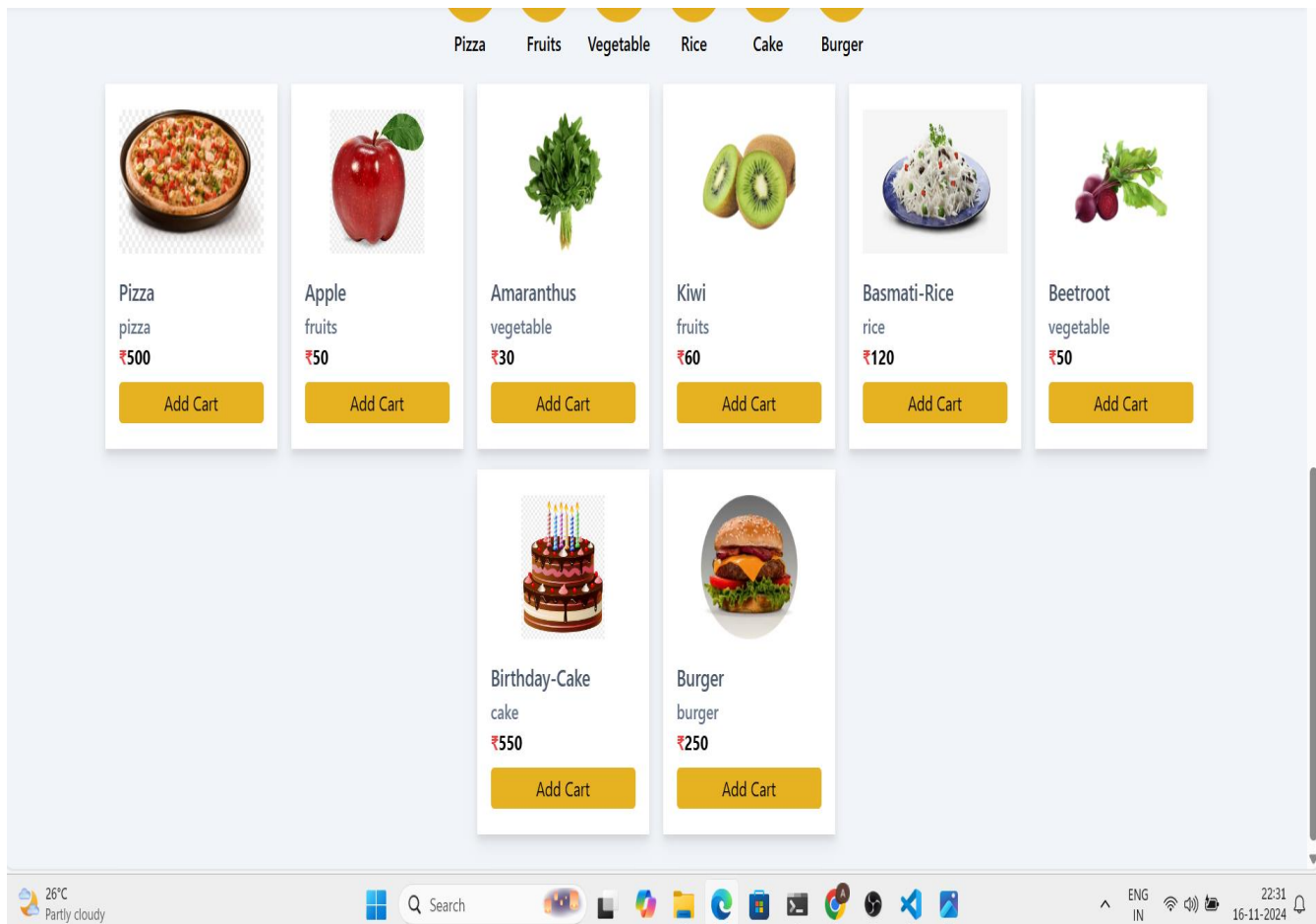
- A homepage with grocery items
- Menu pages with filtering options
- Cart and checkout

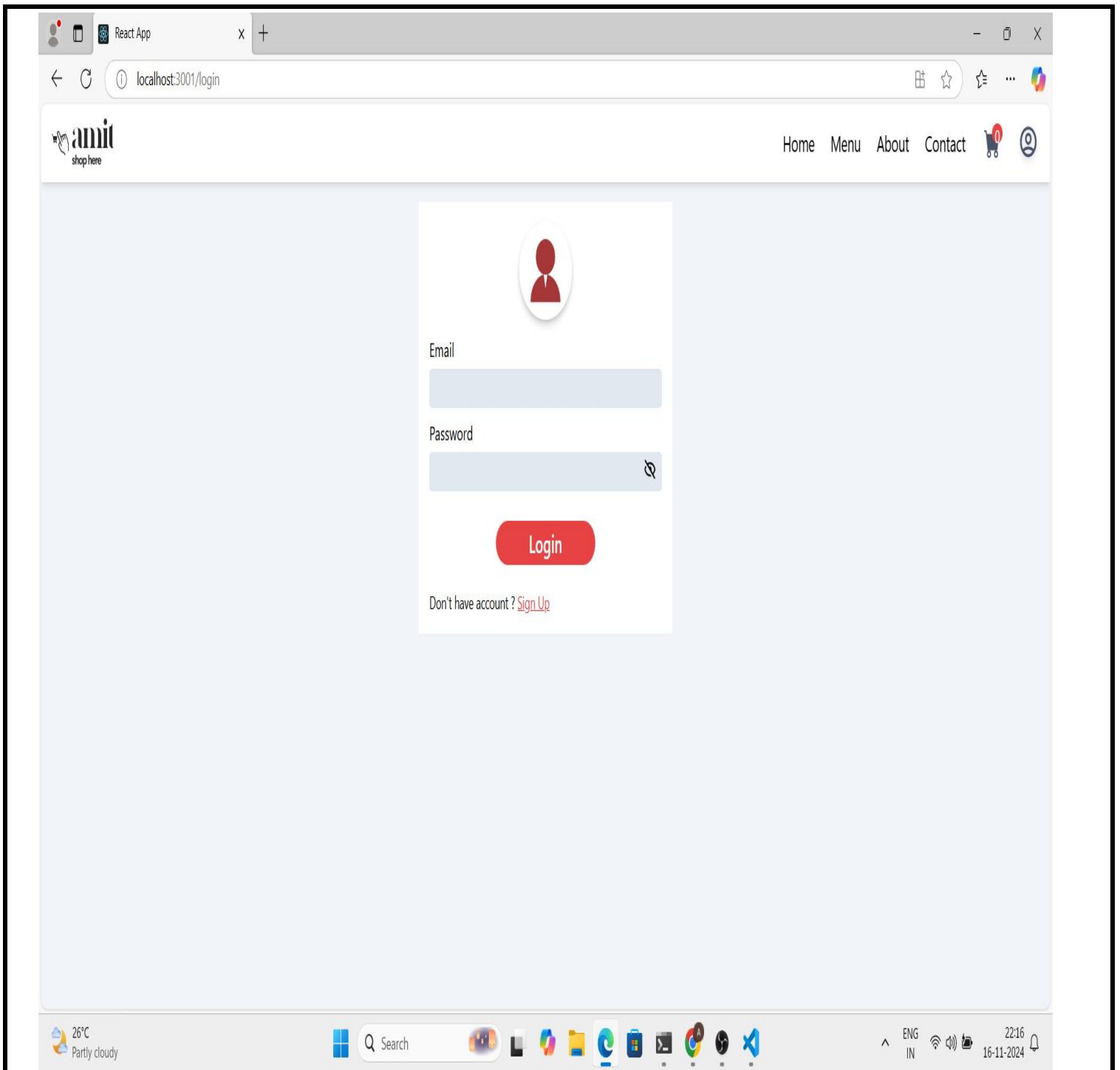
pagesScreenshots:

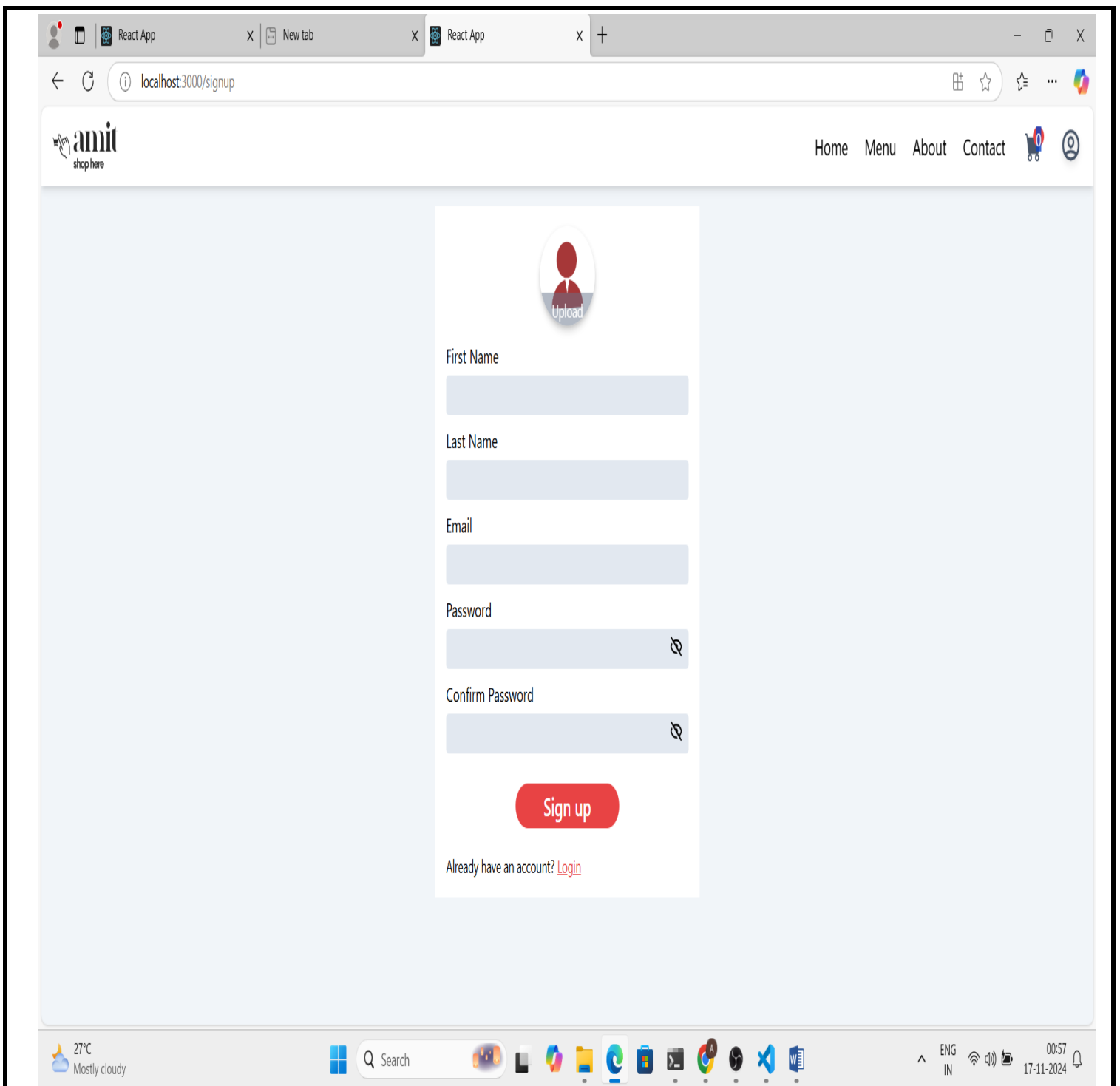
The screenshots of the app will give you a clear vision about the appearance and layout of the application.  
It includes the screenshots of:

1. Admin
2. User
3. Grocery items and
4. App's UI

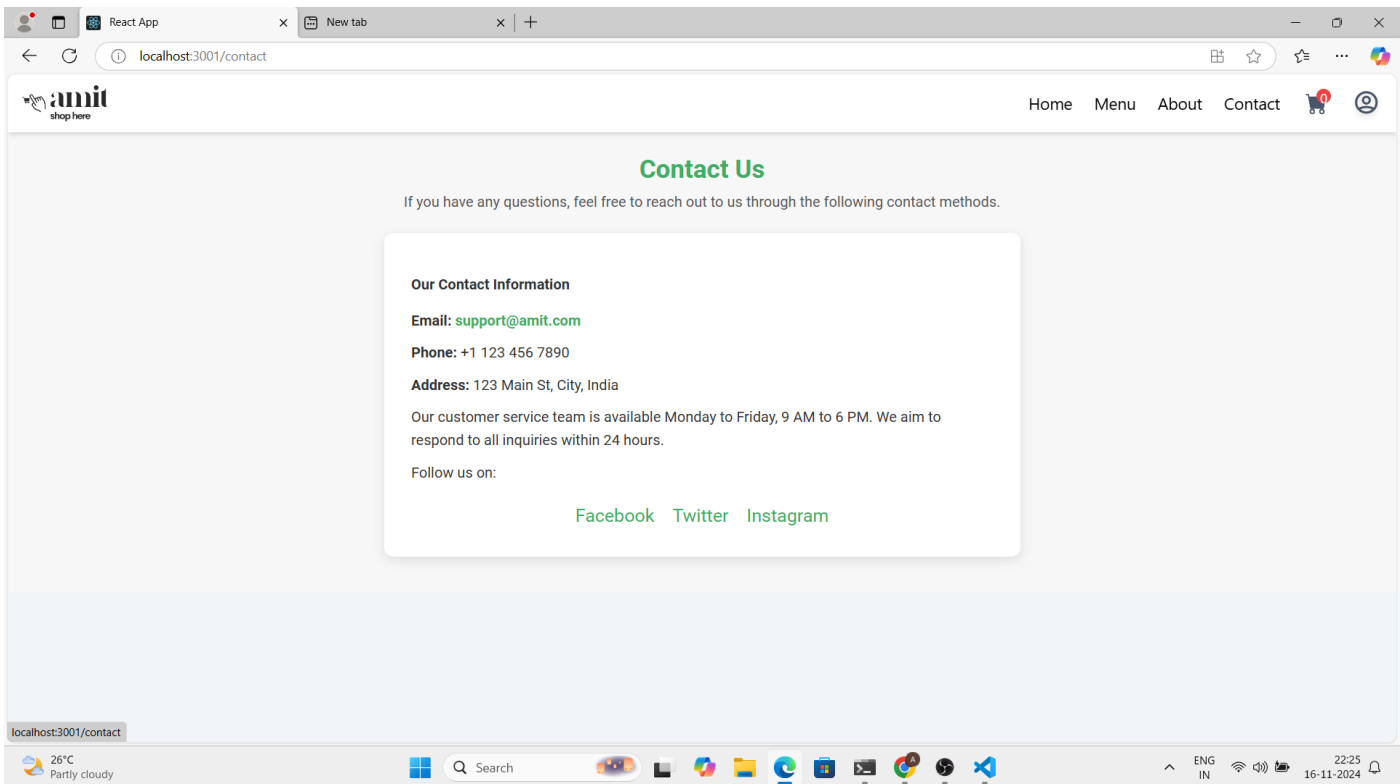
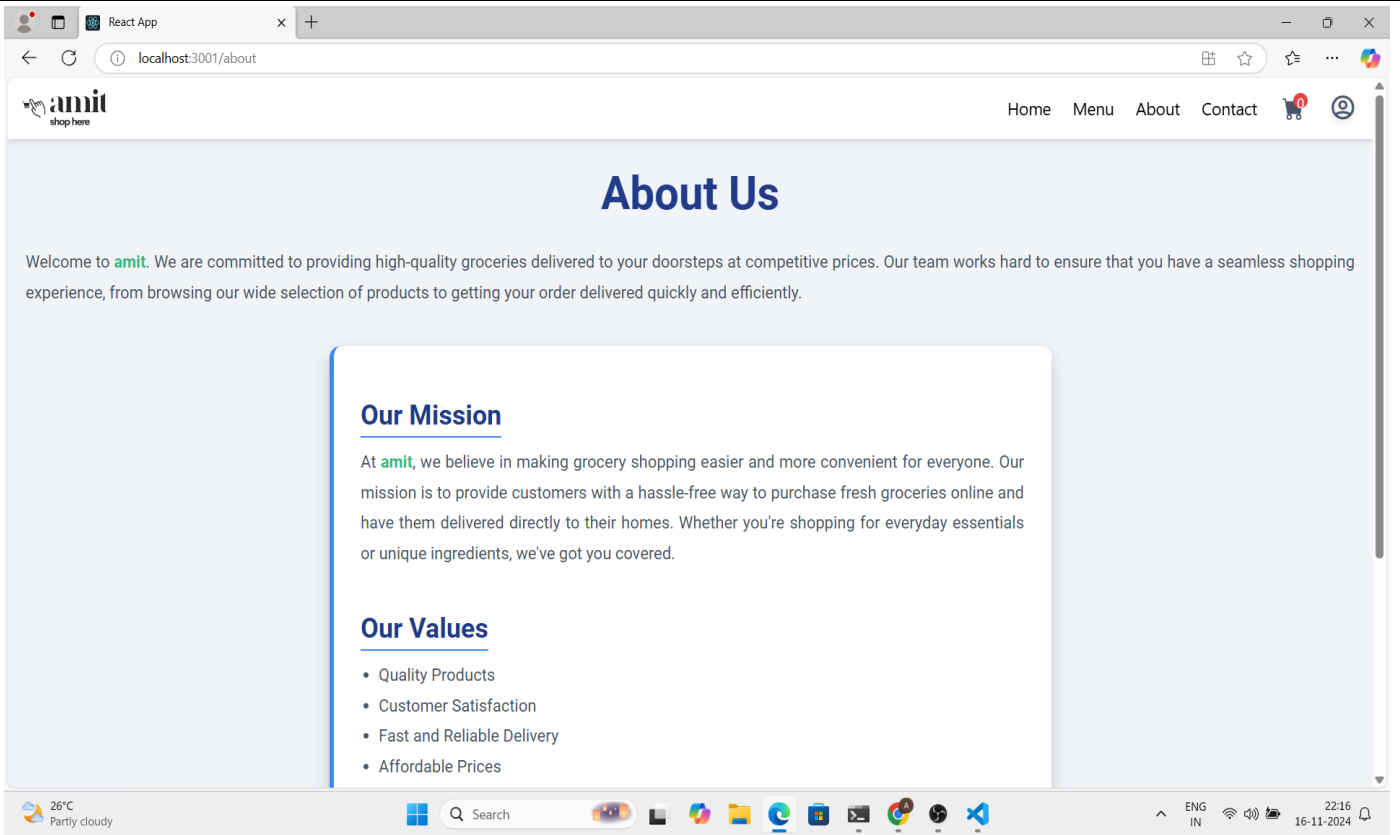












React App


New tab

localhost:3001/menu/6737ad56c2b1030d9b88a0ab

amitshop here

HomeMenuAboutContact

0



## Kiwi

fruits

**₹60**

Buy Now


Add to Cart

Description:  
Baby kiwi - fruits

Related Products

26°C  
Partly cloudy

Search



ENG  
IN

22:34  
16-11-2024

React App


New tab

localhost:3001/menu/6737ac9ec2b1030d9b88a0a5

amitshop here

HomeMenuAboutContact

0



## Apple

fruits

**₹50**

Buy Now


Add to Cart

Description:  
Fresh Apple

Related Products

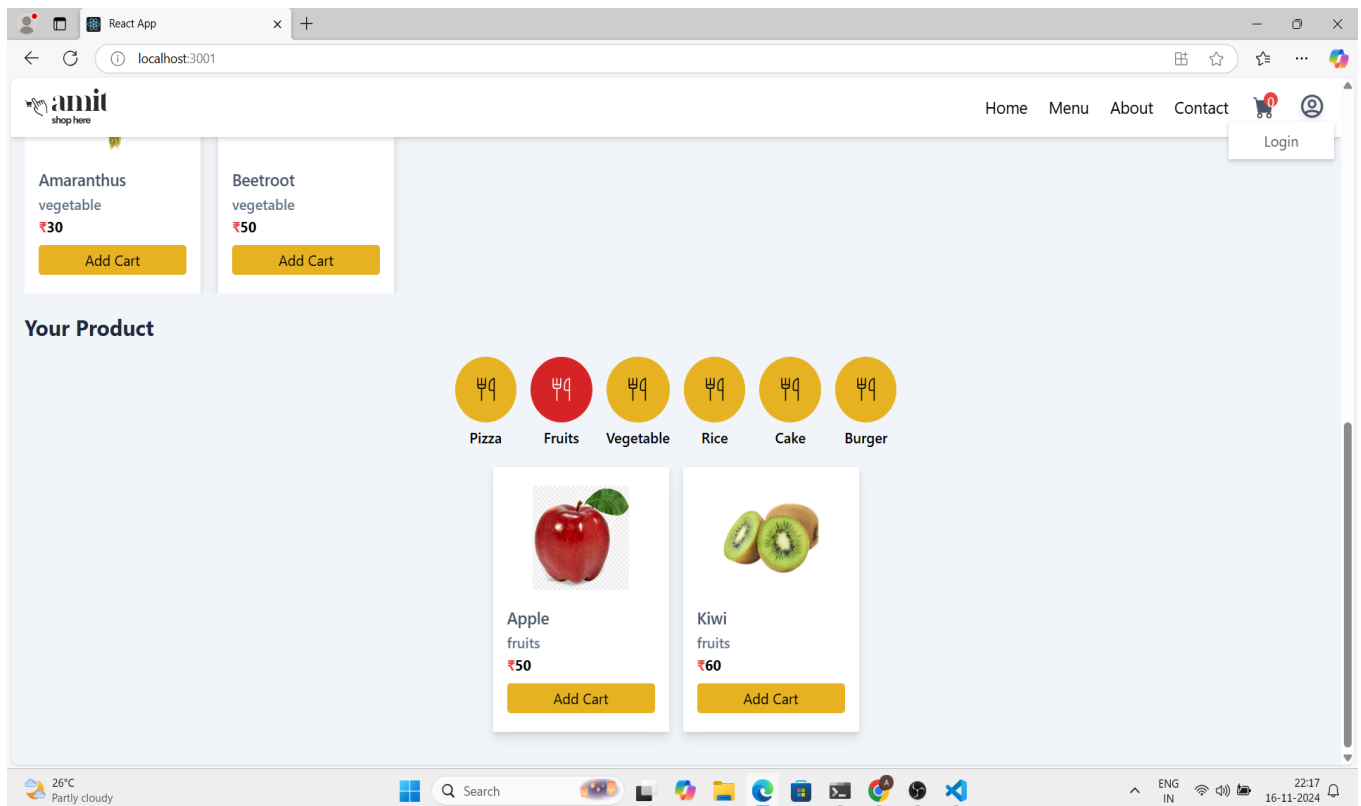
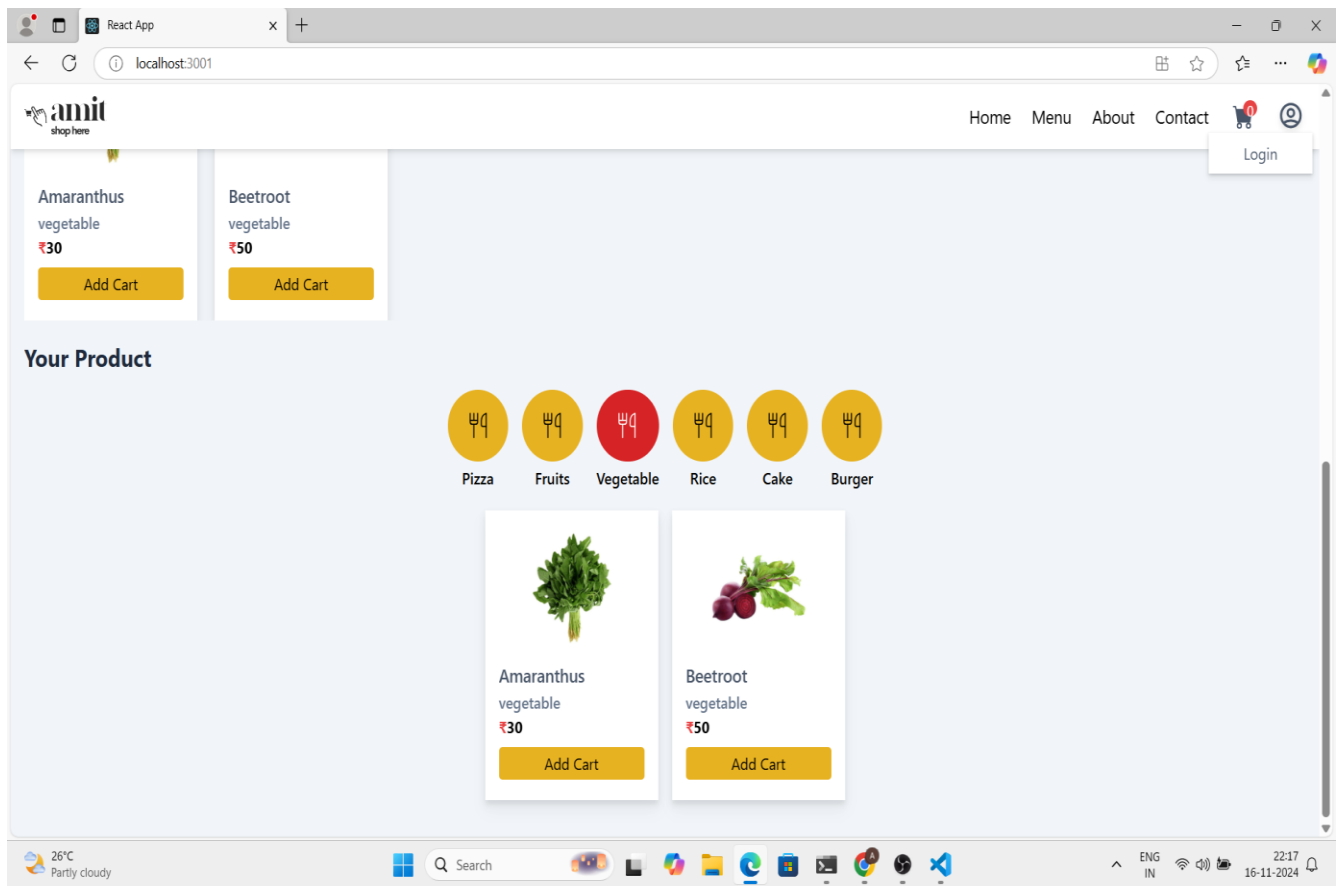
26°C  
Partly cloudy

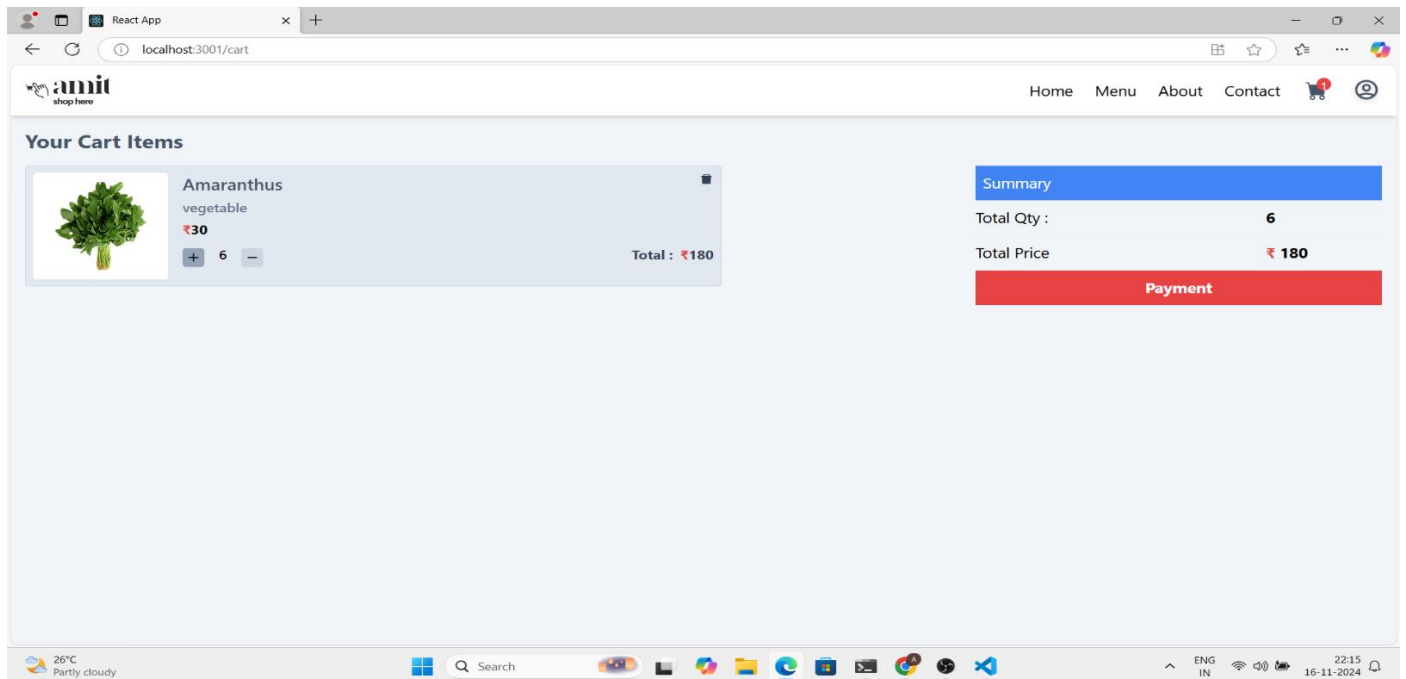
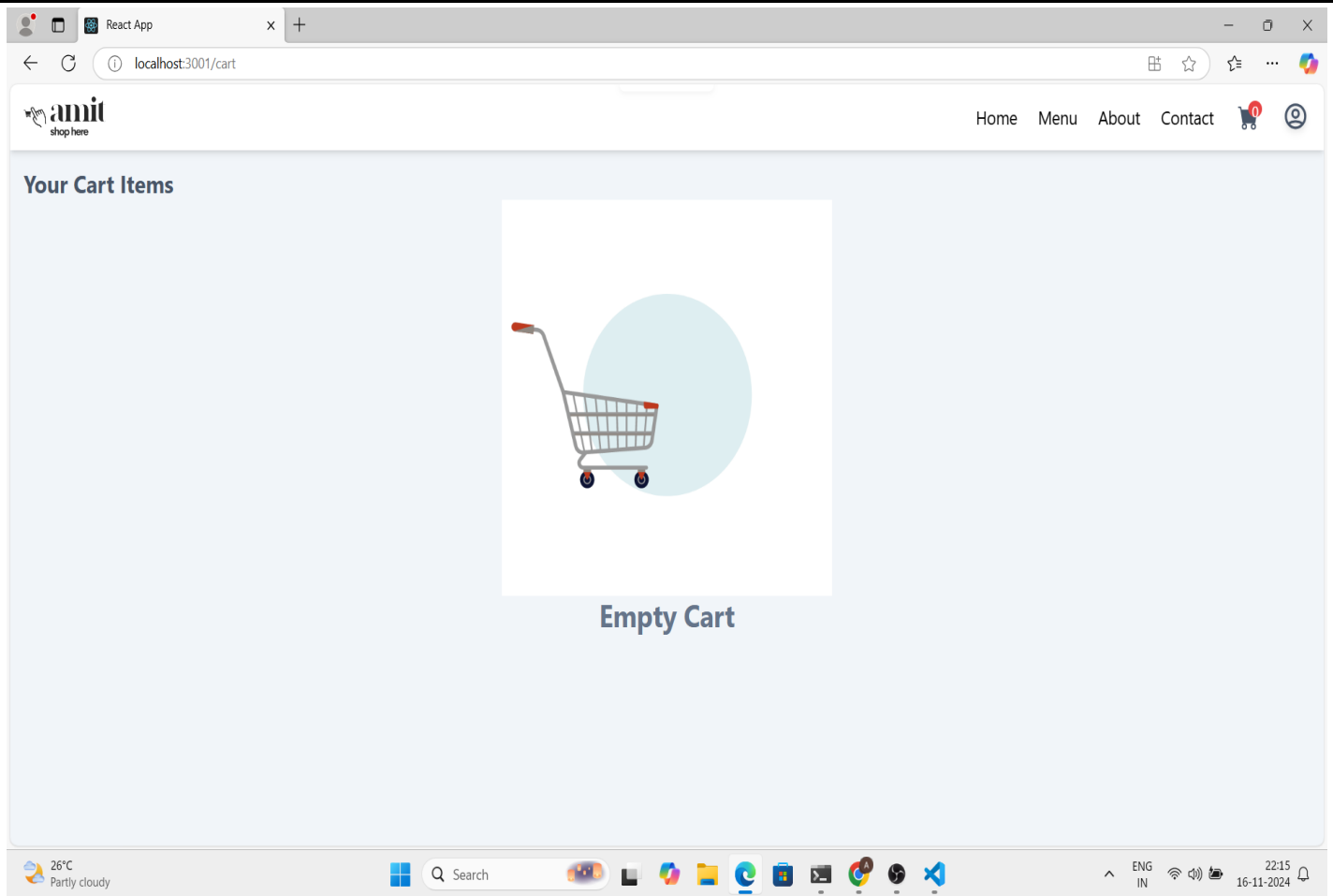
Search

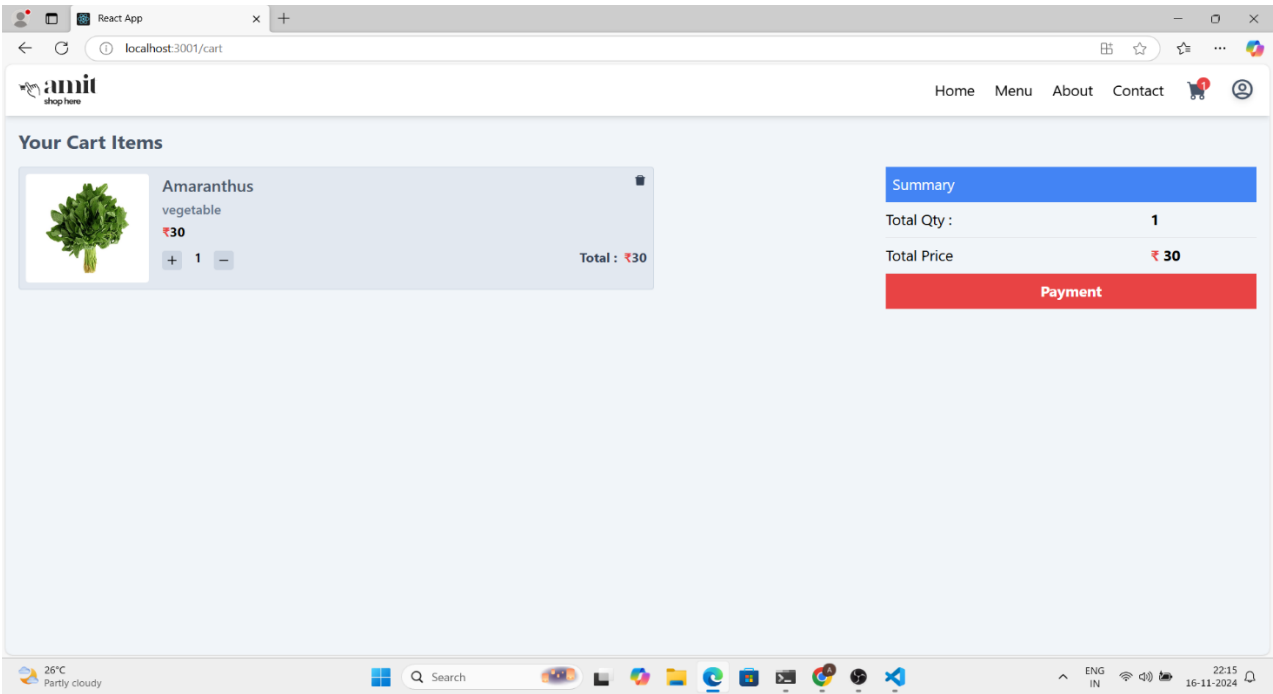


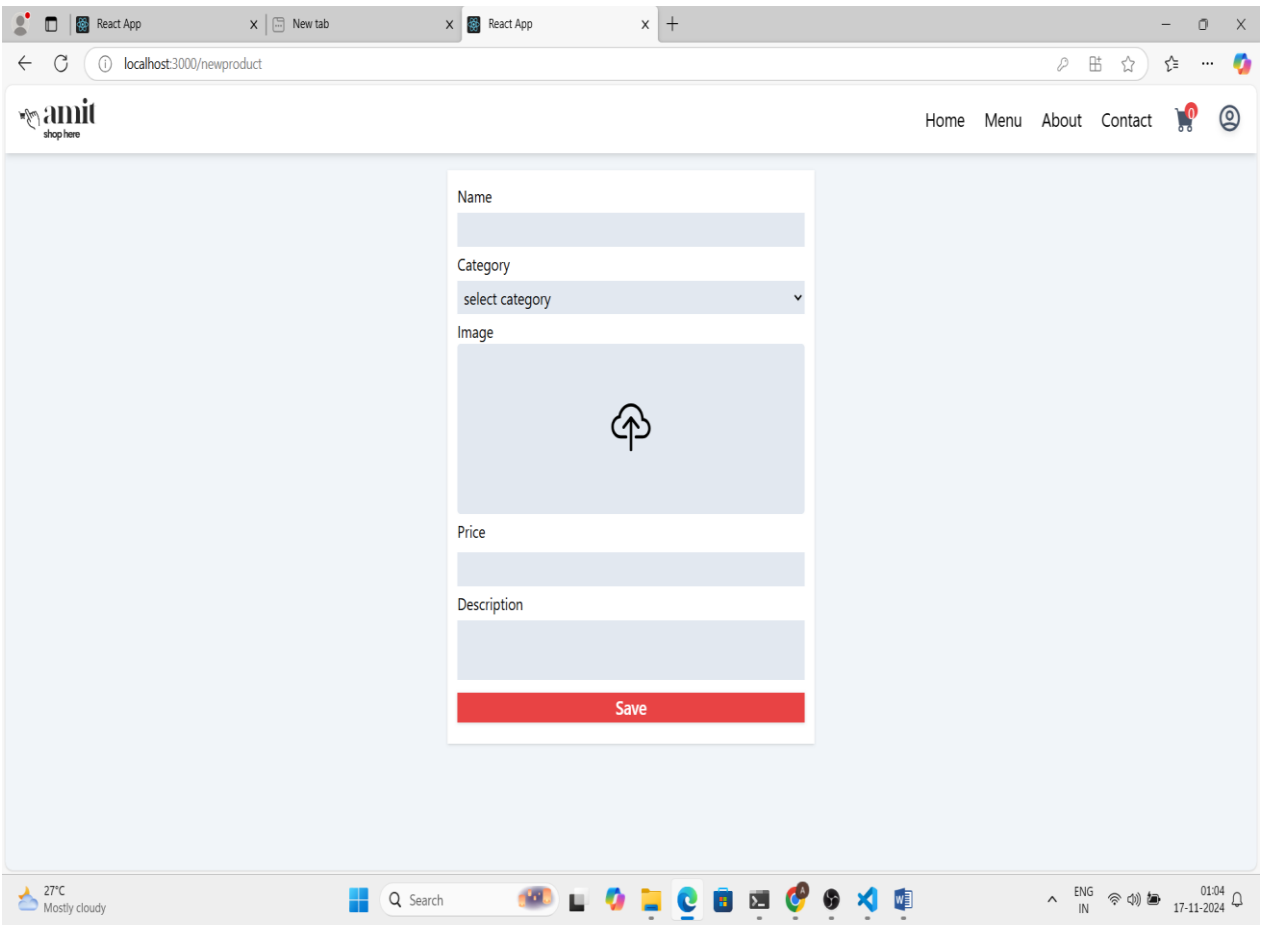
ENG  
IN

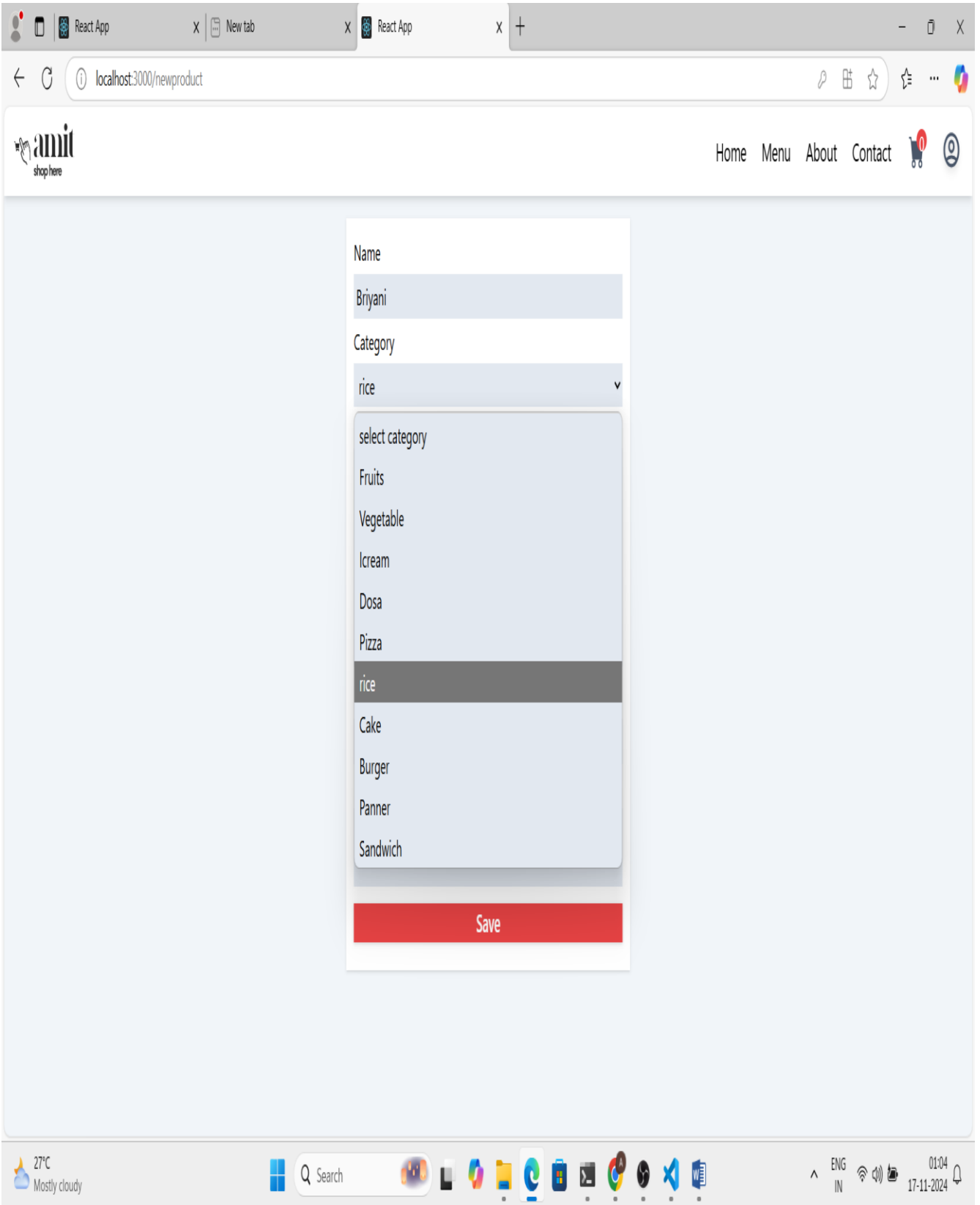
22:33  
16-11-2024

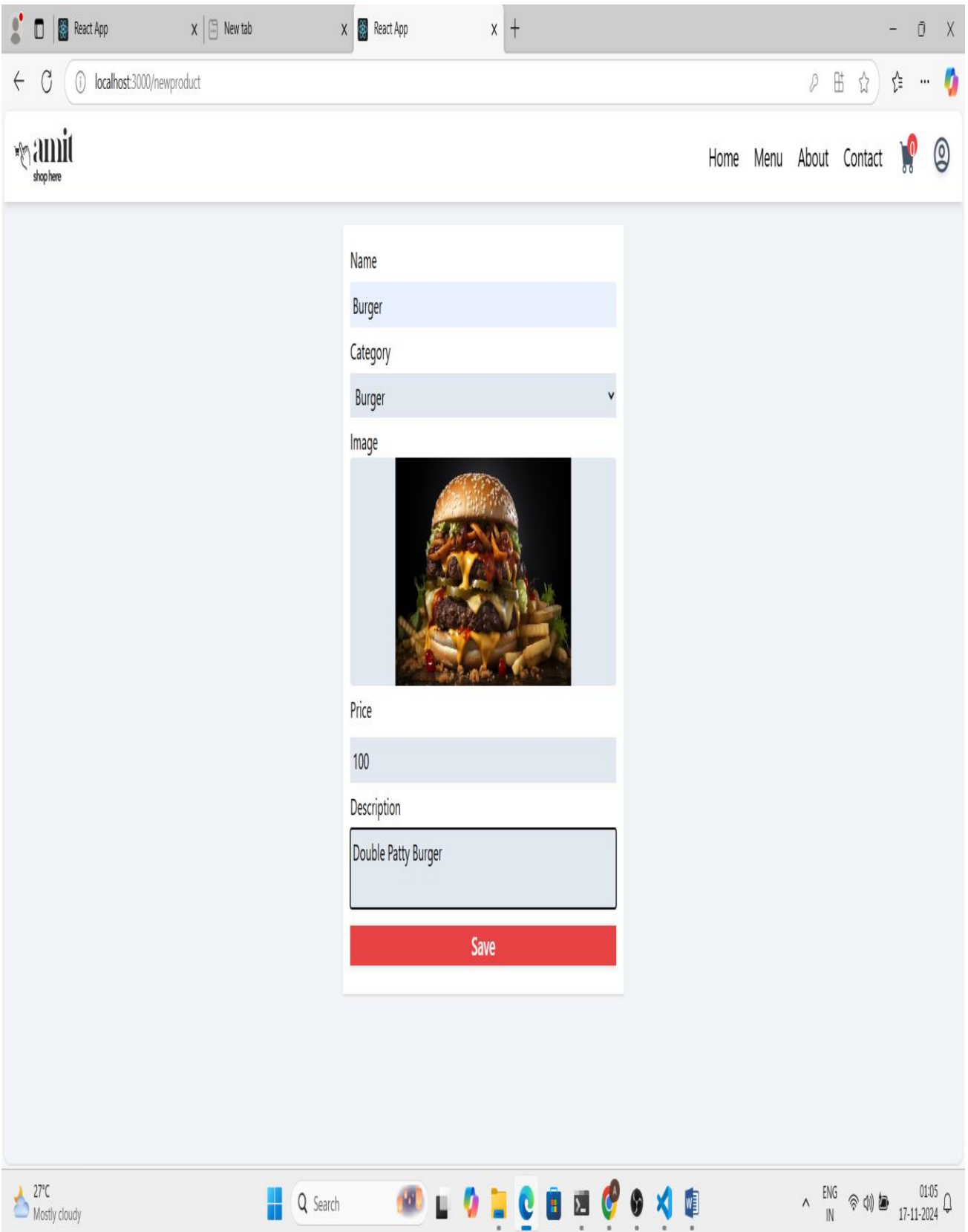














PIZZA


fruits

vegetable

rice


cake

burger




Pizza  
pizza  
₹500

Add Cart




Apple  
fruits  
₹50

Add Cart




Amaranthus  
vegetable  
₹30

Add Cart




Kiwi  
fruits  
₹60

Add Cart




Basmati-Rice  
rice  
₹120

Add Cart




Beetroot  
vegetable  
₹50

Add Cart



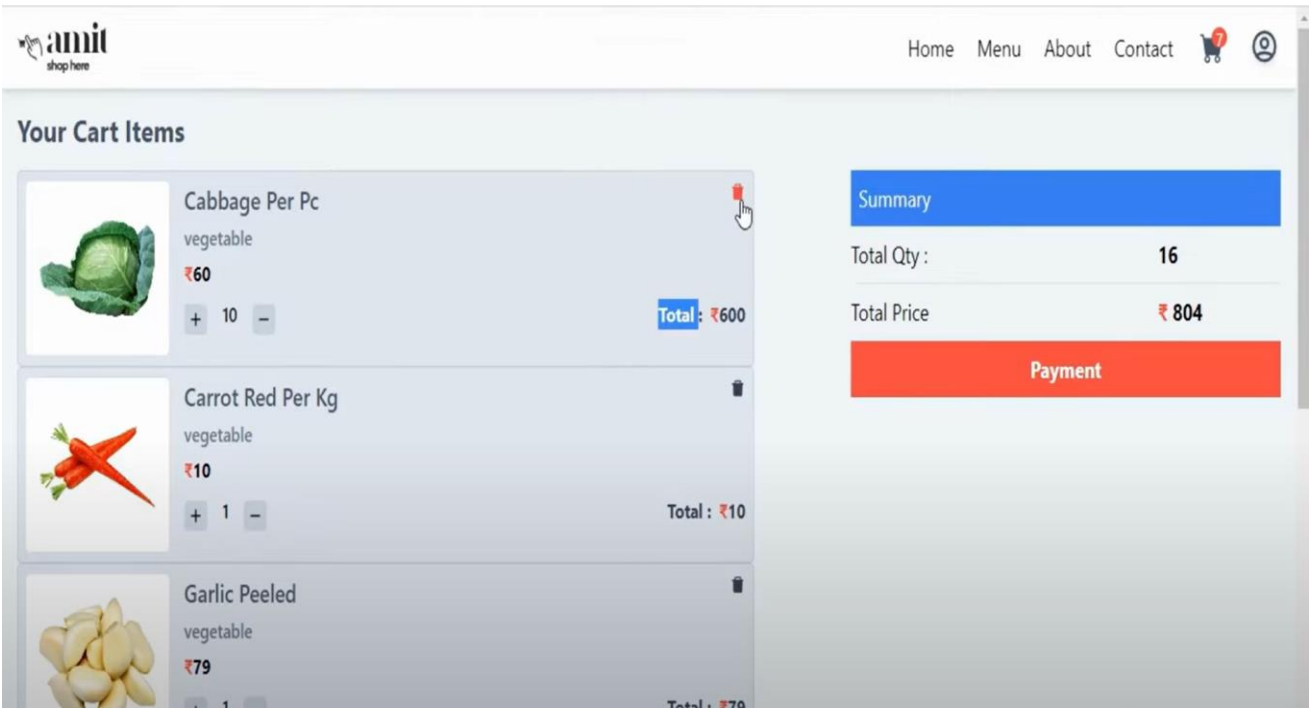
Birthday-Cake  
cake  
₹550

Add Cart



Burger  
burger  
₹250

Add Cart



## 12)Testing:

### Tools:

- **Jest:**

This comprehensive testing framework is widely used with React applications, providing a powerful combination of assertion, mocking, and snapshot testing. Jest offers efficient test running, built-in mocking capabilities for functions and modules, and detailed test result outputs, which are critical for React component testing.

### Mocha:

An adaptable testing framework for Node.js, often utilized for server-side and backend testing. Mocha allows for asynchronous testing, supports hooks like `before()`, `after()`, `beforeEach()`, and `afterEach()` for controlling the testing lifecycle, and pairs well with assertion libraries like Chai.

### Strategy:

#### 1. Unit Testing:

- **Objective:** Test individual units such as functions, components, and services to ensure they behave correctly in isolation.
- **Key Activities:**
  - **Testing React Components:** Verify that React components render correctly given a set of props or state changes.
  - **Function Validation:** Test utility functions, input validation, business logic, and transformations, ensuring expected outcomes for various inputs.
  - **Mocking:** Employ Jest's mock functions to simulate dependent services, APIs, or module

behavior during testing.

## 2.Integration Testing:

- **Objective:** Test the interaction between different components, modules, or services, ensuring that they work together as expected.
- **Key Activities:**
  - **API Testing:** Use Mocha to test server-side RESTful API endpoints for correct HTTP responses, payloads, and error handling.
  - **Database Queries:** Validate integration with database operations to ensure queries return expected results and handle edge cases correctly.
  - **User Interaction Flows:** Test user paths that involve multiple components, simulating real-world interactions (e.g., registering, logging in, ordering food).

### Technical Details:

- **Mocking and Stubbing:** Dependency injection and mocking allow isolation of units and controlled environments for reliable testing.
- **Assertions and Matchers:** Leverage Jest's and Mocha's built-in assertions to ensure expected values and states, checking for elements, API status codes, etc.
- **Automated Test Suites:** Organized suites for both unit and integration tests, with continuous integration hooks for automated test execution on code commits.
- **Code Coverage Analysis:** Employ coverage tools like Istanbul (integrated with Jest) to measure the extent of tested code paths, identifying potential gaps in test coverage.

## 2) Technical Issues:

Even though the app is working perfectly, we won't forget the issues faced during this. As Thomas Alva Edison tried 1000 times to build a light bulb, we tried 4 times to create and make our app work. Let's see about that now:

### 1. State Management:

- **Complexity:** Managing and synchronizing application state across numerous components, especially in large applications, can be difficult due to prop drilling and maintaining UI consistency.
- **Solution:**
  - Utilize **Redux** or **React Context** for predictable state updates, centralized state storage, and global state access.
  - Apply middleware like **Redux Thunk** or **Redux Saga** for asynchronous actions and side-effects management.

### 2. Database Schema Complexity:

- **Challenge:**
  - Nesting documents can lead to large data sizes, while referencing can cause slower joins.

- **Solution:**

- Optimize by determining when to **embed** (for denormalized, frequently-read data) vs. **reference** (for more normalized data with less redundancy).

### 3. Handling Edge Cases:

- **Challenge:**

Scenarios such as user input validation, unexpected API response errors, race conditions, server outages, and invalid data can lead to application crashes or poor user experience.

- **Solution:**

- Utilize Mongoose's schema design capabilities, such as schema validation, population (for joins), and indexing to enhance data query efficiency.
- Implement robust input validation using libraries like Joi and enforce type checks.
- Use try-catch blocks and error-handling middleware in Express to manage exceptions gracefully.

### 4) API Design:

- **Challenge:**

Ensuring scalable, secure, and maintainable API design is crucial for the backend architecture in Node.js/Express.

- **Solution:**

- Follow RESTful design principles by structuring routes cleanly (using versioning and semantic endpoints).
- Implement JWT (JSON Web Token) for secure authentication and authorization mechanisms.
- Optimize data fetching using pagination and filtering to minimize payload size and latency.
- Use middleware for input validation, logging, and rate limiting to enforce security best practices.
- Integrate retry mechanisms, fallbacks, and circuit breakers for network errors, and perform comprehensive testing (unit, integration, and boundary testing).
- Use tools like Postman or Swagger for testing and documenting APIs to ensure proper functionality and ease of consumption for developers.

## 14) Future Improvements for Grocery Web App

The **Grocery Web App** is committed to continuously enhancing the user experience by integrating innovative features. The following key future improvements are planned to elevate the app's functionality and service:

1. **Personalized Recommendations**
2. **Subscription and Recurring Orders**
3. **Advanced Search and Filtering**

### 1. Personalized Recommendations

**Description:** Implementing personalized recommendation systems will allow the app to suggest products tailored to each user's preferences and shopping habits. By analyzing past purchases, browsing behavior, and user preferences, the app can offer relevant product suggestions, enhancing the shopping experience.

#### Features:

- **Machine Learning Algorithms:** Utilize machine learning to analyze user data and predict products that users are likely to purchase.
- **Customized Dashboards:** Display personalized product suggestions on the home page and product pages.
- **Targeted Promotions:** Offer special deals and discounts based on individual user preferences and shopping history.
- **Cross-Selling and Up-Selling:** Recommend complementary products to increase average order value.

#### Benefits:

- **Enhanced User Experience:** Users receive relevant product suggestions, making their shopping experience more efficient and enjoyable.
- **Increased Sales:** Personalized recommendations can lead to higher conversion rates and increased sales.
- **Customer Retention:** Tailored experiences foster customer loyalty and encourage repeat purchases.

## 2. Subscription and Recurring Orders

**Description:** Introducing subscription services and the ability to set up recurring orders will provide users with the convenience of automating their regular grocery purchases. This feature is especially beneficial for staple items that customers buy frequently.

### Features:

- **Subscription Plans:** Allow users to subscribe to regular deliveries of specific products, such as milk, bread, or vegetables.
- **Flexible Scheduling:** Enable users to choose delivery frequencies (e.g., weekly, bi-weekly, monthly) and modify them as needed.
- **Auto-Renewal Options:** Offer options for automatic renewal of subscriptions with seamless payment processing.
- **Management Dashboard:** Provide a user-friendly interface for managing subscriptions, including pausing or canceling orders.

### Benefits:

- **Convenience:** Saves users time by automating the ordering process for frequently purchased items.
- **Predictable Revenue:** Provides the business with a steady and predictable revenue stream through recurring orders.
- **Customer Loyalty:** Encourages long-term relationships with customers by simplifying their shopping routine.

## 3. Advanced Search and Filtering

**Description:** Enhancing the search and filtering capabilities will enable users to find products more efficiently and tailor their shopping experience to their specific needs. Advanced search features can significantly improve user satisfaction and reduce cart abandonment rates.

### Features:

- **Intelligent Search Bar:** Implement autocomplete and suggestion features to help users find products quickly.
- **Multi-Filter Options:** Allow users to filter products by categories, brands, price ranges, dietary preferences (e.g., gluten-free, vegan), and more.
- **Voice Search:** Integrate voice-activated search functionality for a hands-free shopping experience.
- **Visual Search:** Enable users to upload images of products they are looking for and find similar items in the inventory.

### Benefits:

- **Improved Usability:** Makes it easier for users to locate specific products, enhancing overall user satisfaction.
- **Increased Efficiency:** Reduces the time users spend searching for products, leading to a more streamlined shopping process.
- **Higher Conversion Rates:** Enhanced search capabilities can lead to higher conversion rates by helping users find exactly what they need.

## Conclusion

The **Grocery Web App** is dedicated to providing a seamless and enjoyable shopping experience by continuously evolving to meet user needs. By focusing on personalized recommendations, subscription services, and advanced search functionalities, the app aims to enhance convenience, efficiency, and user satisfaction. These future improvements align with the core principles of innovation and customer-centric development, ensuring that the app remains competitive and continues to deliver exceptional value to its users.

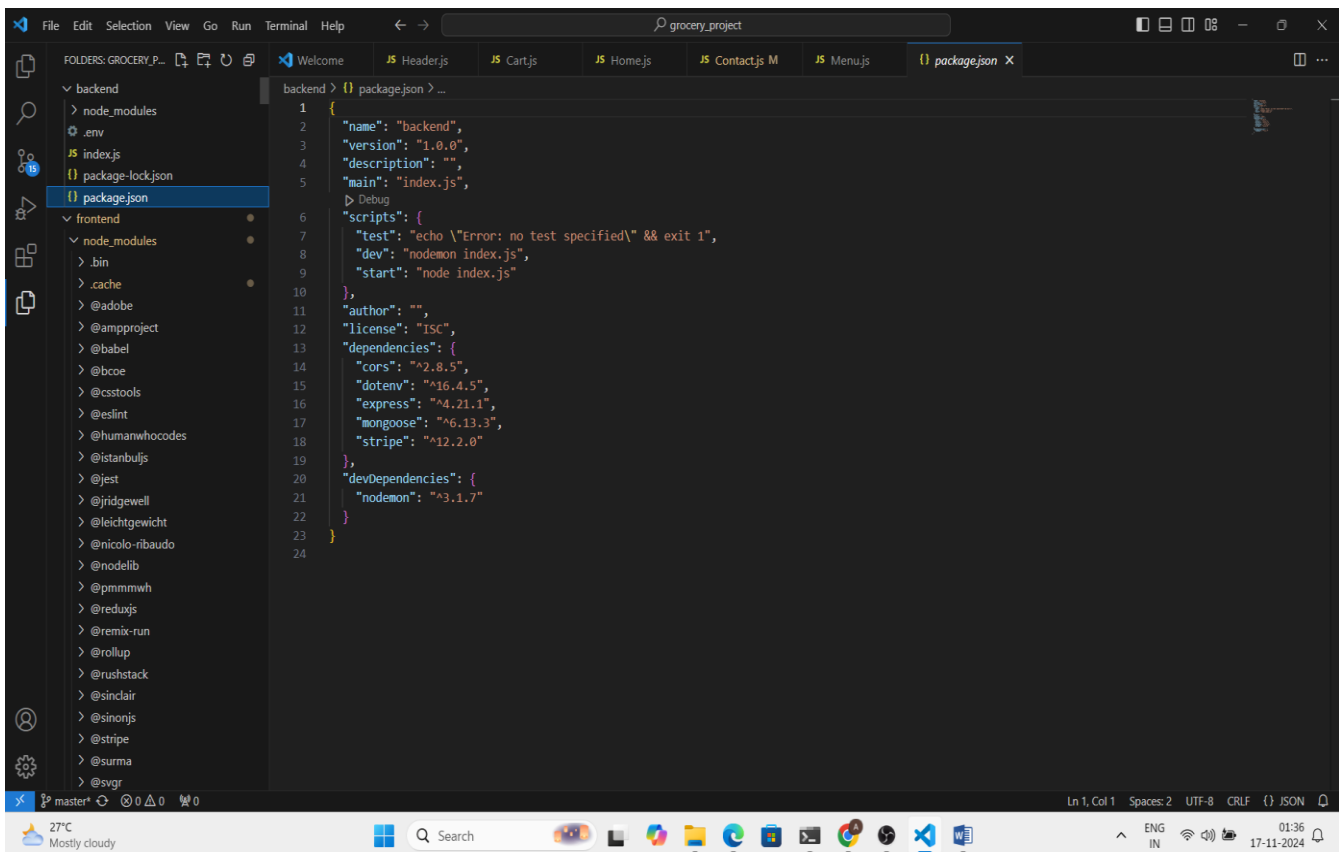
As technology advances, the app will incorporate emerging trends and user feedback to stay ahead, fostering long-term engagement and loyalty.

These planned enhancements will not only improve the functionality and user experience of the Grocery Web App but also position it as a leader in the competitive online grocery market. By prioritizing user needs and leveraging advanced technologies, the app is set to offer unparalleled convenience and satisfaction to its customers.

\*\*\*\*\*

## Appendices

### Code snippets:



The screenshot shows a Visual Studio Code editor window with a project named 'grocery\_project'. The file explorer on the left shows a folder structure with 'backend' and 'frontend' directories. The 'package.json' file in the 'backend' directory is open in the editor. The code is as follows:

```
1 {
2   "name": "backend",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1",
8     "dev": "nodemon index.js",
9     "start": "node index.js"
10  },
11   "author": "",
12   "license": "ISC",
13   "dependencies": {
14     "cors": "^2.8.5",
15     "dotenv": "^16.4.5",
16     "express": "^4.21.1",
17     "mongoose": "^6.13.3",
18     "stripe": "^12.2.0"
19  },
20   "devDependencies": {
21     "nodemon": "^3.1.7"
22  }
23 }
24
```

The status bar at the bottom indicates the file is at line 1, column 1, with 2 spaces, UTF-8 encoding, CRLF line endings, and JSON syntax. The system tray shows a temperature of 27°C, mostly cloudy weather, and the date 17-11-2024.

The screenshot shows the VS Code editor with the 'index.js' file open. The file contains the following code:

```
1 const express = require("express");
2 const cors = require("cors");
3 const mongoose = require("mongoose");
4 const dotenv = require("dotenv").config();
5 const Stripe = require('stripe')
6
7 const app = express();
8 app.use(cors());
9 app.use(express.json({ limit: "10mb" }));
10
11 const PORT = process.env.PORT || 8080;
12
13 //mongodb connection
14 mongoose.set("strictQuery", false);
15 mongoose
16   .connect(process.env.MONGODB_URI)
17   .then(() => console.log("Connect to Database"))
18   .catch((err) => console.log(err));
19
20 //schema
21 const userSchema = mongoose.Schema({
22   firstName: String,
23   lastName: String,
24   email: {
25     type: String,
26     unique: true,
27   },
28   password: String,
29   confirmPassword: String,
30   image: String,
31 });
32
33 //
34 const userModel = mongoose.model("user", userSchema);
35
36 //api
37 app.get("/", (req, res) => {
```

The file explorer on the left shows the project structure with folders like 'backend', 'frontend', and 'public'. The status bar at the bottom indicates 'Ln 1, Col 1', 'Spaces: 2', 'UTF-8', 'CRLF', and 'JavaScript'.

The screenshot shows the VS Code editor with the '.env' file open. The file contains the following environment variables:

```
1 MONGODB_URI=mongodb+srv://akshayabala1273:y7DK3VFuFafJ3Pu2@mygrocery.ry7gj.mongodb.net/mygrocery?retryWrites=true&w=majority&appName=myg
2 STRIPE_SECRET_KEY=cyour_stripe_secret_key
3 FRONTEND_URL=http://localhost:3000
4 MONGODB_USERNAME=akshayabala1273
5 MONGODB_PASSWORD=y7DK3VFuFafJ3Pu2
6 REACT_APP_ADMIN_EMAIL=admin@example.com
7
```

The file explorer on the left shows the project structure with folders like 'backend', 'frontend', and 'public'. The status bar at the bottom indicates 'Ln 7, Col 1', 'Spaces: 4', 'UTF-8', 'CRLF', and 'Properties'.

The screenshot shows the VS Code editor with the 'login.js' file open. The file is located in the 'frontend > src > page' directory. The code is a React component for a login page. It imports various libraries including React, useState, loginSignupImage, BShow, BHide, Link, BEmojiSmileUpsideDown, toast, useNavigate, useDispatch, useSelector, and loginRedux. The component defines a 'Login' function that uses useState for showPassword and data, useNavigate for navigation, and useSelector for user data. It also defines a 'handleShowPassword' function to toggle password visibility and a 'handleOnChange' function to handle input changes. The code is as follows:

```
1 import React, { useState } from "react";
2 import loginSignupImage from "../asset/login-animation.gif";
3 import { BShow, BHide } from "react-icons/bi";
4 import { Link } from "react-router-dom";
5 import { BEmojiSmileUpsideDown } from "react-icons/bs";
6 import { toast } from "react-hot-toast";
7 import { useNavigate } from "react-router-dom";
8 import { useDispatch, useSelector } from "react-redux";
9 import { loginRedux } from "../redux/userSlice";
10
11
12 const Login = () => {
13   const [showPassword, setShowPassword] = useState(false);
14   const [data, setData] = useState({
15     email: "",
16     password: "",
17   });
18   const navigate = useNavigate();
19   const userData = useSelector(state => state)
20
21
22   const dispatch = useDispatch()
23
24
25
26
27   const handleShowPassword = () => {
28     setShowPassword((prev) => !prev);
29   };
30
31   const handleOnChange = (e) => {
32     const { name, value } = e.target;
33     setData((prev) => {
34       return {
35         ...prev,
36         [name]: value
37       }
38     });
39   };
40 }
```

The file explorer on the left shows the project structure, with 'login.js' selected under 'frontend > src > page'. The status bar at the bottom indicates the file is at line 1, column 1, with 2 spaces, UTF-8 encoding, and CRLF line endings.

The screenshot shows the VS Code editor with the 'index.js' file open. The file is located in the 'backend' directory. The code is a Node.js application using Express, CORS, Mongoose, and Stripe. It sets up the application, connects to a MongoDB database, defines a user schema, and sets up API routes. The code is as follows:

```
1 const express = require("express");
2 const cors = require("cors");
3 const mongoose = require("mongoose");
4 const dotenv = require("dotenv").config();
5 const Stripe = require("stripe");
6
7 const app = express();
8 app.use(cors());
9 app.use(express.json({ limit: "10mb" }));
10
11 const PORT = process.env.PORT || 8080;
12
13 //mongodb connection
14 mongoose.set("strictQuery", false);
15 mongoose
16   .connect(process.env.MONGODB_URI)
17   .then(() => console.log("Connect to Databse"))
18   .catch((err) => console.log(err));
19
20 //schema
21 const userSchema = mongoose.Schema({
22   firstName: String,
23   lastName: String,
24   email: {
25     type: String,
26     unique: true,
27   },
28   password: String,
29   confirmPassword: String,
30   image: String,
31 });
32
33 // user model
34 const userModel = mongoose.model("user", userSchema);
35
36 //api
37 app.get("/", (req, res) => {
```

The file explorer on the left shows the project structure, with 'index.js' selected under 'backend'. The status bar at the bottom indicates the file is at line 1, column 1, with 2 spaces, UTF-8 encoding, and CRLF line endings.