# NAAN MUDHALVAN - MERN STACK
# FOOD ORDERING APP

**A PROJECT REPORT**

***Submitted By***

**MOHAMED AADHIL A (410621243019)**

**MOHAMMED YASIN A (410621243025)**

**MANOJ KUMAR G (410621243018)**

**VISHNU R (410621243054)**

***In partial fulfillment for the award of the degree***

**BACHELOR OF TECHNOLOGY**

***In***

**ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

**DHAANISH AHMED COLLEGE OF ENGINEERING**

**TABLE OF CONTENTS**

## CONTENTS

# CHAPTER 1 1. INTRODUCTION PROJECT TITLE

**YaMmY FoOdZ** -FOOD ORDERING APP

YaMmY FoOdZ is a modern food ordering platform designed to make online food ordering simple, fast, and convenient. Whether you're a foodie exploring new cuisines or someone satisfying late-night cravings, YaMmY FoOdZ connects users with a diverse range of restaurants, offering detailed menus, user reviews, and seamless checkout options.

With a user-centric design, secure payment processes, and robust tools for restaurants to manage their menus, YaMmY FoOdZ revolutionizes the food delivery experience. The app also empowers administrators with complete control over user management, restaurant approvals, and platform operations, ensuring smooth and reliable service for all stakeholders.

## TEAM MEMBERS

1. MOHAMED AADHIL A (LEADER)
2. MOHAMMED YASIN A
3. MANOJ KUMAR G
4. VISHNU R

# CHAPTER 2

## 2. PROJECT OVERVIEW
## PURPOSE

The primary purpose of the YaMmY FoOdZ - Food Ordering App is to provide a seamless, efficient, and user-friendly platform for ordering food online. The app is designed to address the needs of three main stakeholders: users, restaurants, and administrators.

## OBJECTIVES:

### a) For users:

**Convenience:** Enable users to browse, select, and order food from a variety of restaurants without leaving their homes.

**Choice and Transparency:** Provide comprehensive details about dishes, including descriptions, reviews, pricing, and discounts, to make informed decisions.

**Efficiency:** Streamline the ordering and payment process with a secure and intuitive system. **Accessibility**: Ensure users can access food options anytime, even during unconventional hours like late-night cravings.

### b) For Restaurants:

**Management:** Offer restaurants a platform to manage their menus, monitor orders, and track sales.

**Exposure**: Help restaurants reach a wider audience and boost their visibility online.

Ease of Use: Provide a dashboard to simplify the addition and management of food listings.

### c) For Admins:

**Oversight:** Enable platform administrators to manage users, restaurants, and product listings effectively.

**Quality Assurance:** Approve and monitor restaurants to maintain service quality and platform standards.

**Operational Control:** Ensure smooth operation of the platform through robust tools and reporting.

## GOALS:

The primary goal of the YaMmY FoOdZ - Food Ordering App is to provide a seamless, user-friendly platform that enhances convenience for users, empowers restaurants, and ensures smooth operations for administrators. For users, the app aims to simplify the food ordering process, offering a wide variety of restaurants and cuisines with easy browsing, detailed product information, and secure, efficient checkout. The goal is to create a personalized and hassle-free experience, allowing users to place orders quickly and track them in real time.

For restaurants, the app seeks to provide a powerful toolset for managing menus, processing orders, and tracking performance. It aims to increase restaurant visibility by connecting them to a large customer base and streamlining the process of adding, editing, or removing food items from their listings. Restaurants will also benefit from valuable insights into customer preferences and sales performance, helping them improve service and grow their business.

From an administrative perspective, the app's goal is to provide robust control over the platform, ensuring smooth operations. Admins will manage user accounts, approve restaurant listings, monitor orders, and generate reports to maintain platform quality and optimize performance. By doing so, the app ensures a high standard of service and fosters trust among all users, restaurants, and administrators.

Overall, the goals of YaMmY FoOdZ are to enhance user satisfaction, support restaurant growth, and create a transparent, efficient system that promotes business success for all parties involved.

## FEATURES:

The YaMmY FoOdZ app offers a simple and convenient food ordering experience for users, restaurants, and admins.

### a) For Users:

**Easy Registration & Login:** Users can quickly create an account or log in for a personalized experience.

**Browse and Order:** Explore a wide range of food items with detailed descriptions, pricing, and customer reviews. Add favorites to your cart and securely check out with multiple payment options.

**Order Tracking:** Stay updated on your order status with real-time notifications and delivery tracking.

**Late-Night Delivery:** Special section for late-night cravings, showcasing restaurants that are open during non-standard hours.

### b) For Restaurants:

**Menu Management:** Restaurants can easily add, update, or remove menu items.

**Order Notifications:** Real-time notifications alert restaurants to new orders.

**Sales Analytics:** Track sales, monitor customer activity, and manage promotions to boost business.

### c) For Admins:

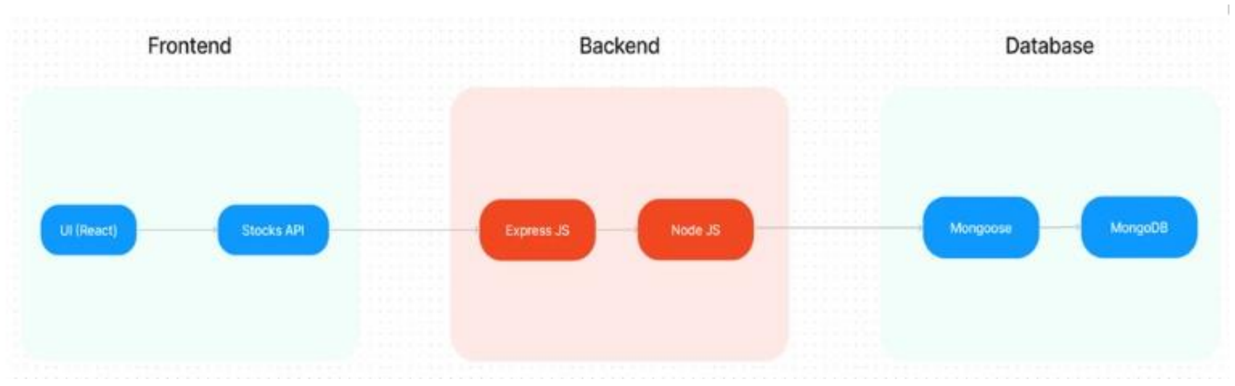
**Control Panel:** Admins can manage users, approve restaurant listings, monitor orders, and track performance.

**Reports & Analytics:** Access detailed reports to optimize platform operations and ensure smooth performance.

# CHAPTER 3

# 3. ARCHITECTURE

## DIAGRAM



## Front -end Architecture of YaMmY FoOdZ - Food Ordering App:

The frontend is the user-facing part of the YaMmY FoOdZ app that interacts directly with users, whether they are customers, restaurant owners, or administrators. It is designed to be intuitive, responsive, and visually appealing.

**Technologies Used:**

**HTML, CSS, JavaScript:** These core web technologies structure the app's pages, styling, and interactivity.

**React.js:** JavaScript frameworks/libraries that make the frontend responsive, dynamic, and efficient by creating interactive user interfaces (UI).

Bootstrap: These are design frameworks to ensure a clean and consistent look across the app.

**Components:**

**User Interface (UI):** The home page, product listings, product details page, cart, checkout, order confirmation, and user profiles are all part of the UI.

Authentication and Registration: A login and signup system for users, restaurants, and admins.

**Order Management:** Customers can place orders, view their cart, and track their order status.

**Admin Dashboard:** A section for administrators to manage users, restaurants, products, and orders.

**User Experience (UX):**

**Navigation:** Easy navigation across sections for a smooth flow from browsing products to placing orders.

**Responsiveness:** The app is designed to work seamlessly on both desktop and mobile devices. Interactive Elements: Features like dropdowns, search filters, and buttons that provide a smooth and efficient user journey.

## Backend Architecture of YaMmY FoOdZ - Food Ordering App

The backend manages the logic, database interactions, user authentication, and more. It is the server-side portion of the app that ensures proper functionality and security.

**Technologies Used:**

**Node.js with Express.js:** A JavaScript runtime and web framework used to handle requests, manage routing, and control application flow.

**Main Components:**

**User Management:** Handles user login, registration, and profile management.

**Order Processing:** Manages the creation, tracking, and updating of orders, including integration with payment systems.

**Product and Menu Management:** Allows restaurants to manage product listings and updates. Admins can add, update, or remove products and restaurants.

**Notification System:** Sends real-time notifications (via push or email) to users and restaurants about order status or promotions.

## Security:

**Authentication and Authorization:** Ensures that only authenticated users can access specific sections of the app (e.g., only restaurants can manage their menus).

**Encryption:** User data, payment details, and sensitive information are encrypted to protect privacy and security.

## Database Architecture of YaMmY FoOdZ - Food Ordering App

The database stores all essential data related to users, orders, products, and restaurants, ensuring everything functions smoothly. It is designed for efficiency, scalability, and security.

**Technologies Used:**
MongoDB: A NoSQL database might be used to handle unstructured or semi-structured data, such as reviews and product descriptions.

**Main Components:**
**Users Table:** Stores user information like email, password, delivery address, and order history.
**Restaurants Table:** Contains details about restaurants, including their name, location, menu items, and operational status.
**Orders Table:** Stores all order information, including the user, products, delivery details, payment method, and order status.
**Products Table:** Contains details about the food items, including name, description, price, restaurant ID, and availability.
**Cart Table:** Holds temporary data about the products added by the user before checkout.
**Admin Table:** Stores admin user credentials and permissions.

**Relationships:**

**One-to-Many Relationship:** One restaurant can have many products, and one user can place many orders.

**Many-to-Many Relationship:** Users can review multiple products, and products can have multiple reviews from different users.

# CHAPTER 4

# 4. SETUP INSTRUCTIONS

## 1. PREREQUISITES SETUP

Before installing the YaMmY FoOdZ - Food Ordering App, ensure the following prerequisites are met:

### HARDWARE REQUIREMENTS:

- A Windows 8 or higher machine with sufficient processing power.
- Internet bandwidth of at least 30 Mbps for smooth operation.

### SOFTWARE REQUIREMENTS:

- **Node.js**: Download and install Node.js (LTS version) from Node.js official website.
- **NPM**: Comes pre-installed with Node.js; used for managing project dependencies.
- **Database**: Set up a MySQL/PostgreSQL server for relational database storage or MongoDB for NoSQL needs.
- **Web Browsers**: Install at least two web browsers (e.g., Google Chrome, Brave) for testing.

### DEVELOPMENT TOOLS:

- **Git**: Install Git for version control. Download from Git official website.
- **Code Editor**: Install Visual Studio Code (or any preferred editor) from VS Code official website.

# 2. INSTALLATION INSTRUCTIONS

### Step 1: Install Dependencies

1. Navigate to the project directory.
2. Run the following command to install dependencies for the frontend:

```
cd frontend
npm install
```

3. Similarly, install backend dependencies:

```
Cd backend npm
install
```

### Step 2: Set Up Environment Variables

1. Create a `.env` file in both the **frontend** and **backend** directories.
2. Add necessary environment variables. Example for the backend:

```
env
MONGO=your-database-url
JWT_SECRET=your-jwt-secret
```

### Step 3: Set Up the Database

1. **For MongoDB**:
   - Start your MongoDB server and create a new database.

### Step 4: Run the Backend Server

1. Navigate to the backend folder:

```
cd backend
```

2. Start the backend server:

```
node injdex.js
```

The server will start on the specified port (e.g., `http://localhost:6001`).

### Step 5: Run the Frontend

1. Navigate to the frontend folder:

```
cd ../frontend
```

2. Start the frontend application:

```
npm start
```

The app will launch in your default web browser at `http://localhost:3000`.

### Step 6: Test the Application

- Open the app in your browser.
- Log in as a user, restaurant, or admin to ensure all functionalities work as expected.

# CHAPTER 5 5. PROJECT STRUCTURE 1) CLIENT

The structure of the React frontend in your Food Ordering System project is well-organized, ensuring modularity and scalability.

1. **`src/components/`**

This folder contains reusable components shared across the application.

- **`Footer.jsx`**: Handles the footer displayed at the bottom of every page.
- **`Login.jsx` and `Register.jsx`**: Manage user authentication interfaces for logging in and registering new users.
- **`Navbar.jsx`**: The navigation bar, providing links for easy access to various pages.
- **`PopularRestaurants.jsx` and `Restaurants.jsx`**: Display popular and general restaurant listings.

2. **`src/context/`**

This folder is likely used for managing global state using React Context API, such as user authentication, cart data, or restaurant data.

3. **`src/pages/`**

The `pages/` directory is divided into subfolders for each user role:

- **`admin/`**: Contains pages for admin operations:
  - **`Admin.jsx`**: The admin dashboard.
  - **`AllOrders.jsx`**,
  - **`AllProducts.jsx`**,
  - **`AllRestaurants.jsx`**,
  - **`AllUsers.jsx`**:

  Pages for managing orders, products, restaurants, and users respectively.
- **`customer/`**: Contains pages for customer-related operations:

    - **`Cart.jsx`**: Displays items in the cart.
    - **`CategoryProducts.jsx`**: Lists products within a specific category. o **`IndividualRestaurant.jsx`**: Displays the menu of a specific restaurant.
    - **`Profile.jsx`**: User profile management page.
- **`restaurant/`**: Contains pages for restaurant operations:
    - **`EditProduct.jsx`**: Allows restaurants to edit existing product details. o **`NewProduct.jsx`**: A form for adding new products to the menu. o **`RestaurantHome.jsx`**: The dashboard for restaurant users. o **`RestaurantMenu.jsx`**: Displays the restaurant's current menu.
    - **`RestaurantOrders.jsx`**: Displays orders placed for that restaurant.
    - **`Authentication.jsx`**: Handles login/registration for restaurant owners.

4. **`src/styles/`**

Holds the stylesheets (`App.css`) to ensure a consistent look and feel across the app.

5. **`src/App.js`**

This is the central file for managing routing and rendering pages. It likely uses **React Router** to navigate between components and pages.

6. **`public/`**

This folder holds static files, such as images or icons, that don't change frequently.

## 2. SERVER

The Node.js backend is structured to handle the server-side logic and interact with the database. Here's an explanation of the organization:

1. **`index.js`**

This is the entry point for the server. It is responsible for:

- Setting up the Express server.
- Defining routes and middleware.
- Connecting to the database (e.g., MongoDB, MySQL, etc.).
- Starting the server on a specified port.

2. `Schema.js`

This file defines the database schema using a library such as **Mongoose** (for MongoDB) or an ORM (like Sequelize for SQL databases). It likely includes models for:

- **Users**: Stores user information (e.g., name, email, hashed passwords).
- **Restaurants**: Contains restaurant details like name, menu items, and operational status.
- **Products**: Stores information about individual food items, including price and availability.
- **Orders**: Tracks order details, including user ID, restaurant ID, and order status.
- **Admin**: Stores admin credentials and permissions.

3. `package.json`

This file lists all the backend dependencies, scripts, and metadata. Key dependencies might include:

- **Express.js**: For building APIs.
- **Mongoose/Sequelize**: For database interaction.
- **JWT (jsonwebtoken)**: For user authentication.
- **Cors**: To allow cross-origin requests from the frontend.

4. `package-lock.json`

This file locks the dependency versions to ensure consistency across installations.

**Backend Workflow**:

1. **API Routing**: Routes are likely defined in `index.js` or separate route files. These routes map frontend requests to the appropriate backend logic (e.g., user login, product listing).
2. **Controller Functions**: Each route calls a corresponding controller function to process the request.
3. **Database Queries**: The controller interacts with models in `Schema.js` to fetch or update data in the database.
4. **Response**: The API sends JSON responses back to the frontend for rendering.

# CHAPTER 6

# 6. WORKING OF APPLICATION

To run the application locally, you need to start both the **frontend server** and the **backend server** separately. Below are the steps and commands for each.

**1. Running the Frontend Server**

The frontend is a React application:

1. Open a terminal and navigate to the `client` folder:

   ```
   cd client
   ```

2. Install all the dependencies:

   ```
   npm install (or) npm i
   ```

   This will download all required packages as defined in the `package.json` file.

3. Start the React development server:

   ```
   npm start
   ```

4. Once started, the application will be available at:

   **`http://localhost:3000`**

## 2. Running the Backend Server

The backend is a Node.js application :

1. Open another terminal and navigate to the `server` folder:

   ```
   cd server
   ```

2. Install all the dependencies:

   ```
   npm install
   ```

3. Start the Node.js server:

   ```
   node index.js
   ```

Make sure both servers are running simultaneously for the application to work correctly. If needed, use tools like Postman to test backend APIs and verify that they respond to the frontend requests.

# CHAPTER 7

# 7. API DOCUMENTATION

Each section covers APIs for different modules like **User**, **Restaurant**, **Products**, **Cart**, and **Orders**.

### 1. User APIs

### 1.1 User Registration

- **Endpoint**: `POST /api/users/register`
- **Description**: Registers a new user.
- **Request Body** (JSON):

```
{
  "name": "SR",
  "email": "SR123@gmail.com",
"password": "securepassword"
}
```

- **Response** (JSON):

```
{
  "message": "User registered successfully"
}
```

### 1.2 User Login

- **Endpoint**: `POST /api/users/login`
- **Description**: Authenticates a user and provides a JWT token.
- **Request Body** (JSON):

```
{
  "email": "SR123@gmail.com",
"password": "securepassword"
}
```

- **Response** (JSON):

```
{
  "token": "jwt_token_here"
}
```

### 1.3 User Profile

- **Endpoint**: `GET /api/users/profile`
- **Description**: Fetches the logged-in user's profile details.

- **Headers**:

```
Authorization: Bearer <jwt_token>
```

- **Response** (JSON):

```
json
Copy code
{
  "id": "123",
  "name": "SR",
  "email": "SR123@gmail.com"
}
```

## 2. Restaurant APIs

### 2.1 List All Restaurants

- **Endpoint**: `GET /api/restaurants`
- **Description**: Fetches a list of all restaurants.
- **Response** (JSON):

```
[
  {
    "id": "1",
    "name": "Pizza Palace",
    "location": "Downtown",
  },
  {
    "id": "2",
    "name": "Burger Haven",
"location": "City Center",
  }
]
```

**2.2 Add a New Restaurant (Admin)**

- **Endpoint**: `POST /api/restaurants`
- **Description**: Adds a new restaurant to the platform (requires admin authentication).
- **Request Body** (JSON):

```
{
  "name": "Sushi Spot",
  "location": "Uptown",
"cuisine": "Japanese"
}
```

- **Response** (JSON):

```
{
  "message": "Restaurant added successfully"
}
```

**3. Product APIs**

**3.1 List Products by Restaurant**

- **Endpoint**: `GET /api/restaurants/:restaurantId/products`
- **Description**: Fetches the menu of a specific restaurant.
- **Response** (JSON):

```
[
  {
    "id": "1",
    "name": "Cheeseburger",
    "price": 5.99,
    "category": "Main Course"
  },
  {
    "id": "2",
    "name": "Fries",
    "price": 2.99,
    "category": "Sides"
  }
]
```

### 3.2 Add a New Product (Restaurant Owner)

- **Endpoint**: `POST /api/products`
- **Description**: Adds a new product to a restaurant's menu.
- **Request Body** (JSON):

```
{
  "restaurantId": "1",
  "name": "Veggie Pizza",
  "price": 9.99,
  "category": "Pizza"
}
```

- **Response** (JSON):

```
{
  "message": "Product added successfully"
}
```

## 4. Cart APIs

### 4.1 Add Item to Cart

- **Endpoint**: `POST /api/cart`
- **Description**: Adds a product to the user's cart.
- **Request Body** (JSON):

```
{
  "userId": "123",
  "productId": "1",
  "quantity": 2
}
```

- **Response** (JSON):

```
{
  "message": "Item added to cart"
}
```

### 4.2 View Cart

- **Endpoint**: `GET /api/cart/:userId`
- **Description**: Fetches all items in the user's cart.
- **Response** (JSON):

```
[
  {
    "productId": "1",
    "name": "Cheeseburger",
    "quantity": 2,
    "price": 5.99
  },
  {
    "productId": "2",
    "name": "Fries",
    "quantity": 1,
    "price": 2.99
  }
]
```

## 5. Order APIs

### 5.1 Place an Order

- **Endpoint**: `POST /api/orders`

- **Description**: Places an order with the current cart items.
- **Request Body** (JSON):

```
json
Copy code
{
  "userId": "123",
  "cartItems": [
    {
      "productId": "1",
      "quantity": 2
    },
    {
      "productId": "2",
      "quantity": 1
    }
  ],
  "totalPrice": 14.97,
  "deliveryAddress": "123 Main Street"
}
```

- **Response** (JSON):

```
{
  "orderId": "456",
  "message": "Order placed successfully"
}
```

### 5.2 Get User Orders

- **Endpoint**: `GET /api/orders/:userId`
- **Description**: Fetches all orders placed by a user.
- **Response** (JSON):

```
[
  {
    "orderId": "456",
    "items": [
      {
        "name": "Cheeseburger",
        "quantity": 2
      },
      {
        "name": "Fries",
        "quantity": 1
      }
    ],
    "totalPrice": 14.97,
"status": "Delivered"
  }
]
```

## 6. Admin APIs

### 6.1 Get All Users

- **Endpoint**: `GET /api/admin/users`
- **Description**: Fetches a list of all registered users (admin-only access).
- **Response** (JSON):

```
[
  {
    "id": "123",
    "name": "SR",
    "email": "SR123@gmail.com"
  },
  {
    "id": "124",
    "name": "Sathya",
    "email": "sathya123@gmail.com"
  }
]
```

# CHAPTER 8 8. AUTHENTICATION

## Authentication and Authorization in the Food Ordering System

Authentication and authorization are critical for ensuring that only verified users and administrators can access specific resources in the Food Ordering System.

### 1. Authentication

Authentication verifies the identity of users (customers, restaurants, and admins). The system uses **JWT (JSON Web Tokens)** for handling authentication securely.

Steps in Authentication:

1. **User Registration**:
   - o New users register by providing their details (e.g., name, email, and password).
   - o Passwords are hashed using a library like **bcrypt** for security before storing them in the database.

   Example Workflow:

   - o Endpoint: `POST /api/users/register` o The password is hashed, and user data is saved in the database.

2. **User Login**:
   - o Users log in with their email and password.
   - o The system checks the credentials against the database.
   - o If credentials are valid:
     -  The token is sent back to the client to be used in subsequent requests.

   Example Workflow:

- Endpoint: `POST /api/users/login` o The system generates a token using a library like `jsonwebtoken`:

```
const token = jwt.sign(
  { id: user._id, role: user.role },
process.env.JWT_SECRET,
  { expiresIn: "1h" }
);
```

- The token is returned in the response.

3. **Token Validation**:
    - For every API request to a protected route, the client includes the token in the **Authorization** header:

```
makefile
Copy code
Authorization: Bearer <JWT_TOKEN>
```

    - The server validates the token using the secret key. If valid, the server grants access; otherwise, the request is rejected.

**2. Authorization**

Authorization ensures that users can access only the resources they are allowed to.

**How Authorization Works**:

1. **Role-Based Access Control (RBAC)**: o Different roles have specific permissions:
    - **Customers** can browse restaurants, view menus, add items to the cart, and place orders.
    - **Restaurant Owners** can manage their menu, view their orders, and update product details.
    - **Admins** have elevated permissions to manage all users, restaurants, and products.

    - The role is stored in the user's JWT token (e.g., `role: "admin"`).

2. **Middleware for Role Verification**:
    - The system uses middleware to verify user roles and restrict access to certain routes:

```
const verifyAdmin = (req, res, next) => {
  const token = req.header("Authorization").split(" ")[1];
const decoded = jwt.verify(token, process.env.JWT_SECRET);
if (decoded.role === "admin") {     next();   } else {
    res.status(403).send("Access denied.");
  }
};
```

3. **Protected Routes**:
    - API routes are protected with middleware to ensure only authorized users can access them. Examples:
        - **Customers**: Can only access routes like `/api/cart` and `/api/orders/:userId`.
        - **Restaurant Owners**: Can only modify their restaurant's data.
        - **Admins**: Can access all resources, like viewing all users and restaurants.

## 3. Key Components of Authentication & Authorization

1. **JWT (JSON Web Token)**: o Tokens are stateless and contain encoded information (e.g., user ID and role). o They are signed using a secret key (stored in environment variables).

```
{
  "id": "12345",
  "role": "customer",
  "iat": 1689413401,
"exp": 1689417001
}
```

2. **Password Security**:
    - o **bcrypt** is used to hash passwords before storing them in the database.
    - o During login, passwords are compared using `bcrypt.compare()`.

3. **Environment Variables**:
    - o Secret keys (e.g., `JWT_SECRET`) are stored securely in `.env` files to prevent unauthorized access.

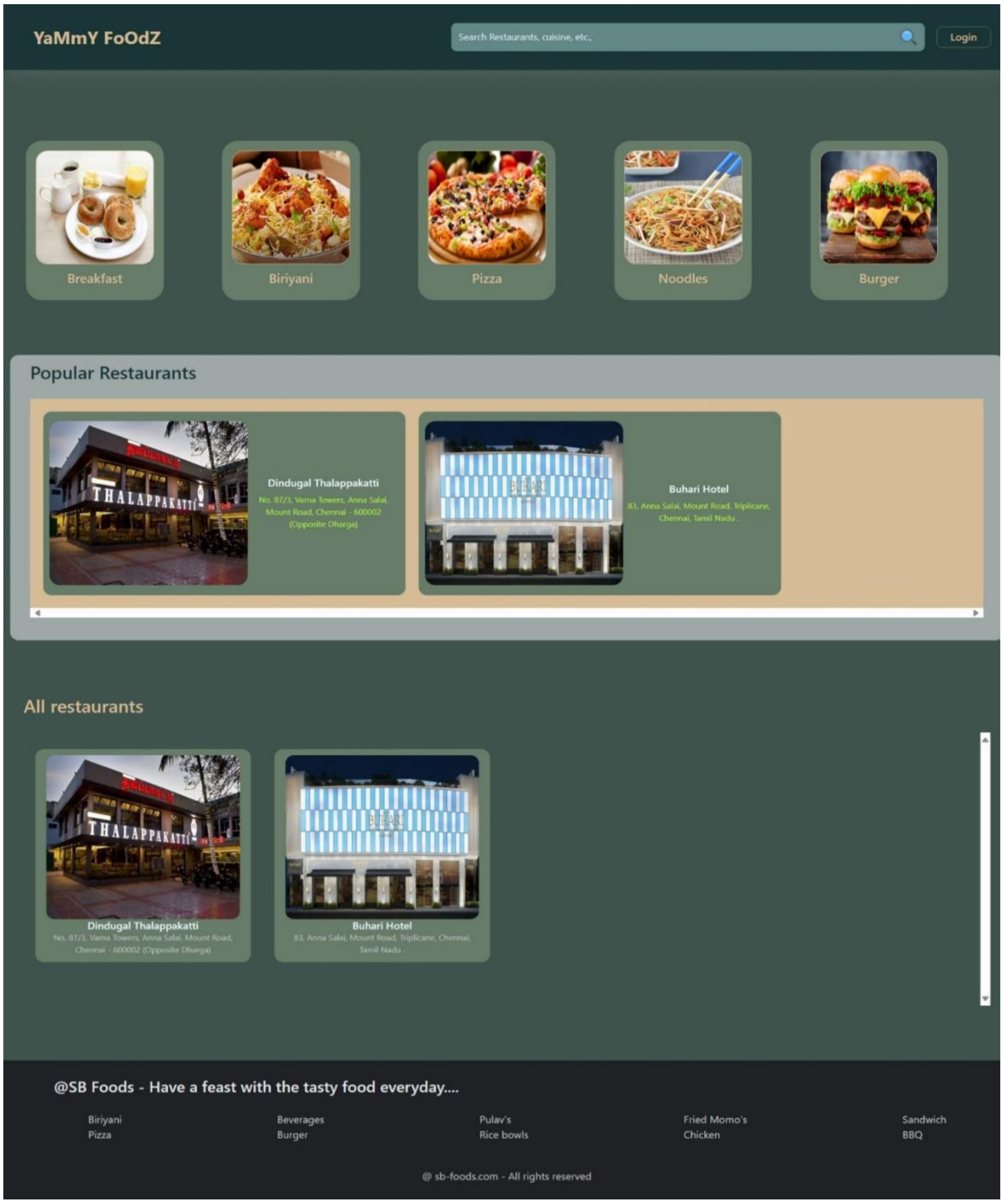
**4. Handling Sessions and Tokens**

1. **Token Expiration**: o Tokens have a defined expiration time (e.g., 1 hour).
    - o After expiration, users must log in again to obtain a new token.
2. **Token Revocation**: o Tokens are stateless, meaning the server does not store session data.
    - o If token revocation is required (e.g., user logout), a blacklist approach can be implemented.
3. **Secure Transmission**: o All requests involving tokens are transmitted over HTTPS to ensure encryption.

# CHAPTER 9

# 9. USER INTERFACE

## a) User-Interface


YaMmY FoOdZ

Search Restaurants, cuisine, etc.,

Login

Breakfast

Biriyani

Pizza

Noodles

Burger

Popular Restaurants

Dindugal Thalappakatti

No. 87/3, Varna Towers, Anna Salai, Mount Road, Chennai - 600002 (Opposite Dharga)

Buhari Hotel

83, Anna Salai, Mount Road, Triplicane, Chennai, Tamil Nadu .

All restaurants

Dindugal Thalappakatti

No. 87/3, Varna Towers, Anna Salai, Mount Road, Chennai - 600002 (Opposite Dharga)

Buhari Hotel

83, Anna Salai, Mount Road, Triplicane, Chennai, Tamil Nadu .

@SB Foods - Have a feast with the tasty food everyday....

Biriyani
Pizza

Beverages
Burger

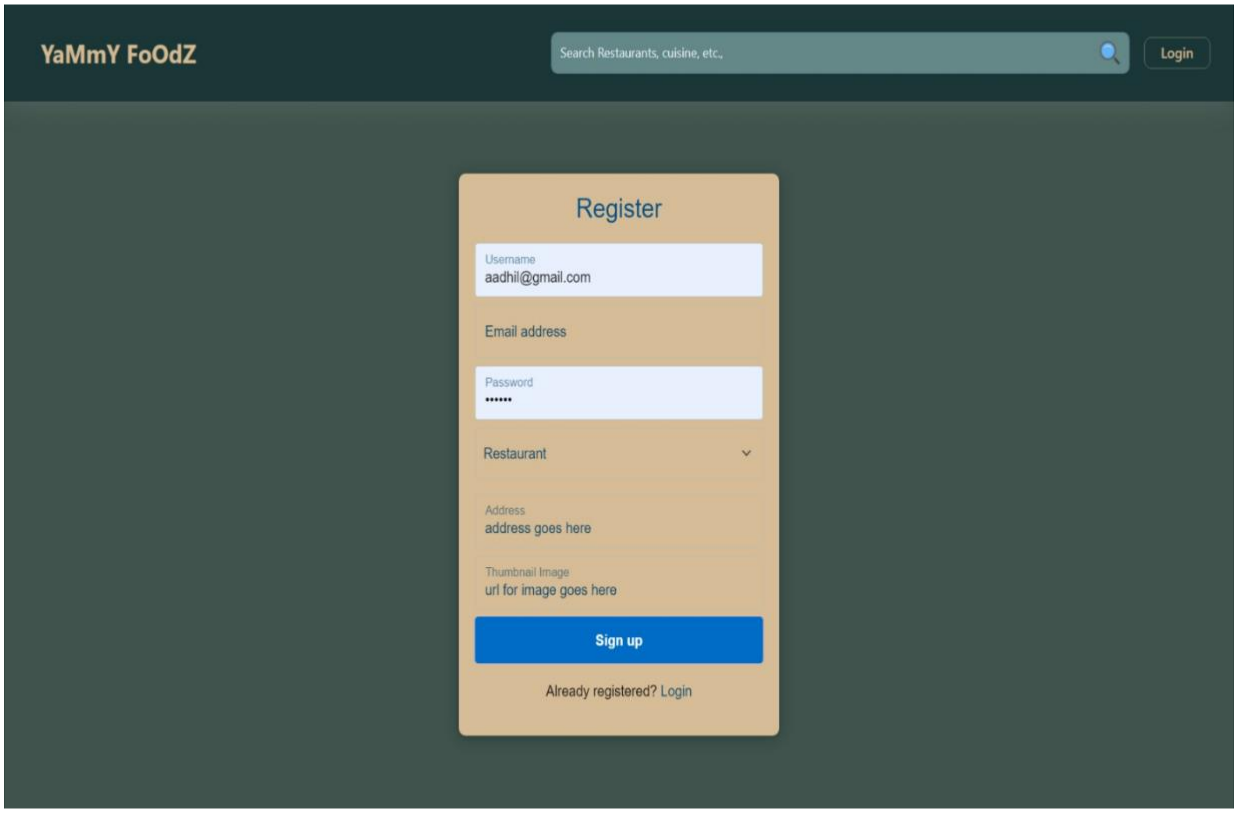
Pulav's
Rice bowls

Fried Momo's
Chicken

Sandwich
BBQ

@ sb-foods.com - All rights reserved

## b) User Registration



Register

Username
aadhil@gmail.com

Email address

Password
••••••

Restaurant

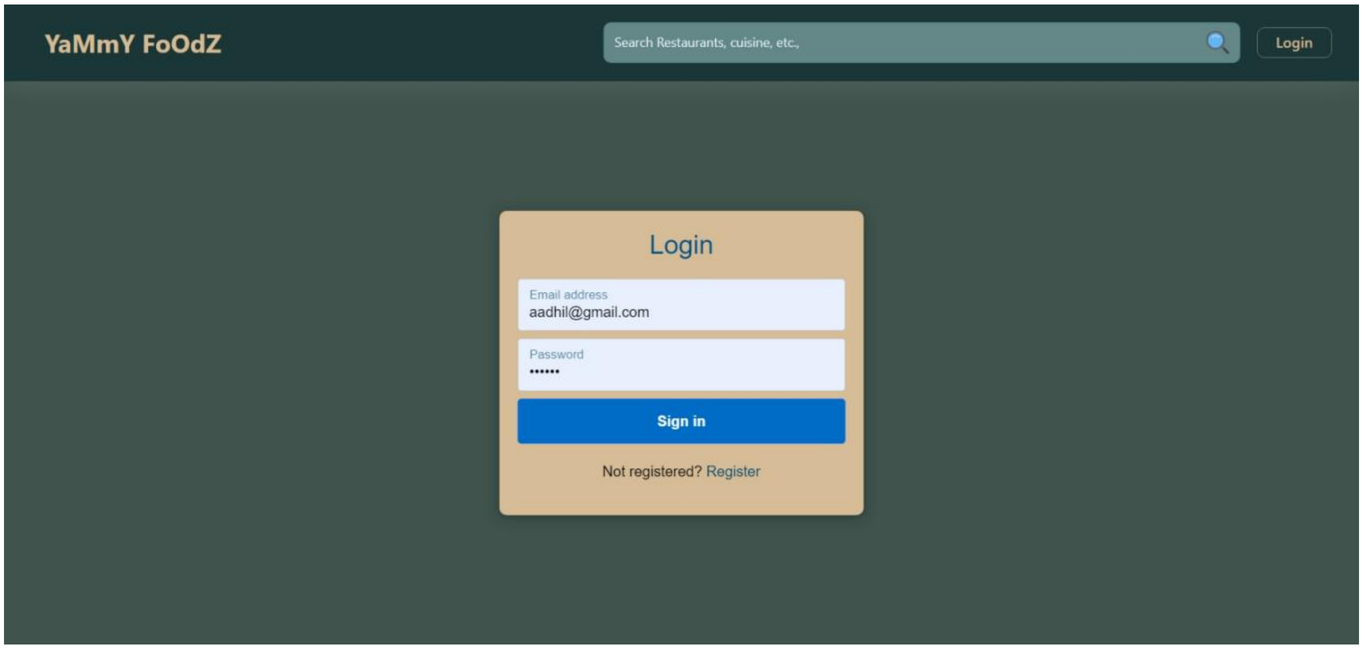
Address
address goes here

Thumbnail Image
url for image goes here

Sign up

Already registered? Login

## c)User Login Page



Login

Email address
aadhil@gmail.com

Password
••••••

Sign in

Not registered? Register

# CHAPTER 10

# 10. TESTING

**Testing Strategy for Food Ordering System**

Testing ensures that the Food Ordering System functions correctly, meets user requirements, and provides a seamless experience for customers, restaurant owners, and admins. Below is a description of the testing strategy and tools used for this project:

**1. Testing Strategy a. Unit Testing**

- **Purpose**: Validate individual components or modules of the application (e.g., APIs, UI components, database queries).
- **Scope**:
  - Frontend: Test React components like `Cart.jsx`, `Profile.jsx`, and `Register.jsx` to ensure they render correctly and handle state changes.
  - Backend: Test API routes (e.g., `/api/users/login`, `/api/orders/:id`) to verify proper data flow and error handling. o Example: Checking if a login API returns a JWT token for valid credentials.

**b. Integration Testing**

- **Purpose**: Verify that different modules of the application (frontend, backend, and database) work together as expected.
- **Scope**:
  - Ensure that data flows correctly from the React frontend to the Node.js backend and the MongoDB database. o Test the user flow, such as adding items to the cart, proceeding to checkout, and placing an order.
  - Example: Verifying that an item added to the cart reflects correctly in the database and frontend UI.

**c. End-to-End (E2E) Testing**

- **Purpose**: Simulate real-world user interactions and test the entire system workflow from start to finish.
- **Scope**:
    - Test scenarios such as user registration, logging in, browsing restaurants, placing an order, and receiving order confirmation.
    - Validate that role-based access works correctly (e.g., admins can view all users, but customers cannot).

**d. Manual Testing**

- **Purpose**: Detect edge cases and usability issues not covered by automated tests.
- **Scope**: o Test responsiveness and design consistency across different devices and browsers.
    - Perform ad-hoc tests for rare cases, such as failed payment handling or invalid login attempts.

**e. Regression Testing**

- **Purpose**: Ensure new features or updates do not break existing functionality.
- **Scope**:
    - Re-run unit and integration tests after making changes to ensure no unintended side effects.
    - Example: If the cart feature is updated, verify that checkout and profile functionalities remain intact.

**2. Tools Used Frontend Testing Tools**

1. **Jest**:
    - A JavaScript testing framework for React components. o Used to test frontend functionality, such as button clicks, state changes, and conditional rendering.

    - Example: Writing tests for the `Register.jsx` component to ensure it displays validation errors for incorrect input.
2. **React Testing Library**:
    - Focuses on testing React components from the user’s perspective. o Used to simulate user interactions like form submissions and button clicks.
    - Example: Testing that the "Place Order" button triggers the correct API call.

**Backend Testing Tools**

1. **Mocha**:
    - A Node.js testing framework used for testing backend functionality. o Used to test API routes and middleware logic.
    - Example: Verifying that the `/api/products` endpoint returns the correct list of products.
2. **Chai**:
    - An assertion library used with Mocha for writing test cases.
    - Example: Ensuring that a ‘403’status code is returned for unauthorized API access.
3. **Supertest**:
    - A library for testing HTTP requests to Node.js APIs. o Example: Testing that a POST request to `/api/orders` creates an order in the database.

**Database Testing Tools**

1. **MongoDB Atlas**:
    - Provides a testing environment for database queries and operations.
    - Example: Verifying that an order is saved with the correct user and product details.
2. **Mongoose**:
    - A library for mocking MongoDB in unit tests. o Example: Simulating database queries to test API logic without requiring a live database.

**End-to-End Testing Tools**

1. **Cypress**:
   - A powerful E2E testing framework for simulating user interactions.
   - Used to test full workflows, such as user login, adding items to the cart, and completing checkout. o Example: Automating the process of logging in as a user and verifying that the profile page displays the correct data.
2. **Postman**:
   - A tool for manually testing APIs.
   - Used to test backend endpoints and ensure they return the correct data and status codes.
   - Example: Sending a `GET /api/restaurants` request and verifying the response format.

**3. Sample Test Scenarios Frontend**

- Verify that invalid email input in the registration form displays an error message.
- Check that the cart page updates dynamically when items are added or removed.

**Backend**

- Ensure that the `/api/users/login` route rejects invalid credentials with a 401 error.
- Verify that only admins can access the `/api/admin/users` endpoint. **Database**

- Validate that deleted products are removed from all associated user carts.
- Ensure that orders are saved with accurate timestamps and status values.

**End-to-End**

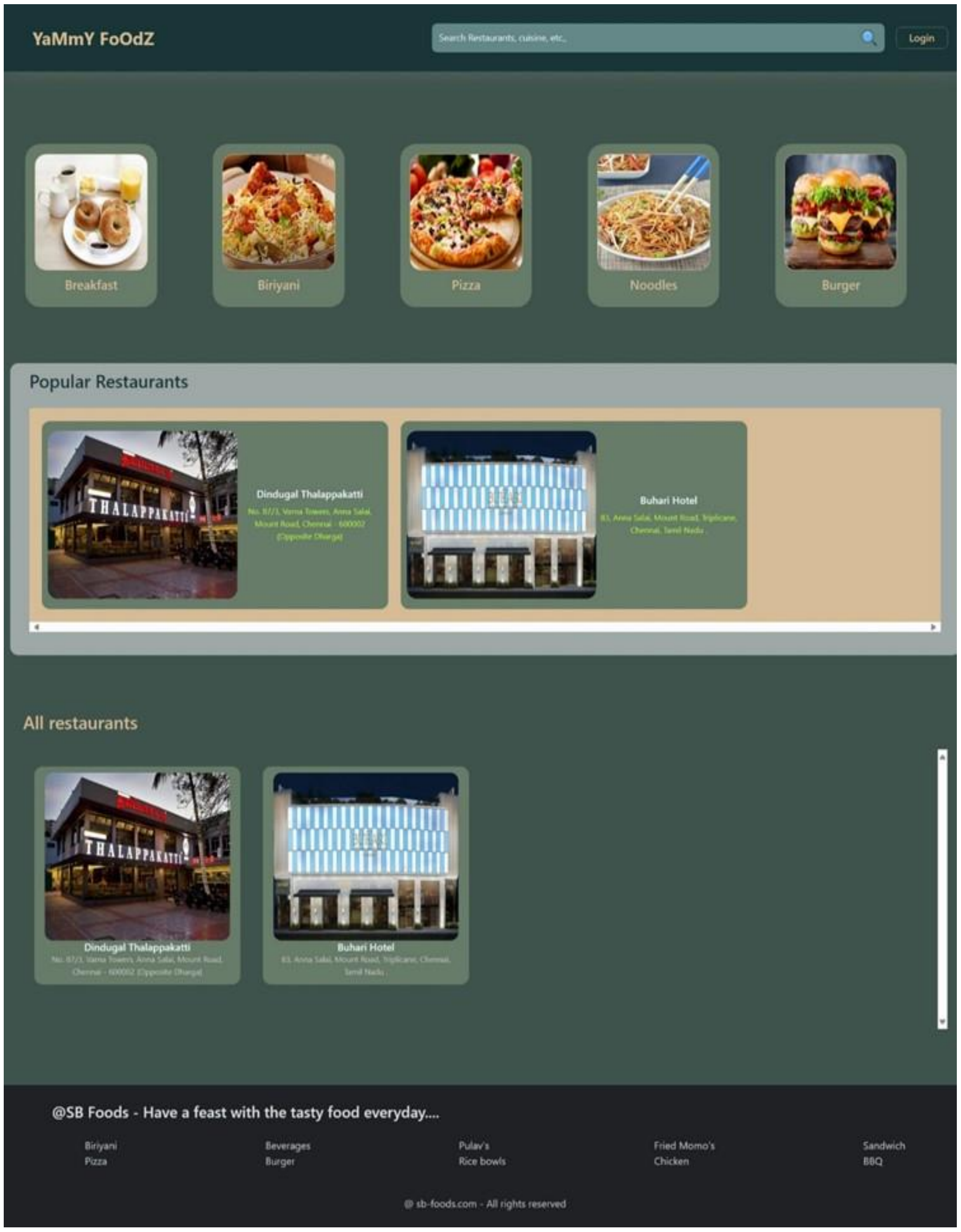
- Test the workflow for a customer placing an order, from login to receiving order confirmation.

- Verify that a restaurant owner can add a product and see it listed on the frontend.

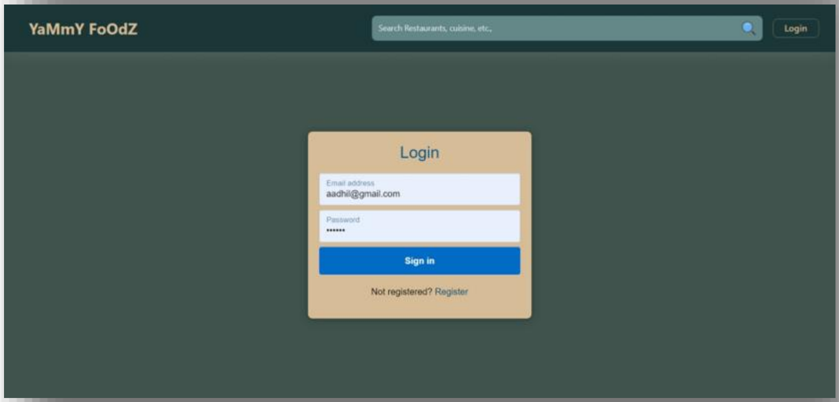
# CHAPTER 11

# 11. PROJECT RESULTS

## I. USER INTERFACE


YaMmY FoOdZ
Search Restaurants, cuisine, etc.,
Login
Breakfast
Biriyani
Pizza
Noodles
Burger
Popular Restaurants
Dindugal Thalappakatti
No. 87/3, Varna Towers, Anna Salai, Mount Road, Chennai - 600002 (Opposite Dharga)
Buhari Hotel
83, Anna Salai, Mount Road, Triplicane, Chennai, Tamil Nadu .
All restaurants
Dindugal Thalappakatti
No. 87/3, Varna Towers, Anna Salai, Mount Road, Chennai - 600002 (Opposite Dharga)
Buhari Hotel
83, Anna Salai, Mount Road, Triplicane, Chennai, Tamil Nadu .
@SB Foods - Have a feast with the tasty food everyday....
Biriyani
Pizza
Beverages
Burger
Pulav's
Rice bowls
Fried Momo's
Chicken
Sandwich
BBQ
@ sb-foods.com - All rights reserved

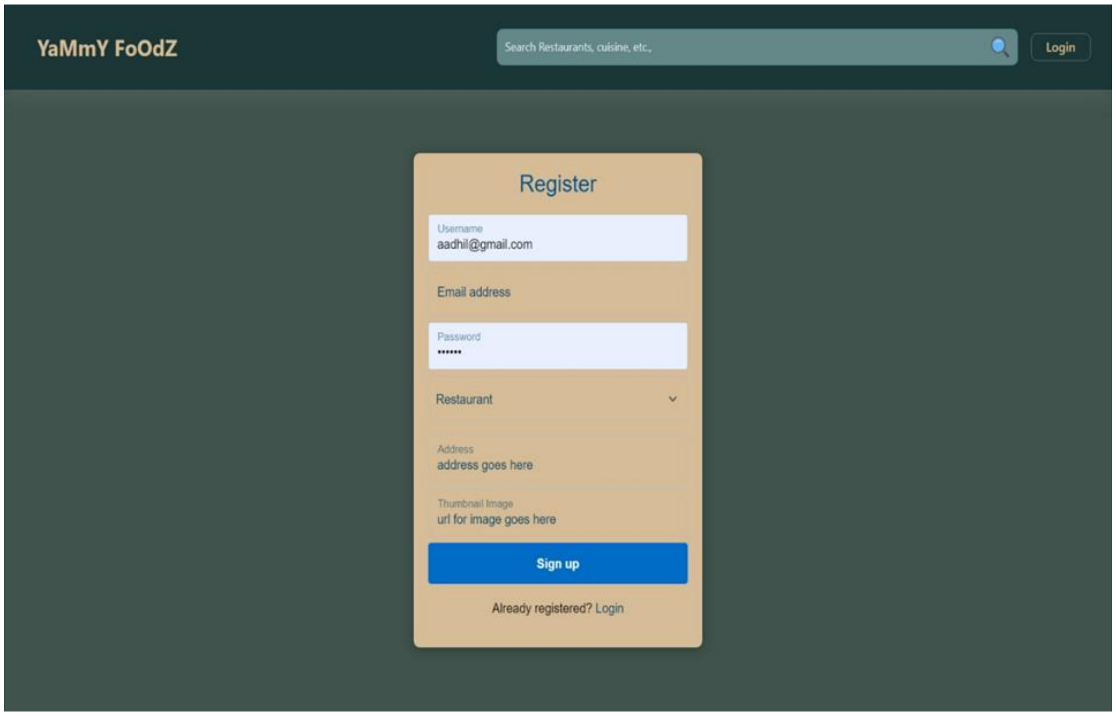
## II. LOGIN



Email address
aadhil@gmail.com
Password
••••••

Not registered? Register

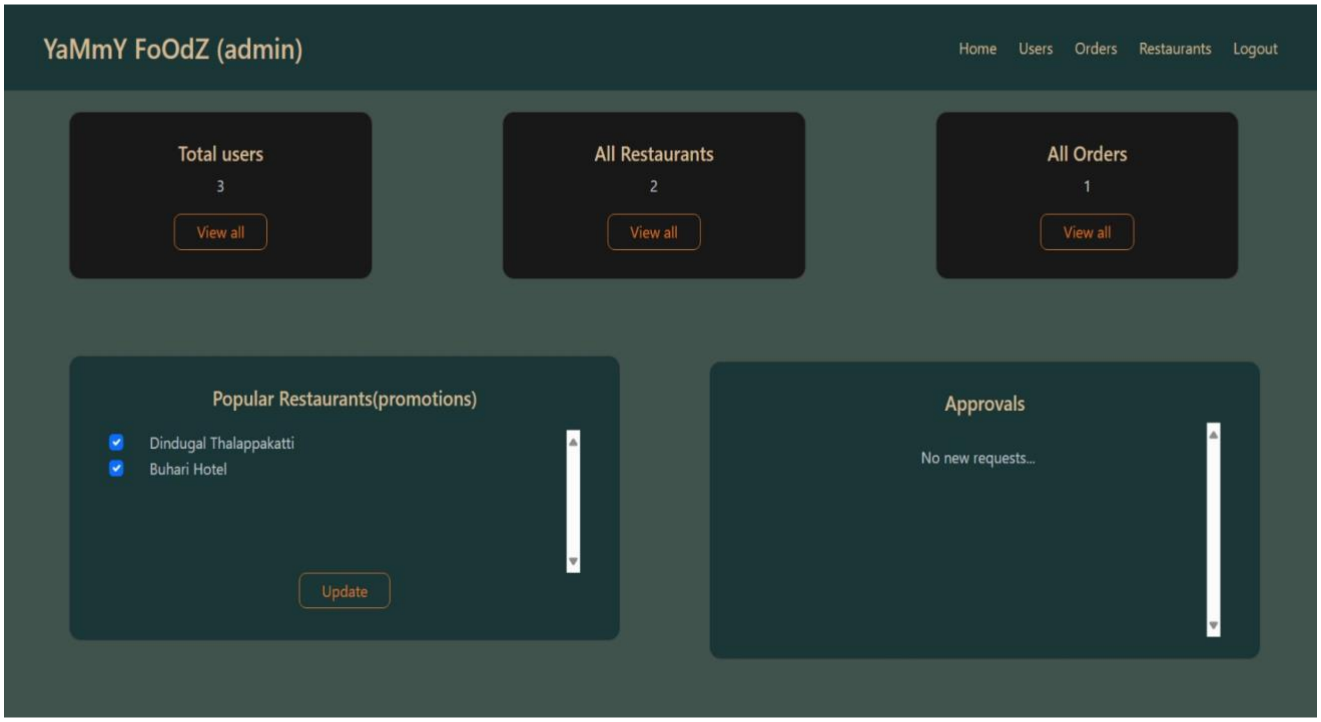
## III. REGISTER



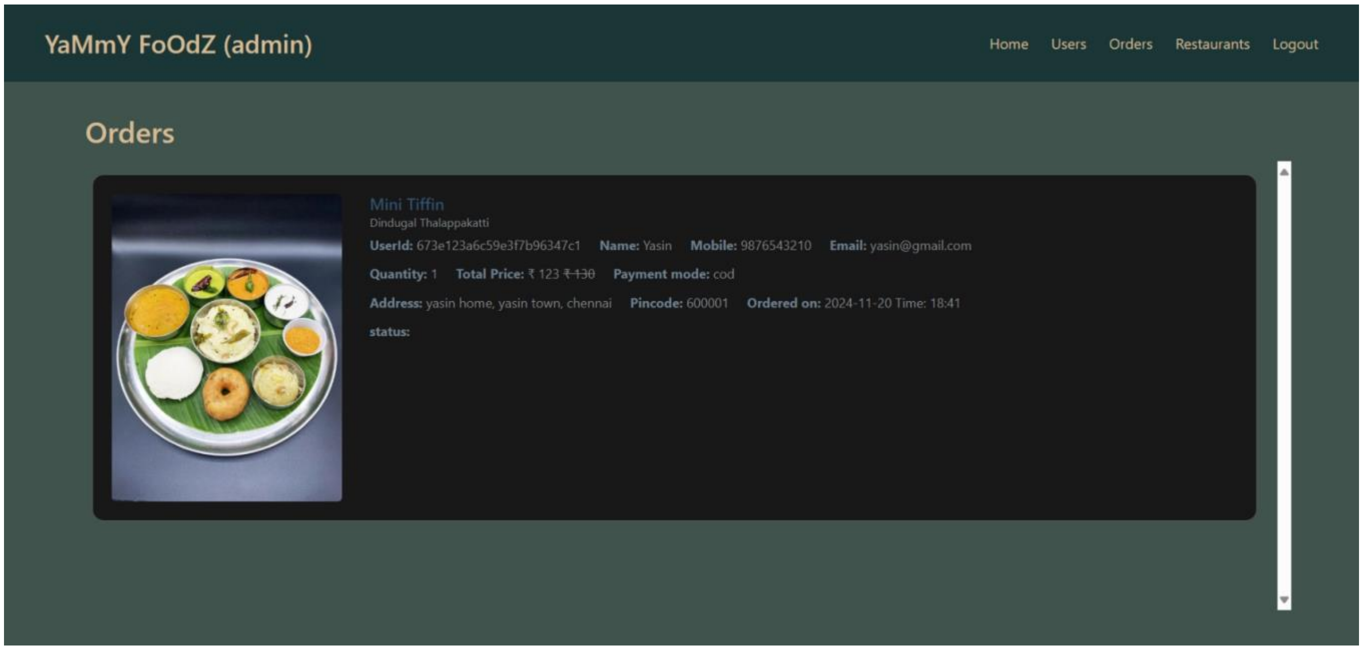
Username
aadhil@gmail.com
Email address
Password
••••••
Restaurant
Address
address goes here
Thumbnail Image
url for image goes here

Already registered? Login

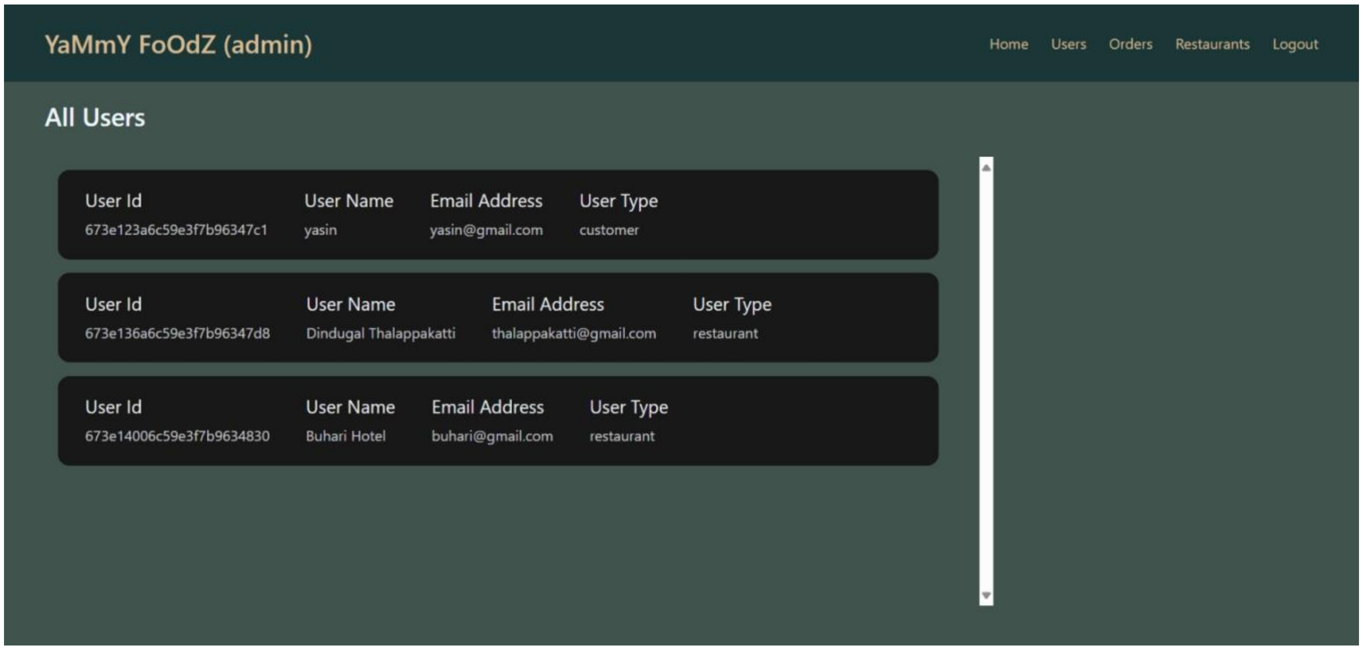
## IV. ADMIN HOME PAGE



## ADMIN VIEW

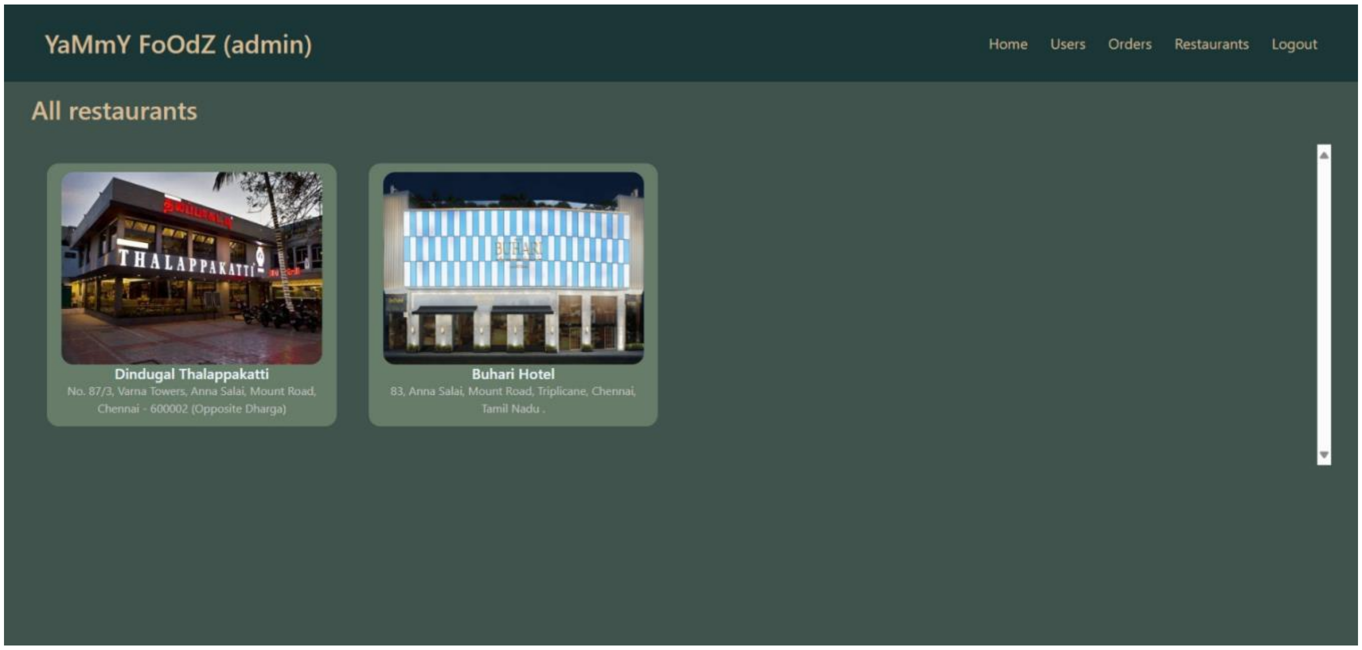


Mini Tiffin
Dindugal Thalappakatti
**UserId:** 673e123a6c59e3f7b96347c1 **Name:** Yasin **Mobile:** 9876543210 **Email:** yasin@gmail.com
**Quantity:** 1 **Total Price:** ₹ 123 ~~₹ 130~~ **Payment mode:** cod
**Address:** yasin home, yasin town, chennai **Pincode:** 600001 **Ordered on:** 2024-11-20 Time: 18:41
**status:**



All Users

| User Id | User Name | Email Address | User Type |
|---|---|---|---|
| 673e123a6c59e3f7b96347c1 | yasin | yasin@gmail.com | customer |
| 673e136a6c59e3f7b96347d8 | Dindugal Thalappakatti | thalappakatti@gmail.com | restaurant |
| 673e14006c59e3f7b9634830 | Buhari Hotel | buhari@gmail.com | restaurant |

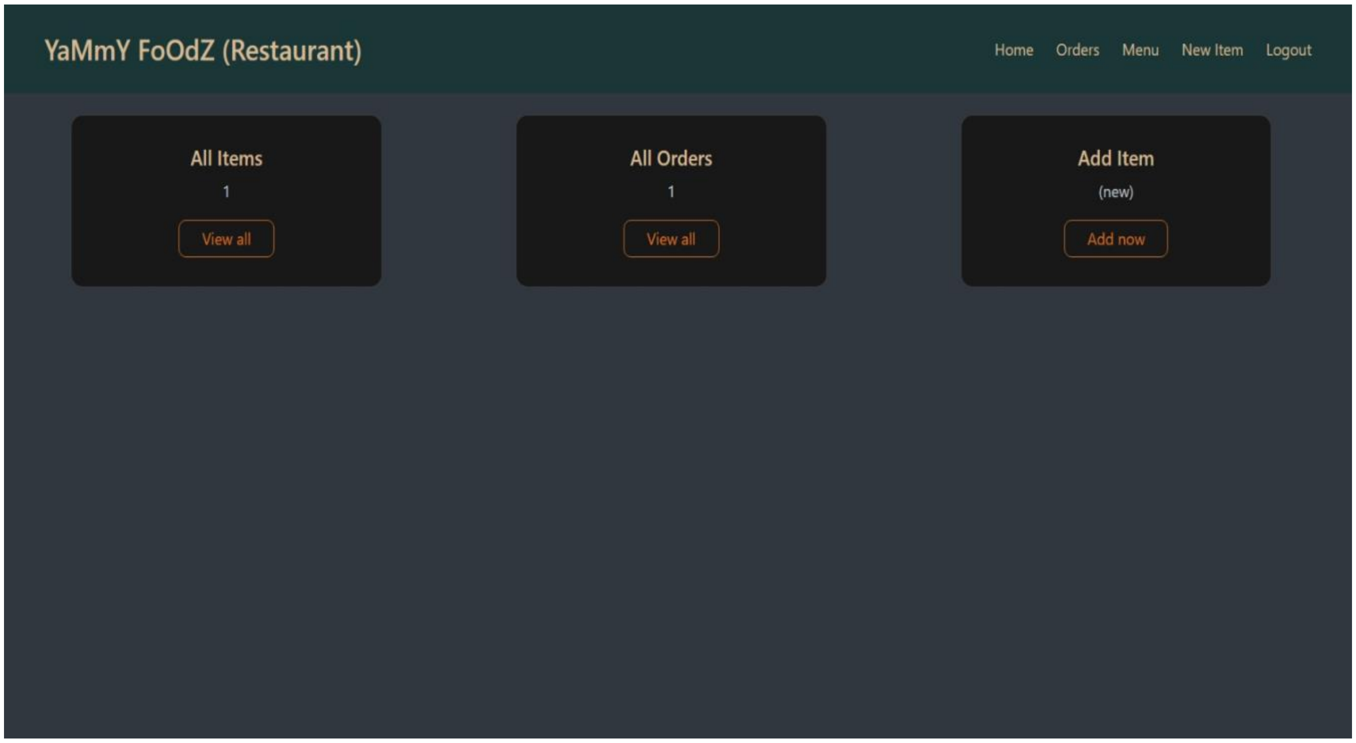


All restaurants

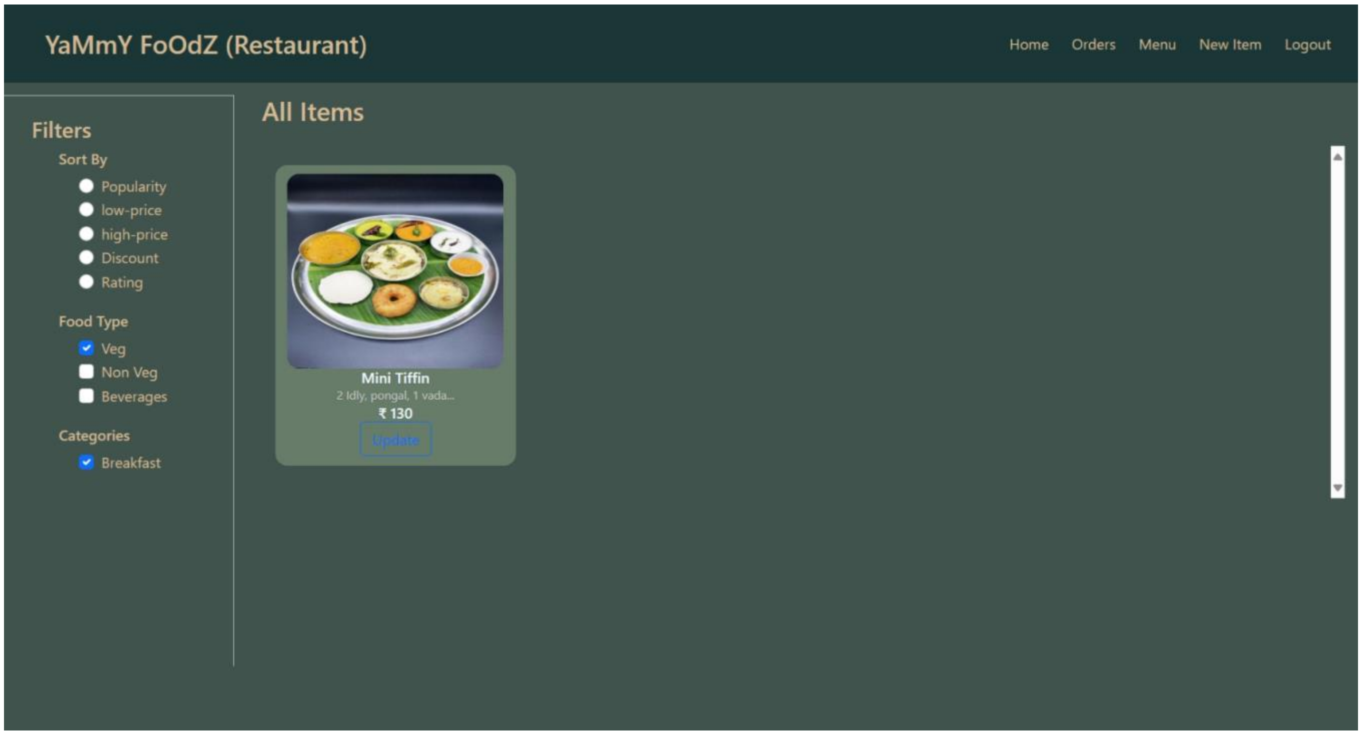
**Dindugal Thalappakatti**
No. 87/3, Varna Towers, Anna Salai, Mount Road, Chennai - 600002 (Opposite Dharga)

**Buhari Hotel**
83, Anna Salai, Mount Road, Triplicane, Chennai, Tamil Nadu .

## V. RESTURANT HOME



## RESTAURANT PAGES



All Items

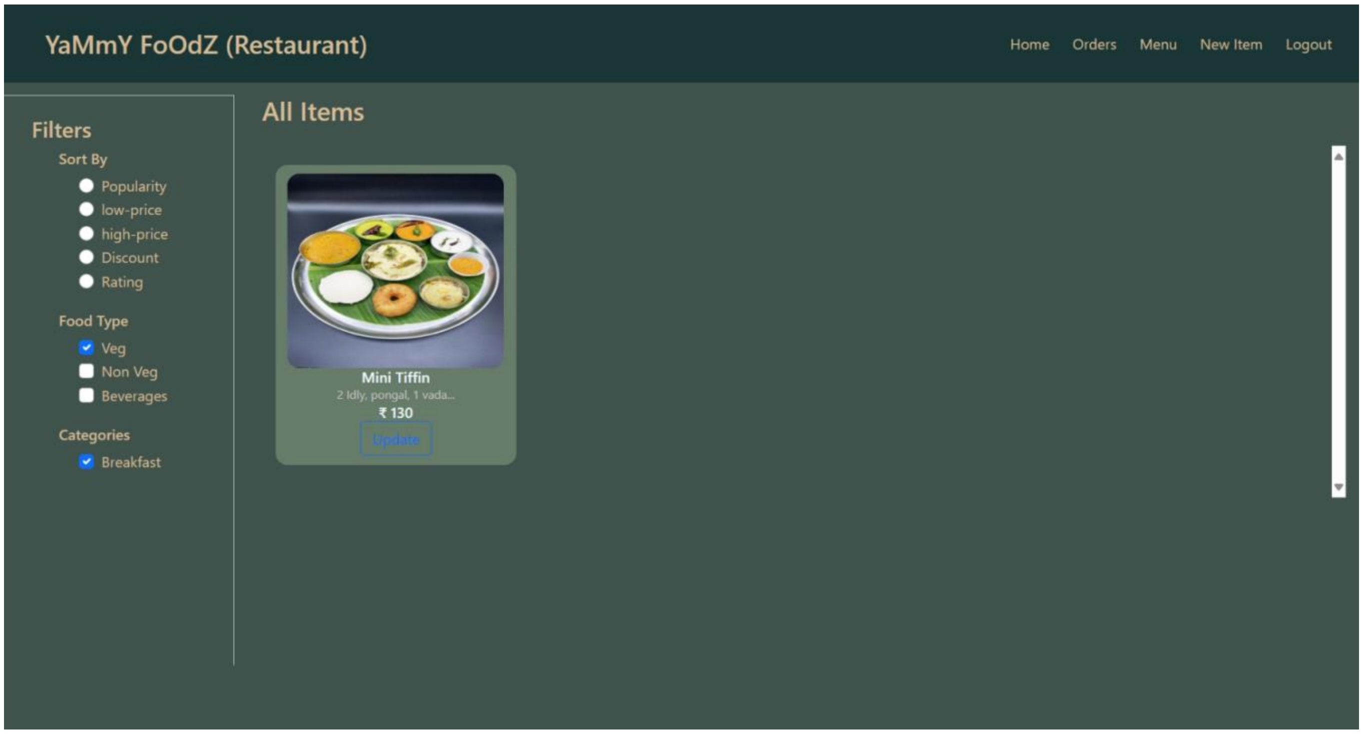
Filters

Sort By
- ○ Popularity
- ○ low-price
- ○ high-price
- ○ Discount
- ○ Rating

Food Type
- [✓] Veg
- [ ] Non Veg
- [ ] Beverages

Categories
- [✓] Breakfast

Mini Tiffin
2 Idly, pongal, 1 vada...
₹ 130


YaMmY FoOdZ (Restaurant)
Home Orders Menu New Item Logout

All Items

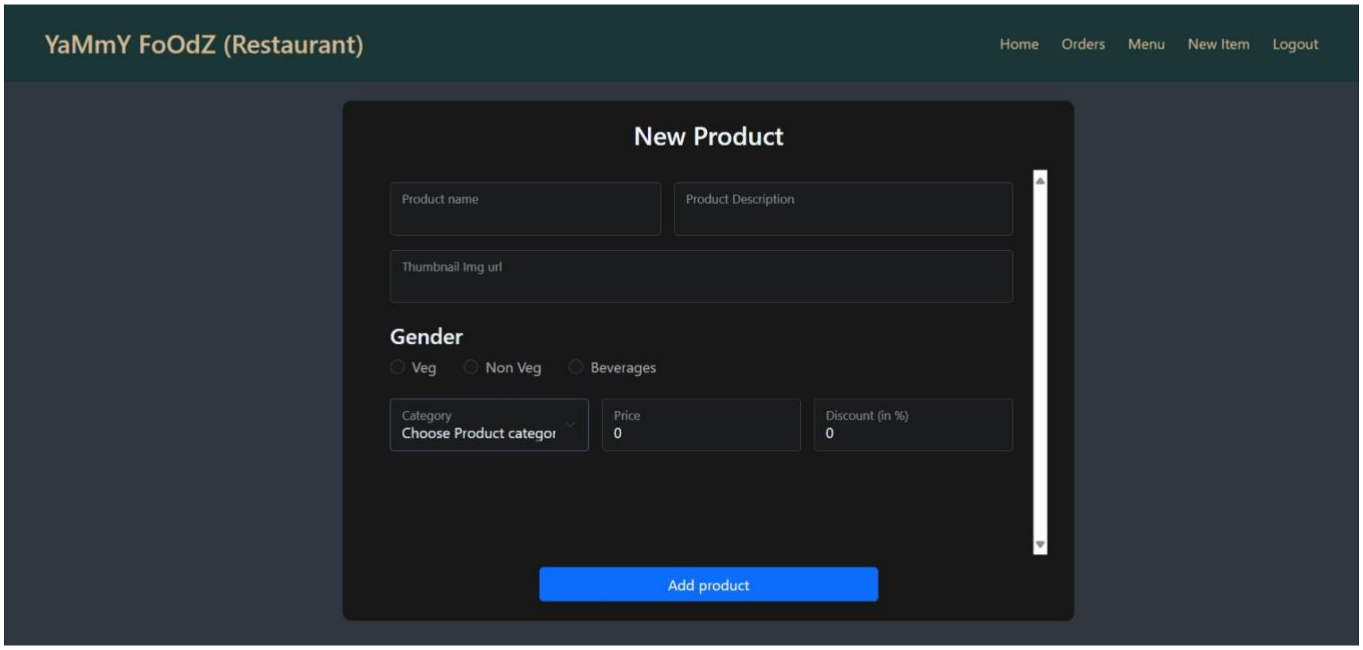
Filters

Sort By
- ○ Popularity
- ○ low-price
- ○ high-price
- ○ Discount
- ○ Rating

Food Type
- [✓] Veg
- [ ] Non Veg
- [ ] Beverages

Categories
- [✓] Breakfast

Mini Tiffin
2 Idly, pongal, 1 vada...
₹ 130
Update



New Product

Product name

Product Description

Thumbnail Img url

Gender

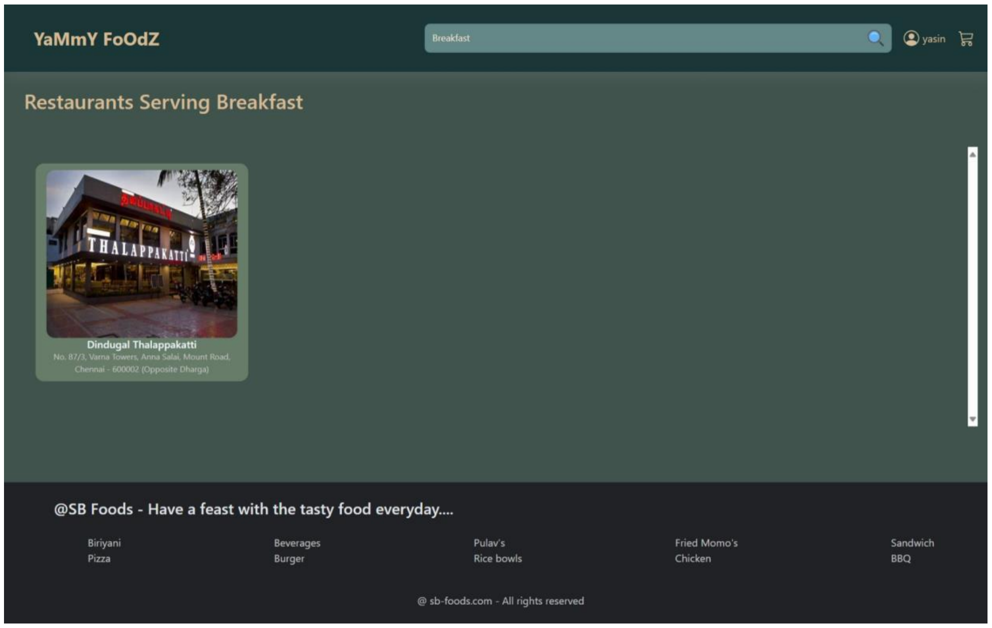
Veg Non Veg Beverages

Category: Choose Product categor

Price: 0

Discount (in %): 0

## VI. CUSTOMER LOGIN

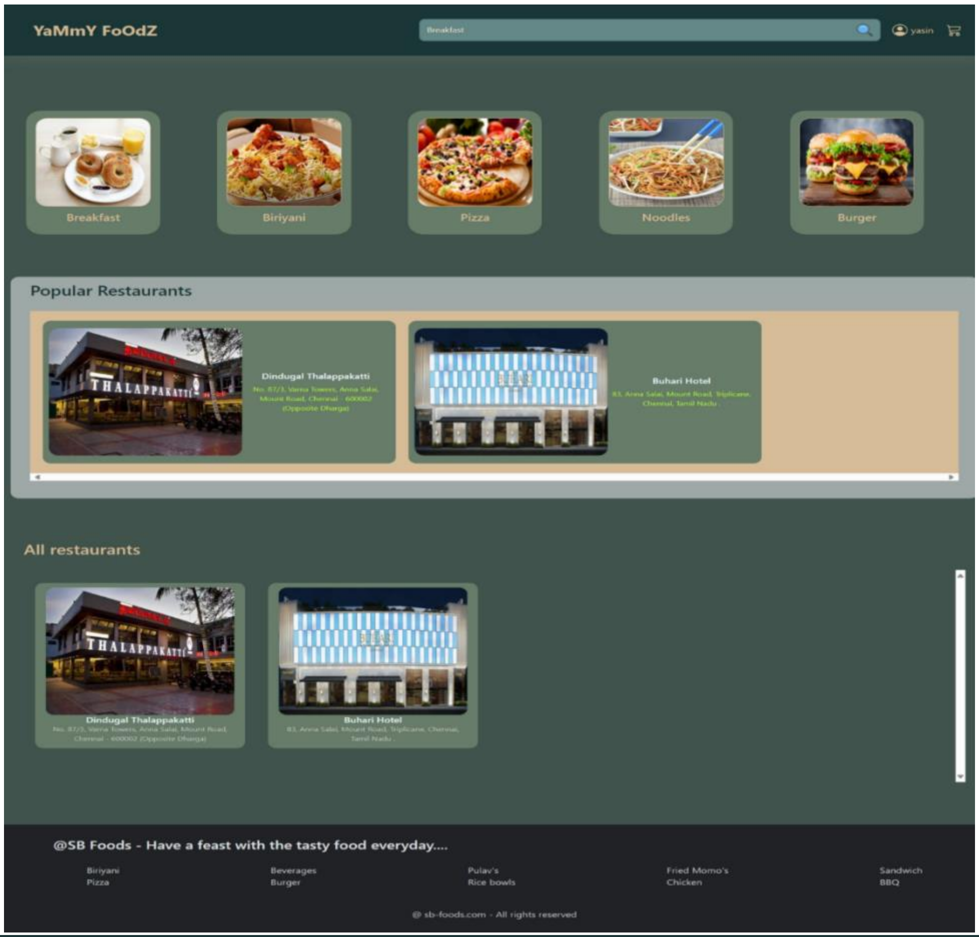


Restaurants Serving Breakfast

Dindugal Thalappakatti
No. 87/3, Varna Towers, Anna Salai, Mount Road, Chennai - 600002 (Opposite Dharga)

@SB Foods - Have a feast with the tasty food everyday....

@ sb-foods.com - All rights reserved



YaMmY FoOdZ

Breakfast

yasin

Popular Restaurants

Dindugal Thalappakatti
No. 87/3, Varna Towers, Anna Salai, Mount Road, Chennai - 600002 (Opposite Dharga)

Buhari Hotel
83, Anna Salai, Mount Road, Triplicane, Chennai, Tamil Nadu .

All restaurants

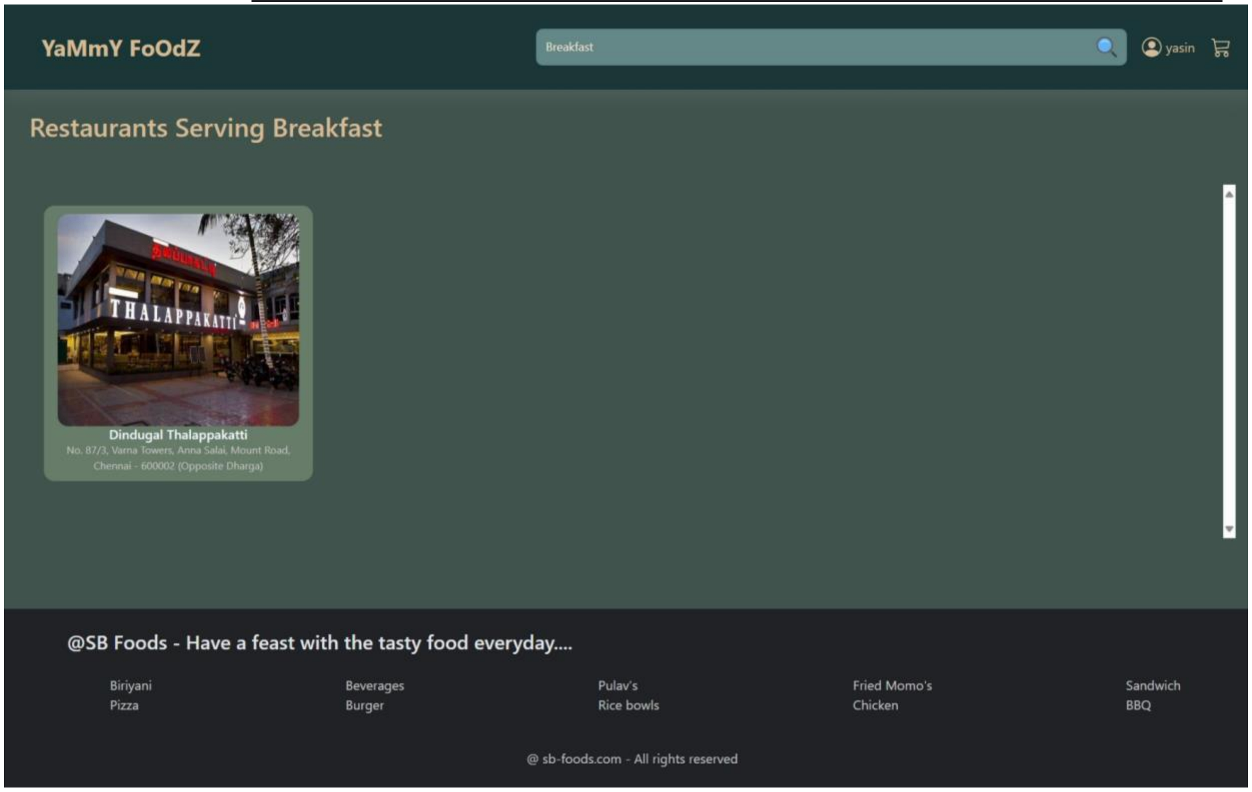
Dindugal Thalappakatti
No. 87/3, Varna Towers, Anna Salai, Mount Road, Chennai - 600002 (Opposite Dharga)

Buhari Hotel
83, Anna Salai, Mount Road, Triplicane, Chennai, Tamil Nadu .

@SB Foods - Have a feast with the tasty food everyday....

| | | | | |
|---|---|---|---|---|
| Biriyani | Beverages | Pulav's | Fried Momo's | Sandwich |
| Pizza | Burger | Rice bowls | Chicken | BBQ |

@ sb-foods.com - All rights reserved



YaMmY FoOdZ

Breakfast

yasin

Restaurants Serving Breakfast

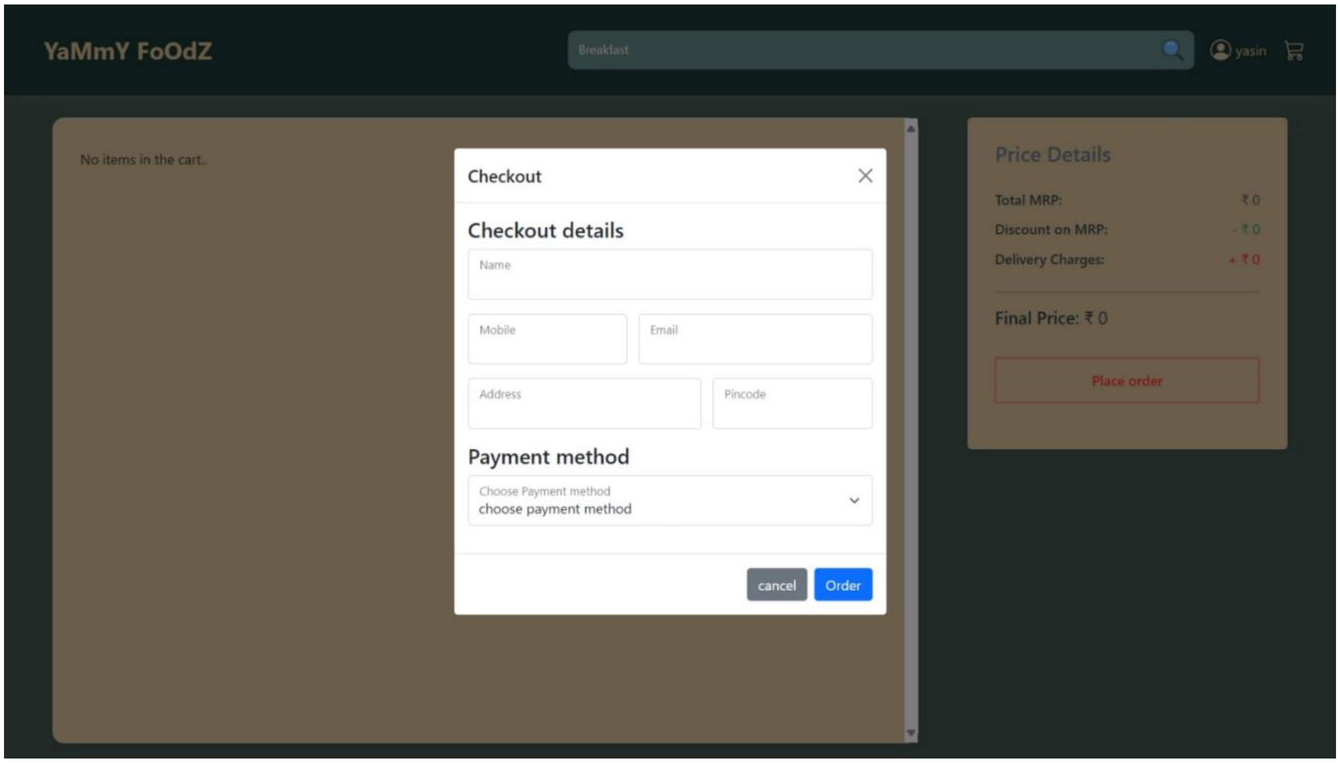
Dindugal Thalappakatti
No. 87/3, Varna Towers, Anna Salai, Mount Road, Chennai - 600002 (Opposite Dharga)

@SB Foods - Have a feast with the tasty food everyday....

| | | | | |
|---|---|---|---|---|
| Biriyani | Beverages | Pulav's | Fried Momo's | Sandwich |
| Pizza | Burger | Rice bowls | Chicken | BBQ |

@ sb-foods.com - All rights reserved

## VII. CART - **PAYMENT PAGE**



No items in the cart..

Checkout

Checkout details

Name

Mobile

Email

Address

Pincode

Payment method

Choose Payment method
choose payment method

cancel Order

Price Details

| | |
|---|---|
| Total MRP: | ₹ 0 |
| Discount on MRP: | - ₹ 0 |
| Delivery Charges: | + ₹ 0 |

Final Price: ₹ 0

Place order

# CHAPTER 12

# 12. KNOWN ISSUES

While the Food Ordering System is functional and user-friendly, there are a few known issues that may affect certain aspects of the application. These issues are actively being worked on or documented for further improvements.

## 1. User Interface Issues

- **Issue**: The website may experience responsiveness problems on smaller screen sizes or older browsers.
    - **Impact**: Some components like the navigation bar or buttons may not align properly on mobile devices.
    - **Workaround**: Recommend using modern browsers and larger screen resolutions for the best experience.
- **Issue**: Minor delay in loading components like restaurant menus or product details due to unoptimized images.
    - **Impact**: Users might experience slight lag while browsing menus with large images. o **Workaround**: Optimize images before uploading them to the database.

## 2. Authentication and Authorization

- **Issue**: Tokens do not expire automatically if a user logs out without clearing their local storage.
    - **Impact**: Users can manually reuse the token from local storage to bypass login until it expires. o **Workaround**: Implement token invalidation on logout.
- **Issue**: Admin access permissions are not granular. All admins currently have the same level of access.
    - **Impact**: This may pose challenges in larger teams requiring role-based access.

  - **Workaround**: Introduce role-based permissions for admins in future updates.

**3. Backend/API**

- **Issue**: API error messages are generic in some cases (e.g., login failures or invalid product requests).
  - **Impact**: Developers and users may find it difficult to debug specific issues.
  - **Workaround**: Improve error-handling mechanisms to provide more descriptive messages.
- **Issue**: High traffic may result in slower response times due to insufficient backend optimization.
  - **Impact**: Performance could degrade when handling simultaneous requests.
  - **Workaround**: Implement caching mechanisms (e.g., Redis) for frequently accessed endpoints.

**4. Database**

- **Issue**: No cascading delete for associated data (e.g., removing a user does not automatically remove their cart or order history).
  - **Impact**: Leaves orphaned data in the database, increasing storage usage.
  - **Workaround**: Ensure manual cleanup of associated data or implement cascading delete functionality.
- **Issue**: Search functionality for restaurants and products is limited.
  - **Impact**: Users cannot use partial matches or advanced filters effectively. o **Workaround**: Plan to implement full-text search indexing in MongoDB.

**5. Payment Integration**

- **Issue**: Payment gateway sandbox mode sometimes fails during testing.
  - **Impact**: Test payments might not process consistently, affecting QA testing.

    - **Workaround**: Use alternative sandbox accounts or verify with live payment credentials (in a controlled environment).
- **Issue**: Payment failures do not automatically retry or provide detailed feedback to the user.
    - **Impact**: Users need to restart the order process in case of payment errors. o **Workaround**: Add an order retry mechanism for failed payments.

**6. Notifications**

- **Issue**: Notifications for order updates (e.g., "Order Accepted" or "Order Delivered") are delayed.
    - **Impact**: Users might not receive timely status updates for their orders.
    - **Workaround**: Optimize notification triggers and improve WebSocket configurations for real-time updates.

# CHAPTER 13

# 13. FUTURE ENHANCEMENTS

The Food Ordering System has a solid foundation, but there is always room for improvement and innovation to provide an even better user experience. Below are some potential future enhancements that can be implemented:

**1. Advanced Search and Filtering Options**

- **Description**: Enhance the search functionality to include filters based on cuisine, dietary preferences (e.g., vegetarian, vegan, gluten-free), price range, and restaurant ratings.
- **Impact**: Makes it easier for users to find exactly what they are looking for quickly.

**2. Real-Time Order Tracking**

- **Description**: Integrate real-time GPS tracking for orders, allowing users to track the exact location of their food delivery.
- **Impact**: Increases user satisfaction by providing transparency and reducing uncertainty about delivery times.

**3. AI-Powered Recommendations**

- **Description**: Use machine learning to recommend restaurants and dishes based on users' past orders, browsing history, and preferences.
- **Impact**: Provides a personalized user experience and increases user engagement.

**4. Multi-Language and Currency Support**

- **Description**: Add support for multiple languages and currencies to make the platform accessible to a global audience.
- **Impact**: Expands the reach of the application and enhances usability for users in different regions.

**5. Loyalty and Rewards Program**

- **Description**: Implement a rewards system where users earn points for each order, which can be redeemed for discounts or free items.
- **Impact**: Encourages repeat usage and builds customer loyalty.

**6. Push Notifications**

- **Description**: Add push notifications to inform users about order updates, exclusive deals, or discounts from their favorite restaurants.
- **Impact**: Keeps users engaged and improves communication.

**7. Enhanced Restaurant Dashboard**

- **Description**: Improve the restaurant dashboard with analytics for sales, customer preferences, and feedback.
- **Impact**: Empowers restaurant owners to make data-driven decisions and improve their offerings.

**8. Offline Functionality**

- **Description**: Allow users to browse previously loaded menus and add items to their cart even when offline, with orders syncing when the connection is restored.
- **Impact**: Provides a seamless user experience, especially in areas with unstable internet connections.

**9. Integration with Voice Assistants**

- **Description**: Enable users to order food using voice commands through popular assistants like Alexa, Google Assistant, or Siri.
- **Impact**: Offers a convenient, hands-free ordering experience for tech-savvy users.

**10. Subscription Plans**

- **Description**: Introduce subscription models (e.g., premium users get free delivery, exclusive offers, and priority customer support).
- **Impact**: Provides additional revenue streams while offering value-added services to customers.

**11. Social Media Integration**

- **Description**: Allow users to share their favorite dishes or restaurant experiences directly on social media platforms.

- **Impact**: Increases the app’s visibility and attracts new users through organic marketing.

## 12. Advanced Security Features

- **Description**: Incorporate advanced authentication methods, such as biometric login (fingerprint or facial recognition) and two-factor authentication (2FA).
- **Impact**: Improves data security and builds user trust.

## 13. Dark Mode

- **Description**: Add a dark mode theme to the application for users who prefer a low-light interface.
- **Impact**: Enhances usability and provides a modern, user-friendly design option.

## 14. Multi-Vendor Support

- **Description**: Allow users to order from multiple restaurants in a single transaction.
- **Impact**: Provides greater convenience and enhances user satisfaction.

## 15. Feedback and Review System

- **Description**: Enable users to leave detailed feedback for restaurants and dishes, along with a rating system.
- **Impact**: Helps restaurants improve their services and assists users in making informed decisions.

## Conclusion

These future enhancements aim to expand the functionality, usability, and user satisfaction of the Food Ordering System. By addressing user needs and adopting modern technologies, the application can stay competitive and grow in popularity.