**RESTAURANT WEBSITE**

**ABSTRACT:**

The *Restaurant Website Backend* project is developed to digitalize and streamline the essential operations of a restaurant, enhancing both customer experience and internal efficiency. In today’s fast-paced, technology-driven world, restaurants are increasingly expected to offer online services that are responsive, secure, and user-friendly. This backend system is designed as a centralized engine that powers key restaurant functionalities such as user registration, secure login, menu browsing, food ordering, and table reservations, all through a seamless web interface.

The backend has been implemented using the MEAN stack—MongoDB, Express.js, Angular (assumed frontend), and Node.js. This powerful combination of technologies enables a full-stack JavaScript environment that ensures fast development, real-time updates, and efficient data communication between the client and the server. MongoDB serves as the non-relational database system, which is ideal for storing flexible, schema-less data structures such as food items, user profiles, and orders. Express.js is used to create RESTful API endpoints, handling server-side logic and routing, while Node.js provides the runtime environment, managing asynchronous operations and scaling efficiently. Though Angular is assumed to be used on the frontend, the backend is designed to support API-based integration with any modern frontend framework.

Key functionalities offered by the backend include dynamic loading of menu items, managing customer sessions, handling shopping cart and checkout processes, processing food orders, and managing reservations with appropriate date and time validations. Security is also a priority in this system—user authentication is handled securely with encrypted credentials and session management to protect user data and privacy.

One of the distinguishing features of the project is its modular architecture, which ensures flexibility and scalability. The backend is capable of handling multiple users simultaneously, updating menu information in real-time, and supporting future enhancements like online payments, review systems, and delivery tracking. This makes it a robust foundation for developing a full-fledged restaurant management system.

The documentation outlines the complete backend development lifecycle, starting from system analysis and requirement gathering to design, implementation, testing, and deployment. It also discusses the challenges faced during the development process, such as data modeling, API structuring, and error handling, and how they were resolved. Furthermore, it proposes future improvements, including admin dashboards, push notifications, loyalty programs, and third-party service integrations.

Ultimately, this project aims to create a practical and scalable backend system that not only facilitates online interaction between restaurants and customers but also optimizes the internal workflow, thus contributing to the overall digital transformation of the restaurant business.

**INTRODUCTION:**

**Project Overview:**

The *Restaurant Website Backend* project provides a robust and scalable foundation for managing a restaurant's core digital operations. Designed to serve as the central processing unit of the web application, the backend handles all server-side logic and ensures smooth interaction between the user interface and the database. It encompasses critical functionalities such as user account management, menu data handling, order placement, and table reservation systems. By integrating these components, the backend not only supports user-facing features but also lays the groundwork for operational automation and efficiency. This system is essential for ensuring real-time data updates, managing large volumes of transactions, and maintaining secure access to sensitive information.

**Objectives:**

The primary goal of the project is to build a reliable, efficient, and secure backend system that enables seamless interaction for both customers and administrators. The specific objectives include:

* Enable secure user registration and login:  
  To implement authentication mechanisms that protect user credentials and prevent unauthorized access through encrypted passwords and session control.
* Allow users to place food orders and reserve tables online:  
  To create intuitive endpoints that handle food ordering and reservation requests in real-time, ensuring availability checks, validation, and data consistency.
* Store and manage restaurant data securely:  
  To utilize a flexible database (MongoDB) for storing structured and unstructured data related to users, orders, menus, and reservations, with secure access and integrity control.
* Facilitate backend scalability and maintainability:  
  To adopt a modular architecture that allows future feature additions and system upgrades without major overhauls, ensuring long-term maintainability and performance.

**Scope:**

The scope of this backend project encompasses the entire server-side infrastructure for a restaurant web application. It acts as a bridge between the frontend interface (assumed to be built with Angular) and the MongoDB database. This backend handles all HTTP API requests, processes business logic, performs database queries, and returns structured responses to the frontend. The system includes modules for:

* User account management (sign-up, login, and authentication)
* Menu item retrieval and management
* Order processing and storage
* Table reservation management
* Administrative data operations (future scope)

While the current implementation focuses on core restaurant operations, the backend is designed with extensibility in mind to support features such as online payment integration, customer reviews, and inventory tracking.

**Methodology:**

The development of the backend system follows the Agile methodology, promoting iterative development, continuous testing, and adaptive planning. Key methodological practices include:

* Agile Development Approach  
  Development tasks were divided into smaller sprints, enabling incremental feature delivery, rapid prototyping, and flexibility in response to changing requirements.
* Modular Structure for Reusability and Clarity  
  The system is structured into self-contained modules (user, order, reservation, etc.), improving code readability, testing, and maintainability.
* RESTful API Design Principles  
  All communication between frontend and backend is based on REST principles, using standardized HTTP methods (GET, POST, PUT, DELETE) and clean endpoint structures to ensure scalability and consistent data exchange.

**SYSTEM ANALYSIS:**

**Problem Statement:**

In traditional restaurant environments, managing orders and table reservations manually presents numerous challenges. Restaurant staff often rely on physical registers or basic spreadsheets to keep track of orders, available tables, and customer information. This manual system is highly prone to human error, such as double bookings, lost records, and incorrect order details. Additionally, during peak hours, staff can become overwhelmed, resulting in slower service, misplaced orders, and overall customer dissatisfaction.

From the customer’s perspective, the lack of an online interface often leads to inconvenience. Customers may have to call the restaurant to place an order or inquire about table availability, which may not always be efficient. Without real-time information, there is often uncertainty regarding menu options, wait times, and availability. These inefficiencies collectively impact the restaurant’s reputation and operational performance.

In the modern digital era, where online interactions are the norm, the absence of a streamlined digital platform places traditional restaurants at a competitive disadvantage. Therefore, there is a growing need to adopt a digital solution that not only improves customer convenience but also optimizes internal workflow and data management.

**Proposed System:**

To address the inefficiencies of the manual system, a centralized backend solution has been proposed. The *Restaurant Website Backend* system aims to automate key restaurant operations by integrating essential features such as:

* User Authentication: Secure user registration and login system using encrypted credentials, ensuring that each customer has a unique account to manage their orders and reservations.
* Menu Management: Dynamic retrieval of food items from the database, allowing real-time updates and easy modifications by restaurant administrators.
* Order Placement: A structured process for selecting items, managing the cart, and placing food orders directly through the website interface.
* Reservation System: Table booking functionality that allows customers to reserve seats for specific dates and time slots, reducing the chance of double bookings and confusion.
* Centralized Data Handling: Efficient backend APIs that manage all customer interactions and store data in a secure MongoDB database for easy access and analytics.

The proposed system not only automates restaurant operations but also enhances user experience by offering a responsive, real-time, and interactive platform. This improves customer satisfaction, operational transparency, and business scalability.

**Feasibility Study:**

Before implementing any software system, it is essential to evaluate its feasibility from different perspectives—technical, operational, and economic. This ensures that the system is not only viable but also practical to implement and beneficial in the long run.

**Technical Feasibility:**

The backend system is developed using the MEAN stack—specifically focusing on Node.js, Express.js, and MongoDB for backend operations, while Angular is assumed for the frontend interface. These technologies are widely used and supported, making them ideal for building modern web applications.

* Node.js offers asynchronous, non-blocking I/O operations, making it suitable for handling multiple user requests simultaneously without performance degradation.
* Express.js simplifies backend routing and middleware integration, allowing clean and maintainable API creation.
* MongoDB, a NoSQL database, provides schema flexibility and high scalability. It is well-suited for dynamic data models such as menu items, user orders, and reservations.

The project uses RESTful APIs to communicate between the client-side application and the backend, adhering to modern web standards. These APIs can easily be consumed by any frontend, mobile app, or even third-party services in the future.

All development tools and platforms used are open-source and cross-platform, ensuring high availability and easy adoption. As such, the system is technically feasible with low infrastructure constraints.

**Operational Feasibility:**

From an operational standpoint, the proposed backend system significantly enhances the restaurant's daily functioning by automating time-consuming tasks. The transition from a manual to a digital system minimizes staff involvement in routine administrative tasks and reduces the risk of errors.

* Orders can be accurately recorded, tracked, and processed in real time.
* Reservations are managed systematically, with instant confirmations and conflict avoidance.
* The backend dashboard (in future scope) will offer administrative tools to update menu items, track sales, and manage customer data.

This results in a smoother, faster, and more efficient workflow. Employees can focus more on service quality, while customers benefit from a more convenient and modern dining experience.

Moreover, the system requires minimal training, thanks to its modular and intuitive design. As the backend is built to support real-time interaction with the frontend, any updates or changes made by the staff are instantly reflected on the user interface.

**Economic Feasibility:**

One of the major advantages of the proposed backend system is its cost-effectiveness. Since the system is built using open-source technologies, there are no licensing costs involved. Hosting and deployment can be managed on affordable cloud platforms, making the overall setup economical even for small to medium-sized restaurants.

The long-term financial benefits are also notable:

* Reduced administrative overhead: Automating ordering and reservations reduces the need for dedicated staff handling these functions manually.
* Increased order accuracy and customer satisfaction: Improved service can lead to higher customer retention and positive word-of-mouth.
* Scalability for growth: The backend is designed to scale, meaning additional features like online payments, loyalty programs, or delivery integration can be added without rewriting the core system.

By investing in this backend solution, restaurants can increase their return on investmentthrough operational savings and enhanced customer engagement.

**SYSTEM DESIGN:**

* The system design phase outlines how the backend architecture and modules are structured to meet the requirements identified during analysis. It translates functional needs into technical specifications, offering a clear blueprint for developers and stakeholders. This chapter covers the architectural pattern, module breakdown, non-functional requirements, and the overall database schema.

**System Architecture:**

The architecture of the Restaurant Website Backend is designed around the Model-View-Controller (MVC) design pattern. Although the frontend is assumed to be handled separately (with Angular or another framework), the MVC pattern is applied primarily within the backend to organize code into logical components and ensure a clean separation of concerns.

**MVC Pattern:**

* **Model**: Represents the data layer and defines schema structures for entities such as users, orders, and reservations using Mongoose (MongoDB ODM). It handles database operations, queries, and data validation.
* **View:** While this component is primarily managed by the frontend in an Angular environment, the backend may serve template files or JSON responses (if used in a full-stack JS environment).
* **Controller:** Manages the application logic, processes requests from the frontend, interacts with models, and returns appropriate responses via API endpoints.

**Key Components:**

* **MongoDB (Database Layer):**MongoDB is used for storing structured yet flexible data. Collections represent entities such as users, orders, and reservations, allowing for fast reads/writes and scalability across clusters.
* **Node.js and Express.js (Server Layer):**Node.js provides the runtime environment for executing JavaScript code on the server.Express.js is used to handle routing, middleware integration, and HTTP request management efficiently.
* **REST API Endpoints:**The backend exposes RESTful APIs, ensuring stateless communication between frontend and backend. Endpoints follow naming conventions and HTTP methods like GET, POST, PUT, and DELETE to manipulate data consistently.
* System Flow Diagram (optional for your document).
* User sends a request (e.g., login or order placement) via frontend.
* The request hits the appropriate REST API endpoint in Express.
* The controller processes the request, accesses the model (MongoDB) if needed, and returns a response.
* The frontend updates the UI based on the received JSON data.

**Functional Modules:**

The system is broken into modular components, each handling a distinct area of functionality. This modularization improves maintainability, reusability, and clarity in the development process.

**1. Authentication Module:**

* Handles user registration and login.
* Uses bcrypt to hash passwords for secure storage.
* Performs input validation to ensure only legitimate data is stored.
* On successful login, generates a session or token (JWT in future upgrades).
* Endpoints:
* POST /api/register
* POST /api/login

**2. Menu Module:**

* Retrieves food items stored in the MongoDB collection.
* Allows admin (in future scope) to add, update, or remove items.
* Returns menu data dynamically to the frontend.
* Endpoints:
* GET /api/menu
* (Future scope: POST, PUT, DELETE endpoints for admin)

**3. Order Module**

* Manages customer orders submitted from the frontend.
* Stores order details like customer name, contact info, selected items, and total cost.
* Calculates and stores order totals.
* In future versions, can be extended to handle online payments.
* Endpoints:
* POST /api/orders
* GET /api/orders *(for admin or user order history)*

**4. Reservation Module**

* Enables users to book tables for a specific date and time.
* Validates availability and avoids double bookings.
* Stores any special requests made by customers.
* Endpoints:
* POST /api/reservations
* GET /api/reservations *(for admin)*
* Each of these modules is designed to work independently but can communicate through shared middleware, models, or helper functions for tasks like error handling and logging.

**Non-functional Requirements:**

* While functional requirements define *what* the system should do, non-functional requirements ensure *how well* the system performs, scales, and maintains its quality under different conditions.
* Security
* **Password Hashing:** User passwords are hashed using bcrypt before storage, preventing unauthorized access even if database contents are exposed.
* **Input Validation:** All incoming data (especially from forms) is sanitized to prevent XSS, SQL injection, or NoSQL injection attacks.
* **HTTPS (optional in deployment):** Ensures data is encrypted during transmission to prevent interception.
* Performance
* The backend is optimized to respond to typical API requests within 500 milliseconds under normal load conditions.
* Efficient use of asynchronous functions ensures non-blocking I/O operations, allowing multiple requests to be handled concurrently.
* MongoDB indexing can be applied (future scope) to optimize query performance for large datasets.
* Scalability
* The system is designed with horizontal scaling in mind. MongoDB can be deployed in sharded clusters, and Node.js instances can be load-balanced using tools like PM2 or NGINX.
* Modular architecture allows easy expansion—new features like online payments, user roles, or feedback systems can be added without impacting core functionality.
* Maintainability
* Clean code practices, modularization, and consistent naming conventions make the codebase easy to maintain and debug.
* Use of middleware in Express allows for reusable logic across multiple routes (e.g., logging, authentication checks, error handling).

**Database Schema**

The backend uses MongoDB for data storage, employing flexible JSON-like documents in collections. The key schemas used in the system are defined below.

**1. Users Collection:**

* Stores registered users with basic credentials.
* **code**

{

name: String,

email: String (unique),

password: String (hashed)

}

* The password is stored in a hashed format using bcrypt.
* Email addresses are required to be unique to prevent duplicate accounts.

**2. Orders Collection:**

* Captures order details submitted by customers.
* **code**

{

name: String,

email: String,

phone: String,

items: [

{

itemName: String,

quantity: Number,

price: Number

}

],

total: Number,

createdAt: { type: Date, default: Date.now }

}

* Orders contain an array of items with quantity and pricing.
* createdAt helps in filtering and generating order history or reports**.**

**3. Reservations Collection:**

* Stores reservation details made through the website.
* **code**

{

name: String,

email: String,

phone: String,

tableNumber: Number,

date: String,

time: String,

request: String

}

* Each reservation is associated with a date and time slot.
* tableNumber can be assigned dynamically or manually depending on availability.

**SYSTEM IMPLEMENTATION:**

System implementation involves the actual development and setup of the backend components that power the restaurant website. This stage translates the design architecture into a working system by writing server-side code, connecting to the database, and integrating supporting libraries. This chapter discusses the tools used, setup procedures, project structure, and key backend logic.

**Tools and Technologies:**

To build a secure, scalable, and modern restaurant backend, a set of proven tools and technologies were utilized. These tools not only facilitate rapid development but also ensure reliability and maintainability in the long run.

**Node.js:**

Node.js is a JavaScript runtime environment that allows running JavaScript on the server side. It’s event-driven and non-blocking, making it suitable for handling concurrent connections with high performance.

* **Purpose:** Acts as the core backend runtime.
* **Benefits:** Lightweight, scalable, and ideal for building real-time APIs.

**Express.js:**

Express.js is a minimalist web framework built on top of Node.js. It simplifies the process of creating robust and modular server-side applications.

* **Purpose:** Manages routes, middleware, and API endpoints.
* **Benefits:** Simplifies request handling, improves readability, and supports middleware chaining.

**MongoDB:**

MongoDB is a NoSQL database that stores data in JSON-like documents, offering flexibility and performance for dynamic applications.

* **Purpose:** Stores user data, orders, menu items, and reservations.
* **Benefits:** Schema-less design supports rapid development and scalability.

**Mongoose:**

Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It provides a structured way to define schemas and interact with MongoDB collections.

* **Purpose:** Manages schemas and validation.
* **Benefits:** Adds structure to data models and simplifies CRUD operations.

**Nodemailer:**

Nodemailer allows sending emails directly from the Node.js application.

* **Purpose:** Sends confirmation emails for reservations or order placements.
* **Benefits:** Improves communication with users and adds professionalism to the service.

**Installation:**

Setting up the backend environment is a crucial step to ensure the project runs as intended. Below are the necessary steps to get started:

**Step 1:**

Install Node.js and MongoDB

* Download and install Node.js from <https://nodejs.org>
* Install MongoDB Community Edition <https://www.mongodb.com/try/download/community>

**Step 2:**

Clone the Project and Install Dependencies

Navigate to our project directory and install dependencies:

bash

npm install

This command will install packages listed in the package.json file, such as Express, Mongoose, and nodemailer.

**Step 3:**

Start the Server

To launch the backend server:

bash

node server.js

If using nodemon for development:

bash

Copy code

nodemon server.js

The server will start on the configured port (e.g., http://localhost:3000), ready to handle API requests.

**Directory Structure:**

A clean and organized directory structure improves maintainability and scalability. Below is the standard folder hierarchy used in the project:

php

restaurant-auth/

│

├── backend/ # Backend folder (Node.js, Express, MongoDB)

│ │

│ ├── config/ # Configuration files

│ │ └── db.js # MongoDB connection setup

│ │

│ ├── models/ # Mongoose models (schemas)

│ │ ├── User.js # User schema

│ │ ├── Order.js # Order schema

│ │ ├── Reservation.js # Reservation schema (table bookings)

│ │ └── Subscribe.js # Subscription schema (email subscriptions)

│ │

│ ├── routes/ # Express route definitions

│ │ ├── auth.js # Login/Register routes

│ │ ├── order.js # Order-related endpoints

│ │ ├── reservation.js # Reservation-related endpoints

│ │ └── subscription.js # Subscription routes

│ │

│ └── server.js # Main application entry point

│

├── public/ # Static frontend files

│ ├── index.html # Landing page

│ ├── login.html # Login/Register page

│ ├── menu.html # Menu and order interface

│ ├── reservation.html # Table booking page

│ ├── css/ # Stylesheets

│ │ ├── style.css

│ │ ├── login.css

│ │ ├── menu.css

│ │ └── reservation.css

│ └── images/ # UI images

│

├── .env # Environment variables (MONGO\_URI, etc.)

├── package.json # Project metadata and dependencies

└── package-lock.json # Lock file for dependencies

This modular structure follows best practices and separates concerns for each logical area of the application.

**Backend Logic Snippets:**

Below are core code snippets that demonstrate how the system handles critical backend functionality:

**User Registration:**

During registration, user passwords are hashed using bcrypt before saving to the database:

**Code:**

constbcrypt = require('bcryptjs');

const User = require('./models/User');

consthashedPassword = await bcrypt.hash(password, 10);

constnewUser = new User({

name,

email,

password: hashedPassword

});

await newUser.save();

* Ensures passwords are not stored in plaintext.
* The salt rounds (10) provide a balance between security and performance.

**User Login:**

User login verifies credentials by comparing the entered password with the hashed password stored in MongoDB:

**Code:**

const user = await User.findOne({ email });

if (!user) return res.status(400).send('User not found');

constisMatch = await bcrypt.compare(password, user.password);

if (!isMatch) return res.status(401).send('Incorrect password');

* bcrypt.compare() ensures a secure comparison.
* Proper error handling avoids leaking sensitive details.

**Table Reservation with Email Notification:**

Upon successful table reservation, a confirmation email is sent using nodemailer:

**Code:**

constnodemailer = require('nodemailer');

const transporter = nodemailer.createTransport({

service: 'gmail',

auth: {

user: 'yourrestaurant@gmail.com',

pass: 'yourpassword'

}

});

constmailOptions = {

from: 'yourrestaurant@gmail.com',

to: reservation.email,

subject: 'Table Reservation Confirmation',

text: `Dear ${reservation.name}, your reservation for ${reservation.date} at ${reservation.time} has been confirmed.`

};

transporter.sendMail(mailOptions, (error, info) => {

if (error) {

console.error('Email failed:', error);

} else {

console.log('Confirmation email sent:', info.response);

}

});

* Email templates can be customized or replaced with HTML-based designs.
* SMTP credentials should ideally be stored securely or replaced with OAuth2-based authentication for Gmail.

**API SPECIFICATIONS:**

The backend API of the Restaurant Website is built using Node.js and Express.js, with MongoDB as the database. It is designed to handle key server-side operations such as user registration, authentication, order processing, and reservation management. These APIs serve as endpoints that connect the frontend with the backend, enabling dynamic and interactive user experiences.

All the endpoints follow RESTful architecture principles, use the POST method, and accept and return data in JSON format for seamless integration with client-side applications (such as an Angular frontend or mobile app).

**POST /api/register — User Registration:**

This endpoint allows new users to register on the platform by providing their name, email, and password. The backend uses the bcryptjs library to securely hash passwords before storing them in the MongoDB database.

**Backend Functionality:**

* Accepts user details from the request body.
* Validates input data.
* Checks if the email already exists.
* Hashes the password.
* Creates a new user in the users collection.

**POST /api/login — User Login:**

This endpoint verifies user credentials and grants access to registered users.

Backend Functionality:

* Accepts email and password.
* Checks if the user exists in the database.
* Compares the hashed password using bcrypt.
* Returns user information if login is successful.

**POST /api/orders — Place a Food Order:**

This endpoint enables users to place food orders by submitting selected menu items and personal contact information.

Backend Functionality:

* Accepts an array of items, total cost, and user contact details.
* Validates input.
* Saves the order in the orders collection in MongoDB.
* Can be extended to send order confirmation email or SMS.

**POST /api/reserve — Table Reservation:**

This endpoint is used for booking a table at the restaurant. Users provide their personal details, preferred date/time, table number, and any special requests.

**Backend Functionality:**

* Accepts reservation details via the request body.
* Checks if the table is already booked (you may implement availability check).
* Stores the reservation in the reservations collection.
* Sends confirmation email via nodemailer.

**JSON-Based Communication:**

**All endpoints:**

* Require Content-Type: application/json in headers.
* Respond in application/json format.
* Use clear and concise success/error messages for frontend integration.

**Security and Validation:**

* **Password Security:** Passwords are hashed using bcryptjs.
* **Input Validation:** Required fields are checked before processing.
* **Error Handling:** Proper error codes and messages are returned for robustness.

**TESTING :**

Testing is a critical phase in the software development lifecycle to ensure that the application performs as expected under various scenarios. For the Restaurant Website Backend, rigorous testing was conducted to validate functionality, detect bugs, and ensure overall system reliability. The testing strategy included unit testing, integration testing, and manual testing using tools like Postman. These methods helped verify individual components, their interactions, and the correctness of API behavior with actual data.

**Testing Techniques:**

To ensure the robustness and reliability of the backend system, a combination of different testing methods was adopted:

**Unit Testing:**

Unit testing involves testing individual functions or modules in isolation. For the backend system:

* Functions like user registration logic, password hashing, order creation, and reservation input validation were tested independently.
* JavaScript testing frameworks such as Mocha, Jest, or Chai can be integrated (if extended further) to automate unit tests.

**Examples:**

* Verifying if the bcrypt.hash() function correctly encrypts the password.
* Checking if the User.findOne() method retrieves a user by email.

**Integration Testing:**

Integration testing focuses on verifying that different modules of the system work together correctly. In this case:

* All RESTful API endpoints were tested to ensure that data flows correctly from client requests to MongoDB through Express.js logic.
* Endpoints like /api/register, /api/login, /api/orders, and /api/reserve were tested with various input combinations to validate request handling, business logic execution, and data persistence.

**Manual Testing via Postman:**

Manual testing was carried out using Postman, a popular API testing tool:

* Testers manually sent JSON requests to the API endpoints.
* Different edge cases were simulated to check error responses.
* It was particularly useful in the absence of a frontend during early development stages.

**Postman allowed developers to:**

* Test endpoints with valid and invalid payloads.
* Check status codes and JSON responses.
* Debug error messages and refine validation rules.

### **Test Case Scenarios :**

#### **1. Registration Test Case**

* **Functionality**: User Registration
* **Test Input**: Valid user details (e.g., username, email, password)
* **Expected Output**: A success message indicating that the user has been registered successfully.
* **Purpose**: To verify that the system correctly processes valid registration information and provides appropriate confirmation.

#### **2. Login Test Case**

* **Functionality**: User Login
* **Test Input**: Valid username but an invalid password
* **Expected Output**: An error message indicating incorrect login credentials.
* **Purpose**: To ensure the authentication system correctly rejects login attempts with invalid passwords, safeguarding user accounts.

#### **3. Order Test Case**

* **Functionality**: Placing an Order
* **Test Input**: No items in the cart or order form
* **Expected Output**: A validation error prompting the user to add items before proceeding.
* **Purpose**: To test the system’s validation logic for order submission and prevent empty orders from being processed.

#### **4. Reservation Test Case**

* **Functionality**: Making a Reservation
* **Test Input**: Attempt to book a slot that is already reserved
* **Expected Output**: A conflict error message notifying the user that the slot is already booked.
* **Purpose**: To validate the system’s handling of reservation conflicts and ensure it prevents double bookings.

The deployment phase ensures that the full-stack restaurant web application – The Hungry Table – is hosted on a live server, connected to the database, and properly monitored to deliver a seamless user experience. This section outlines the deployment steps, tools used, configuration, and a checklist for successful implementation.

## **Deployment Steps:**

To make The Hungry Table accessible to users, the system must be deployed to a live environment. This involves multiple stages including hosting, database connection, process management, and environment setup.

### **Step 1: Hosting the Code on a Cloud Platform**

The source code of both frontend and backend is hosted on a cloud-based platform such as:

* **Render** – Ideal for Node.js applications with easy CI/CD setup.
* **Heroku** – Supports quick deployment using Git.
* **Vercel/Netlify** – Typically used for frontend, static sites, or serverless functions.

**Procedure**:

* Push the latest code to a GitHub repository.
* Link the repository to the cloud platform.
* Set up build and start commands for backend (npm run build, npm start) and frontend (npm run build).
* Define environment variables (MongoDB URI, PORT, etc.) in the deployment settings.

### **Step 2: Connecting to MongoDB Atlas or Local Instance**

MongoDB Atlas is used for managing the cloud-hosted database securely and efficiently.

**Procedure**:

* Create a MongoDB cluster using Atlas.
* Add user credentials and whitelist server IP addresses.
* Copy the **MongoDB connection URI**.
* Replace the local URI in the backend .env file with the Atlas URI:

pgsql

CopyEdit

MONGO\_URI=mongodb+srv://<username>:<password>@cluster.mongodb.net/hungry-table

* Ensure Mongoose (ODM) is configured to connect using mongoose.connect(process.env.MONGO\_URI).

### **Step 3: Setting Up Environment Variables**

Use environment variables to securely manage configurations. These include:

* PORT: The port on which the backend server will listen.
* MONGO\_URI: The database connection string.
* JWT\_SECRET: Secret key for authentication tokens (if applicable).

These are set in the deployment platform's settings or within a .env file (for local testing).

### **Step 4: Using PM2 for Process Management**

**PM2** is used for production-level process management. It ensures that:

* The server restarts if it crashes.
* Logs are accessible for debugging.
* Load balancing is possible (if scaling is needed).

**Installation**:

bash

CopyEdit

npm install pm2 -g

**Starting the App**:

bash

CopyEdit

pm2 start server.js --name hungry-table

pm2 save

**Monitoring**:

bash

CopyEdit

pm2 logs

pm2 list

### **Step 5: SSL and HTTPS (Optional but Recommended)**

For better security and trust, enable SSL certificates using services like **Let's Encrypt** or through the deployment platform settings.

## **Deployment Checklist**

To ensure that all components of The Hungry Table are functional after deployment, the following checklist is used:

| **Task** | **Status** | **Details** |
| --- | --- | --- |
| Codebase pushed to GitHub | ✅ Completed | Latest stable version committed and pushed to main branch. |
| Cloud hosting setup | ✅ Completed | Connected to GitHub repository and configured with start commands. |
| MongoDB URI connected | ✅ Completed | Atlas URI securely stored in environment variables. |
| Environment variables defined | ✅ Completed | PORT, MONGO\_URI, and JWT\_SECRET added to .env file or dashboard. |
| Application port listening | ✅ Completed | Server listens on defined port (e.g., 5000 or 8080). |
| PM2 process running | ✅ Completed | Application running and monitored using PM2. |
| Test endpoints working | ✅ Completed | Routes such as /login, /register, /order tested post-deployment. |
| Logs monitored | ✅ Completed | PM2 or platform logs checked for errors. |
| Frontend and backend linked | ✅ Completed | API endpoints connected properly with frontend fetch requests. |
| Backup & rollback plan ready | ✅ Completed | Git branches and snapshots created for version rollback if needed. |

# **USER MANUAL:**

This user manual provides step-by-step guidance for navigating and interacting with the features of The Hungry Table, a full-stack restaurant website. The goal is to help users—both new and returning—understand how to register, log in, browse the menu, place orders, and reserve tables using the user-friendly interface.

## **Getting Started: Login and Registration**

The first interaction begins on the login.html page. This page is designed with a toggle system that allows users to switch between **Login** and **Register** forms seamlessly.

### **How to Register (For New Users):**

1. Navigate to the login.html page.
2. Click the **"Register"** toggle to open the registration form.
3. Fill in the required fields:
   * **Username**
   * **Email**
   * **Password**
4. Click **"Register"** to create your account.
5. A success message will appear, and you can proceed to log in.

Note: Passwords are securely hashed using bcrypt and stored in MongoDB to protect user credentials.

### **How to Log In (For Existing Users):**

1. From login.html, ensure the **Login** tab is selected.
2. Enter your registered **email** and **password**.
3. Click **"Login"** to access your account.
4. Upon successful login, you will be redirected to the main interface (usually menu.html).

❗ Incorrect login attempts will display an error message. Make sure your credentials are correct.

## **Browsing the Menu**

Once logged in, users are directed to menu.html, which showcases the restaurant’s dishes categorized into types (e.g., Starters, Main Course, Desserts, Beverages).

### **How to Browse the Menu:**

1. Navigate through food categories using navigation tabs or filter buttons.
2. Each item displays:
   * **Food image**
   * **Name**
   * **Price**
   * **Add to Cart** button
3. Scroll freely to explore all options or use category filters to narrow down choices.

The menu dynamically loads from a database, ensuring real-time updates and availability.

## **8.3 Placing an Order: Cart and Checkout**

Users can select their desired food items and add them to a virtual cart, accessible from the same menu.html page.

### **How to Add Items to Cart:**

1. Click the **"Add to Cart"** button on any food item.
2. The item will be added to the cart, and a small popup/cart icon will display the count.
3. Continue browsing or proceed to checkout.

### **How to Review and Submit an Order:**

1. Click the **Cart icon** to open the cart modal or sidebar.
2. Review all selected items.
3. Adjust quantities or remove items if necessary.
4. Click **"Checkout"** or **"Place Order"**.
5. A confirmation message will appear upon successful submission.

The backend validates orders to ensure no empty submissions and stores them in MongoDB.

## **Table Reservation**

The system allows users to reserve a dining table via reservation.html, simplifying the process of booking in advance.

### **How to Reserve a Table:**

1. Open reservation.html from the navigation menu.
2. Fill in the reservation form:
   * **Name**
   * **Contact Number**
   * **Date and Time**
   * **Number of Guests**
3. Click **"Reserve"** to book your slot.

### **Conflict Handling:**

* If a time slot is already booked, the system will return a **conflict error message**, asking the user to select another slot.
* Successfully reserved slots are stored in the database and cannot be double-booked.

All reservations are stored in MongoDB and fetched on the backend using Express.js for validation.

## **Additional Features and Notes**

* **Persistent Sessions**: User sessions persist until logout or browser close.
* **Validation**: All forms include client-side and backend validation to prevent errors and ensure data integrity.
* **Responsive Design**: The UI adapts to both desktop and mobile screens for seamless access.

## **Troubleshooting Tips**

| **Issue** | **Possible Cause** | **Solution** |
| --- | --- | --- |
| Unable to log in | Incorrect email/password | Use the correct credentials or reset password. |
| Cannot place order | Empty cart | Add items before checking out. |
| Table slot unavailable | Duplicate booking | Choose a different date or time. |
| Site not loading | Network error or server offline | Refresh or check your internet connection. |

# **FUTURE ENHANCEMENTS:**

The current version covers essential features such as user login/registration, menu browsing, ordering, and table reservations, there is significant scope for scaling and enriching the platform. This section outlines proposed future enhancements aligned with modern full-stack development practices.

## **Admin Dashboard**

### **Objective**:

Introduce a secured **admin panel** to manage the restaurant’s backend operations more efficiently.

### **Planned Features**:

* View and manage all food items (CRUD operations)
* Monitor user orders and reservation logs
* Access analytics such as total orders, popular items, and revenue
* Ability to add/remove food items and update menu categories

### **Implementation Plan**:

* Create a protected route (/admin) using Express.js
* Build a separate frontend (admin.html) with charts, tables, and input forms
* Authenticate admins with a special role flag in MongoDB
* Add role-based access control (RBAC) to restrict unauthorized access

This will enable restaurant staff to dynamically update the menu and manage operations without modifying source code.

## **Dynamic Menu CRUD via Backend**

### **Objective**:

Enable admins to manage the food menu dynamically using backend APIs, removing the need to hardcode menu items.

### **Current Limitation**:

* Menu items are likely hardcoded or fetched once and manually updated in the database.

### **Enhancement Details**:

* Create RESTful API endpoints:
  + POST /menu – Add new item
  + PUT /menu/:id – Update item
  + DELETE /menu/:id – Delete item
  + GET /menu – Fetch all items (already implemented)
* Implement admin-only access to these routes
* Integrate the functionality into the admin dashboard for real-time updates

Dynamic menu control allows for quicker promotions, seasonal updates, and inventory management.

## **JWT Authentication**

### **Objective**:

Enhance session security and scalability using **JSON Web Tokens (JWT)** for authentication.

### **Why JWT?**

* Currently, sessions may rely on simple form data or cookies without token-based validation.
* JWT provides stateless, secure, and scalable user authentication across frontend and backend.

### **Enhancement Plan**:

* On successful login, generate a JWT token containing the user ID and role.
* Store token in localStorage or as an HTTP-only cookie.
* Attach token in headers for all protected routes (Authorization: Bearer <token>).
* Backend middleware will verify and decode the token for route access.

JWT improves security by ensuring only verified users can access or modify protected data, such as orders or reservations.

## **Payment Integration**

### **Objective**:

Allow customers to pay online during order checkout, improving convenience and reducing manual handling.

### **Proposed Integration**:

* Use payment gateways such as **Razorpay**, **Stripe**, or **PayPal**
* Allow payment via:
  + Credit/Debit cards
  + UPI/Bank transfer
  + Wallets (optional)

### **Workflow**:

1. User clicks "Checkout" → Order is confirmed.
2. Redirect to secure payment gateway or embed payment widget.
3. On successful payment:
   * Order is marked as Paid in MongoDB.
   * Confirmation is shown to user.
4. On failure:
   * Order is canceled or flagged.

Integrating online payments adds convenience and can open options for advance booking and prepaid reservations.

## **SMS/Email Order Confirmation**

### **Objective**:

Notify users automatically after placing an order or reserving a table through **email or SMS**.

### **Benefits**:

* Provides reassurance and receipt to the customer
* Reduces customer support queries
* Adds professionalism to the restaurant's online presence

### **Implementation Options**:

* **Email**: Use **Nodemailer** with Gmail or a service like SendGrid
* **SMS**: Integrate with services like **Twilio** or **MSG91**

### **Example Use Cases**:

* "Thank you for your order! Your food will be ready shortly."
* "Your table reservation for 7:30 PM is confirmed. See you soon!"

Enhancing communication builds trust and keeps customers informed, improving their experience.

**CODE:**

Index.html

<!DOCTYPE html>

<html lang="en">

    <head>

        <meta charset="UTF-8">

        <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.2/css/all.min.css" integrity="sha512-SnH5WK+bZxgPHs44uWIX+LLJAJ9/2PkPKZ5QiAj6Ta86w+fsb2TkcmfRyVX3pBnMFcV7oQPJkl9QevSCWr3W6A==" crossorigin="anonymous" referrerpolicy="no-referrer" />

        <link rel="stylesheet" href="../CSS/style.css">

        <meta name="viewport" content="width=device-width, initial-scale=1.0" >

        <link href="https://fonts.googleapis.com/css2?family=Pacifico&display=swap" rel="stylesheet">

        <title> HOME|RESTUARANT </title>

    </head>

    <body>

        <h1>The Hungry Table</h1>

        <nav>

            <div class="nav1">

                <a href="#">Home</a>

                <a href="menu.html">Menu</a>

                <a href="reservation.html">Reservation</a>

            </div>

            <div class="logo">

                <img src="../images/official logo text bl.png" alt="">

            </div>

            <div class="nav2">

                <div class="search">

                    <input type="text" placeholder="Search">

                    <i class="fa-solid fa-magnifying-glass"></i>

                </div>

                <div class="user">

                    <a href="login.html">

                        <i class="fa-solid fa-user"></i>

                        <span>User</span>

                    </a>

                </div>

                <!-- Inside .nav2 (near the user icon) -->

                <button id="logoutBtn" style="display: none;">Logout</button>

            </div>

        </nav>

        <div class="sec1">

            <div class="content">

                <h1>Enjoy Delicious Food</h1>

                <p>Aromatic, Delicious, Flavorful, mouth-watering, Nutritious, Satisfying, Savory, Tasty, Yummy. Appetizing, Delectable, Saccharine.</p>

                <a href="menu.html" class="btn">Order Now <i class="fa-solid fa-arrow-right"></i></a>

             </div>

             <div class="image">

                <img src="../images/img1.jpg" alt="">

             </div>

        </div>

        <div class="sec2">

            <div class="image">

            <img src="../images/img2.jpeg" alt="">

            </div>

        <div class="content">

            <h2>About us</h2>

            <h1>We Provide Healthy Food</h1>

            <p>Aromatic, Delicious, Flavorful, mouth-watering, Nutritious, Satisfying, Savory, Tasty, Yummy. Appetizing, Delectable, Saccharine.</p>

       </div>

     </div>

     <div class="sec3">

        <h1>Popular Dishes</h1>

        <div class="cardbox">

            <div class="card">

                <img src="../images/img13.jpeg" alt="">

                <h4>Breakfast</h4>

                <p>Healthy and Home-Made</p>

                <a href="menu.html" class="btn">Order Now <i class="fa-solid fa-arrow-right"></i></a>

            </div>

            <div class="card">

                <img src="../images/img6.jpeg" alt="">

                <h4>Veg Thali</h4>

                <p>Lunch time!</p>

                <a href="menu.html" class="btn">Order Now <i class="fa-solid fa-arrow-right"></i></a>

            </div>

            <div class="card">

                <img src="../images/img7.jpeg" alt="">

                <h4>Biryani Specials</h4>

                <p>Delicious and Awefull</p>

                <a href="menu.html" class="btn">Order Now <i class="fa-solid fa-arrow-right"></i></a>

            </div>

            <div class="card">

                <img src="../images/img14.jpeg" alt="">

                <h4>Snacks</h4>

                <p>Its! Fun Time</p>

                <a href="menu.html" class="btn">Order Now <i class="fa-solid fa-arrow-right"></i></a>

            </div>

            <div class="card">

                <img src="../images/img15.jpeg" alt="">

                <h4>Refreshers</h4>

                <p>Hot & Refresh</p>

                <a href="menu.html" class="btn">Order Now <i class="fa-solid fa-arrow-right"></i></a>

            </div>

        </div>

     </div>

     <div class="sec4">

        <div class ="content">

            <h2>Newly Added</h2>

            <h1>MilkShakes</h1>

            <p>Cool!!!</p>

            <a href="menu.html" class="btn" >Order Now <i class="fa-solid fa-arrow-right"></i></a>

        </div>

        <div class="image">

            <img src="../images/img11.jpeg" alt="">

        </div>

     </div>

     <div class="sec5">

        <div class="newsletter">

            <div class="image">

                <img src="../images/img12.jpeg" alt="">

            </div>

            <div class="content">

                <h2>newsletter</h2>

                <h1>Subscribe for offer updates</h1>

                <div class="subscribe">

                    <input type="email" placeholder="Enter Email Address">

                    <button type="submit">Subscribe</button>

                </div>

            </div>

        </div>

     </div>

     <div class="sec6">

        <div>

            <img src="../images/official logo text bl.png" alt="">

            <p>RESTUARANT</p>

        </div>

        <div>

            <ul>

                <li>Main Menu</li>

                <li><a href="#">About</a></li>

                <li><a href="#">Menu</a></li>

                <li><a href="#">Events</a></li>

                <li><a href="#">Offers</a></li>

            </ul>

        </div>

        <div>

            <ul>

                <li>Information</li>

                <li><a href="#">Contact</a></li>

                <li><a href="#">Order & Returns</a></li>

                <li><a href="#">Vidoes</a></li>

                <li><a href="#">Reservation</a></li>

            </ul>

        </div>

        <div>

            <ul>

                <li>Address</li>

                <p>Guntur, Andhra Pradesh, India</p>

                <p>9:00AM-11:00PM</p>

            </ul>

            </div>

     </div>

     <div class="footer">

        <p>Copyright &copy;2024 RESTUARANT | All Rights are Reserved</p>

     </div>

     <script>

        document.addEventListener('DOMContentLoaded', () => {

          const subscribeBtn = document.querySelector('.subscribe button');

          const emailInput = document.querySelector('.subscribe input');

          subscribeBtn.addEventListener('click', async (e) => {

            e.preventDefault();

            const email = emailInput.value.trim();

            if (!email) {

              alert('Please enter your email');

              return;

            }

            try {

              const res = await fetch('/api/subscribe', {

                method: 'POST',

                headers: {

                  'Content-Type': 'application/json'

                },

                body: JSON.stringify({ email })

              });

              const data = await res.json();

              alert(data.message);

              if (res.ok) {

                emailInput.value = ''; // Clear input on success

              }

            } catch (err) {

              console.error('❌ Subscription error:', err);

              alert('Something went wrong. Please try again later.');

            }

          });

        });

        const isLoggedIn = localStorage.getItem("userLoggedIn");

        const logoutBtn = document.getElementById("logoutBtn");

if (!isLoggedIn || isLoggedIn !== "true") {

  alert("Please login to access this page.");

  window.location.href = "login.html";

}

else if (logoutBtn) {

    logoutBtn.style.display = "inline-block";

    logoutBtn.addEventListener("click", () => {

      localStorage.clear();

      window.location.href = "login.html";

    });

  }

</script>

</body>

</html>

Login.html

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8" />

  <meta name="viewport" content="width=device-width, initial-scale=1.0" />

  <title>Login & Register | RESTAURANT</title>

  <link rel="stylesheet" href="../CSS/style.css" />

  <link rel="stylesheet" href="../CSS/login.css" />

  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.2/css/all.min.css" crossorigin="anonymous" referrerpolicy="no-referrer" />

  <link href="https://fonts.googleapis.com/css2?family=Pacifico&display=swap" rel="stylesheet">

  <style>

    .form-toggle-buttons {

      text-align: center;

      margin-bottom: 20px;

    }

    .form-toggle-buttons button {

      margin: 0 10px;

      padding: 10px 20px;

      border: none;

      background-color: navy;

      color: white;

      cursor: pointer;

      border-radius: 5px;

    }

    .form-toggle-buttons button.active {

      background-color: #1e90ff;

    }

    .form-wrapper {

      display: none;

    }

    .form-wrapper.active {

      display: block;

    }

  </style>

</head>

<body>

  <h1>The Hungry Table</h1>

  <nav>

    <div class="nav1">

      <a href="index.html">Home</a>

      <a href="menu.html">Menu</a>

      <a href="reservation.html">Reservation</a>

    </div>

    <div class="nav2">

      <div class="search">

        <input type="text" placeholder="Search" />

        <i class="fa-solid fa-magnifying-glass"></i>

      </div>

      <div class="user">

        <a href="#">

          <i class="fa-solid fa-user"></i>

          <span>User</span>

        </a>

      </div>

    </div>

  </nav>

  <div class="login-container">

    <div class="login-image">

      <img src="../images/img25.png" alt="Login Image" />

    </div>

    <div class="login">

      <div class="form-toggle-buttons">

        <button id="loginBtn" class="active" onclick="showForm('login')">Login</button>

        <button id="registerBtn" onclick="showForm('register')">Register</button>

      </div>

      <!-- Login Form -->

      <div id="loginForm" class="form-wrapper active">

        <h2>Login</h2>

        <form id="login-form">

          <label for="email">Email</label>

          <input type="email" id="login-email" placeholder="abcdeg@gmail.com" required />

          <label for="password">Password</label>

          <input type="password" id="login-password" placeholder="Abcd@12345" required />

          <p class="forgot">Forgot password?</p>

          <button type="submit" class="btn">Login</button>

          <p class="signup">Don't have an account? <a href="#" onclick="showForm('register')">Register for Free</a></p>

        </form>

      </div>

      <!-- Register Form -->

      <div id="registerForm" class="form-wrapper">

        <h2>Register</h2>

        <form id="register-form">

          <label for="name">Full Name</label>

          <input type="text" id="register-name" placeholder="Enter your full name" required />

          <label for="email">Email</label>

          <input type="email" id="register-email" placeholder="Enter your email" required />

          <label for="password">Password</label>

          <input type="password" id="register-password" placeholder="Create a strong password" required />

          <label for="confirm-password">Confirm Password</label>

          <input type="password" id="register-confirm-password" placeholder="Re-enter your password" required />

          <button type="submit" class="btn">Register</button>

          <p class="signup">Already have an account? <a href="#" onclick="showForm('login')">Login here</a></p>

        </form>

      </div>

    </div>

  </div>

  <script>

    function showForm(form) {

      document.getElementById("loginForm").classList.remove("active");

      document.getElementById("registerForm").classList.remove("active");

      document.getElementById("loginBtn").classList.remove("active");

      document.getElementById("registerBtn").classList.remove("active");

      if (form === "login") {

        document.getElementById("loginForm").classList.add("active");

        document.getElementById("loginBtn").classList.add("active");

      } else {

        document.getElementById("registerForm").classList.add("active");

        document.getElementById("registerBtn").classList.add("active");

      }

    }

    document.getElementById("register-form").addEventListener("submit", async (event) => {

      event.preventDefault();

      const name = document.getElementById("register-name").value;

      const email = document.getElementById("register-email").value;

      const password = document.getElementById("register-password").value;

      const confirmPassword = document.getElementById("register-confirm-password").value;

      if (password !== confirmPassword) {

        alert("Passwords do not match");

        return;

      }

      const response = await fetch("http://localhost:5000/api/register", {

        method: "POST",

        headers: { "Content-Type": "application/json" },

        body: JSON.stringify({ name, email, password })

      });

      const data = await response.json();

      alert(data.message);

    });

    document.getElementById("login-form").addEventListener("submit", async (event) => {

      event.preventDefault();

      const email = document.getElementById("login-email").value;

      const password = document.getElementById("login-password").value;

      const response = await fetch("http://localhost:5000/api/login", {

        method: "POST",

        headers: { "Content-Type": "application/json" },

        body: JSON.stringify({ email, password })

      });

      const data = await response.json();

      if (response.ok) {

  // ✅ Save login status in localStorage

  localStorage.setItem("userLoggedIn", "true");

  localStorage.setItem("userEmail", email); // Optional: To send email with orders

  alert("Login Successful!");

  window.location.href = "index.html"; // or homepage

}

    });

  </script>

</body>

</html>

Menu.html

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.2/css/all.min.css" integrity="sha512-SnH5WK+bZxgPHs44uWIX+LLJAJ9/2PkPKZ5QiAj6Ta86w+fsb2TkcmfRyVX3pBnMFcV7oQPJkl9QevSCWr3W6A==" crossorigin="anonymous" referrerpolicy="no-referrer" />

    <link rel="stylesheet" href="css/style.css">

    <link rel="stylesheet" href="css/menu.css">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Menu | RESTAURANT</title>

</head>

<body>

    <!-- Navbar -->

    <nav>

        <div class="nav1">

            <a href="index.html">Home</a>

            <a href="menu.html">Menu</a>

            <a href="reservation.html">Reservation</a>

        </div>

        <div class="nav2">

            <div class="search">

                <input type="text" placeholder="Search">

                <i class="fa-solid fa-magnifying-glass"></i>

            </div>

            <div class="user">

                <a href="login.html">

                    <i class="fa-solid fa-user"></i>

                    <span>User</span>

                </a>

            </div>

        </div>

    </nav>

    <!-- Menu Section -->

    <div class="menu">

        <h1>Our Delicious Food</h1>

        <div class="menu-box">

            <!-- Example Food Item -->

            <div class="menu-card" data-name="Idly" data-price="50">

                <img src="../images/img4.jpg" alt="">

                <h4>Idly</h4>

                <p>Plate of 4 Idly Serves-1</p>

                <span>50 Rs</span>

                <button class="add-to-cart">Add to Cart</button>

            </div>

            <div class="menu-card" data-name="Dosa" data-price="50">

                <img src="../images/img3.jpg" alt="">

                <h4>Dosa</h4>

                <p>Plate of 1 Dosa Serves-1</p>

                <span>50 Rs</span>

                <button class="add-to-cart">Add to Cart</button>

            </div>

            <div class="menu-card" data-name="Puri" data-price="75">

                <img src="../images/img5.jpg" alt="">

                <h4>Puri</h4>

                <p>Plate of 2 Puri Serves-1</p>

                <span>75 Rs</span>

                <button class="add-to-cart">Add to Cart</button>

            </div>

            <div class="menu-card" data-name="Veg Thali" data-price="150">

                <img src="../images/img6.jpeg" alt="">

                <h4>Veg Thali</h4>

                <p>Rice, Dal, Fry, Curry, Curd, Sweet</p>

                <span>150 Rs</span>

                <button class="add-to-cart">Add to Cart</button>

            </div>

            <div class="menu-card" data-name="Veg Biryani" data-price="160">

                <img src="../images/img17.jpeg" alt="">

                <h4>Veg Biryani</h4>

                <p>Serves-1</p>

                <span>160 Rs</span>

                <button class="add-to-cart">Add to Cart</button>

            </div>

            <div class="menu-card" data-name="Fry Piece Biryani" data-price="280">

                <img src="../images/img18.jpeg" alt="">

                <h4>Fry Piece Biryani</h4>

                <p>Serves-1</p>

                <span>280 Rs</span>

                <button class="add-to-cart">Add to Cart</button>

            </div>

            <div class="menu-card" data-name="Chicken Biryani" data-price="250">

                <img src="../images/img7.jpeg" alt="">

                <h4>Chicken Biryani</h4>

                <p>Serves-1</p>

                <span>250 Rs</span>

                <button class="add-to-cart">Add to Cart</button>

            </div>

            <div class="menu-card" data-name="Mutton Biryani" data-price="350">

                <img src="../images/img23.jpeg" alt="">

                <h4>Mutton Biryani</h4>

                <p>Serves-2</p>

                <span>350 Rs</span>

                <button class="add-to-cart">Add to Cart</button>

            </div>

            <div class="menu-card" data-name="Noodles" data-price="80">

                <img src="../images/img8.jpeg" alt="">

                <h4>Noodles</h4>

                <p>Serves-1</p>

                <span>80 Rs</span>

                <button class="add-to-cart">Add to Cart</button>

            </div>

            <div class="menu-card" data-name="Gobi Manchurian" data-price="80">

                <img src="../images/img16.jpeg" alt="">

                <h4>Gobi Manchurian</h4>

                <p>Serves-1</p>

                <span>80 Rs</span>

                <button class="add-to-cart">Add to Cart</button>

            </div>

            <div class="menu-card" data-name="Coffee" data-price="40">

                <img src="../images/img9.jpeg" alt="">

                <h4>Coffee</h4>

                <p>Serves-1</p>

                <span>40 Rs</span>

                <button class="add-to-cart">Add to Cart</button>

            </div>

            <div class="menu-card" data-name="Special Tea" data-price="45">

                <img src="../images/img20.jpeg" alt="">

                <h4>Special Tea</h4>

                <p>Serves-1</p>

                <span>45 Rs</span>

                <button class="add-to-cart">Add to Cart</button>

            </div>

            <div class="menu-card" data-name="Hot Badam Milk" data-price="65">

                <img src="../images/img21.jpeg" alt="">

                <h4>Hot Badam Milk</h4>

                <p>Serves-1</p>

                <span>65 Rs</span>

                <button class="add-to-cart">Add to Cart</button>

            </div>

            <div class="menu-card" data-name="Special Rabdi" data-price="110">

                <img src="../images/img22.jpeg" alt="">

                <h4>Special Rabdi</h4>

                <p>Serves-1</p>

                <span>110 Rs</span>

                <button class="add-to-cart">Add to Cart</button>

            </div>

            <div class="menu-card" data-name="Pizza" data-price="280">

                <img src="../images/img19.jpg" alt="">

                <h4>Pizza</h4>

                <p>Serves-4</p>

                <span>280 Rs</span>

                <button class="add-to-cart">Add to Cart</button>

            </div>

        </div>

    </div>

    <div id="cart-sidebar">

        <h2>Your Cart</h2>

        <ul id="cart-items"></ul>

        <p><strong>Total:</strong> ₹<span id="cart-total">0</span></p>

        <button id="checkout-button">Proceed to Checkout</button>

    </div>

    <!-- 🧾 Checkout Modal -->

    <div id="checkout-modal" class="modal">

        <div class="modal-content">

            <span class="close">&times;</span>

            <h2>Place Your Order</h2>

            <form id="order-form">

                <input type="text" id="name" placeholder="Your Name" required>

                <input type="email" id="email" placeholder="Your Email" required>

                <input type="tel" id="phone" placeholder="Your Phone Number" required>

                <button type="submit">Submit Order</button>

            </form>

        </div>

    </div>

    <!-- Footer Section -->

    <div class="sec6">

        <div>

            <img src="../images/official\_logo\_text\_bl.png" alt="">

            <p>RESTAURANT</p>

        </div>

        <div>

            <ul>

                <li>Main Menu</li>

                <li><a href="#">About</a></li>

                <li><a href="#">Menu</a></li>

                <li><a href="#">Events</a></li>

                <li><a href="#">Offers</a></li>

            </ul>

        </div>

        <div>

            <ul>

                <li>Information</li>

                <li><a href="#">Contact</a></li>

                <li><a href="#">Order & Returns</a></li>

                <li><a href="#">Videos</a></li>

                <li><a href="#">Reservation</a></li>

            </ul>

        </div>

        <div>

            <ul>

                <li>Address</li>

                <p>Guntur, Andhra Pradesh, India</p>

                <p>9:00 AM - 11:00 PM</p>

            </ul>

        </div>

    </div>

    <div class="footer">

        <p>Copyright &copy;2024 RESTAURANT | All Rights Reserved</p>

    </div>

    <script>

        document.addEventListener("DOMContentLoaded", () => {

            const cart = [];

            const cartItemsEl = document.getElementById("cart-items");

            const cartTotalEl = document.getElementById("cart-total");

            const modal = document.getElementById("checkout-modal");

            const closeModalBtn = document.querySelector(".close");

            const checkoutBtn = document.getElementById("checkout-button");

            const orderForm = document.getElementById("order-form");

            // Add items to cart

            document.querySelectorAll(".add-to-cart").forEach(button => {

                button.addEventListener("click", () => {

                    const card = button.closest(".menu-card");

                    const name = card.dataset.name;

                    const price = parseInt(card.dataset.price);

                    const existingItem = cart.find(item => item.name === name);

                    if (existingItem) {

                        existingItem.quantity += 1;

                    } else {

                        cart.push({ name, price, quantity: 1 });

                    }

                    updateCartUI();

                });

            });

            // Update cart UI

            function updateCartUI() {

                cartItemsEl.innerHTML = "";

                let total = 0;

                cart.forEach(item => {

                    const itemTotal = item.price \* item.quantity;

                    total += itemTotal;

                    const li = document.createElement("li");

                    li.textContent = `${item.name} x${item.quantity} - ₹${itemTotal}`;

                    cartItemsEl.appendChild(li);

                });

                cartTotalEl.textContent = total;

            }

            // Show modal

            checkoutBtn.addEventListener("click", () => {

                if (cart.length === 0) {

                    alert("Cart is empty!");

                    return;

                }

                modal.style.display = "block";

            });

            // Close modal

            closeModalBtn.addEventListener("click", () => {

                modal.style.display = "none";

            });

            window.addEventListener("click", (e) => {

                if (e.target === modal) {

                    modal.style.display = "none";

                }

            });

            // Submit order

            orderForm.addEventListener("submit", async (e) => {

                e.preventDefault();

                const name = document.getElementById("name").value.trim();

                const email = document.getElementById("email").value.trim();

                const phone = document.getElementById("phone").value.trim();

                const total = cart.reduce((sum, item) => sum + item.price \* item.quantity, 0);

                const orderData = {

                    name,

                    email,

                    phone,

                    items: cart,

                    total

                };

                try {

                    const response = await fetch("/api/orders", {

                        method: "POST",

                        headers: { "Content-Type": "application/json" },

                        body: JSON.stringify(orderData)

                    });

                    const result = await response.json();

                    if (response.ok) {

                        alert(result.message || "Order placed successfully!");

                        modal.style.display = "none";

                        cart.length = 0;

                        updateCartUI();

                        orderForm.reset();

                    } else {

                        alert("Error placing order: " + (result.message || "Try again."));

                    }

                } catch (err) {

                    console.error("Order submission error:", err);

                    alert("Failed to place order. Server error.");

                }

            });

        });

        const isLoggedIn = localStorage.getItem("userLoggedIn");

if (!isLoggedIn || isLoggedIn !== "true") {

  alert("Please login to access this page.");

  window.location.href = "login.html";

}

        </script>

     </body>

</html>

Reservation.html

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8" />

  <meta name="viewport" content="width=device-width, initial-scale=1.0" />

  <title>Reservation | RESTAURANT</title>

  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.2/css/all.min.css" integrity="sha512-SnH5WK+bZxgPHs44uWIX+LLJAJ9/2PkPKZ5QiAj6Ta86w+fsb2TkcmfRyVX3pBnMFcV7oQPJkl9QevSCWr3W6A==" crossorigin="anonymous" referrerpolicy="no-referrer" />

  <link rel="stylesheet" href="../CSS/style.css" />

  <link rel="stylesheet" href="../CSS/reservation.css" />

  <link href="https://fonts.googleapis.com/css2?family=Pacifico&display=swap" rel="stylesheet">

</head>

<body>

<h1>The Hungry Table</h1>

  <nav>

    <div class="nav1">

      <a href="index.html">Home</a>

      <a href="menu.html">Menu</a>

      <a href="reservation.html">Reservation</a>

    </div>

    <div class="nav2">

      <div class="search">

        <input type="text" placeholder="Search" />

        <i class="fa-solid fa-magnifying-glass"></i>

      </div>

      <div class="user">

        <a href="login.html">

          <i class="fa-solid fa-user"></i>

          <span>User</span>

        </a>

      </div>

    </div>

  </nav>

  <div class="reservation-container">

    <div class="reservation">

      <form id="reservation-form">

        <h2>Your Reservation</h2>

        <div>

          <input type="text" id="name" placeholder="Your Name" required />

          <input type="email" id="email" placeholder="Your Email" required />

        </div>

        <div>

          <input type="text" id="phone" placeholder="Phone Number" required />

          <input type="number" id="tableNumber" placeholder="Table Number" required />

        </div>

        <div>

          <input type="date" id="date" required />

          <input type="time" id="time" required />

        </div>

        <div>

          <textarea id="request" placeholder="Your Request (Optional)" cols="30" rows="7"></textarea>

        </div>

        <div>

          <button type="submit" class="btn">BOOK NOW</button>

        </div>

      </form>

    </div>

  </div>

  <div class="sec6">

    <div>

      <img src="../images/official logo text bl.png" alt="" />

      <p>RESTAURANT</p>

    </div>

    <div>

      <ul>

        <li>Main Menu</li>

        <li><a href="#">About</a></li>

        <li><a href="#">Menu</a></li>

        <li><a href="#">Events</a></li>

        <li><a href="#">Offers</a></li>

      </ul>

    </div>

    <div>

      <ul>

        <li>Information</li>

        <li><a href="#">Contact</a></li>

        <li><a href="#">Order & Returns</a></li>

        <li><a href="#">Videos</a></li>

        <li><a href="#">Reservation</a></li>

      </ul>

    </div>

    <div>

      <ul>

        <li>Address</li>

        <p>Guntur, Andhra Pradesh, India</p>

        <p>9:00AM - 11:00PM</p>

      </ul>

    </div>

  </div>

  <div class="footer">

    <p>&copy; 2024 RESTAURANT | All Rights Reserved</p>

  </div>

  <!-- Reservation Form Script -->

  <script>

    document.getElementById("reservation-form").addEventListener("submit", async function(e) {

      e.preventDefault();

      const name = document.getElementById("name").value;

      const email = document.getElementById("email").value;

      const phone = document.getElementById("phone").value;

      const tableNumber = document.getElementById("tableNumber").value;

      const date = document.getElementById("date").value;

      const time = document.getElementById("time").value;

      const request = document.getElementById("request").value;

      try {

        const response = await fetch("http://localhost:5000/api/reserve", {

          method: "POST",

          headers: { "Content-Type": "application/json" },

          body: JSON.stringify({ name, email, phone, tableNumber, date, time, request })

        });

        const result = await response.json();

        alert(result.message);

      } catch (error) {

        alert("Error submitting reservation. Please try again.");

        console.error("Reservation error:", error);

      }

    });

    const isLoggedIn = localStorage.getItem("userLoggedIn");

if (!isLoggedIn || isLoggedIn !== "true") {

  alert("Please login to access this page.");

  window.location.href = "login.html";

}

  </script>

</body>

</html>

Style.css

@import url('https://fonts.googleapis.com/css2?family=Cedarville+Cursive&display=swap');

\*{

    margin:0;

    padding:0;

    box-sizing: border-box;

    list-style: none;

    text-decoration: none;

    font-family: Verdana, Geneva, Tahoma, sans-serif;

    max-width: 1500px;

    overflow-x: hidden;

}

nav{

    position: relative;

    display: flex;

    align-items: center;

    justify-content: space-around;

    width: 100%;

    height: 10vh;

    background-color: transparent;

}

nav .nav1 a{

    color: black;

    font-size: 13px;

    padding-right: 20px;

    transition: all 0.3s;

}

nav .nav1 a:hover{

    color: rgb(239,122,30);

}

nav .user a{

    color: black;

}

.logo img{

    width: 200px;

}

nav .nav2{

    display: flex;

    align-items: center;

    font-size: 13px;

}

.nav2 .search{

    border-bottom: 1px solid rgb(59, 59, 50);

    padding-bottom: 10px;

    margin-right: 20px;

}

.nav2 .search input{

    width: 150px;

    outline: none;

    background-color: transparent;

    border: none;

}

.nav2 .search input::placeholder{

    color:black;

}

.nav2 .user{

    margin-left: 20px;

    padding-bottom: 10px;

    cursor: pointer;

}

.sec1{

    display: flex;

    align-items: center;

    justify-content: space-between;

}

.sec1 .image img{

    width: 50vw;

    max-width: 700px;

    margin-right: -5vw;

}

.sec1 .content{

    width: 50vw;

    padding: 7%;

}

.sec1 .content h1{

    color: rgb(57,33,55);

    font-size: 55px;

    margin-bottom: 15px;

    font-family: Cambria, Cochin, Georgia, Times, 'Times New Roman', serif;

}

.sec1 .content p{

    color: rgb(109, 97, 110);

    margin-bottom: 30px;

}

.btn{

    background-color: rgb(239,122,30);

    color: white;

    display: inline-flex;

    align-items: center;

    padding: 20px 30px;

    border-radius: 50px;

    transition: all 0.3s;

}

.btn i{

    margin-left: 20px;

    font-size: 20px;

}

.btn:hover{

    background-color: rgba(239, 121, 30,0.712);

    transform: translateX(20px);

}

.sec2{

    display: flex;

    align-items: center;

    justify-content: space-between;

    padding: 0 5%;

}

.sec2 .image img{

    width: 40vw;

    max-width: 600px;

}

.sec2 .content{

    max-width: 40vw;

}

.sec2 h2{

    font-family: cedarville cursive, cursive;

    font-size: 30px;

    color: rgb(241,147,73);

}

.sec2 content h1{

    color: rgb(57,33,55);

    font-size: 45px;

    margin-bottom: 15px;

    font-family: Cambria, Cochin, Georgia, Times, 'Times New Roman', serif;

}

.sec2 .content p{

    color: rgb(109, 97, 110);

    margin-bottom: 30px;

    font-size: 15px;

    line-height: 25px;

}

.sec3{

    padding: 7% 5%;

    text-align: center;

}

.sec3 h1{

    font-family: Cambria, Cochin, Georgia, Times, 'Times New Roman', serif;

    font-size: 45px;

}

.sec3 .cardbox{

    padding: 70px 0;

    display: flex;

    flex-wrap: wrap;

    align-items: center;

    gap: 20px;

    justify-content: space-evenly;

}

.card{

    width: 270px;

    padding: 40px 20px;

    border-radius: 15px;

    box-shadow: 0 0 40px rgba(254,253,209,0.6);

    cursor: pointer;

    transition: all 0.3s;

    text-align: center;

}

.card:hover{

    box-shadow: 0 0 100px rgb(239,122,30);

    transform: scale(1.02);

}

.card img{

    width: 150px;

    height: 150px;

    margin-bottom: 45px;

}

.card h4, .card p, .card span{

    text-align: left;

    margin-bottom: 15px;

    display: block;

}

.card p{

    font-size: 13px;

    display: -webkit-box;

    -webkit-box-orient: vertical;

    -webkit-line-clamp: 2;

    overflow: hidden;

    text-overflow: ellipsis;

}

.card span{

    color: rgb(241,147,73);

    font-weight: 700;

}

.sec4{

    display: flex;

    align-items: center;

    justify-content: space-between;

}

.sec4 .image img{

    width: 40vw;

    max-width: 700px;

    margin-top: -50px;

}

.sec4 .content{

    width: 60vw;

    padding: 7%;

}

.sec4 .content h1{

    color: rgb(57,33,55);

    font-size: 55px;

    margin-bottom: 15px;

    font-family: Cambria, Cochin, Georgia, Times, 'Times New Roman', serif;

}

.sec4 .content p{

    color: rgb(109, 97, 110);

    margin-bottom: 30px;

}

.sec4 h2{

    font-family: cedarville cursive,cursive;

    font-size: 30px;

    color: rgb(241,147,73);

}

.sec5{

    padding: 5% 10%;

    margin: 10% 0;

}

.newsletter{

    background-color: black;

    border-radius: 20px;

    min-height: 200px;

    width: 100%;

    display: flex;

    align-items: center;

    justify-content: space-around;

}

.newsletter img{

    width: 30vw;

    max-width: 450px;

}

.newsletter .content{

    width: 30vw;

}

.newsletter .content h1{

    color: white;

    font-size: 40px;

    margin-bottom: 40px;

    width: 80%;

    font-family: Cambria, Cochin, Georgia, Times, 'Times New Roman', serif;

}

.newsletter h2{

    font-family: cedarville cursive, cursive;

    font-size: 30px;

    color: rgb(241,147,73);

}

.newsletter .subscribe{

    padding: 10px;

    background-color: white;

    border-radius: 40px;

    display: flex;

    align-items: center;

    justify-content: space-between;

}

.newsletter .subscribe input{

    font-size: 15px;

    width: 70%;

    outline: none;

    border: none;

    padding: 10px;

}

.newsletter .subscribe button{

    background-color: rgb(239,122,30);

    border: none;

    color: white;

    border-radius: 20px;

    font-size: 15px;

    padding: 10px 20px;

    cursor: pointer;

}

.sec6{

    padding: 0% 10%;

    display: flex;

    justify-content: space-between;

}

.sec6 div{

    max-width: 200px;

}

.sec6 ul li :first-child{

    font-weight: 700;

}

.sec6 li, sec6 p{

    padding: 10px 0;

}

.sec6 p, .sec6 a{

    color: gray;

    font-size: 14px;

    line-height: 22px;

}

.sec6 a:hover{

    color: rgb(239,122,30);

}

.footer{

    text-align: center;

    padding: 1.5% 0;

    margin-top: 7%;

    font-size: 14px;

    background-color: rgba(239, 121, 30,0.3);

}

#logoutBtn {

    all: unset;

    cursor: pointer;

    font-family: inherit;

    margin-left: 15px;

    margin-top: -8px;

    color: black;

  }

  #logoutBtn:hover {

    text-decoration: underline;

  }

  h1 {

    text-align: center;

    font-size: 45px;

    color: rgb(239,122,30);

    margin-top: 20px;

    font-family:  'Pacifico', cursive;

  }

Login.css

\*{

    padding: 0;

    margin: 0;

    box-sizing: border-box;

text-decoration: none;

font-family: Verdana, Geneva, Tahoma, sans-serif;

}

.login-container{

    min-height: 100vh;

    padding: 0 7%;

    margin-top: 50px;

    display: flex;

    align-items: center;

    justify-content: space-around;

}

.login-image {

    width: 300px; /\* Small fixed width \*/

    height: auto;

}

.login-image img {

    width: 100%;

    height: auto;

    border-radius: 20px;

    box-shadow: 0 0 20px rgba(239, 121, 30, 0.2);

    object-fit: contain;

}

.login{

    display: flex;

    align-items: center;

    flex-direction: column;

    box-shadow: 0 0 150px rgba(239, 121, 30, 0.5);

    border-radius: 20px;

    padding: 40px;

    width: 40%;

}

.form{

    width: 100%;

}

.login h2{

    margin-bottom: 20px;

    font-size: 30px;

    color: rgba(239, 121, 30);

}

.login-container label, .login-container input{

    display: block;

}

.login label{

    font-size: 13px;

    font-weight: 600;

    margin-top: 15px;

    margin-bottom: 5px;

}

.login input{

    padding: 10px;

    width: 100%;

    outline: none;

    border: 1px solid rgba(128 128 128 0.1);

    border-radius: 5px;

    background-color: rgba(128 128 128 0.1);

}

.login .forgot{

    text-align: right;

    color: rgb(82, 82, 82);

    font-size: 12px;

    margin-top: 5px;

    cursor: pointer;

}

.login .btn{

    width: 100%;

    padding: 10px;

    border-radius: 5px;

    margin-top: 20px;

    display: block;

    text-align: center;

}

.login .btn:hover{

    transform: translateX(0);

}

.login .signup{

    color: rgb(82, 82, 82);

    font-size: 13px;

    text-align: center;

    margin-top: 20px;

    cursor: pointer;

}

.login .signup span{

    color: black;

    font-weight: 30px;

    cursor: pointer;

}

Menu.css

body {

    background: linear-gradient(45deg, white 70%, rgb(254, 233, 209));

}

.menu {

    text-align: center;

    align-items: flex-start;

    justify-content: space-between;

    gap: 20px;

    padding: 20px;

    box-sizing: border-box;

}

.menu h1 {

    font-family: Cambria, Cochin, Georgia, Times, 'Times New Roman', serif;

    font-size: 45px;

}

.menu .menu-box {

    padding: 100px 5%;

    display: flex;

    align-items: center;

    justify-content: space-between;

    flex-wrap: wrap;

    width: 70%;

}

.menu-card {

    min-width: 270px;

    width: 270px;

    padding: 40px 20px;

    background-color: white;

    padding-top: 0;

    border-radius: 10px;

    cursor: pointer;

    transition: all 0.3s;

    overflow: visible;

    margin-bottom: 100px;

    box-shadow: 0 0 50px rgb(254, 233, 209);

}

.menu-card:hover img {

    margin-top: 80px;

}

.menu-card img {

    margin-bottom: 40px;

    width: 180px;

    height: 180px;

    transition: all 0.3s;

}

.menu-card h4,

.menu-card p,

.menu-card span {

    text-align: left;

    margin-bottom: 15px;

    display: block;

}

.menu-card p {

    font-size: 13px;

    display: -webkit-box;

    -webkit-box-orient: vertical;

    -webkit-line-clamp: 2;

    overflow: hidden;

    text-overflow: ellipsis;

}

.menu-card span {

    color: rgb(241, 147, 73);

    font-weight: 700;

}

/\* NEW: Add to Cart Button \*/

.add-to-cart {

    background-color: #0a4275;

    color: white;

    border: none;

    padding: 10px 15px;

    border-radius: 8px;

    cursor: pointer;

    font-weight: bold;

    transition: background-color 0.3s ease;

    width: 100%;

    margin-top: 10px;

}

.add-to-cart:hover {

    background-color: #06385f;

}

/\* Cart Sidebar \*/

#cart-sidebar {

    position: fixed;

    display: flex;

    flex-direction: column;

    top: 0;

    right: 0;

    width: 350px;

    height: 100vh;

    background-color:rgb(254, 233, 209);

    box-shadow: -5px 0 15px rgba(221, 205, 205, 0.2);

    z-index: 999;

    padding: 20px;

    overflow-y: auto;

}

#cart-sidebar h2 {

    font-size: 24px;

    margin-bottom: 20px;

    color: #0a4275;

}

.cart-items {

    max-height: 60vh;

    overflow-y: auto;

    margin-bottom: 20px;

}

.cart-item {

    display: flex;

    justify-content: space-between;

    align-items: center;

    margin-bottom: 15px;

    border-bottom: 1px solid #ddd;

    padding-bottom: 10px;

}

.cart-item-name {

    font-weight: bold;

    flex: 1;

}

.cart-item-price {

    color: #0a4275;

    margin-left: 10px;

}

.remove-item {

    color: red;

    cursor: pointer;

    margin-left: 10px;

}

/\* Checkout Button \*/

.checkout-btn {

    background-color: #0a4275;

    color: white;

    width: 100%;

    padding: 12px;

    border: none;

    border-radius: 8px;

    font-size: 16px;

    font-weight: bold;

    cursor: pointer;

    transition: background-color 0.3s ease;

    margin-top: auto;

}

.checkout-btn:hover,

#checkout-button:hover {

    background-color: #06385f;

}

/\* Floating Cart Button \*/

.cart-toggle-btn {

    position: fixed;

    top: 20px;

    right: 20px;

    background-color: #0a4275;

    color: white;

    padding: 12px 16px;

    border-radius: 50%;

    cursor: pointer;

    font-size: 18px;

    z-index: 1000;

    box-shadow: 0 0 15px rgba(0, 0, 0, 0.3);

}

.modal {

    display: none;

    position: fixed;

    top: 0; left: 0;

    width: 100%; height: 100%;

    background-color: rgba(0, 0, 0, 0.5);

    z-index: 999;

}

.modal-content {

    background-color: white;

    margin: 10% auto;

    padding: 20px;

    width: 90%;

    max-width: 400px;

    border-radius: 10px;

}

Reservation.css

\* {

    margin: 0;

    padding: 0;

    box-sizing: border-box;

  }

  .reservation-container {

    display: flex;

    min-height: 100vh;

    background: url('../images/img10.jpg') no-repeat center center/cover;

  }

  .reservation {

    width: 100%;

    height: 100vh;

    background: linear-gradient(to bottom, white -100%, transparent);

    display: flex;

    align-items: center;

    justify-content: center;

    padding-top: 70px;

  }

  .reservation form {

    padding: 50px;

    width: 60%;

    border-radius: 20px;

    backdrop-filter: blur(5px);

    background-color: rgba(255, 255, 255, 0.5);

  }

  .reservation form h2 {

    margin-bottom: 20px;

    font-size: 30px;

    color: rgba(239, 122, 30);

    text-align: center;

  }

  .reservation form div {

    width: 100%;

    display: flex;

    align-items: center;

    justify-content: center;

    flex-wrap: wrap; /\* Responsive layout \*/

  }

  .reservation form input {

    width: 45%;

    padding: 15px;

    margin: 5px;

    outline: none;

    border: none;

    border-radius: 5px;

    font-size: 16px;

  }

  .reservation form textarea {

    resize: none;

    width: 95%;

    padding: 15px;

    margin: 5px;

    outline: none;

    border: none;

    border-radius: 5px;

    font-size: 16px;

  }

  .reservation form .btn {

    display: block;

    border-radius: 10px;

    margin: 20px auto 0;

    width: 250px;

    padding: 15px;

    background-color: rgba(239, 122, 30, 0.9);

    color: white;

    font-size: 16px;

    border: none;

    cursor: pointer;

    transition: all 0.3s ease;

  }

  .reservation form .btn:hover {

    transform: scale(1.05);

    background-color: rgba(239, 122, 30, 1);

  }

  /\* Responsive Improvements \*/

  @media (max-width: 768px) {

    .reservation form {

      width: 90%;

      padding: 30px;

    }

    .reservation form input {

      width: 90%;

    }

    .reservation form textarea {

      width: 90%;

    }

    .reservation form .btn {

      width: 90%;

    }

  }

Db.js

  const mongoose = require("mongoose");

const dotenv = require("dotenv");

// Load environment variables

dotenv.config();

const connectDB = async () => {

  try {

    console.log("🔍 Connecting to MongoDB:", process.env.MONGO\_URI); // Debugging line

    await mongoose.connect(process.env.MONGO\_URI, {

      useNewUrlParser: true,

      useUnifiedTopology: true,

    });

    console.log("✅ MongoDB Connected");

  } catch (error) {

    console.error("❌ MongoDB Connection Failed:", error.message);

    process.exit(1);

  }

};

module.exports = connectDB;

Models/Order.js

const mongoose = require('mongoose');

const orderSchema = new mongoose.Schema({

  name: String,

  email: String,

  phone: String,

  items: [

    {

      name: String,

      price: Number,

      quantity: Number

    }

  ],

  total: Number,

  createdAt: {

    type: Date,

    default: Date.now

  }

});

module.exports = mongoose.model('Order', orderSchema);

Models/Reservation.js

const mongoose = require('mongoose');

const ReservationSchema = new mongoose.Schema({

  name: String,

  email: String,

  phone: String,

  tableNumber: Number,

  date: String,

  time: String,

  request: String

});

module.exports = mongoose.model('Reservation', ReservationSchema);

Models/subscribe.js

const mongoose = require('mongoose');

const subscribeSchema = new mongoose.Schema({

  email: {

    type: String,

    required: true,

    unique: true

  }

});

module.exports = mongoose.model('Subscribe', subscribeSchema);

Models/User.js

const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({

  name: String,

  email: { type: String, required: true, unique: true },

  password: { type: String, required: true }

});

module.exports = mongoose.model('User', userSchema);

routes/auth.js

const express = require("express");

const router = express.Router();

const bcrypt = require("bcryptjs");

const User = require("../models/User");

// Register

router.post("/register", async (req, res) => {

  const { name, email, password } = req.body;

try {

    const existingUser = await User.findOne({ email });

    if (existingUser) return res.status(400).json({ message: "User already exists" });

const hashedPassword = await bcrypt.hash(password, 10);

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

    await newUser.save();

res.status(201).json({ message: "User registered successfully" });

  } catch (err) {

    res.status(500).json({ message: "Server error" });

  }

});

// Login

router.post("/login", async (req, res) => {

  const { email, password } = req.body;

try {

    const user = await User.findOne({ email });

    if (!user) return res.status(400).json({ message: "Invalid credentials" });

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

    if (!isMatch) return res.status(400).json({ message: "Invalid credentials" });

res.status(200).json({ message: "Login successful" });

  } catch (err) {

    res.status(500).json({ message: "Server error" });

  }

});

module.exports = router;

routes/order.js

const express = require('express');

const router = express.Router();

module.exports = (db) => {

  // ✅ Place a new order

  router.post('/', async (req, res) => {

    const { name, email, phone, items, total } = req.body;

    if (!name || !email || !phone || !items || !total) {

      return res.status(400).json({ error: 'Missing order data' });

    }

    const order = {

      name,

      email,

      phone,

      items,

      total,

      createdAt: new Date()

    };

    try {

      const result = await db.collection('orders').insertOne(order);

      res.status(201).json({ message: 'Order placed successfully', orderId: result.insertedId });

    } catch (err) {

      console.error('Error placing order:', err);

      res.status(500).json({ error: 'Failed to place order' });

    }

  });

  // ✅ Get all orders (for admin)

  router.get('/', async (req, res) => {

    try {

      const orders = await db.collection('orders').find().toArray();

      res.json(orders);

    } catch (err) {

      console.error('Error fetching orders:', err);

      res.status(500).json({ error: 'Failed to fetch orders' });

    }

  });

  // ✅ Get orders for a specific email (user)

  router.get('/user/:email', async (req, res) => {

    const { email } = req.params;

    try {

      const orders = await db.collection('orders').find({ email }).toArray();

      res.json(orders);

    } catch (err) {

      console.error('Error fetching user orders:', err);

      res.status(500).json({ error: 'Failed to fetch user orders' });

    }

  });

  return router;

};

Routes/reservation.js

const express = require("express");

const router = express.Router();

const Reservation = require("../models/Reservation");

router.post("/reserve", async (req, res) => {

    try {

        const newReservation = new Reservation(req.body);

        await newReservation.save();

        res.status(200).json({ message: "Reservation successful!" });

    } catch (error) {

        res.status(500).json({ error: "Reservation failed!" });

    }

});

module.exports = router;

module.exports = router;

routes/Subscription.js// routes/Subscription.js

const express = require('express');

const router = express.Router();

const Subscribe = require('../models/subscribe');

// POST /api/subscribe

router.post('/', async (req, res) => {

  const { email } = req.body;

  if (!email) {

    return res.status(400).json({ message: "Email is required" });

  }

  try {

    const existing = await Subscribe.findOne({ email });

    if (existing) {

      return res.status(409).json({ message: "Already subscribed" });

    }

    const newSubscriber = new Subscribe({ email });

    await newSubscriber.save();

    res.status(201).json({ message: "Subscribed successfully" });

  } catch (err) {

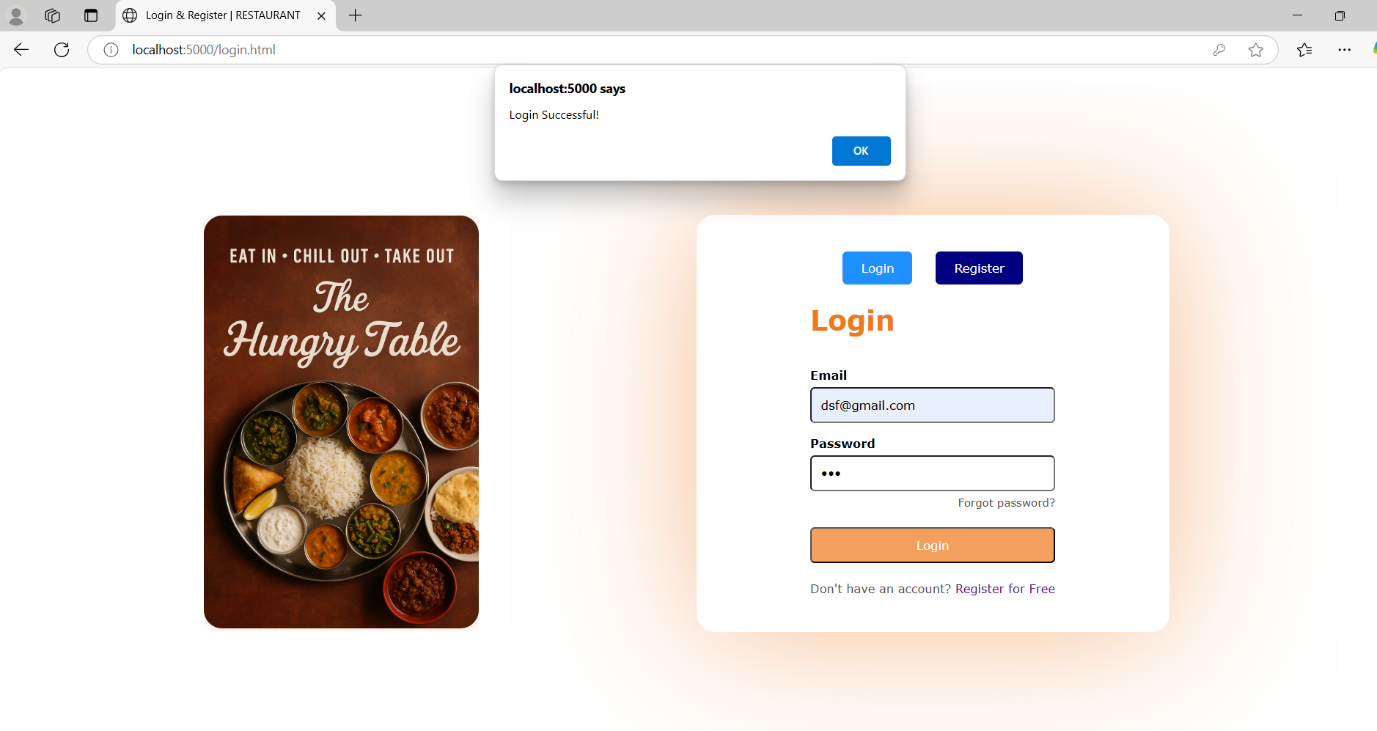
    console.error("❌ Subscription error:", err);

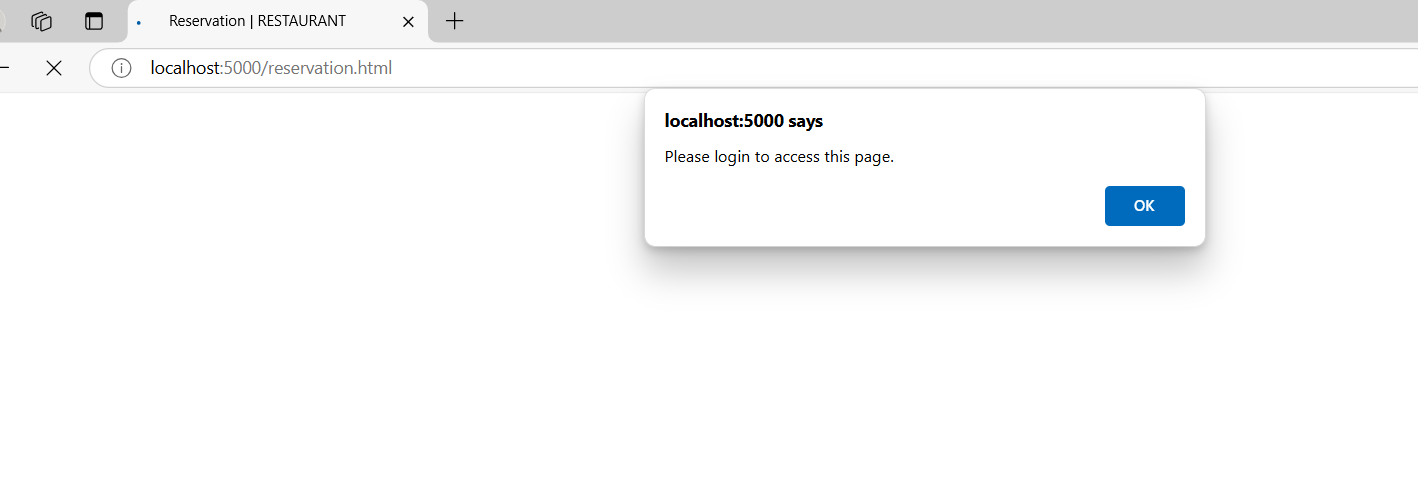
    res.status(500).json({ message: "Subscription failed", error: err.message });

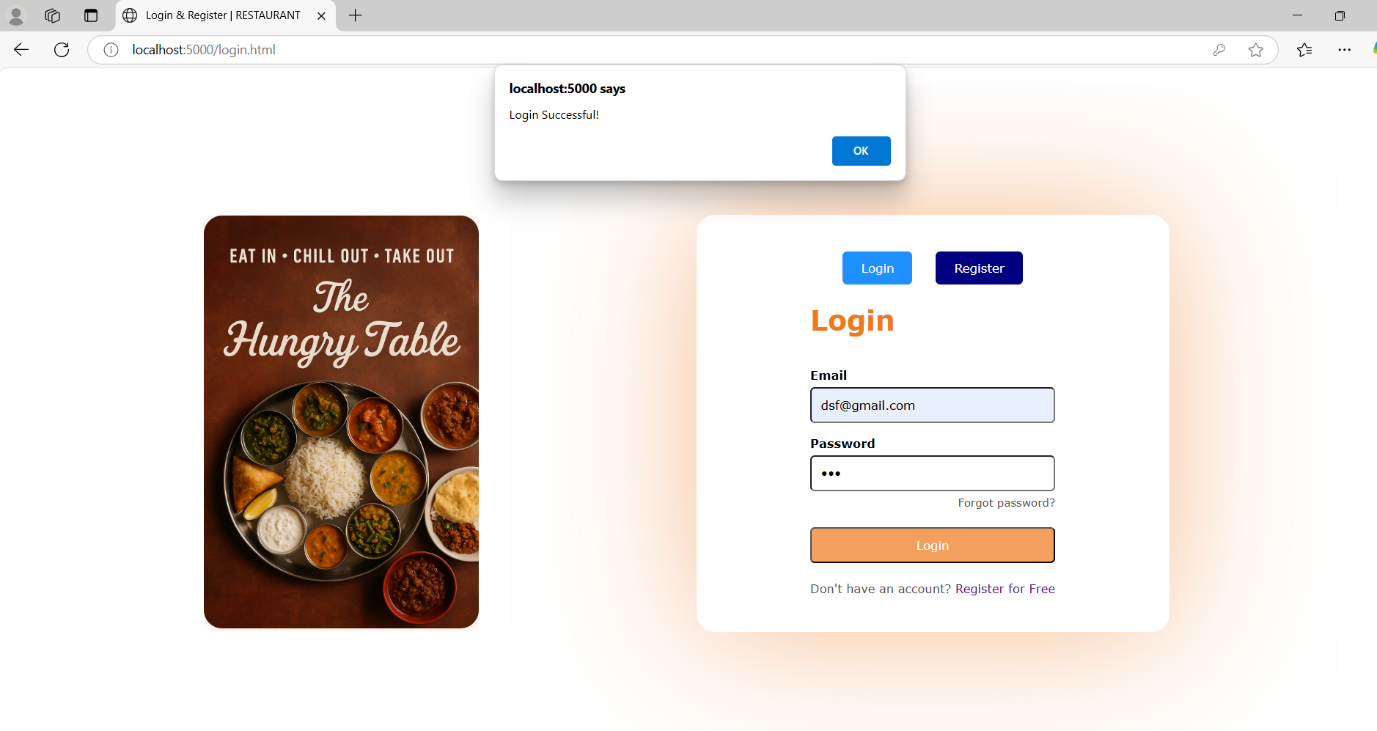
  }

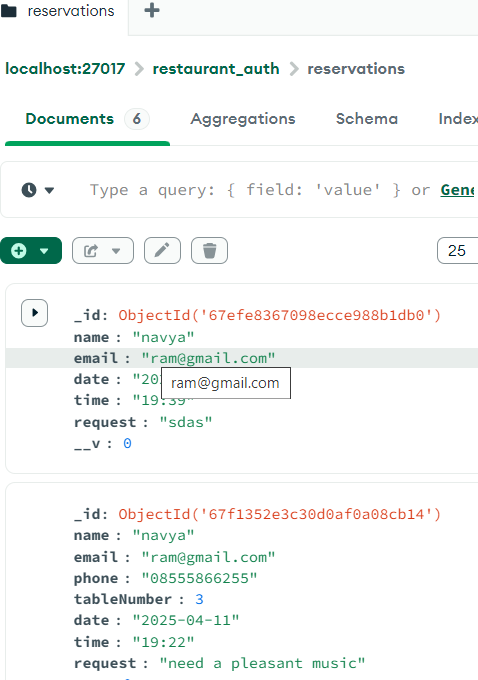
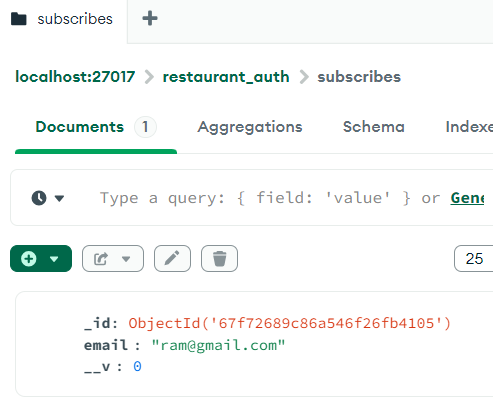
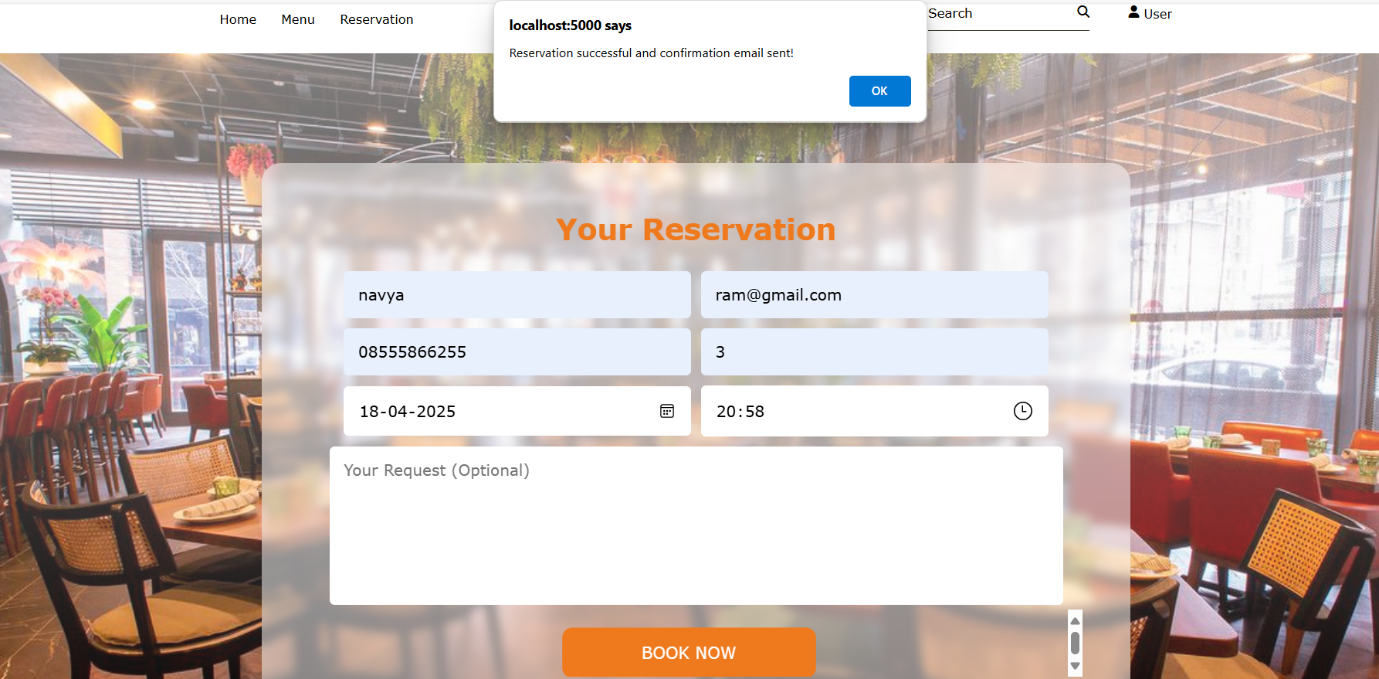
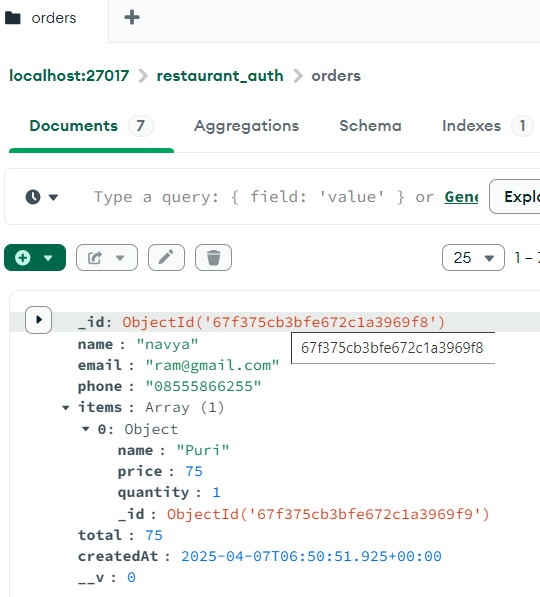
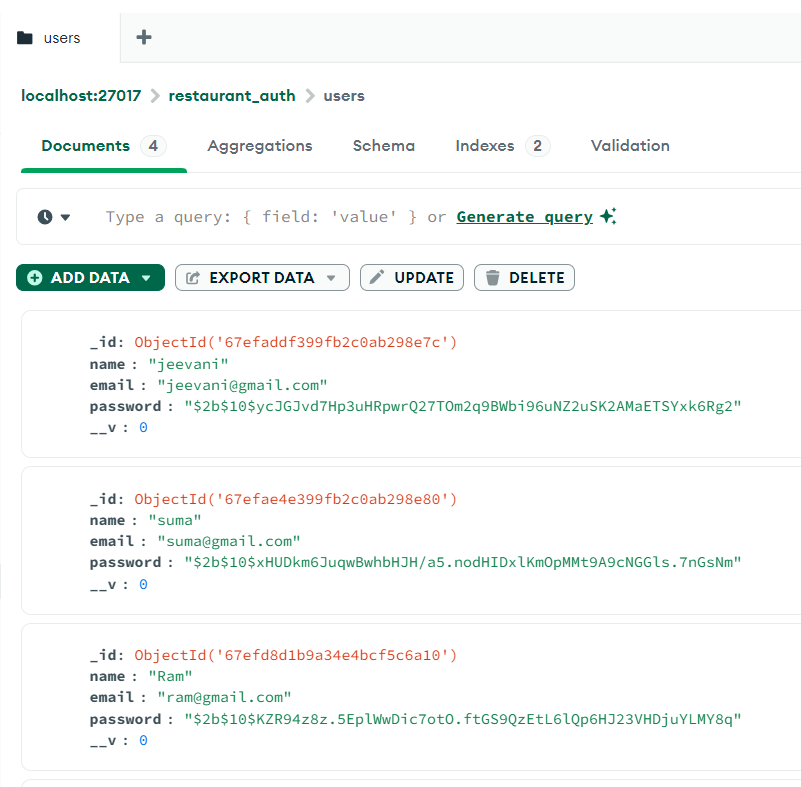
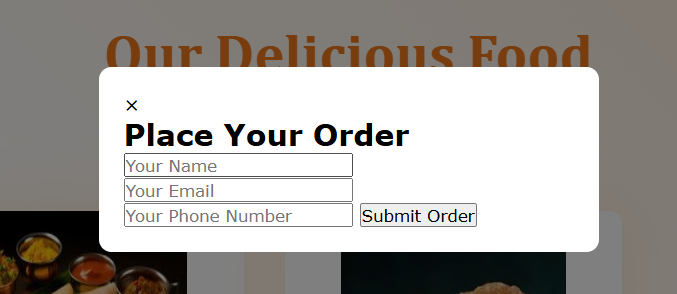
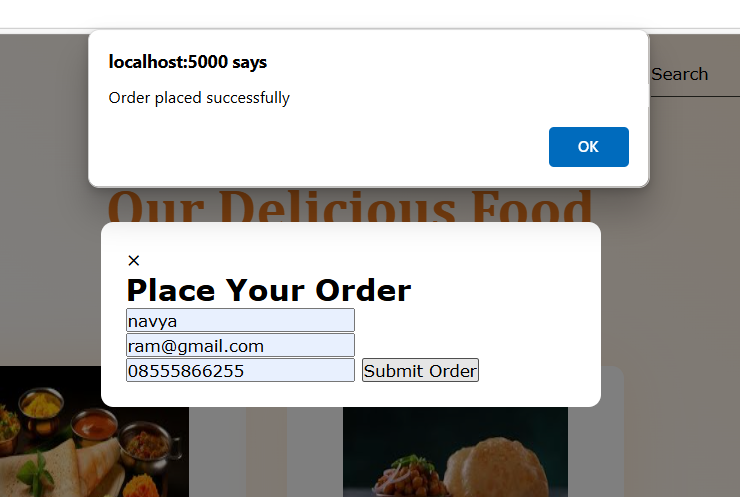
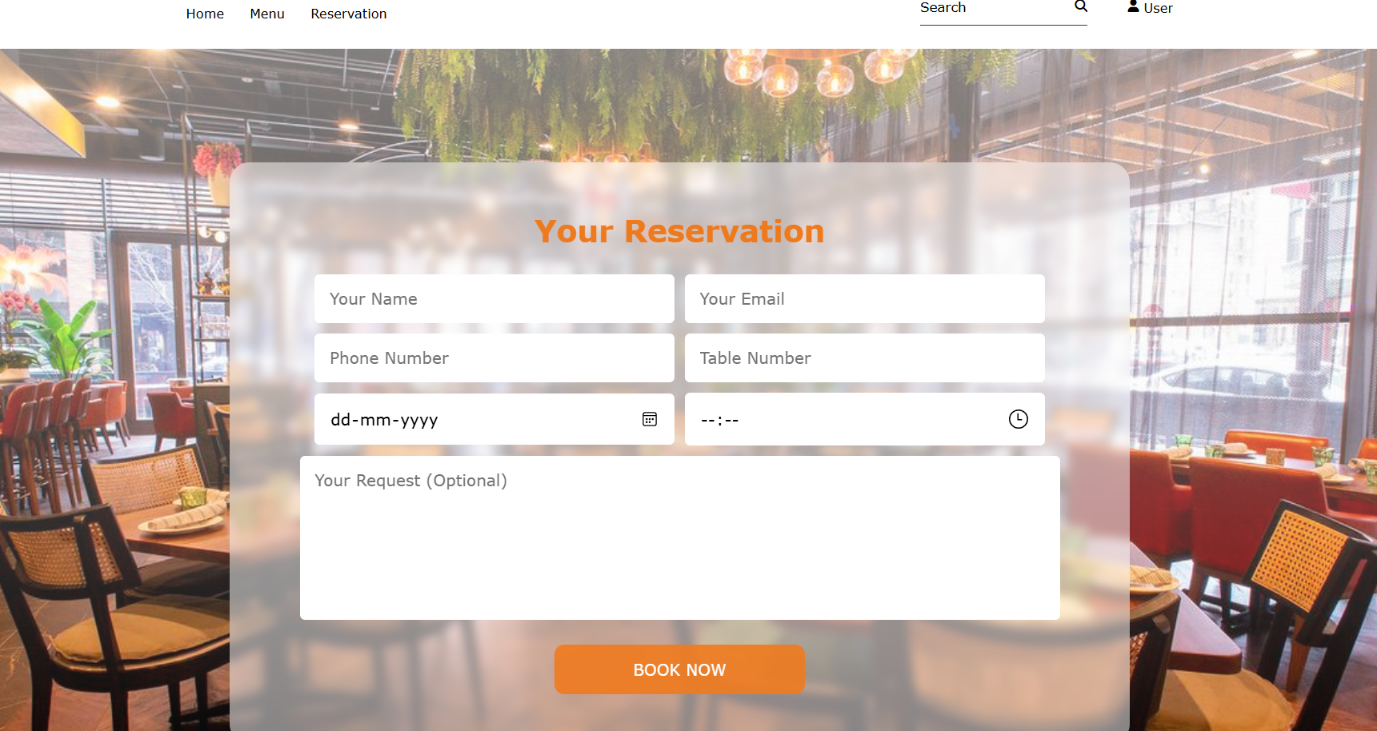
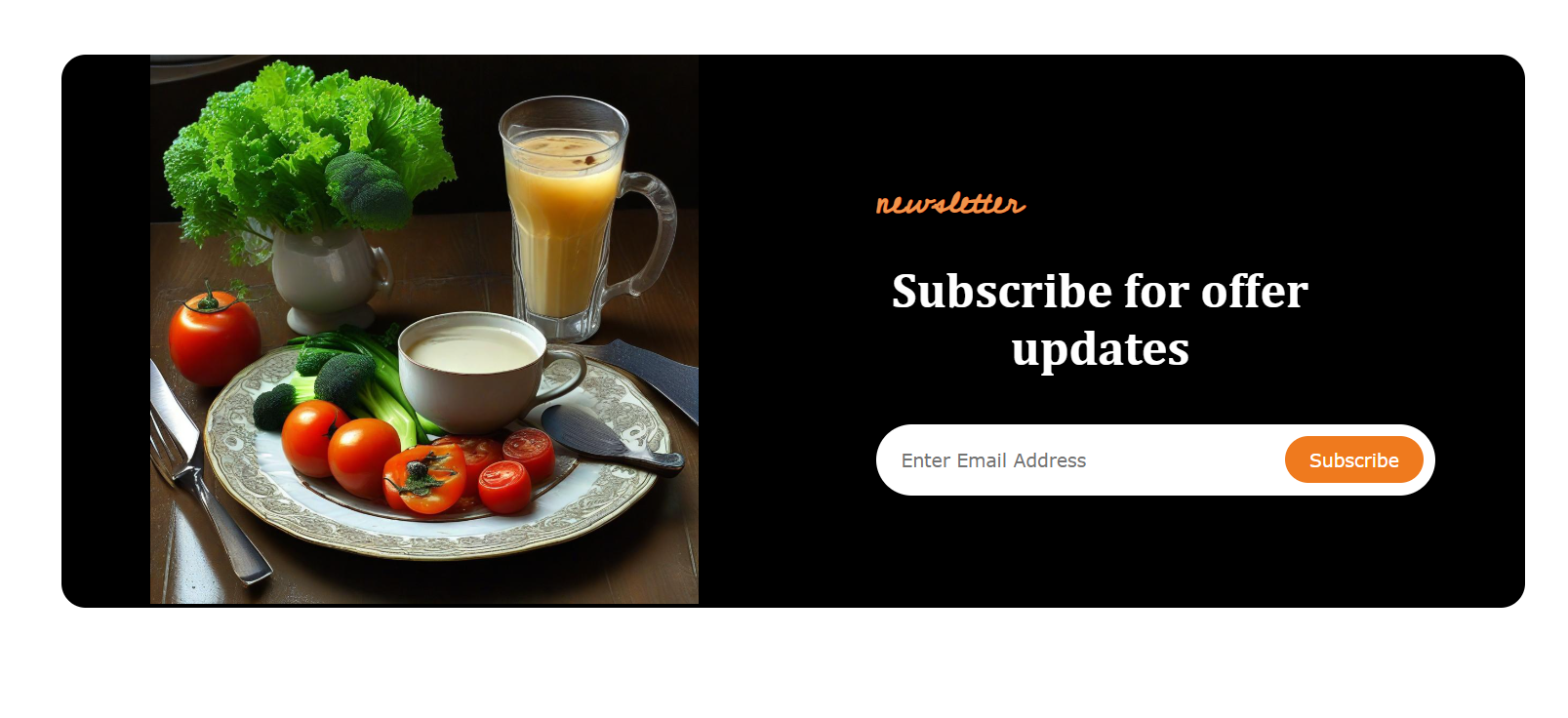
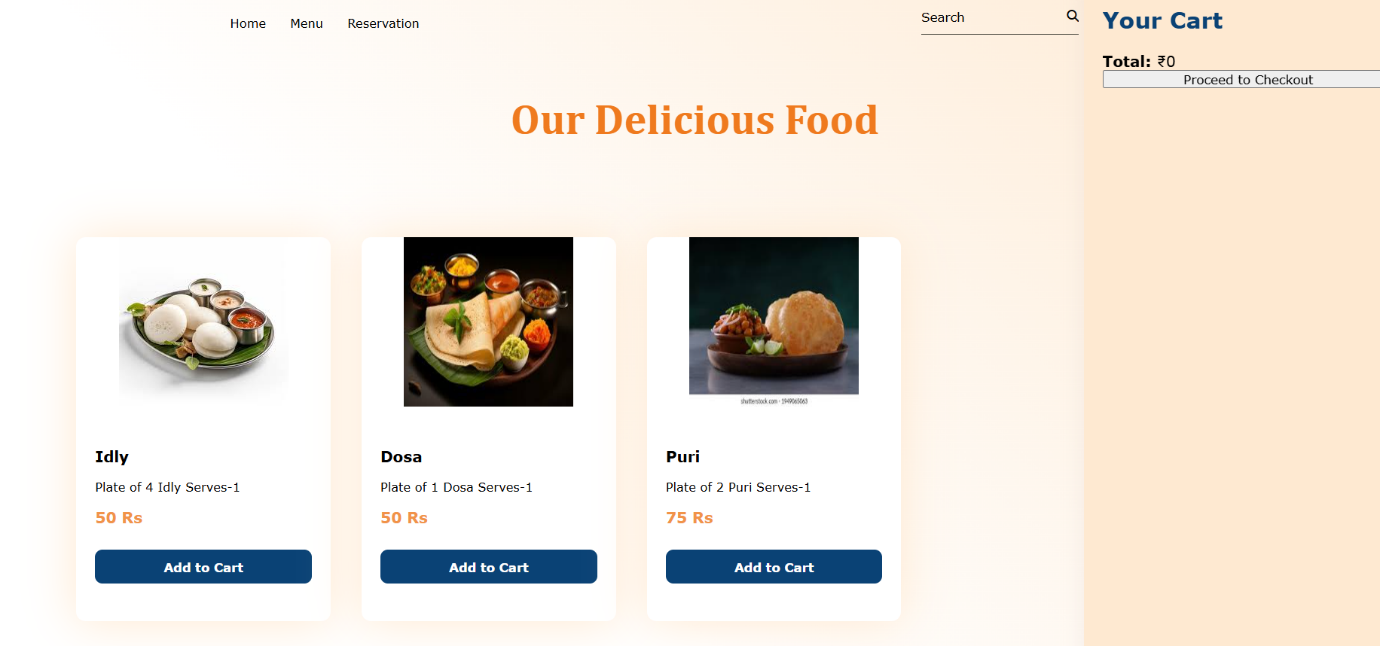
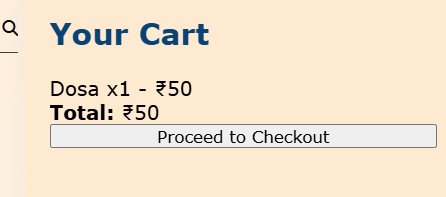
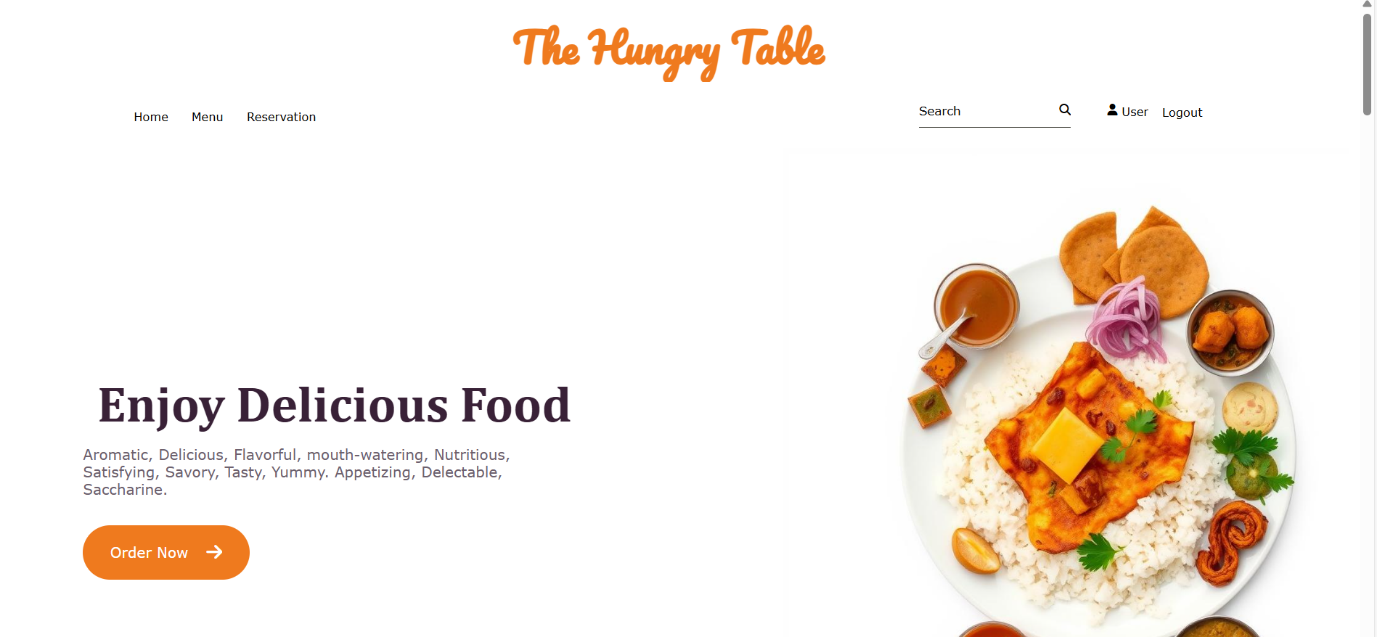
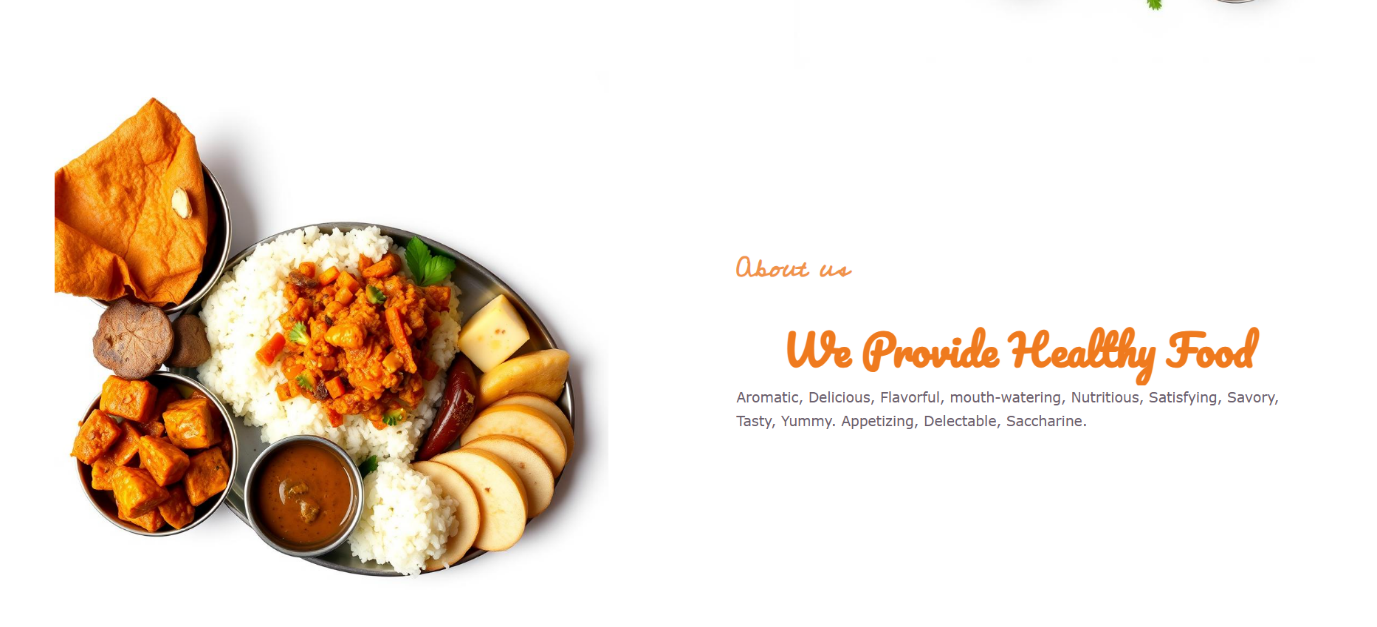
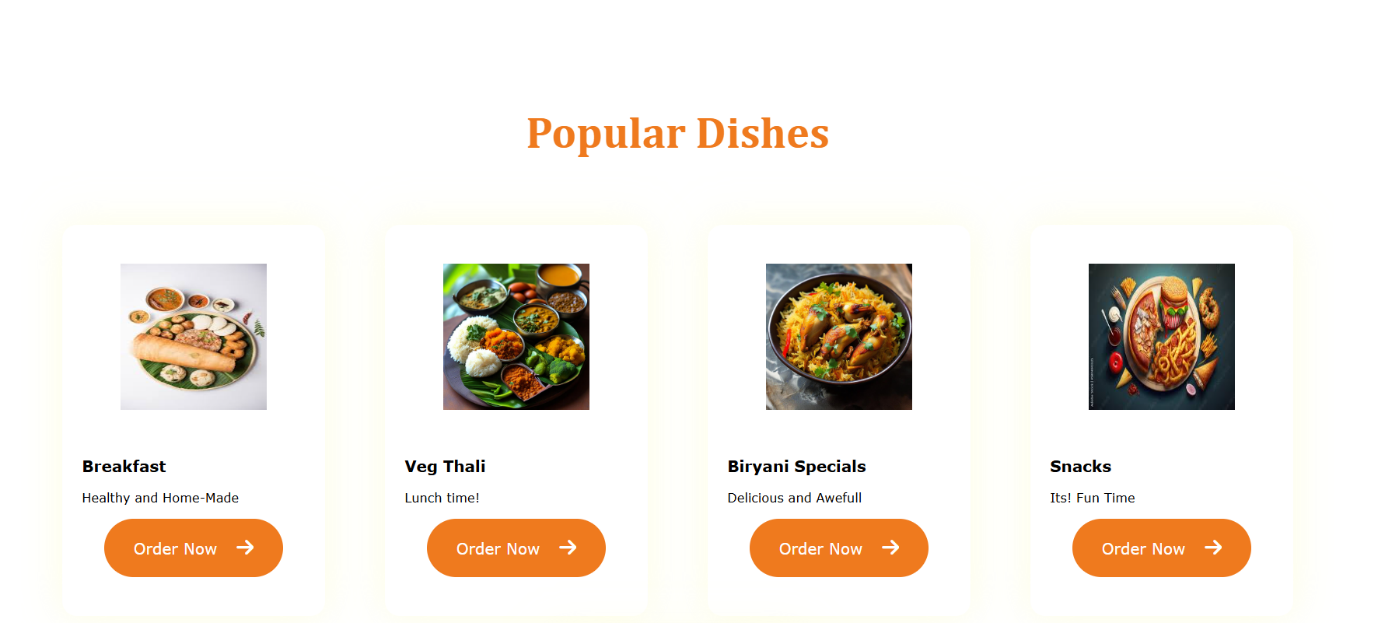
});

module.exports = router;







**CONCLUSION:**

### **Overview**

The Hungry Table is a full-stack restaurant web application that integrates modern web development technologies to deliver a seamless dining and ordering experience. This project was developed with the goal of simplifying restaurant interactions through an intuitive digital interface that allows users to **register/login, browse a dynamic menu, place food orders, and reserve tables online**.

Built using **HTML, CSS, and JavaScript** for the frontend and **Node.js, Express.js, and MongoDB** on the backend, the system follows a modular and scalable architecture that lays the groundwork for future expansion and real-world deployment.

### **Key Features Recap**

1. **User Authentication System**:
   * Secure registration and login with hashed passwords using **bcrypt**.
   * Dynamic form toggling between login and registration using JavaScript.
2. **Menu Browsing & Ordering**:
   * A categorized and visually rich menu interface.
   * Users can add items to a cart, adjust quantities, and place orders.
   * Backend validation ensures that no empty orders are processed.
3. **Table Reservation**:
   * Users can book tables in advance using a dedicated reservation form.
   * The system checks for slot conflicts and provides instant feedback.
4. **Database Integration**:
   * All user, order, and reservation data is stored in **MongoDB**, either hosted locally or via **MongoDB Atlas**.
   * Real-time fetching and updates ensure consistency across sessions.
5. **Deployment**:
   * The system can be hosted on platforms like **Render** or **Heroku**.
   * **PM2** is used for backend process management to ensure uptime and logging.

### **User Experience**

The platform is designed to prioritize ease of use, clarity, and responsiveness. From new users registering accounts to returning customers placing repeat orders, every feature is designed with accessibility and simplicity in mind. The system has been tested with valid and invalid inputs to ensure that error handling and user feedback are well-implemented.

The responsive layout ensures the application functions across various devices, including mobile phones, tablets, and desktops—catering to all types of users.

### **Scalability & Future Vision**

The project architecture supports future scalability. Several enhancements have already been identified and outlined, including:

* An **admin dashboard** for real-time content and order management.
* Full **CRUD support** for managing the food menu dynamically.
* **JWT-based authentication** for secure and scalable session management.
* **Payment gateway integration** for online transactions.
* **Email/SMS notifications** for order confirmations and updates.

These planned additions will transform The Hungry Table from a basic ordering platform into a fully operational digital restaurant management system, suitable for deployment in real-world restaurant chains or cloud kitchens.

### **Final Thoughts**

The Hungry Table demonstrates a strong foundation in full-stack web development, database integration, user management, and modular backend design. It offers practical solutions to real-life problems encountered by both restaurant owners and customers in today’s digital age.

Whether it’s reducing wait times, allowing contactless interaction, or enhancing customer service, this project stands as a testament to how technology can modernize traditional dining experiences. With future enhancements and continuous iteration, The Hungry Table has the potential to grow into a complete restaurant management ecosystem.