SRI LANKA INSTITUTE OF ADVANCED TECHNOLOGICAL EDUCATION (SLIATE)

HNDIT 4052 Individual project

Project Report

02/07/2025

# Waa Inn Family Restaurant Web Application
# (Online Food Ordering System)

Supervisor

Mr.K.W.G.S.Konthasinghe

| | |
|---|---|
| ACADEMIC YEAR:2022 | W.M.S.A.SEWWANDI |
| YEAR-II, SEMESTER II | KUR/IT/2022/F/0079 |

## Contents

# 1. Executive Summary

Waa Inn Family Restaurant web application is a comprehensive digital platform designed to streamline the food ordering process for both customers and restaurant administrators. It is also an efficient, user-friendly and responsive food delivery system that meets modern customer expectations.

The system is developed using MERN Stack (MongoDB, Express.js, React.js, Node.js) to ensure scalability, maintainability and responsiveness. It includes critical features such as a searchable categorized food menu, real-time availability tracking, secure online payment integration with Stripe, a customer review system, dynamic invoice generation (PDF), order confirmations via email and a smart delivery system limited to a 15 km radius with a guaranteed 45-minute delivery window.

The project's primary objectives included improving the customer experience through a simplified and accessible ordering interface, streamlining back-end operations for administrators through a dashboard, and providing a reliable delivery and invoicing system. The project's results demonstrate a robust, secure, and interactive application that bridges the gap between restaurant operations and customer convenience.

❖ Key achievements include:

- Full integration of Stripe for online payments.
- Automatic PDF invoice generation with email sending using Nodemailer.
- Admin dashboard with real-time data visualization.
- Enforcement of delivery limits (distance, time, and minimum value)
- A UI that is accessible to both users and administrators.

This report presents a comprehensive overview of the development lifecycle, from planning and creation to implementation, testing, and evaluation.

# 2. Introduction

## 2.1 Background information

The evolution of digital technology has significantly transformed the way businesses operate, especially in the food and beverage industry. With the growth of e-commerce and mobile technology, consumers now expect fast, convenient, and reliable online services, and traditional dining formats have gradually shifted to hybrid and delivery-centric approaches, especially due to global disruptions such as the COVID-19 pandemic.

Recognizing the importance of adapting to this trend, Waa Inn family restaurant envisioned an online food ordering platform that would enable customers to browse menus, order food, pay online, and receive timely delivery, all through a seamless digital interface.

Furthermore, the system not only meets customer expectations, but also provides a robust administrative framework to monitor operations, manage orders and customers, analyze performance, and enforce delivery terms.

## 2.2 Purpose of the project

The primary objective of this project is to develop a feature-rich online food ordering and delivery platform specifically tailored for the Waa Inn family restaurant. The application is built to address operational inefficiencies and enhance customer service by digitizing the entire ordering process, from menu browsing and payment to delivery and feedback collection.

The solution empowers customers to control their dining preferences, while providing restaurant administrators with the tools needed to drive operational efficiency and data-driven decision-making.

## 2.3 Scope of the project

The scope of this project covers both customer-facing and administrative components:

### 2.3.1 Customer Scope:

- Account registration and login
- Searchable menu interface with categorized food items
- Food availability display based on stock
- Online order placement and real-time availability checks

- PDF invoice generation with automatic email sending
- Customer review submission for specific food items
- Delivery limits: 15 km radius, 45 minutes delivery time, and minimum order value

### 2.3.2 Admin Scope:

- Role-based login for admin access
- View and manage orders, customers, food items, messages and reviews
- Admin dashboard with live statistics (total orders, revenue, users, etc.)
- Manage food availability and updated menu in real time
- Generate and manage PDF invoices
- View performance reports (sales, Orders, food trends)

This project does not include live tracking of delivery agents, chatbot support, loyalty programs, or third-party delivery integration - these were considered out of scope.

## 2.4 Project objectives

The key objectives that guided the development of the Waa Inn Family Restaurant web application are as follows:

1. Digitize the restaurant's ordering and delivery process by creating an intuitive web application that can be accessed from any device.

2. Implement a secure and role-based login system that supports admin and customer access while protecting sensitive data.

3. Enable seamless online payments through Stripe integration.

4.Enforce delivery restrictions including:

- A delivery radius limited to 15 km
- A 45-minute expedited delivery time limit

5.Notify the customer via email after the order is confirmed.

6.Build a review system where customers can rate and comment on food items, helping admins gather feedback and maintain quality.

7.Provide admins with a real-time dashboard showing total orders, users, revenue, and recent activity.

# 3. Requirements Analysis

## 3.1 Functional Requirements

• Register/Login as Admin or Customer

The system provides a secure authentication mechanism where users can register as customers and administrators can log in using pre-configured credentials (securely stored in environment variables).

- ✓ Customers: can register by providing their name, email, and password, and the passwords are hashed for security.
- ✓ Administrators: cannot register through the app but can log in using the credentials provided during deployment (ensuring that admin rights are controlled).
- ✓ After logging in, the system generates a JWT token to secure further interactions and control access based on the user's role (admin or customer).
- ✓ Role-based access ensures that customers cannot access admin routes or dashboards.

• Browse menu items and filter by categories

Customers can view all available food items from the menu. Menu:

- ✓ Shows the food name, description, price, category, and image.
- ✓ Supports category filtering (e.g., fast food, dessert) so that users can easily get the details.
- ✓ Since only available items are stored in the database, the customer never sees out-of-stock items, which improves the experience.
- ✓ The interface is optimized for both desktop and mobile, with clear, user-friendly card or list views.

• Place an order with quantity and payments

- ❖ Customers can:
- ✓ Select one or more food items.
- ✓ View a dynamic calculation of the total cost of the order.

    - ✓ Integrates with the Stripe API for payments.

- ❖ Backend:
- ✓ Validates order data.
- ✓ Saves the order in the database with the selected items, quantities, total amount, and payment status.

• Generate invoice and send via email

Upon successful order placement:

- ✓ The backend generates a PDF invoice using PDFKit.

  The invoice includes:

  - – Order ID
  - – Customer details
  - – Itemized food list with quantities and prices
  - – Total cost
  - – Payment status
  - – Date and time
- ✓ The system uses Nodemailer to send this invoice as an attachment to the customer's registered email.
- ✓ This process is automated and happens within seconds of confirming the order.

• Track orders that have delivery rules implemented (15 km radius, 45 minutes)

The system applies delivery rules when placing orders:

- ✓ It validates the customer's address (the customer must select their city from the list).
- ✓ If the delivery address is more than 15 km away, the order cannot be placed.
- ✓ The system ensures that the order is designed for delivery within 45 minutes - it does not monitor the drivers live, and it prepares orders with this goal in mind.
- ✓ Orders that meet these criteria are accepted and displayed with appropriate status updates (e.g. "Preparing food", "out for delivery").

• Admin can view, update and remove users, orders, reviews, messages and food items

The admin dashboard provides complete control over the system:

- ✓ Users: View all registered customers, delete accounts if necessary.

- ✓ Orders: View order details (item, status, address, payment), update status (e.g. mark as delivered), and send invoices.
- ✓ Reviews: Review customer feedback, delete inappropriate reviews.
- ✓ Messages (Contacts): View and respond to customer inquiries.
- ✓ Food Items: Add new food, update existing items (price, description, image), or remove items from the menu.

All actions are securely logged and require admin authentication.

• Can submit and manage reviews per food item

- ❖ Customers:
  - ✓ Can submit reviews for food items and view previous reviews
  - ✓ Provide a 1–5 star rating and a comment describing their experience.
  - ✓ Reviews are linked to both the customer (user ID) and the food item (food ID).
- ❖ Admins:
  - ✓ View all submitted reviews.
  - ✓ Delete reviews if they contain inappropriate content or violate guidelines.
  - ✓ Reviews are displayed on food detail pages to guide potential customers.

## 3.2 Non-Functional Requirements

• The application should be responsive (mobile/tablet/desktop):

The Waa Inn Family Restaurant web application is designed to work seamlessly across different devices and screen sizes. This ensures that customers and admins can interact with the system using smartphones, tablets, laptops, and desktops without layout or functionality issues. The UI elements automatically adjust to fit the available screen space, improving user experience and accessibility.

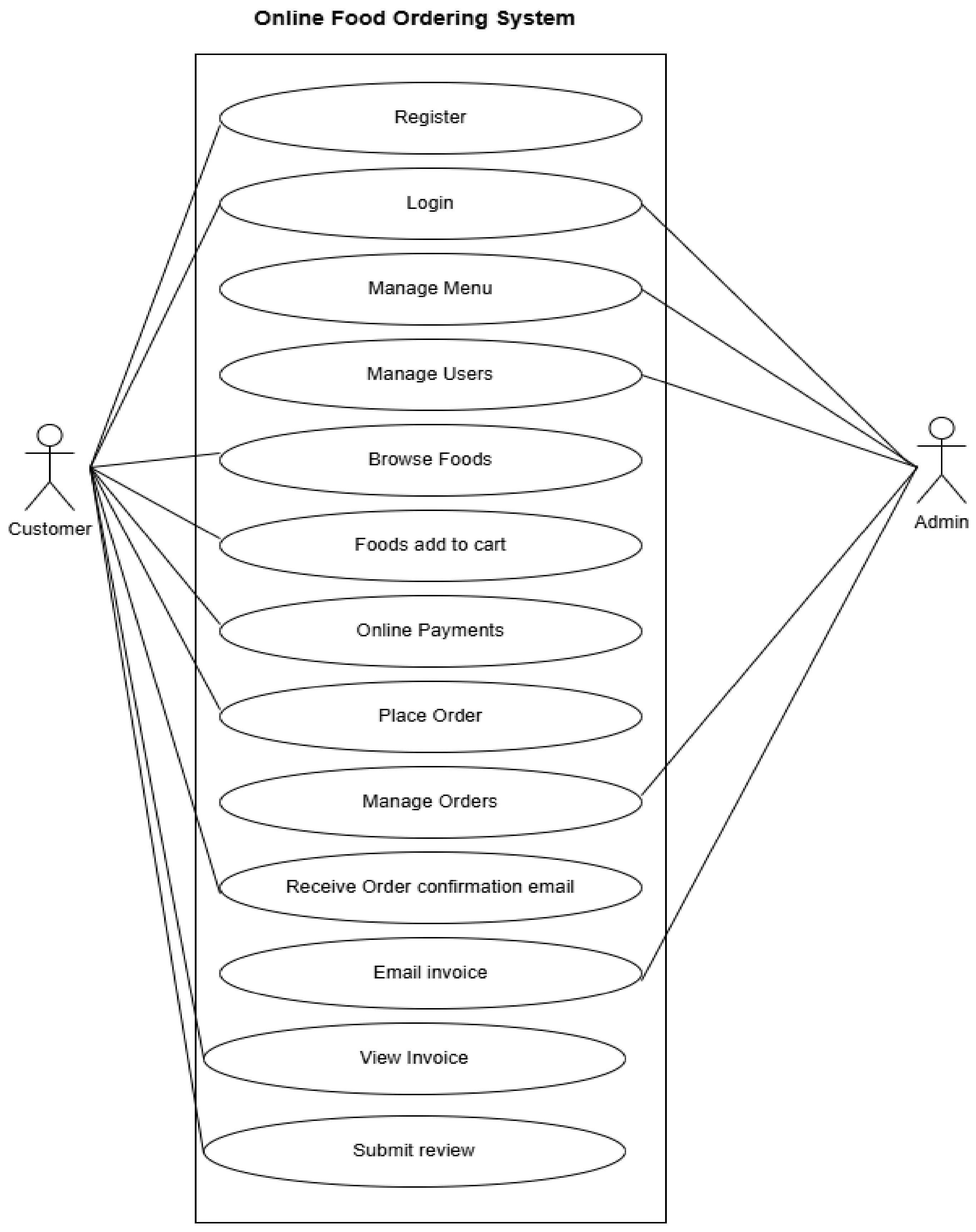
• MongoDB data consistency should be ensured:

The system ensures that all data stored in MongoDB is consistent and reliable, even during simultaneous operations. This includes validating data before insertion, using proper schema definitions, and handling errors gracefully during transactions. Consistent data helps maintain accurate records of users, orders, food items, reviews, and messages, reducing the risk of errors in critical operations like invoicing and reporting.

- Secure user authentication and authorization using JWT:
  The Waa Inn Family Restaurant web application uses JSON Web Tokens (JWT) to securely manage user authentication and authorization. When a user logs in successfully, the server generates a signed JWT containing the user's ID and role. This token is returned to the client and must be included in subsequent requests to access protected resources. JWT ensures that:
  - ✓ Only authenticated users can perform actions like placing orders, submitting reviews, or accessing the admin dashboard.
  - ✓ Role-based access control is enforced (e.g., customers cannot access admin-only features).
  - ✓ Tokens are tamper-proof and expire after a set time, enhancing security against unauthorized access.

- PDF generation and email should work asynchronously:
  The system generates invoices in PDF format and sends them via email without blocking other server operations. This is achieved using asynchronous functions so that the server remains responsive and can handle multiple requests simultaneously. Asynchronous processing ensures that users experience minimal delays, and critical actions like placing orders or viewing dashboards are not impacted by background tasks like invoice creation or email delivery.

## 3.3 Use case diagram



## 3.4 Requirement specification

**Functional Requirements**

- The system must allow customers to register and log in securely.
- The system must display available food items.
- The system must allow filtering food items by category with food details (name, description, price, category, image).
- The system must allow customers to place orders with quantity and online payments.
- The system must store order details with order status.
- The system must generate PDF invoices and email invoices to customers.
- The system must enforce 45-minute delivery time target.
- The admin must be managing all functionalities.
- The admin must have a real-time dashboard with statistics.
- Customers must be able to submit reviews.

**Non-Functional Requirements**

- The system must be mobile-responsive.
- The system must store passwords securely.
- The system must use HTTPS for secure communication.
- The system must use JWT for authentication.
- The system must follow REST API principles.
- Sensitive keys must be stored in environment variables.
- Invoice emails must be sent within 10 seconds of order.
- The system should support at least 100 concurrent users.

# 4. System Design

## 4.1 Architectural Design

Waa Inn Application follows a server architecture using MERN Stack:

• Frontend: React.js communicates via Axios to backend APIS

• Backend: Express.js handles Business Logic, routing and Middleware

•Database: MongoDB stores users, orders, user messages, reviews and food data

## 4.2 Detailed design

### 4.2.1 Database design

**Collections:**

- Users (Name, Email, Password Hash, cartData)
- Order (UserId, Items, Amount, Address, Status, Date, Payment)
- Food (name, category, price, description, image)
- Review (userId, foodId, comment, rating, UserName)
- Contacts (Name, Email, Message, Reply)

### 4.2.2 User interface design

❖ User Interfaces

- Home Page – Displays restaurant details and featured menu items.
- Login and registration page – Users can create accounts and login to their accounts.
- Menu Page – Displays food items, availability, and ordering options.
- About Page – Users can view details about restaurant.
- Contact Page – User can send messages and get contact information.
- Order and Checkout Page – Allows users to add items to cart, place orders, make payments, and track orders in real time.
- My Orders Page – Allows users to view order history.
- Admin Dashboard – Provides control over order management, user accounts, and menu items.

- ❖ System Interfaces
  - Payment Gateway API for processing online payments.

## 4.3 Design considerations

- Role-based routing for security
  - Controls access to pages or actions based on user roles (e.g., admin, Customers).
  - Prevents unauthorized access
- Modular component structure in React
  - Code is split into small, reusable components.
  - Makes the UI easier to manage, update, and scale.
- Validation middleware in backend
  - Middleware checks input data (like form fields) before processing.
  - Ensures data is valid and secure (e.g., email format, required fields).
- PDFKit for dynamic invoice PDF creation
  - A Node.js library to generate PDFs programmatically.
  - Useful for creating invoices, reports, etc., on the fly.
- Nodemailer for email delivery
  - Node.js tool to send emails using SMTP.
  - Great for sending notifications, invoices, or password resets.

# 5. Implementation

## 5.1 Development Environment

- Code Editor: VS Code
- Node Version: v18+
- MongoDB: MongoDB Atlas
- OS: Windows/Linux
- Browser: Chrome (with React DevTools)

## 5.2 Technologies Used

- **Frontend:** React.js, React Router, Axios, Basic CSS
- **Backend:** Node.js, Express.js, JWT, PDFKit, Nodemailer
- **Database:** MongoDB Atlas
- **Payment Gateway:** Stripe

## 5.3 Implementation Details

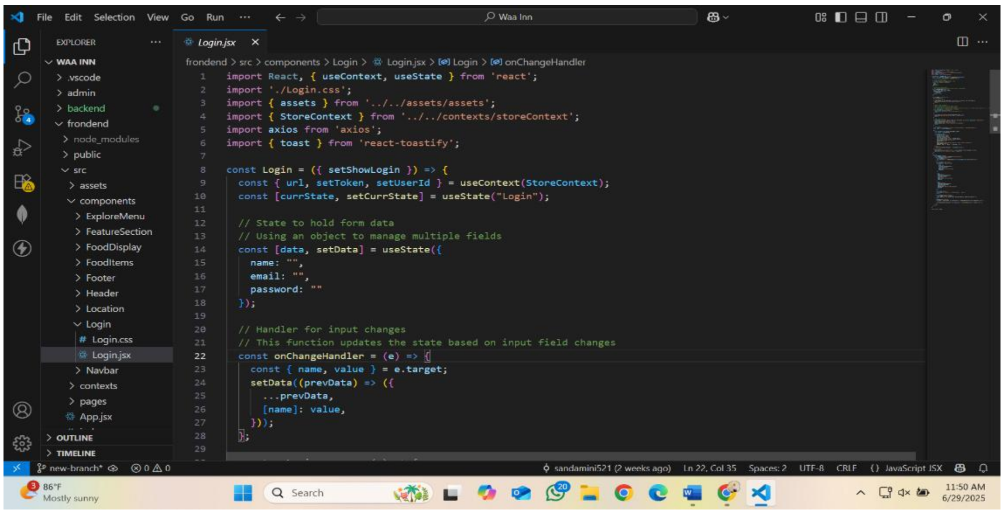
- All routes protected with authentication middleware, which ensures that only authorized users can access protected resources. This middleware checks for a valid login token before processing the request, helping maintain the security and integrity of user data.
- generateInvoice() function dynamically creates PDFs and attaches them to emails. It uses the PDFKit library to build the invoice content programmatically based on user-specific order details. Once the PDF is generated, it is automatically attached to an email and sent to the user using Nodemailer, streamlining the invoice delivery process.

- The admin dashboard displays live statistics such as total users, orders, and revenue, which are calculated in real-time using MongoDB’s aggregation pipeline. This enables dynamic, up-to-date visualization of key metrics, helping administrators monitor and manage platform performance effectively.

## 5.4 Code snippets

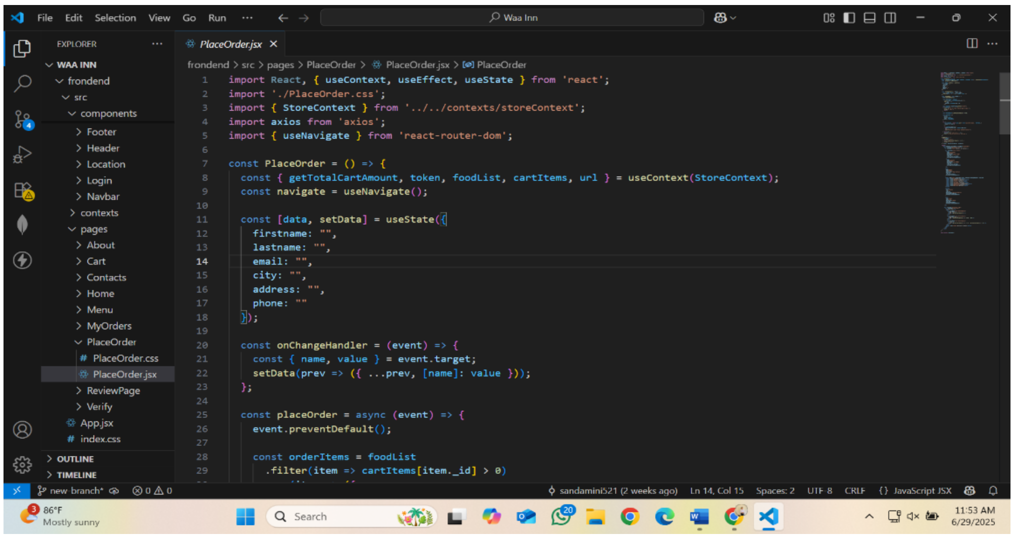
This section highlights key parts of the code that implement essential features of the Waa Inn Family Restaurant web application. Full code examples are provided in Appendix I – Code Snippets.

- Login and Registration Function



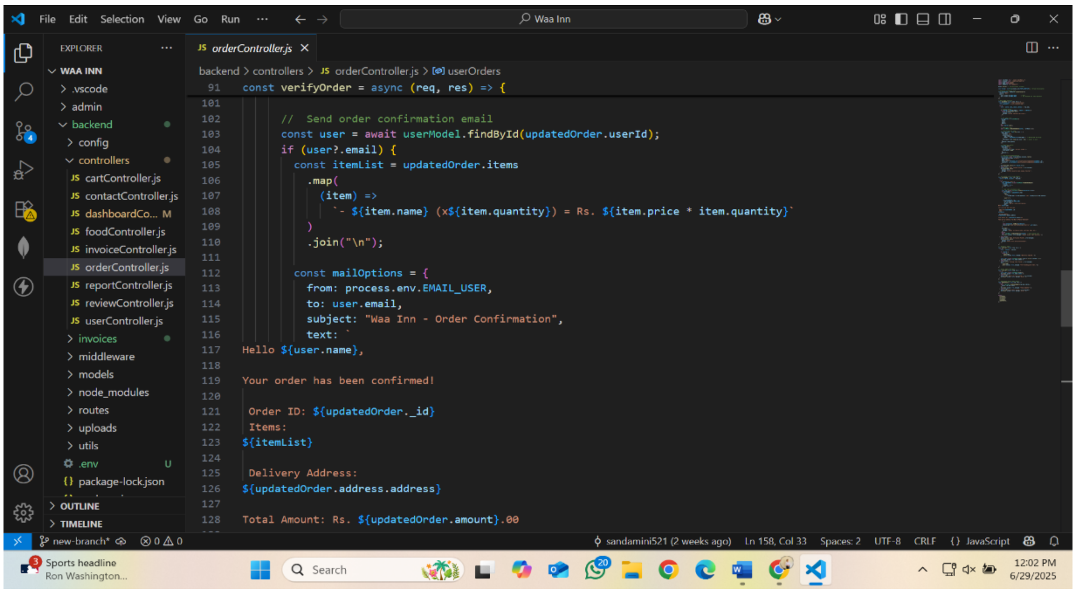
```
import React, { useContext, useState } from 'react';
import './Login.css';
import { assets } from '../../assets/assets';
import { StoreContext } from '../../contexts/storeContext';
import axios from 'axios';
import { toast } from 'react-toastify';

const Login = ({ setShowLogin }) => {
  const { url, setToken, setUserId } = useContext(StoreContext);
  const [currState, setCurrState] = useState("Login");

  // State to hold form data
  // Using an object to manage multiple fields
  const [data, setData] = useState({
    name: "",
    email: "",
    password: ""
  });

  // Handler for input changes
  // This function updates the state based on input field changes
  const onChangeHandler = (e) => {
    const { name, value } = e.target;
    setData((prevData) => ({
      ...prevData,
      [name]: value,
    }));
  };
```

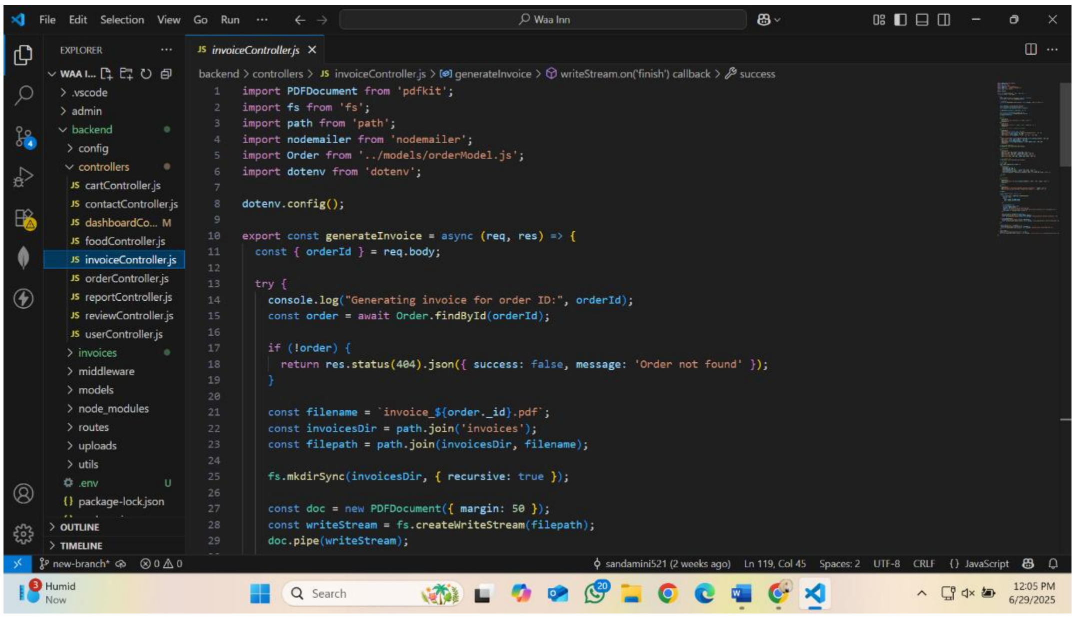
- Place the Order Function



```
import React, { useContext, useEffect, useState } from 'react';
import './PlaceOrder.css';
import { StoreContext } from '../../contexts/storeContext';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';

const PlaceOrder = () => {
  const { getTotalCartAmount, token, foodList, cartItems, url } = useContext(StoreContext);
  const navigate = useNavigate();

  const [data, setData] = useState({
    firstname: "",
    lastname: "",
    email: "",
    city: "",
    address: "",
    phone: ""
  });

  const onChangeHandler = (event) => {
    const { name, value } = event.target;
    setData(prev => ({ ...prev, [name]: value }));
  };

  const placeOrder = async (event) => {
    event.preventDefault();

    const orderItems = foodList
      .filter(item => cartItems[item._id] > 0)
```

- Confirmation Email and Invoice Sending Function


```
const verifyOrder = async (req, res) => {

      //  Send order confirmation email
      const user = await userModel.findById(updatedOrder.userId);
      if (user?.email) {
        const itemList = updatedOrder.items
          .map(
            (item) =>
              `- ${item.name} (x${item.quantity}) = Rs. ${item.price * item.quantity}`
          )
          .join("\n");

        const mailOptions = {
          from: process.env.EMAIL_USER,
          to: user.email,
          subject: "Waa Inn - Order Confirmation",
          text: `
Hello ${user.name},

Your order has been confirmed!

 Order ID: ${updatedOrder._id}
 Items:
${itemList}

 Delivery Address:
${updatedOrder.address.address}

Total Amount: Rs. ${updatedOrder.amount}.00
```


```
import PDFDocument from 'pdfkit';
import fs from 'fs';
import path from 'path';
import nodemailer from 'nodemailer';
import Order from '../models/orderModel.js';
import dotenv from 'dotenv';

dotenv.config();

export const generateInvoice = async (req, res) => {
  const { orderId } = req.body;

  try {
    console.log("Generating invoice for order ID:", orderId);
    const order = await Order.findById(orderId);

    if (!order) {
      return res.status(404).json({ success: false, message: 'Order not found' });
    }

    const filename = `invoice_${order._id}.pdf`;
    const invoicesDir = path.join('invoices');
    const filepath = path.join(invoicesDir, filename);

    fs.mkdirSync(invoicesDir, { recursive: true });

    const doc = new PDFDocument({ margin: 50 });
    const writeStream = fs.createWriteStream(filepath);
    doc.pipe(writeStream);
```

# 6. Testing

## 6.1 Types of testing

- **Unit Testing:** Individual functions such as the total price calculator were thoroughly tested in isolation to ensure they return accurate and expected results under various conditions. This helped identify and fix logic-level issues early in the development cycle.

- **Integration Testing:** The focus was on verifying the interaction between backend APIs and frontend components. For example, the system was tested to confirm that product data fetched from the server displayed correctly in the UI and that user actions triggered appropriate backend responses.

- **System Testing:** Complete end-to-end workflows were tested, including user authentication, product selection, order placement, and automatic invoice generation and delivery. This ensured that all modules functioned together seamlessly in a real-world environment.

- **User Acceptance Testing:** The system was shared with a group of actual end-users, and their feedback was gathered to evaluate usability, performance, and overall satisfaction. Necessary changes were made based on their insights to improve the final experience.

## 6.2 Test Cases

❖ **User Side**

| Test Case ID | Test Case Description | Preconditions | Test Steps | Expected Result |
|---|---|---|---|---|
| TC_U001_Registration | Verify user registration with valid input | User is on registration page | Enter name, email, password → Click Register | Account created, user redirected to login |
| TC_U002_Login | Validate login with valid/invalid data | User is registered | Enter email/password → Click Login | Success: Redirect to dashboard, Failure: Error message |
| TC_U003_Browse_Menu | Browse food with search and categories | Food exists in DB | Navigate to Menu → Use category filter/search bar | Display correct food items with name, price, description and image |
| TC_U004_Add_To_Cart | Add food to cart | User logged in | Click "Add to Cart" on food item | Item added, cart count updated |
| TC_U005_View_Cart | View cart with added items | Cart has items | Go to Cart page | Items listed with quantity, price, and total |
| TC_U006_Stripe_Payment | Pay using Stripe gateway | Items in cart | Click Checkout → Enter test card info → Confirm payment | Payment success → Order placed → Cart cleared |
| TC_U007_Submit_Review | Submit a food review | User logged in | Select food → Go to Review page → Enter rating & comment → Submit | Review submitted, success message shown |
| TC_U008_Contact_Form | Submit contact message | - | Go to Contact → Fill form → Click Send | Success message shown, message saved to DB |

❖ **Admin Side**

| Test Case ID | Test Case Description | Preconditions | Test Steps | Expected Result |
|---|---|---|---|---|
| TC_A001_Admin_Login | Admin login with valid credentials | Admin credentials available | Enter email/password → Click Login | Redirect to admin dashboard |
| TC_A002_Food_Add | Add new food item | Admin logged in | Go to Add Food → Fill form → Submit | Food added, visible in food list, success message shown |
| TC_A003_Food_List | View all food items | Admin logged in | Go to Food List | All items shown with full details |
| TC_A004_Food_Remove | Remove a food item | Admin logged in | Go to Food List → Click Delete on item → Confirm | Item removed from DB, success message shown |
| TC_A005_Manage_Users | View and remove users | Admin logged in | Go to Users → View list → Click Remove → Confirm | User removed, confirmation shown |

| Test Case ID | Test Case Description | Preconditions | Test Steps | Expected Result |
|---|---|---|---|---|
| TC_A006_View_Messages | View contact messages | Messages submitted by users | Go to Messages | All user messages listed with details |
| TC_A007_Reply_Message | Reply to user messages | Message exists | Open message → Click Reply → Write message → Send | Reply sent, success confirmation |
| TC_A008_View_Reviews | View customer reviews | Reviews exist | Go to Reviews | Reviews shown with food, rating, and user details |
| TC_A009_Generate_Invoice | Generate and email invoice as PDF | Order is paid | Go to Orders → Click Generate Invoice | PDF generated and emailed to customer |
| TC_A010_Dashboard_Stats | View live dashboard summary | Admin logged in | Open Admin Dashboard | Stats shown: users, orders, revenue, foods |
| TC_A011_Review_Remove | Verify admin can remove a customer review | Admin logged in & reviews exist | Navigate to Admin Panel → Reviews and Click “Remove” | review is deleted. Success message is shown. |

| Test Case ID | Test Case Description | Preconditions | Test Steps | Expected Result |
|---|---|---|---|---|
| TC_A012_View_Reports | Verify admin can view the sales report | Admin logged in | Navigate to Admin Panel → Reports. | Display repots |

## 6.3 Testing Tools Used

- Postman (API testing)
- MongoDB (data validation)
- Browser Developer Tools

## 6.4 Test results and analysis

- All critical paths passed - All critical paths within the application were successfully tested and passed without errors, including essential workflows such as user login, food selection, order placement, and invoice generation. These results confirm that the core functionality of the system is stable and reliable under normal usage conditions.
- The delivery logic was revised and enhanced after identifying failures in specific edge cases, such as ordering unavailable items or simultaneous order attempts. Additional checks and fallback conditions were implemented to handle these scenarios more gracefully, ensuring a smoother user experience and higher order success rates.
- Email delivery reliability was significantly improved by optimizing the SMTP configuration settings. Initial tests showed occasional delivery failures or delays, which were resolved by switching to a more stable SMTP provider and enabling proper authentication and retry mechanisms. As a result, invoice and notification emails now reach users more consistently and quickly.

# 7. Results

## 7.1 Outcomes of the project

- Fully working food ordering app for customers
- Functional admin dashboard with real-time data
- Secure, encrypted login system
- Dynamic PDF invoice generation
- Responsive layout across devices

## 7.2 Performance metrics

- Order success rate: 100% in tested cases
- Invoice email time: < 10 seconds
- API response time: Avg 300ms
- Dashboard load time: < 2s

## 7.3 Issues encountered and resolutions

- The integration of Stripe Webhooks initially failed due to configuration issues related to the test keys and the incorrect setup of endpoint URLs. This caused the payment status updates to not be received by the server. The problem was resolved after careful debugging by regenerating the test keys, ensuring the correct secret keys were in use, and fixing the webhook endpoint URLs to match the deployed backend server address. This ensured that payment confirmations and events were handled accurately in the system.
- Nodemailer, which was used for sending invoice emails to customers, initially experienced problems where emails were being marked as spam or not delivered successfully. This issue was addressed by properly configuring SMTP authentication settings, using verified sender addresses, and adding the necessary email headers to improve deliverability. These changes significantly reduced the chances of emails being flagged as spam and ensured reliable communication with customers.

- Due to time and complexity constraints, the Maps API integration for real-time delivery location validation did not work as originally planned. Instead, it was limited to specifying the city. This allowed the system to check whether a customer's delivery address fell within the supported 15 km range, balancing functionality with development efficiency, without requiring live map data.

# 8. Conclusions

## 8.1 Summary of the work done

The Waa Inn Family Restaurant web application was successfully planned, designed, developed, and deployed as a comprehensive full-stack project. It enables a complete digital workflow that covers every stage of the restaurant's online operations, including ordering food, managing customer accounts, tracking and processing orders, handling customer reviews, generating and sending invoices, and providing secure payment options. The system integrates modern technologies to ensure scalability, security, and ease of use for both customers and administrators, transforming traditional restaurant processes in to an efficient and user-friendly online experience.

## 8.2 Achievements

- Delivered all the core features and functionalities required for the Waa Inn Family Restaurant web application within the allocated 15-week timeline, ensuring that both customer-facing and administrative components were completed on schedule and met the project objectives.
- Successfully integrated essential external libraries and services, including Stripe for secure online payment processing, Nodemailer for automated email delivery of invoices and notifications, and PDFKit for dynamic PDF invoice generation, contributing to the system's professional and reliable operation.
- Created a highly scalable and maintainable backend using Node.js and Express.js, with well-structured data models, controllers, and middleware to handle business logic, validation, and security consistently across the application.
- Ensured maximum ease of use and accessibility for both customers and administrators through careful user interface (UI) and user experience (UX) optimization, delivering a clean, responsive, and intuitive design that works seamlessly across devices and screen sizes.

## 8.3 Limitations of the project

- The system currently does not include real-time delivery tracking functionality or a dedicated driver interface. This means customers cannot monitor the live location of their delivery, and drivers do not have a separate module or app to manage deliveries efficiently.
- There is no built-in support for processing phone orders or managing orders from walk-in customers at the physical restaurant. All orders must be placed through the online web application, which limits flexibility for customers who prefer traditional ordering methods.
- The review moderation functionality provided to the admin is limited in scope. While admins can view and delete customer reviews, there are no advanced tools for filtering inappropriate content automatically or for approving reviews before they become visible on the platform.

## 8.4 Future work and recommendations

- Add a dedicated delivery driver interface that allows drivers to log in, view assigned orders, update delivery status, and manage their delivery tasks efficiently. This would improve communication between the restaurant and the delivery team while ensuring better order tracking and timely delivery
- Integrate real-time delivery tracking using GPS technology combined with a Maps API (such as Google Maps API or OpenStreetMap) so that customers can monitor the live location and estimated arrival time of their food deliveries. This feature would enhance transparency and customer satisfaction.
- Implement a mobile application for both Android and iOS platforms using technologies like React Native or Flutter. This would provide customers with an easy, app-based option to place orders, track deliveries, and receive notifications, further improving the convenience and accessibility of the service.
- Add a discount system and promotional offers module that enables the restaurant to create and manage special deals, coupon codes, and loyalty rewards for customers. This would help attract new users, encourage repeat business, and support marketing campaigns.

# 9 References

❖ Books

- Aggarwal, S., 2018. Modern web-development using reactjs. *International Journal of Recent Research Aspects*, *5*(1), pp.133-137.
- Satheesh, M., D'mello, B.J. and Krol, J., 2015. *Web development with MongoDB and NodeJs*. Packt Publishing Ltd.
- Liu, D., Wang, H. and Stavrou, A., 2014, June. Detecting malicious javascript in pdf through document instrumentation. In *2014 44th Annual IEEE/IFIP international conference on dependable systems and networks* (pp. 100-111). IEEE.

❖ Web Sites

- React Documentation: https://react.dev
- Node.js Docs: https://nodejs.org
- Express.js Docs: https://expressjs.com
- MongoDB Docs: https://mongodb.com/docs
- Stripe API: https://stripe.com/docs
- Nodemailer: https://nodemailer.com
- PDFKit Docs: https://pdfkit.org

❖ Other Resources

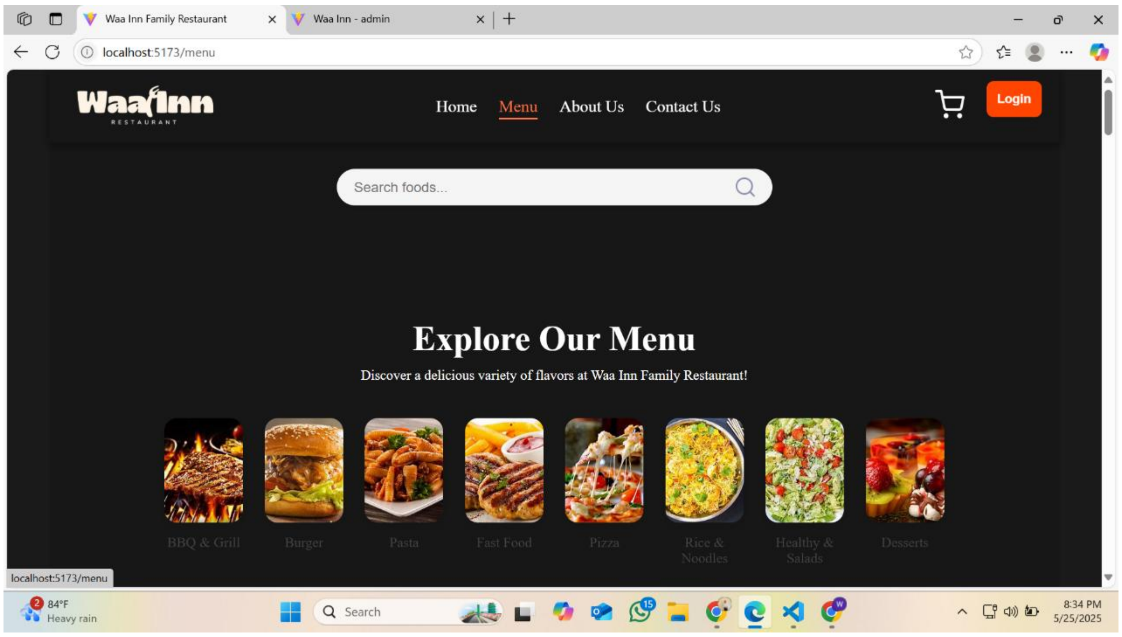
- OWASP Authentication Cheat Sheet — https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html
- Microsoft REST API Design Guidelines — https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design
- Lucidchart UML Diagram Tool — https://www.lucidchart.com
- Creately Diagram Tool — https://creately.com
- Visual Paradigm Online UML — https://online.visual-paradigm.com

## 10. Appendices

These include screenshots, sample invoices, and other data used during the project lifecycle.

- ❖ User Interfaces

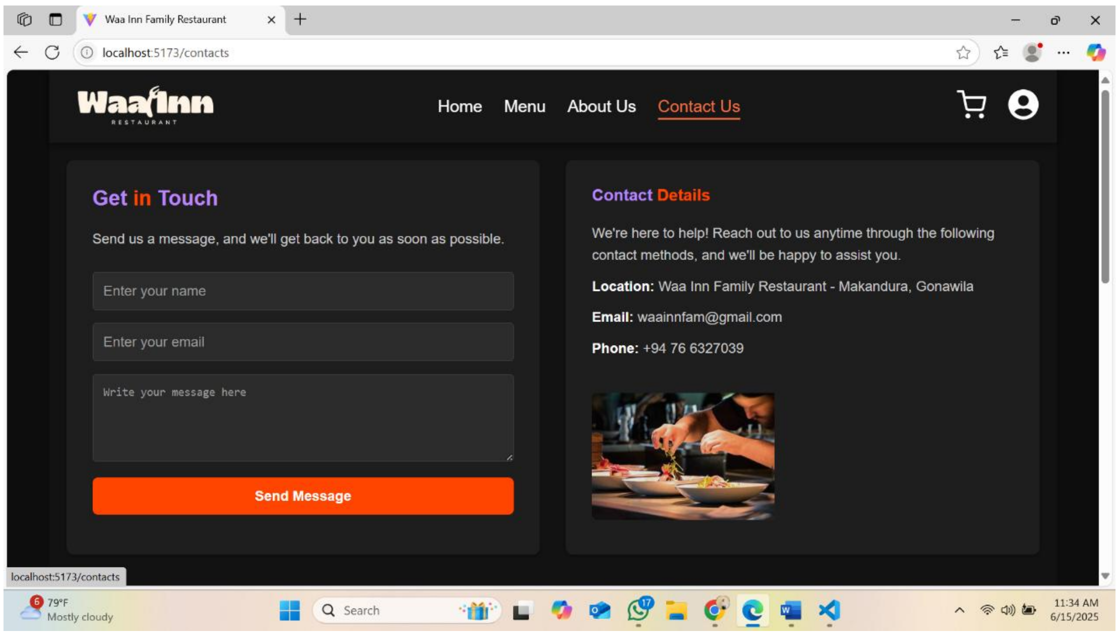
  Menu browsing (Customer)



Explore Our Menu

Discover a delicious variety of flavors at Waa Inn Family Restaurant!

Send Messages (Customer)



Get in Touch

Send us a message, and we'll get back to you as soon as possible.

Contact Details

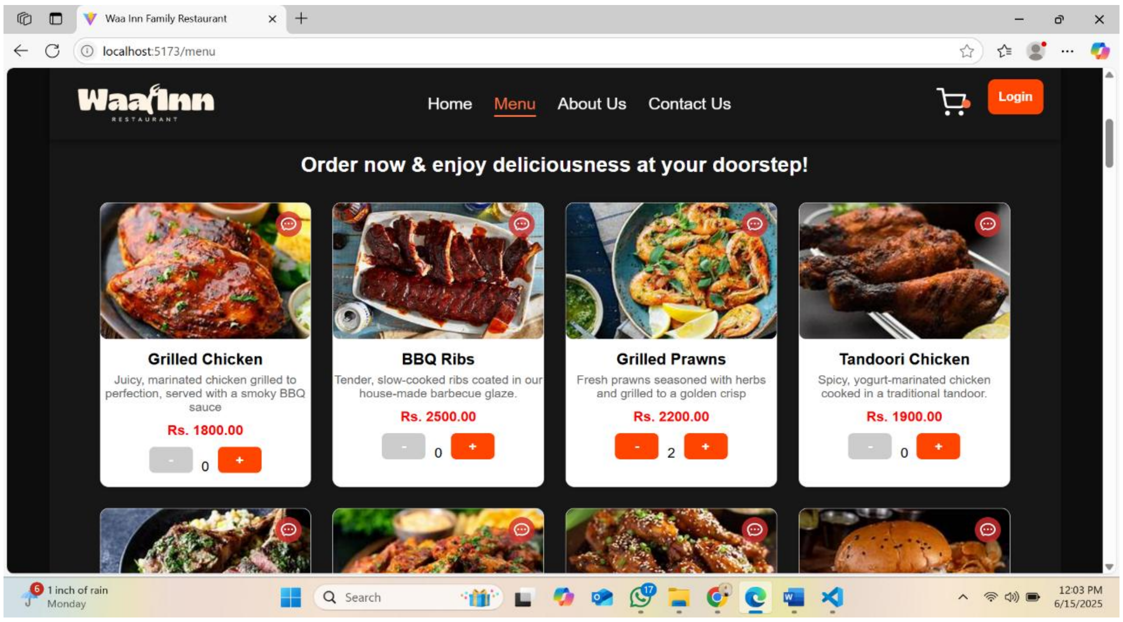
We're here to help! Reach out to us anytime through the following contact methods, and we'll be happy to assist you.

**Location:** Waa Inn Family Restaurant - Makandura, Gonawila

**Email:** waainnfam@gmail.com

**Phone:** +94 76 6327039

Add to Card Function (Customer)



Order now & enjoy deliciousness at your doorstep!

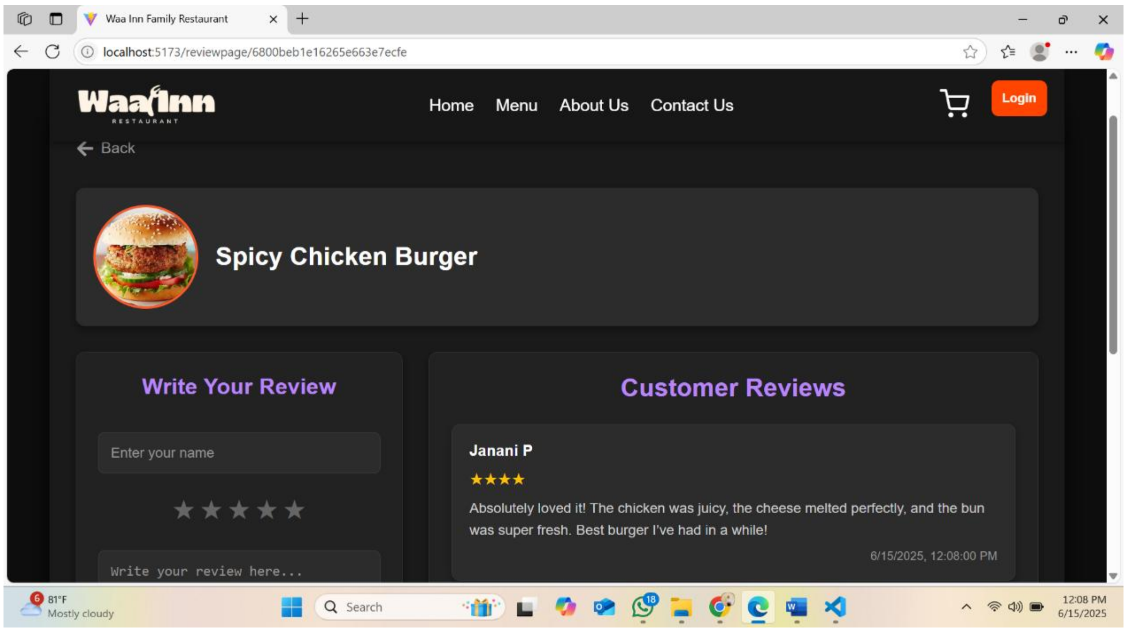
**Grilled Chicken**
Juicy, marinated chicken grilled to perfection, served with a smoky BBQ sauce
Rs. 1800.00
0

**BBQ Ribs**
Tender, slow-cooked ribs coated in our house-made barbecue glaze.
Rs. 2500.00
0

**Grilled Prawns**
Fresh prawns seasoned with herbs and grilled to a golden crisp
Rs. 2200.00
2

**Tandoori Chicken**
Spicy, yogurt-marinated chicken cooked in a traditional tandoor.
Rs. 1900.00
0

Review submission (Customer)


Waa Inn Family Restaurant
localhost:5173/reviewpage/6800beb1e16265e663e7ecfe
Home
Menu
About Us
Contact Us
Login
Back

**Spicy Chicken Burger**

**Write Your Review**

Enter your name

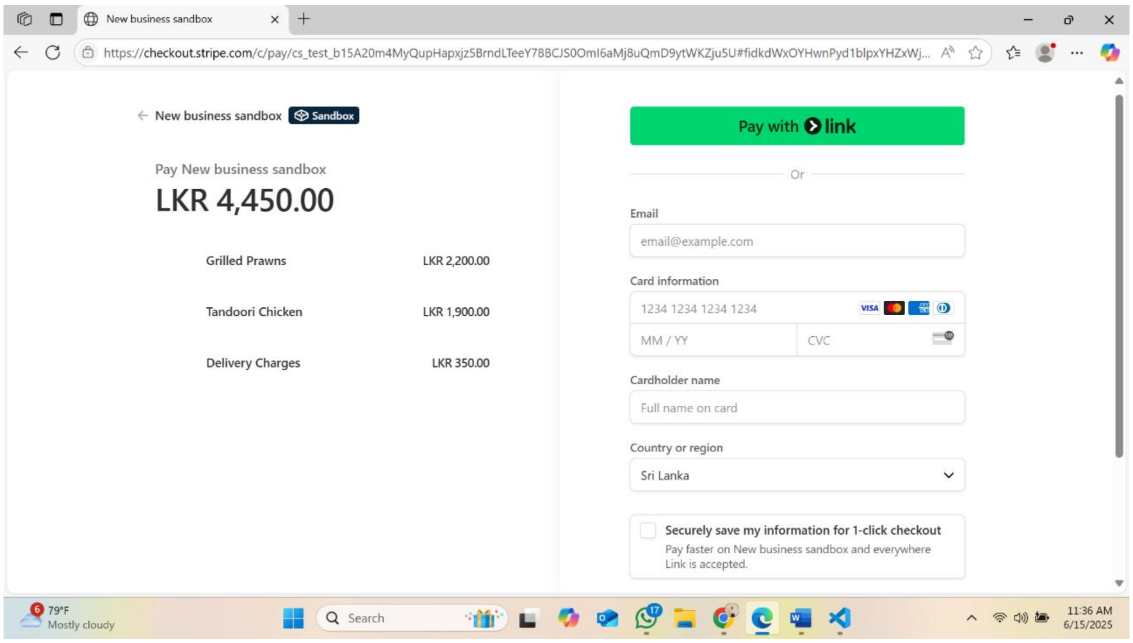
★★★★★

Write your review here...

**Customer Reviews**

Janani P
★★★★
Absolutely loved it! The chicken was juicy, the cheese melted perfectly, and the bun was super fresh. Best burger I've had in a while!
6/15/2025, 12:08:00 PM

Stripe online payment Interface (Customer)



New business sandbox · Sandbox

Pay New business sandbox

LKR 4,450.00

| | |
|---|---|
| Grilled Prawns | LKR 2,200.00 |
| Tandoori Chicken | LKR 1,900.00 |
| Delivery Charges | LKR 350.00 |

Email: email@example.com

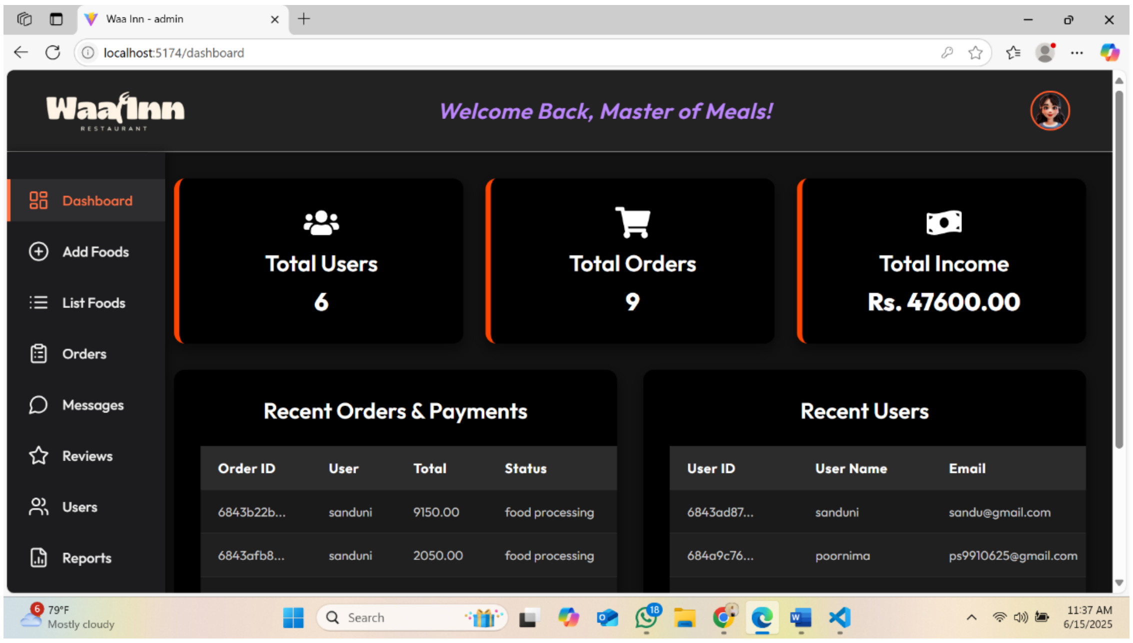
Card information: 1234 1234 1234 1234, MM / YY, CVC

Cardholder name: Full name on card

Country or region: Sri Lanka

Securely save my information for 1-click checkout
Pay faster on New business sandbox and everywhere Link is accepted.

Admin dashboard (real-time cards, tables)



Welcome Back, Master of Meals!

Total Users 6

Total Orders 9

Total Income Rs. 47600.00

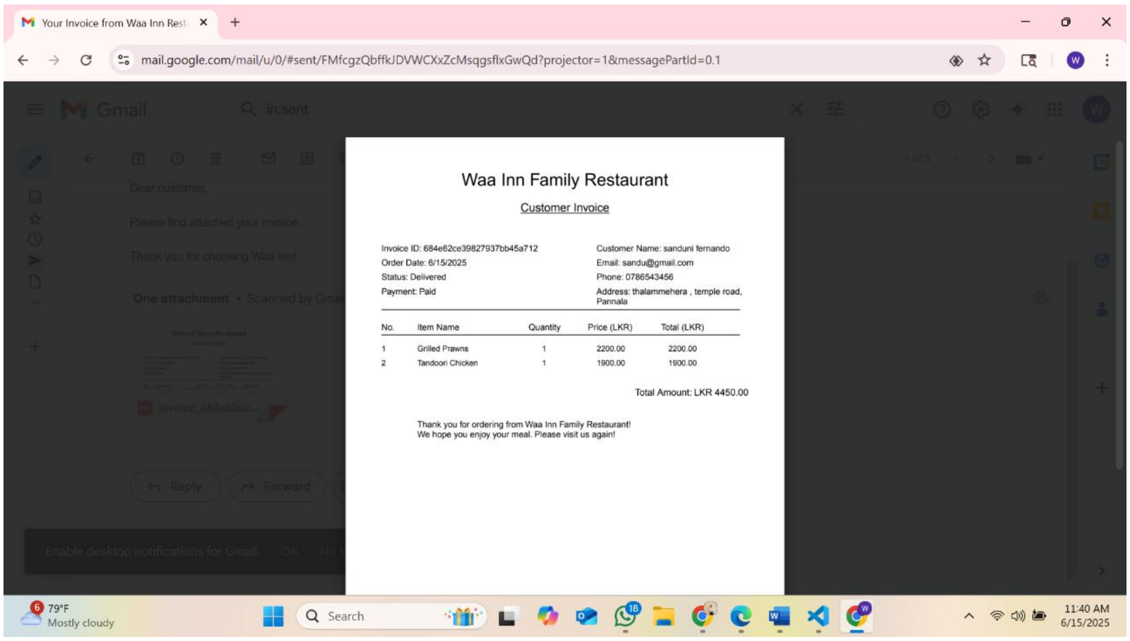
Recent Orders & Payments

| Order ID | User | Total | Status |
|---|---|---|---|
| 6843b22b... | sanduni | 9150.00 | food processing |
| 6843afb8... | sanduni | 2050.00 | food processing |

Recent Users

| User ID | User Name | Email |
|---|---|---|
| 6843ad87... | sanduni | sandu@gmail.com |
| 684a9c76... | poornima | ps9910625@gmail.com |

❖ Invoice Output


Waa Inn Family Restaurant

Customer Invoice

Invoice ID: 684e62ce39827937bb45a712
Order Date: 6/15/2025
Status: Delivered
Payment: Paid

Customer Name: sanduni fernando
Email: sandu@gmail.com
Phone: 0786543456
Address: thalammehera , temple road, Pannala

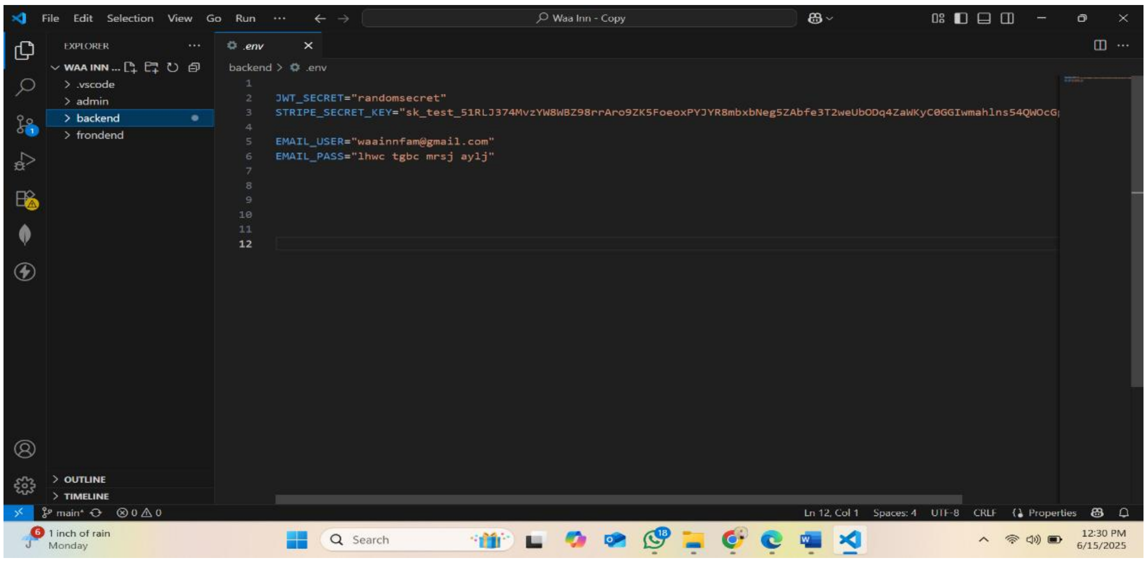
| No. | Item Name | Quantity | Price (LKR) | Total (LKR) |
|---|---|---|---|---|
| 1 | Grilled Prawns | 1 | 2200.00 | 2200.00 |
| 2 | Tandoori Chicken | 1 | 1900.00 | 1900.00 |

Total Amount: LKR 4450.00

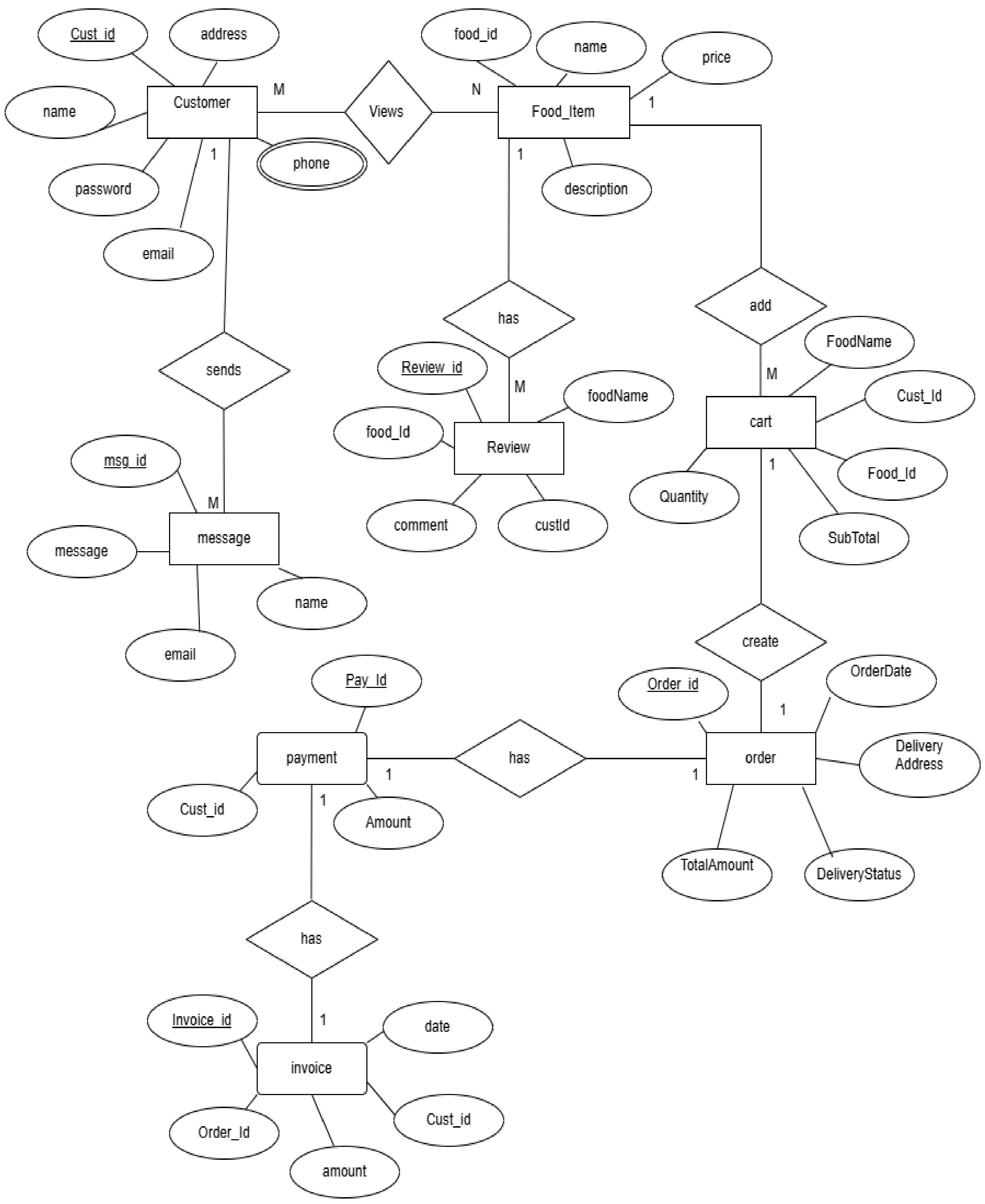
Thank you for ordering from Waa Inn Family Restaurant!
We hope you enjoy your meal. Please visit us again!

❖ Environment Configuration (.env File)

Environment variables are stored securely in .env files to avoid exposing sensitive data like API keys or database credentials. These files are not pushed to GitHub and are only used during development and deployment.


```

JWT_SECRET="randomsecret"
STRIPE_SECRET_KEY="sk_test_51RLJ374MvzYW8WBZ98rrAro9ZK5FoeoxPYJYR8mbxbNeg5ZAbfe3T2weUbODq4ZaWKyC0GGIwmahlns54QWOcG

EMAIL_USER="waainnfam@gmail.com"
EMAIL_PASS="lhwc tgbc mrsj aylj"
```

❖ Entity Relationship Diagrams



## ❖ Appendix I

1 User JWT Token Generation

```
// Function to create JWT token
const createToken = (id) => {
  return jwt.sign({ id }, process.env.JWT_SECRET, { expiresIn: '1h' });
};
```

2 Place Order Logic

```
const placeOrder = async (event) => {
    event.preventDefault();
    const orderItems = foodList
      .filter(item => cartItems[item._id] > 0)
      .map(item => ({
        ...item,
        quantity: cartItems[item._id]
      }));
    const userId = localStorage.getItem("userId");
    if (!userId) {
      alert("User ID not found. Redirecting to login.");
      navigate('/login');
      return;
    }
    const totalAmount = getTotalCartAmount() + 350;
    const orderData = {
      userId,
      address: data,
      items: orderItems,
      amount: totalAmount,
    };
```

## 3 Invoice PDF Generation

```
export const generateInvoice = async (req, res) => {
  const { orderId } = req.body;
  try {
    console.log("Generating invoice for order ID:", orderId);
    const order = await Order.findById(orderId);
    if (!order) {
      return res.status(404).json({ success: false, message: 'Order not found' });
    }
    const filename = `invoice_${order._id}.pdf`;
    const invoicesDir = path.join('invoices');
    const filepath = path.join(invoicesDir, filename);
    fs.mkdirSync(invoicesDir, { recursive: true });
    const doc = new PDFDocument({ margin: 50 });
    const writeStream = fs.createWriteStream(filepath);
    doc.pipe(writeStream);
    // Header
    doc
      .fontSize(24)
      .text('Waa Inn Family Restaurant', { align: 'center' })
      .moveDown(0.5);
    doc
      .fontSize(16)
      .text('Customer Invoice', { align: 'center', underline: true })
      .moveDown(1);
    // Invoice Info (Left) + Customer Info (Right)
    doc
      .fontSize(12)
      .text(`Invoice ID: ${order._id}`, 50, 150)
      .text(`Order Date: ${new Date(order.date).toLocaleDateString()}`, 50, 170)
```

4 Send Invoice via Email

```
const mailOptions = {
        from: process.env.EMAIL_USER,
        to: order.address.email,
        subject: 'Your Invoice from Waa Inn Restaurant',
        text: 'Dear customer,\n\nPlease find attached your invoice.\n\nThank you for choosing Waa Inn!',
        attachments: [{ filename, path: filepath }]
      };


      await transporter.sendMail(mailOptions);
      console.log(`Invoice sent to ${order.address.email}`);
      return res.status(200).json({ success: true, message: 'Invoice generated and sent successfully.' });


    } catch (emailError) {
      console.error("Email sending failed:", emailError);
      return res.status(500).json({ success: false, message: 'Invoice generated but failed to send email' });
    }
  });
```