Assignment Report

Web Engineering

Course Code: CSE415

Submitted To- Engr. Nishat Sadaf Lira

Submitted By-

Lulu Marjan Dina	221-15-5218
Rabina Akhter	221-15-5506
Kanij Fatema	221-15-4884
Nur Mohammad Hridoy	221-15-5952

1. Introduction

Project Name: DineEase – Smart Restaurant Ordering System

Project Type: Web-based food ordering and restaurant management system

Purpose: To streamline the restaurant experience by allowing customers to order food online and enabling admins to manage orders and menus efficiently.

2. Overall Description

DineEase is a standalone web application that provides both customer-facing and admin interfaces.

2.1 Product Perspective:

DineEase is a web-based restaurant management system designed to simplify food ordering and menu management. It allows customers to explore, filter, and order food online with integrated payment options. The system also enables administrators to manage menu items and handle order approvals efficiently. The primary goal of this project is to create a dynamic and user-friendly platform that improves the overall restaurant experience for both customers and restaurant staff. It aims to reduce manual order handling, waiting time, and administrative workload through an interactive web interface.

2.2 Target Users:

- ❖ Customers: Order food, track delivery, make payments
- ❖ Admins: Manage orders, menus, and customer queries

2.3 Product Functions:

- ❖ User registration/login
- ❖ Menu browsing & filtering
- ❖ Add to cart & checkout
- ❖ Payment simulation
- ❖ Order status tracking
- ❖ Admin dashboard for order/menu/user management

3. Engineering Requirements

The requirements for a system are the descriptions of what the system should do, the services that it provides and the constraints on its operation. Requirements should describe what the system should do, but not how it should do it. The process of finding out, analyzing, documenting and checking these services and constraints is called requirements engineering (RE).

Three kinds of requirements based on the intended purpose and target audience: -

1. User Requirements

- Users should be able to browse the food menu.
- Users must have options to filter food items.
- Users can place orders and make online payments securely.

- Users should be able to track their order status in real-time.
- Admins should be able to add/update/delete food items from the menu.
- Admins must be able to accept or reject orders and manage user accounts.

2. System Requirements

System requirements define the capabilities and constraints that a system must possess to satisfy stakeholder needs and fulfill its intended purpose. They are a critical part of systems engineering, outlining what the system must do (functionality) and how it should perform (non-functional aspects like performance, security, etc.). These requirements act as a bridge between business needs and the technical design of the software.

Functional Requirements:

- The system shall allow users to register and log in.
- The system shall allow users to add food items to the cart.
- The system shall process payments via a simulated payment gateway.
- The system shall show the current status of the user's order.
- The admin panel shall provide CRUD operations on food items.
- The system shall send notifications on order updates.

Non-Functional Requirements:

- Usability: The interface should be intuitive and responsive.
- Performance: The system should handle at least 50 concurrent users.
- Security: All user data must be protected; passwords must be encrypted.
- Portability: System must run in Chrome, Firefox, and Safari.
- Maintainability: Code must follow modular structure using MVC.
- Scalability: Designed to allow adding features like delivery tracking.

3. Domain Requirements

- The system must comply with basic food industry digital standards.
- Must support categorization based on cuisine, food type, and offers.

4. Requirement Validation

Create a Google Form to collect feedback from users-**LINK**

5. User Interfaces

- ❖ Customer Dashboard
- ❖ Admin Panel
- ❖ Menu Page
- ❖ Cart Page
- ❖ Order Tracker

6. Technology Stack

Technologies used in DineEase System-

- ❖ Frontend: HTML, CSS, JavaScript
- ❖ Backend: PHP

- ❖ Database: MySQL
- ❖ Hosting: Localhost or free hosting (e.g., 000WebHost)

7. Web Project Management

Web Project Management refers to the structured approach of planning, organizing, and executing a web-based software project to meet specific goals within a defined timeline and scope. It involves coordinating team members, defining tasks, allocating resources, and monitoring progress.

For DineEase – Smart Restaurant Ordering System, the project was divided into clearly defined phases across 7 weeks. Each phase focused on specific objectives, allowing for smooth development, integration, and testing.

7.1. Requirement Analysis

This phase involved gathering functional and non-functional requirements from real users using Google Forms. The form was distributed among target users (students, peers, restaurant visitors), and the feedback was used to validate and refine system features. This ensured the system was user-centered and practically relevant.

Here is some user requirements analysis-

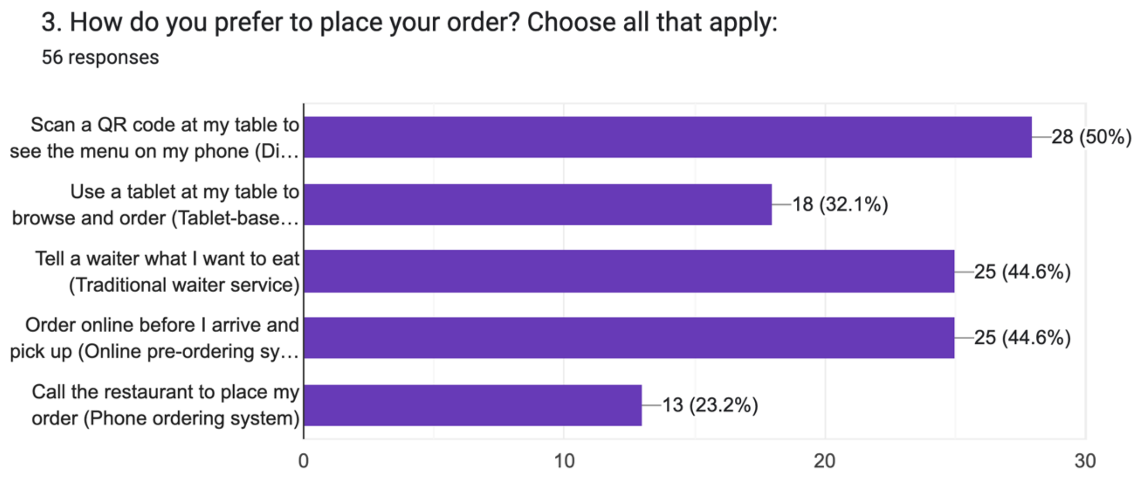


Figure 1. Visual Representation of Question 3

The survey results show that the most preferred way to place an order is by scanning a QR code at the table to view the menu on the phone, chosen by **50%** of respondents. Other popular methods include using a tablet at the table (**32.1%**), traditional waiter service (**44.6%**), and online pre-ordering for pickup (**44.6%**). Calling the restaurant to place an order via phone is the least preferred option (**23.2%**).

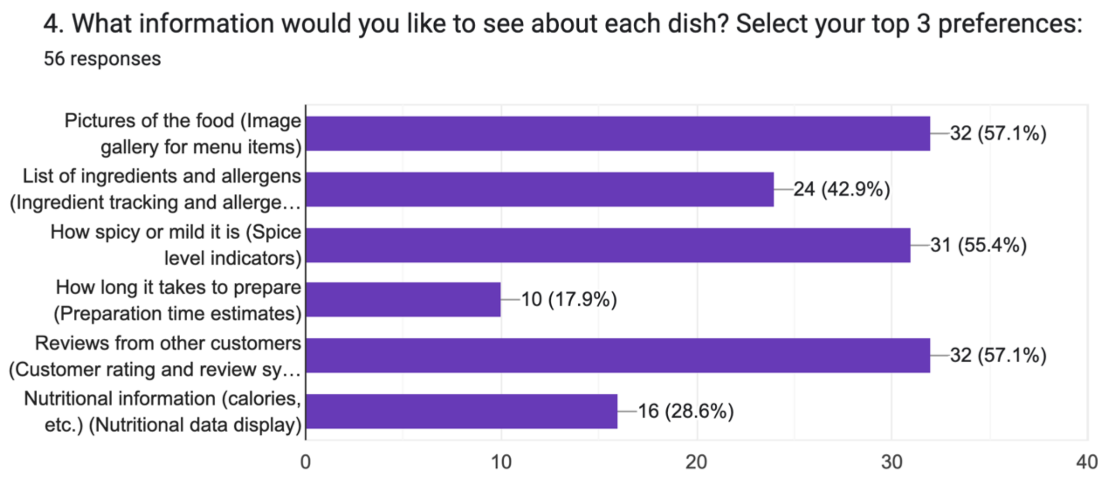


Figure 2. Visual Representation of Question 4

The survey shows that the most popular preferences for dish information are pictures of the food (**57.1%**), reviews from other customers (**57.1%**), and spice level indicators (**55.4%**). Additionally, **42.9%** of respondents prefer to see a list of ingredients and allergens, while **28.6%** are interested in nutritional information such as calories.

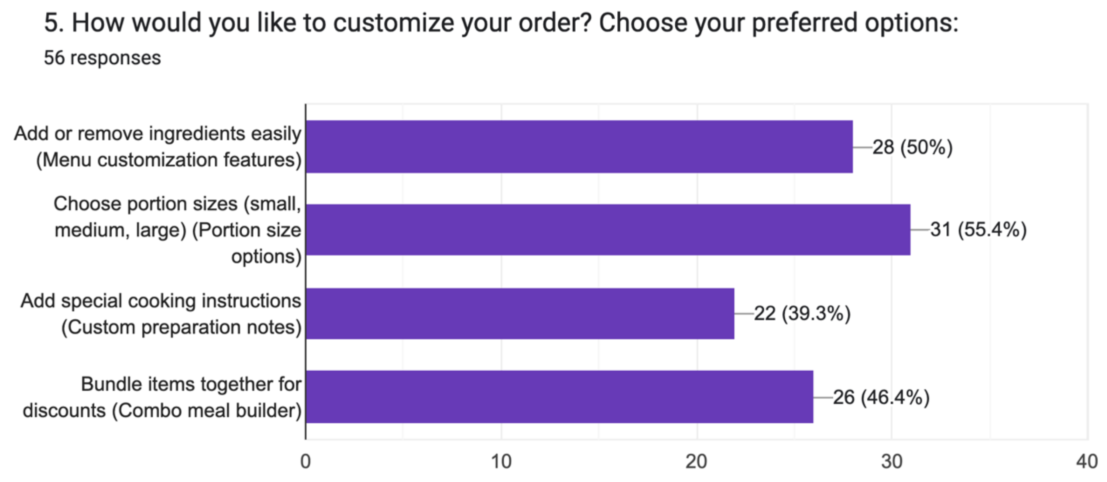


Figure 3. Visual Representation of Question 5

The survey results indicate that the most preferred options for customizing orders are choosing portion sizes (**55.4%**) and adding or removing ingredients (**50%**). Additionally, **46.4%** of respondents favor bundling items for discounts, while **39.3%** prefer adding special cooking instructions for custom preparation.

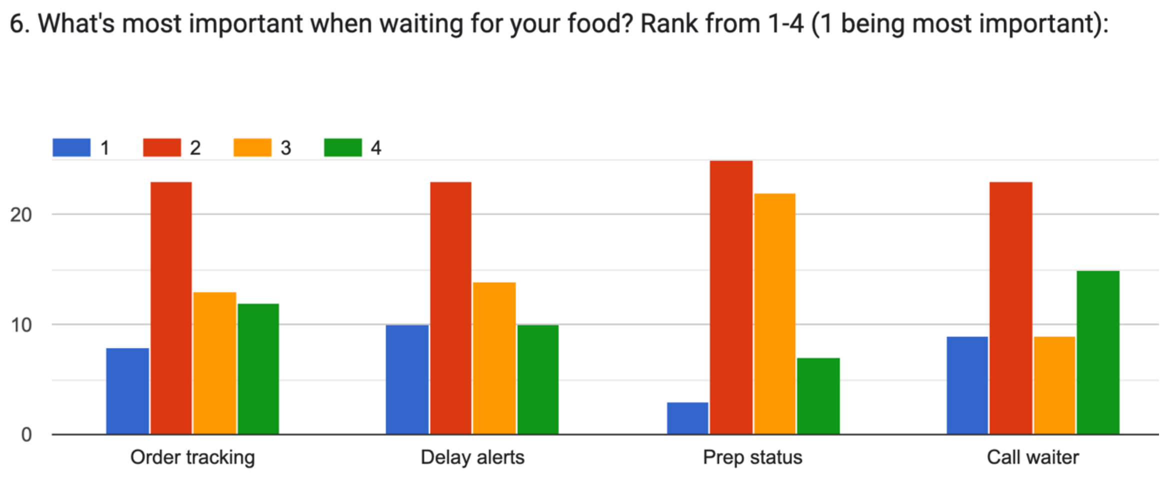


Figure 4. Visual Representation of Question 6

The survey results reveal that **"Order tracking"** is the most important factor when waiting for food, with the majority ranking it as the top priority (**rank 1**). **"Prep status"** follows closely in importance, while **"Delay alerts"** and **"Call waiter"** are ranked lower, with fewer respondents considering them as the highest priority.

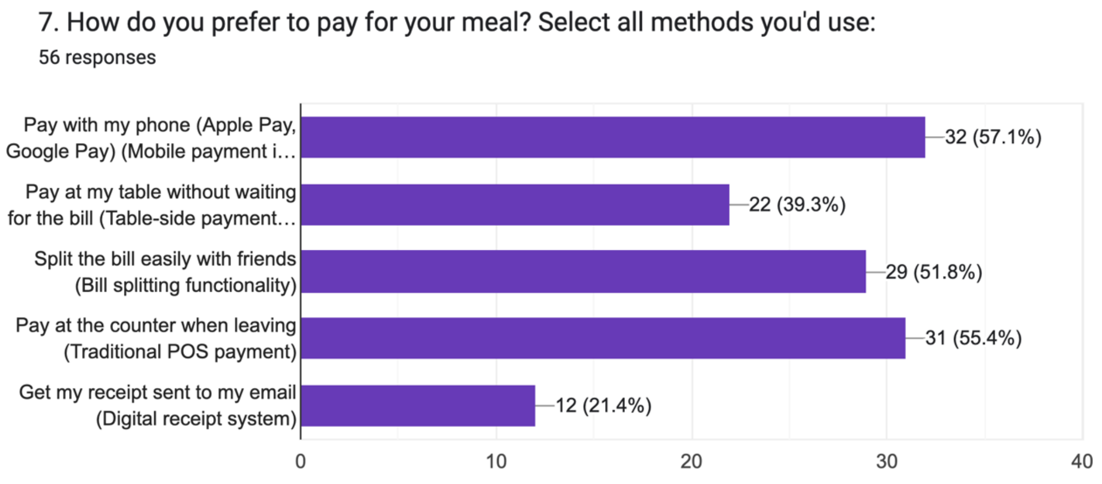


Figure 5. Visual Representation of Question 7

The survey results show that the most preferred payment method is using a phone for mobile payment (**57.1%**), followed by paying at the table without waiting for the bill (**39.3%**) and splitting the bill easily with friends (**51.8%**). A smaller number of respondents prefer paying at the counter (**55.4%**) or receiving a digital receipt by email (**21.4%**).

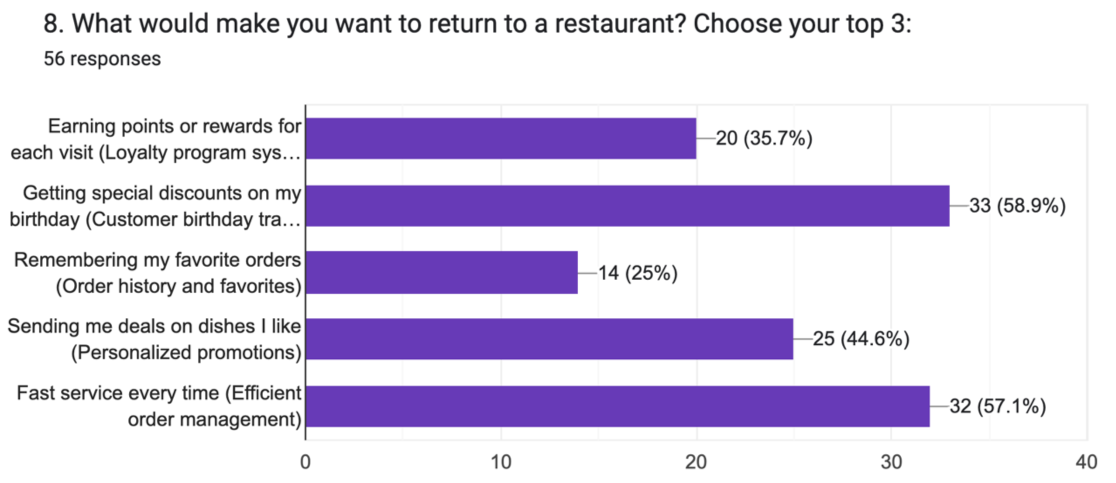


Figure 6. Visual Representation of Question 8

The survey results indicate that the top reasons for wanting to return to a restaurant are receiving special discounts on birthdays (**58.9%**) and earning loyalty points or rewards (**35.7%**). Other factors include receiving personalized promotions (**44.6%**) and fast service (**57.1%**), while remembering favorite orders was less important (**25%**).

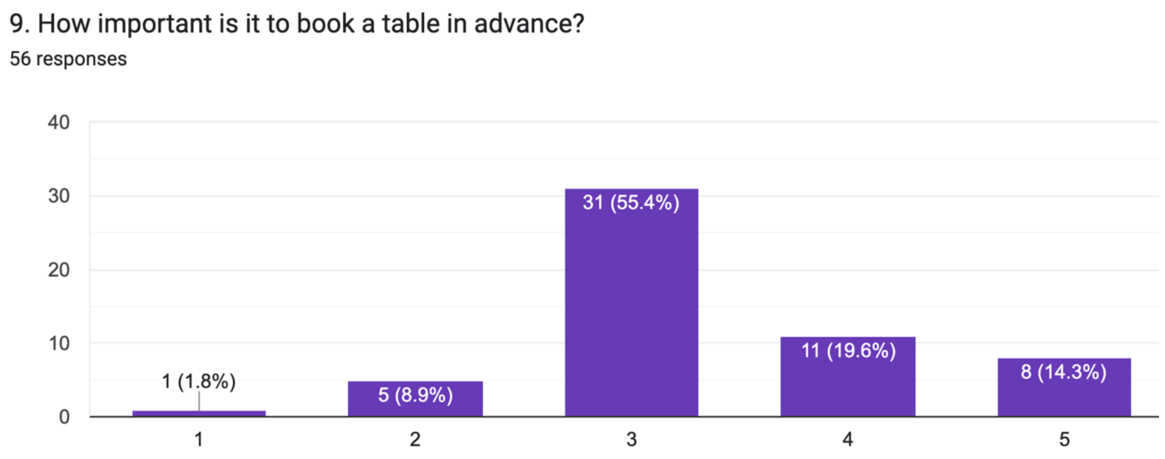


Figure 7. Visual Representation of Question 9

The survey results show that booking a table in advance is considered moderately important by most respondents, with **55.4%** ranking it as a **3**. Only a small percentage (**1.8%**) consider it unimportant (**rank 1**), while **19.6%** rate it as quite important (**rank 4**), and **14.3%** consider it highly important (**rank 5**).

7.2. Task Planning and Scheduling

Using a Gantt chart, the entire development cycle was divided into weekly tasks:

- Week 0-1: Requirement finalization and database design
- Week 2–3: Frontend and backend development
- Week 4–5: Admin panel, payment and order tracking
- Week 6–7: Testing, bug fixing, and final deployment

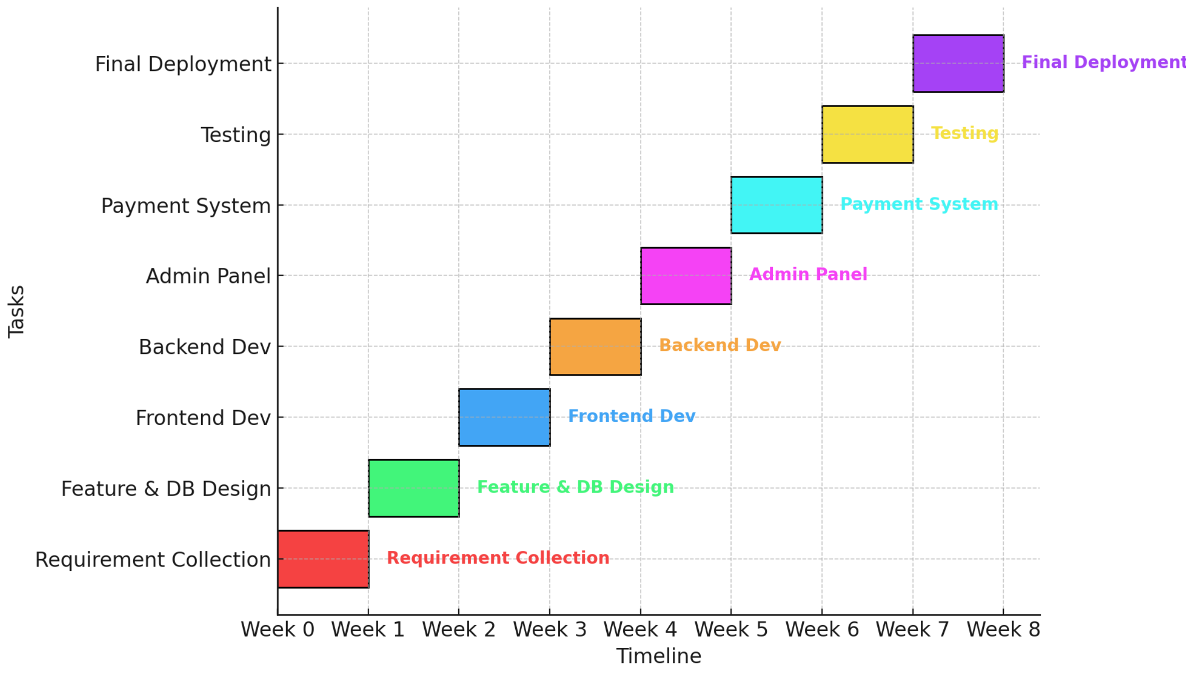


Figure 8. Project Gantt Chart

Each task was planned with clear deliverables and deadlines.

7.3. Design and Diagrams

Basic design and diagrams help simplify the development process by providing a clear visual representation of the system before actual coding begins. These preliminary models act as a blueprint and give both developers and stakeholders an overview of how the system will function.

Using visual diagrams early in the project allows us to identify potential usability issues, structural flaws, and logical gaps before they become costly to fix. It also improves communication among team members and ensures everyone has a shared understanding of the system.

For the DineEase project, the following design elements were created-

7.3.1. Use Case Diagram

Outlines the interaction between users (Customer and Admin) and the system's core functionalities.

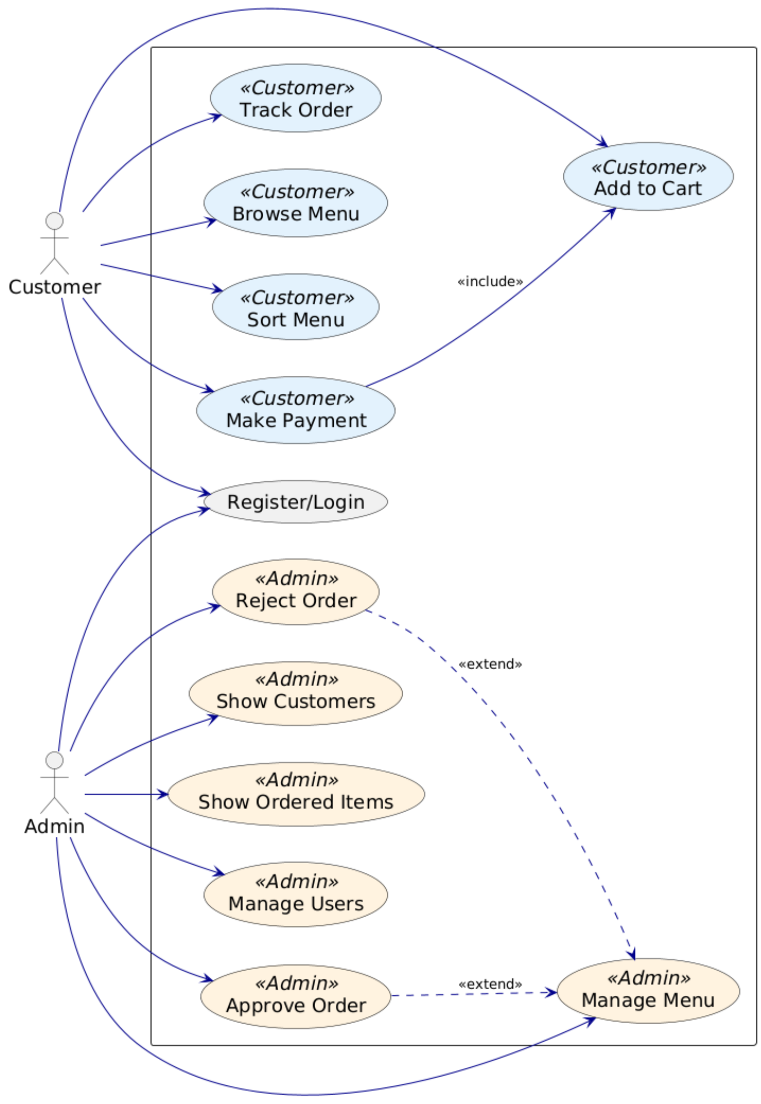


Figure 9. Use Case of DineEase System

The diagram represents the use case model for the DineEase system, illustrating interactions between customers and admins. Customers can browse the menu, add items to the cart, sort the menu, track orders, and make payments. Admins manage users, show customers, approve/reject orders, and handle the menu. Relationships like "include" and "extend" indicate dependencies and optional actions between use cases.

7.3.2. Activity Diagrams

Describes the step-by-step workflows for both customer and admin operations.

For Customer

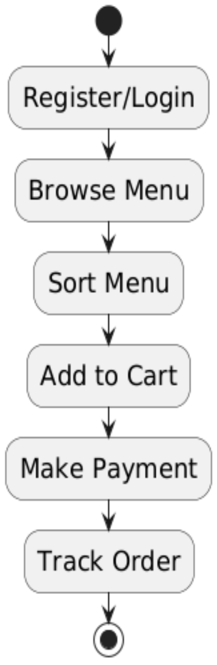


Figure 10. Customer Side Activity Diagram

For Admin

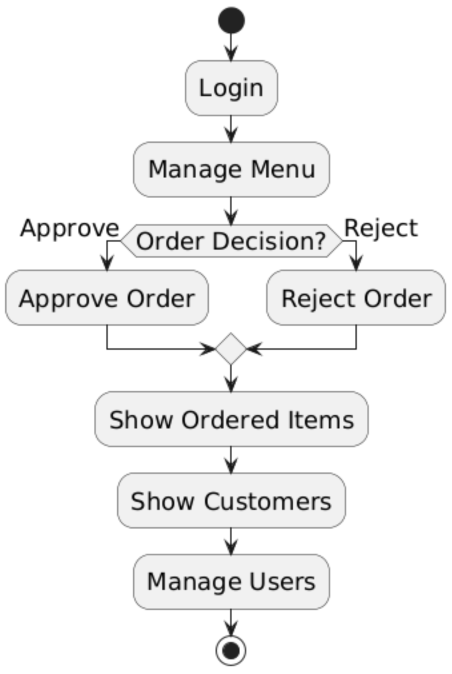


Figure 11. Admin Side Activity Diagram

The Customer begins by registering or logging into the system. They browse and sort the menu before adding items to their cart. After making a payment, they can track their order's status, completing their interaction with the DineEase system.

The Admin logs into the system and manages the menu. After reviewing an order, the admin decides whether to approve or reject it. Once the order decision is made, the admin can view ordered items, show customers, and manage users, ensuring the smooth operation of the system.

These activity diagrams provide a clear visualization of the core functionalities and responsibilities within the DineEase system. On the customer side, the diagram illustrates a streamlined, user-centric workflow—from registration to order tracking—ensuring a smooth and intuitive dining experience. On the admin side, the workflow emphasizes control and management, enabling admins to oversee menu items, handle orders through approval or rejection, and manage customers and users effectively. Together, these diagrams highlight the system's dual focus on user satisfaction and operational efficiency, ensuring seamless interaction between customers and administrators.

7.3.3. Sequence Diagram

Shows the flow of messages between the components (UI, Backend, Database, Payment Gateway, etc.) during a process. For Example- Suppose a Customer want to places an order.

Scenario: Customer places an order

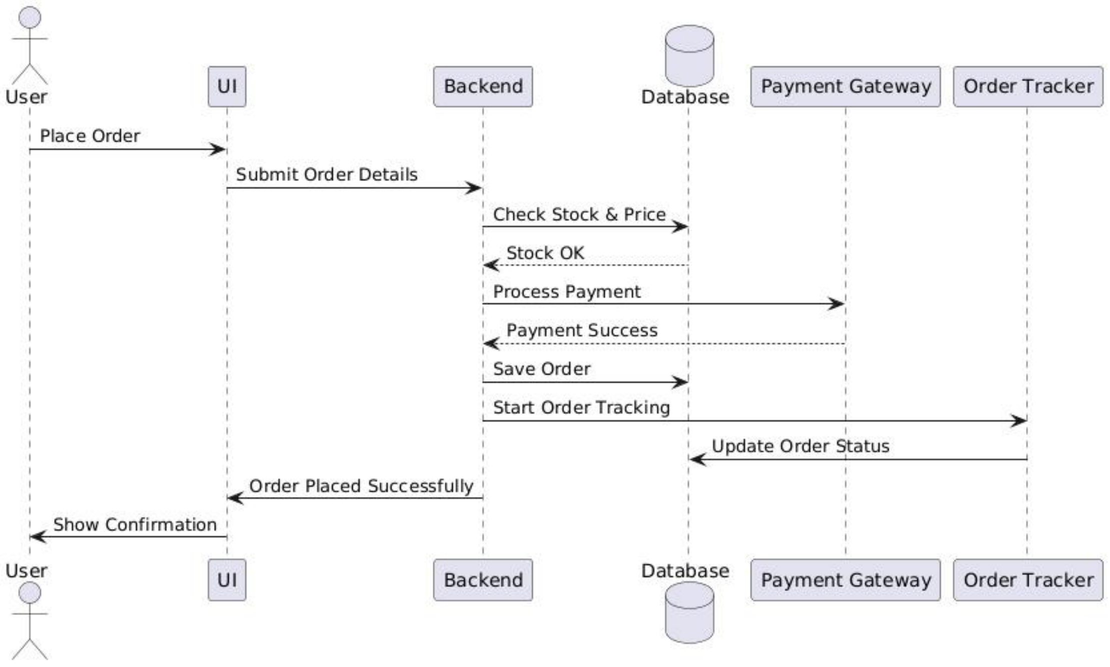


Figure 12. Sequence Diagram of Customer places an order

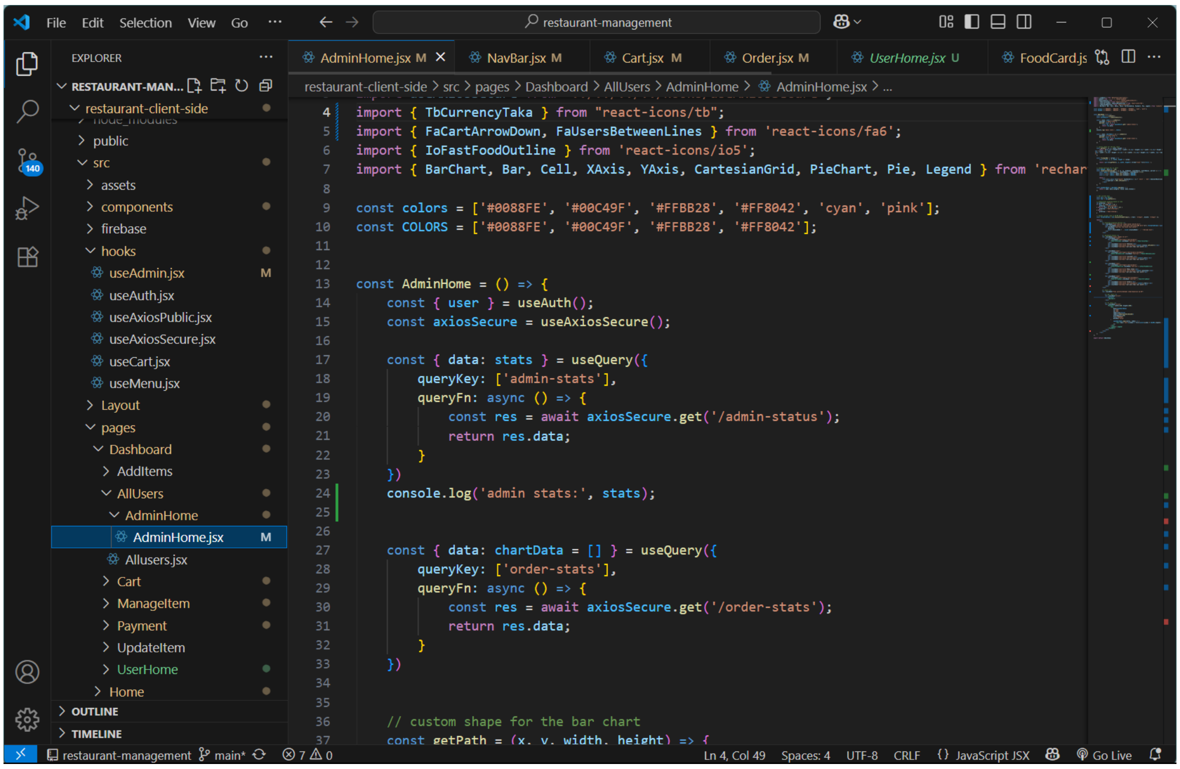
This sequence diagram illustrates the order placement process in an e-commerce system. The user places an order through the UI, which submits order details to the backend. The backend checks stock and pricing, processes payment via a payment gateway, saves the order, and starts order tracking. Finally, the system updates the order status and confirms the order placement to the user.

These diagrams not only made planning more efficient but also served as a guide during implementation and documentation.

7.4. Development and Version Control

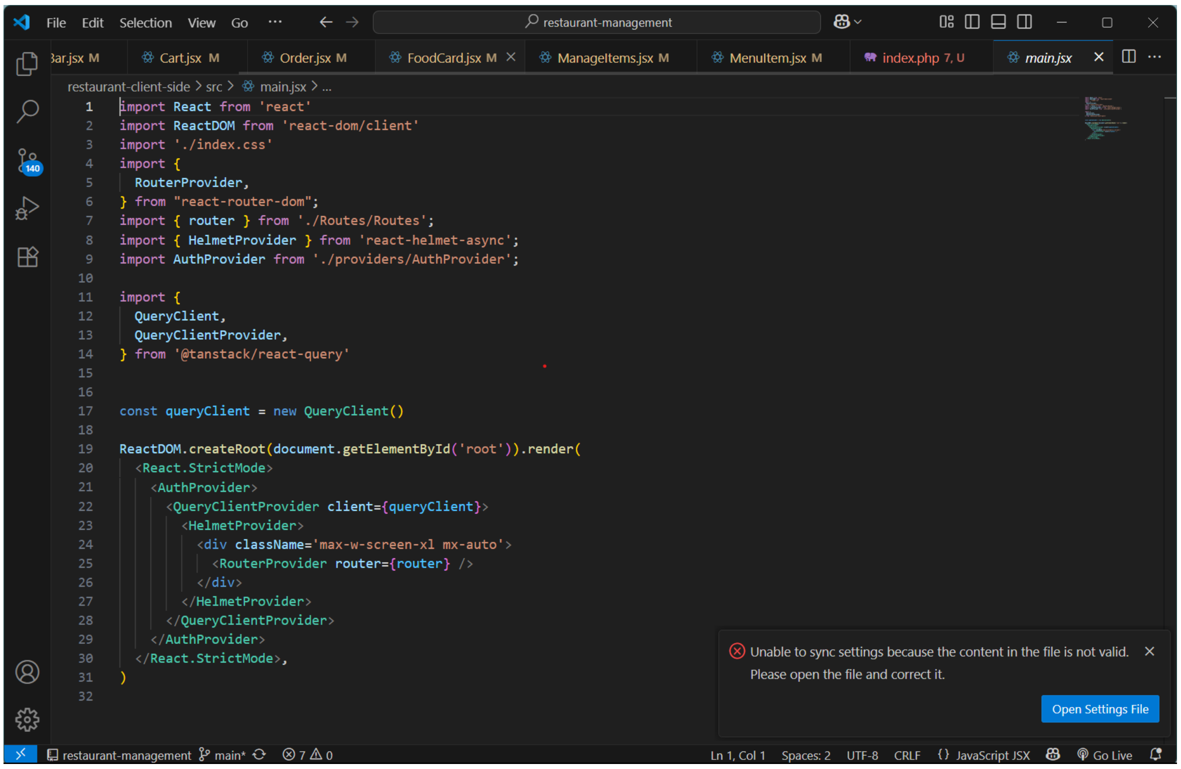
Development was done incrementally using the HTML/CSS/JS frontend and PHP-MySQL backend. All files and versions were stored in local folders and also, we have used GitHub for larger teams. This made it easier to manage changes and revert to previous versions if necessary.

Here are some **codes snippet** –


restaurant-management
restaurant-client-side > src > pages > Dashboard > AllUsers > AdminHome > AdminHome.jsx > ...
Ln 4, Col 49 Spaces: 4 UTF-8 CRLF JavaScript JSX Go Live

```
import { TbCurrencyTaka } from "react-icons/tb";
import { FaCartArrowDown, FaUsersBetweenLines } from 'react-icons/fa6';
import { IoFastFoodOutline } from 'react-icons/io5';
import { BarChart, Bar, Cell, XAxis, YAxis, CartesianGrid, PieChart, Pie, Legend } from 'rechar

const colors = ['#0088FE', '#00C49F', '#FFBB28', '#FF8042', 'cyan', 'pink'];
const COLORS = ['#0088FE', '#00C49F', '#FFBB28', '#FF8042'];


const AdminHome = () => {
    const { user } = useAuth();
    const axiosSecure = useAxiosSecure();

    const { data: stats } = useQuery({
        queryKey: ['admin-stats'],
        queryFn: async () => {
            const res = await axiosSecure.get('/admin-status');
            return res.data;
        }
    })
    console.log('admin stats:', stats);


    const { data: chartData = [] } = useQuery({
        queryKey: ['order-stats'],
        queryFn: async () => {
            const res = await axiosSecure.get('/order-stats');
            return res.data;
        }
    })


    // custom shape for the bar chart
    const getPath = (x, y, width, height) => {
```

Figure 13. Admin Dashboard

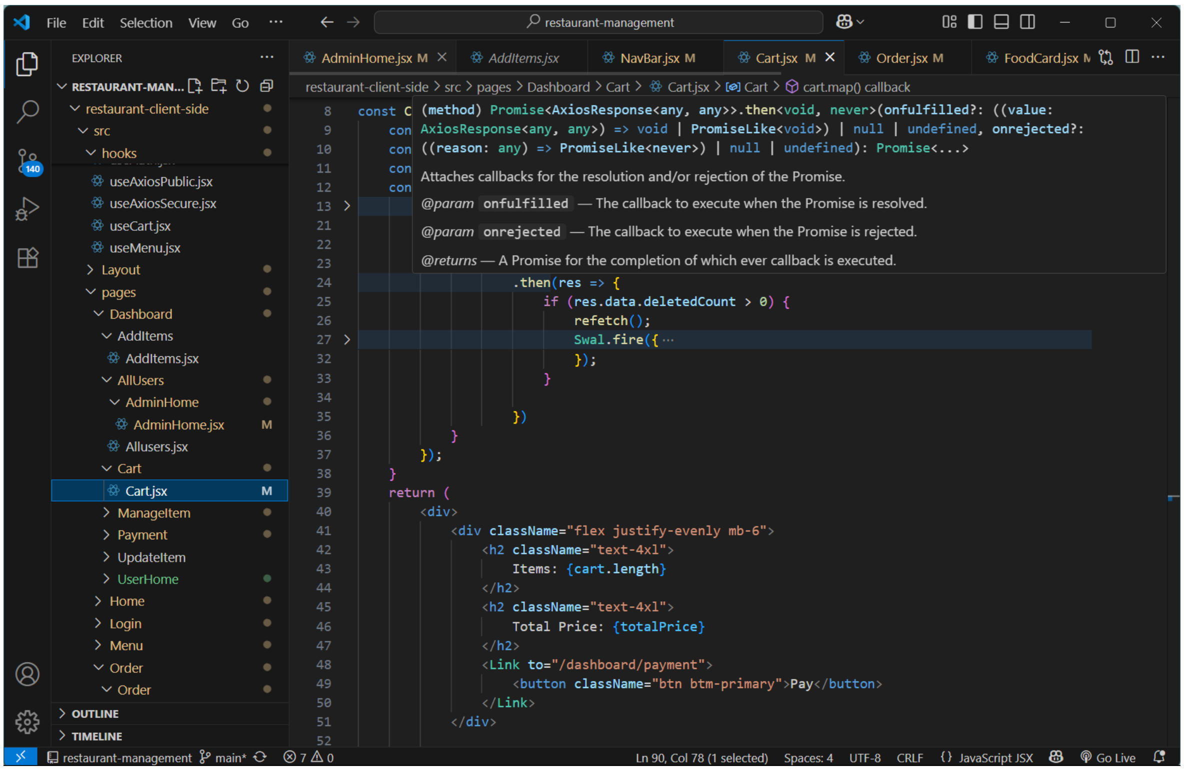

restaurant-management
restaurant-client-side > src > main.jsx > ...
Ln 1, Col 1 Spaces: 2 UTF-8 CRLF JavaScript JSX Go Live

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import './index.css'
import {
  RouterProvider,
} from "react-router-dom";
import { router } from './Routes/Routes';
import { HelmetProvider } from 'react-helmet-async';
import AuthProvider from './providers/AuthProvider';

import {
  QueryClient,
  QueryClientProvider,
} from '@tanstack/react-query'


const queryClient = new QueryClient()

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <AuthProvider>
      <QueryClientProvider client={queryClient}>
        <HelmetProvider>
          <div className='max-w-screen-xl mx-auto'>
            <RouterProvider router={router} />
          </div>
        </HelmetProvider>
      </QueryClientProvider>
    </AuthProvider>
  </React.StrictMode>,
)
```

Unable to sync settings because the content in the file is not valid. Please open the file and correct it.

Open Settings File

Figure 14. MVC Architecture



restaurant-client-side > src > pages > Dashboard > Cart > Cart.jsx > Cart > cart.map() callback

```
(method) Promise<AxiosResponse<any, any>>.then<void, never>(onfulfilled?: ((value:
AxiosResponse<any, any>) => void | PromiseLike<void>) | null | undefined, onrejected?:
((reason: any) => PromiseLike<never>) | null | undefined): Promise<...>
```

Attaches callbacks for the resolution and/or rejection of the Promise.

@param `onfulfilled` — The callback to execute when the Promise is resolved.

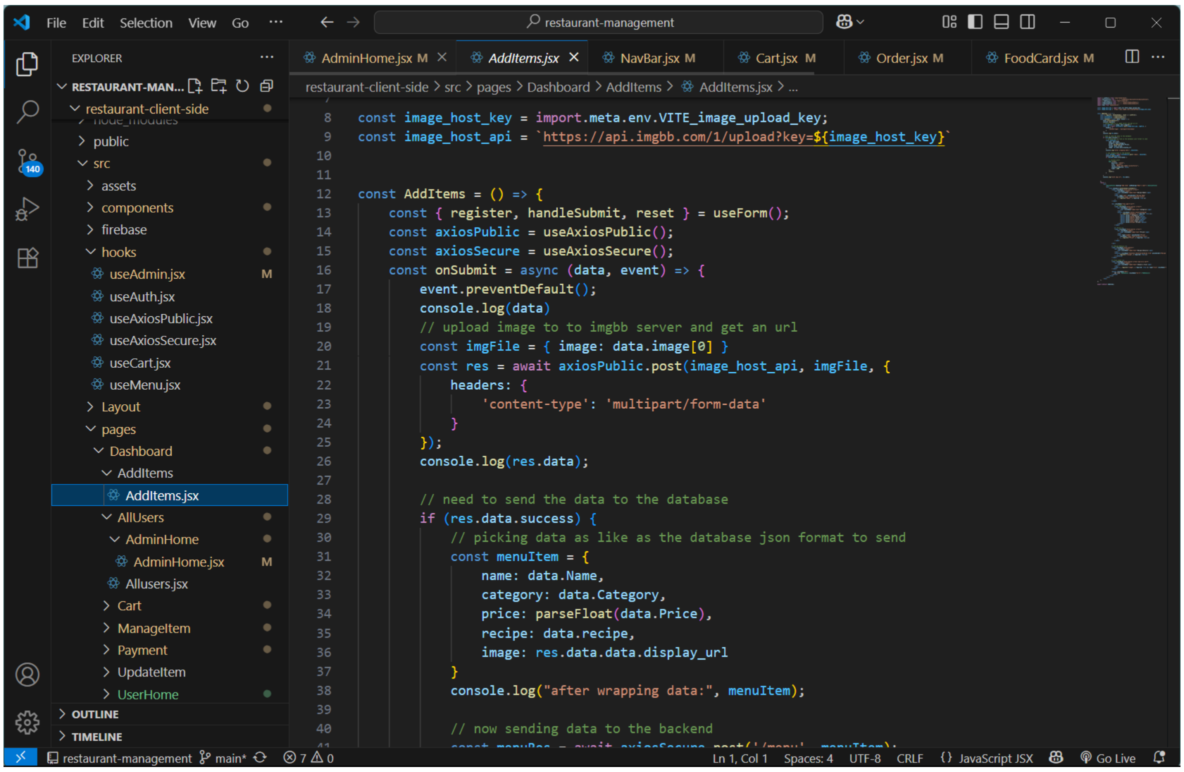
@param `onrejected` — The callback to execute when the Promise is rejected.

@returns — A Promise for the completion of which ever callback is executed.

```
const C
    con
    con
    con
    con

                    .then(res => {
                        if (res.data.deletedCount > 0) {
                            refetch();
                            Swal.fire({ ···
                            });
                        }

                    })
            }
        });
    }
    return (
        <div>
            <div className="flex justify-evenly mb-6">
                <h2 className="text-4xl">
                    Items: {cart.length}
                </h2>
                <h2 className="text-4xl">
                    Total Price: {totalPrice}
                </h2>
                <Link to="/dashboard/payment">
                    <button className="btn btm-primary">Pay</button>
                </Link>
            </div>
```

Ln 90, Col 78 (1 selected) Spaces: 4 UTF-8 CRLF JavaScript JSX

Figure 15. Add to Cart

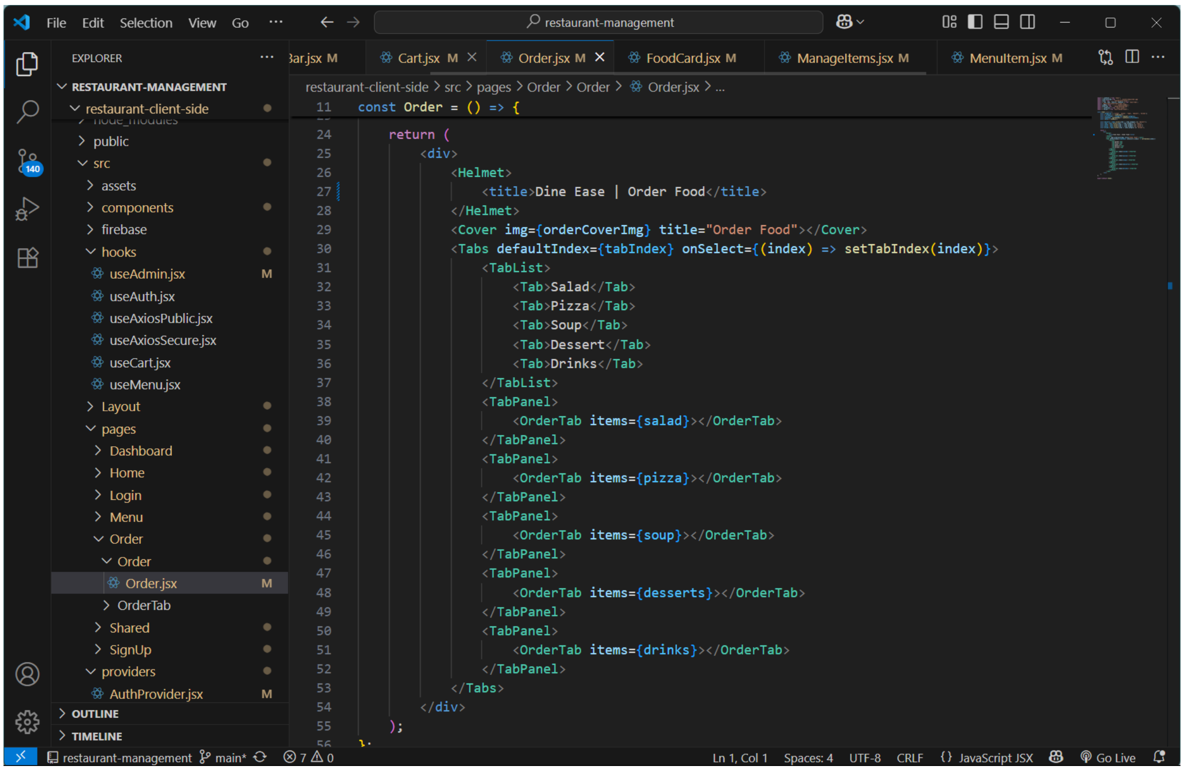


restaurant-client-side > src > pages > Dashboard > AddItems > AddItems.jsx > ...

```
const image_host_key = import.meta.env.VITE_image_upload_key;
const image_host_api = `https://api.imgbb.com/1/upload?key=${image_host_key}`


const AddItems = () => {
    const { register, handleSubmit, reset } = useForm();
    const axiosPublic = useAxiosPublic();
    const axiosSecure = useAxiosSecure();
    const onSubmit = async (data, event) => {
        event.preventDefault();
        console.log(data)
        // upload image to to imgbb server and get an url
        const imgFile = { image: data.image[0] }
        const res = await axiosPublic.post(image_host_api, imgFile, {
            headers: {
                'content-type': 'multipart/form-data'
            }
        });
        console.log(res.data);

        // need to send the data to the database
        if (res.data.success) {
            // picking data as like as the database json format to send
            const menuItem = {
                name: data.Name,
                category: data.Category,
                price: parseFloat(data.Price),
                recipe: data.recipe,
                image: res.data.data.display_url
            }
            console.log("after wrapping data:", menuItem);

            // now sending data to the backend
```

Ln 1, Col 1 Spaces: 4 UTF-8 CRLF JavaScript JSX

Figure 16. Add Items by Admin Side



```jsx
const Order = () => {
    return (
        <div>
            <Helmet>
                <title>Dine Ease | Order Food</title>
            </Helmet>
            <Cover img={orderCoverImg} title="Order Food"></Cover>
            <Tabs defaultIndex={tabIndex} onSelect={(index) => setTabIndex(index)}>
                <TabList>
                    <Tab>Salad</Tab>
                    <Tab>Pizza</Tab>
                    <Tab>Soup</Tab>
                    <Tab>Dessert</Tab>
                    <Tab>Drinks</Tab>
                </TabList>
                <TabPanel>
                    <OrderTab items={salad}></OrderTab>
                </TabPanel>
                <TabPanel>
                    <OrderTab items={pizza}></OrderTab>
                </TabPanel>
                <TabPanel>
                    <OrderTab items={soup}></OrderTab>
                </TabPanel>
                <TabPanel>
                    <OrderTab items={desserts}></OrderTab>
                </TabPanel>
                <TabPanel>
                    <OrderTab items={drinks}></OrderTab>
                </TabPanel>
            </Tabs>
        </div>
    );
```

Figure 17. Order Page

7.5. Team Collaboration

The DineEase project was developed by a team of four members, making effective team collaboration essential throughout the development cycle. Since multiple tasks such as designing, coding, testing, and documentation had to be handled in parallel, smooth coordination among team members played a key role in ensuring timely and quality delivery. To manage our workflow and stay organized, we used several collaboration tools-

- ❖ **Google Docs** - Used for co-editing documents like the Software Requirements Specification (SRS), report writing, and presentation drafts.
- ❖ **Google Drive** - Served as a central repository for storing assets such as UI mockups, images, diagrams, and shared resources.
- ❖ **Git (GitHub)** - Helped us collaborate on the codebase, track changes, and manage version control, especially when working on different modules like frontend and backend.
- ❖ **Group Chat (Telegram)** - Used for day-to-day communication, quick updates, and task coordination.

Regular check-ins and task updates were shared within the group to ensure alignment and progress tracking. This collaborative approach improved the project's efficiency, helped resolve issues quickly, and maintained consistency in both design and development.

7.6. Testing and Feedback Integration

Throughout the development of DineEase, testing and user feedback played a crucial role in improving system functionality and user experience. At the end of each development milestone (typically at the end of each week), we conducted manual testing and informal peer reviews to validate the features completed during that phase.

Key functional areas tested included:

- User authentication (registration/login)
- Cart operations (add, update, remove items)
- Order placement and approval
- Admin menu management
- UI responsiveness across devices and browsers

We also shared early builds with classmates and team members to gather initial user impressions. Their feedback helped us identify minor bugs, usability concerns, and layout inconsistencies.

Changes were incorporated immediately after testing to avoid the accumulation of issues and to ensure continuous improvement. This iterative feedback loop helped keep the project aligned with user expectations and ensured that each feature was stable before moving on to the next.

8. Testing Tools

Testing tools help ensure that a web application functions as expected. They are essential for identifying bugs, verifying functionality, and enhancing both performance and security. In a project like DineEase, testing plays a critical role in maintaining a smooth user experience—ensuring that customers can browse the menu, place orders, and make payments without issues, while admins can effectively manage orders and menu items. Using appropriate testing tools helps detect problems early, reduces deployment risks, and improves the overall reliability of the system.

Table 1. Use for Testing Tools

Type of Testing	Purpose	Tools	Usage in DineEase
Functional Testing	To verify that each feature works correctly	Jest, Cypress	Manual testing of login, menu browsing, cart, payment, and order tracking
Unit Testing	To test individual PHP functions or modules	PHPUnit	Useful for checking core backend logic like login validation and price calculations
Load Testing	To test how the system performs under heavy user traffic	Apache	Simulate multiple users accessing DineEase at once (e.g., peak ordering hours)
Bug Tracking	To record, assign, and fix bugs during development	Trello, GitHub Issues	Track and manage issues such as broken links, payment errors, or UI glitches
Security Testing	To identify potential vulnerabilities in the web application	OWASP	Ensure user input is validated, passwords are encrypted, and no SQL injection allowed

9. Conclusion

The DineEase project successfully delivers a web-based smart restaurant ordering system that streamlines food ordering and restaurant management for both customers and administrators. Designed with a focus on usability, responsiveness, and operational efficiency, the system enables customers to conveniently browse menus, customize orders, make online payments, and track order status, while providing admins with robust tools to manage menus, orders, and users through a dedicated dashboard.

The project follows a user-centered design approach, validated through surveys and iterative feedback integration, ensuring that features align with real-world preferences and needs. The implementation of modern technologies like HTML, CSS, JavaScript, PHP, and MySQL, along with structured project management and collaboration practices, contributed to timely and high-quality delivery.

Through comprehensive planning, requirement analysis, diagram-based system modeling, collaborative development, and rigorous testing, DineEase demonstrates a well-rounded, scalable, and maintainable solution that enhances the restaurant experience, reduces manual workload, and supports digital transformation in the food industry.