RESTAURANT TABLE BOOKING SYSTEM

A PROJECT REPORT

Submitted by

SWAATHI S	**2303811710422164**
THANUJA V	**2303811710422168**
VISHALINI V	**2303811710422180**

in partial fulfillment of the requirements for the award degree of

Bachelor in Engineering

CSB1303-OBJECT ORIENTED ANALYSIS AND DESIGN

in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(An Autonomous Institution, affiliated to Anna University Chennai and Approved by AICTE, New Delhi)

SAMAYAPURAM - 621112

DECEMBER - 2025

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(AUTONOMOUS)

SAMAYAPURAM - 621112

BONAFIDE CERTIFICATE

The work embodied in the present project report entitled "**RESTAURANT TABLE BOOKING SYSTEM**" has been carried out by the students **SWAATHI S, THANUJA V, VISHALINI V**. The work reported here in is original and we declare that the project is their own work, except where specifically acknowledged, and has not been copied from other sources or been previously submitted for assessment.

Date of Viva Voce: ……………………

Mrs.V. KALPANA M.E., (Ph.D.,)	**Mr. R. RAJAVARMAN M.E., (Ph.D.,)**
SUPERVISOR	HEAD OF THE DEPARTMENT
Assistant Professor	Assistant Professor (Sr. Grade)
Department of CSE	Department of CSE
K. Ramakrishnan College of Technology (Autonomous)	K. Ramakrishnan College of Technology (Autonomous)
Samayapuram – 621 112.	Samayapuram – 621 112.

INTERNAL EXAMINER	**EXTERNAL EXAMNIER**

ABSTRACT

The Restaurant Table Booking System is a web-based application that simplifies and modernizes restaurant table reservations. Customers can book tables for a preferred date and time with personalized options such as special occasions, seating preferences, and ambience themes. The system ensures booking authenticity through secure advance online payments and supports reservation cancellation based on restaurant policies. Customers can provide feedback through ratings and comments and view their booking history for future reference. On the administrative side, a comprehensive dashboard allows restaurant staff to efficiently manage reservations, assign tables, update booking statuses, and analyze customer feedback. The system helps restaurants plan seating arrangements better, reduce no-shows, and improve service quality. Overall, the application enhances the dining experience by offering a smooth, interactive, and reliable booking process while supporting efficient restaurant operations.

KEY WORDS:

Restaurant Table Booking System, Web-based Application, Online Reservation, Customer Booking, Seating Preferences, Ambience Selection, Secure Online Payment, Booking Cancellation, Customer Feedback, Booking History, Administrative Dashboard, Reservation Management, Table Assignment, Personalized Service, User Experience, Restaurant Management.

ACKNOWLEDGEMENT

We thank our **DR. N. VASUDEVAN**, Principal, for his valuable suggestions and support during the course of my research work.

We thank our **MR. R. RAJAVARMAN**, Head of the Department, Assistant Professor (Sr. Grade), Department of Computer Science and Engineering, for his valuable suggestions and support during the course of my research work.

We wish to record my deep sense of gratitude and profound thanks to my Guide **Mrs. V. KALPANA,** Assistant Professor , Department of Computer Science and Engineering**,** for her keen interest, inspiring guidance, constant encouragement with my work during all stages, to bring this thesis into fruition.

We are extremely indebted to our project coordinator **Mrs. V. KALPANA,** Assistant Professor, Department of Computer Science and Engineering, for her valuable suggestions and support during the course of my research work.

We also thank the faculty and non-teaching staff members of the Department of Computer Science And Engineering, K.Ramakrishnan College Of Technology, Samayapuram, for their valuable support throughout the course of my research work.

Finally, we thank our parents, friends and our well wishes for their kind support.

SIGNATURE

____

____

____

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

API	-	Application Programming Interface
DB	-	Database
EMI	-	Equated Monthly Instalment
GUI	-	Graphical User Interface
SQL	-	Structured Query Language
MVC	-	Model View Controller

CHAPTER 1
INTRODUCTION

1.1 Introduction about domain

The domain of this project lies in the hospitality and restaurant management sector, focusing specifically on online reservation systems. With the rapid growth of digital technology, restaurants are increasingly adopting web-based solutions to improve customer service and streamline operations. Online booking systems have become essential tools that allow customers to reserve tables easily while helping restaurants manage reservations, reduce waiting times, and avoid overbooking. This domain combines concepts of web development, database management, user experience design, and service automation. It aims to enhance convenience for customers by offering personalized booking options, and it supports restaurant staff by providing organized, real-time reservation data. As the hospitality industry continues to modernize, efficient table booking systems play a crucial role in improving overall service quality and customer satisfaction.

1.2 Problem Description

The Restaurant Table Booking System is a web application developed using Python Flask and MySQL that allows customers to easily reserve tables online by selecting date, time, number of guests, special occasions (like Birthday or Anniversary), seat preferences (Window/Outdoor/AC), and ambience themes (Romantic/Family/Party). Users can register, log in, book tables, make advance payment through a simulated payment page, and later submit star-based feedback with comments. Admins can securely log in, view and manage all bookings, check customer preferences for preparation, and review feedback to improve service.

1.3 Objective

The main objective of the Restaurant Table Booking System is to create an efficient, convenient, and user-friendly platform that allows customers to reserve tables online without the need for phone calls or physical visits. The project aims to simplify the booking process by enabling users to choose their preferred date, time, number of guests, seating area, and special occasion details. It also seeks to enhance the reliability of reservations through features such as advance payment, booking reminders, and cancellation options. Additionally, the system provides customers with access to their booking history and the ability to share feedback after dining. For restaurant administrators, the project aims to offer a centralized dashboard to manage reservations, update statuses, and review customer feedback. Overall, the objective is to improve customer satisfaction and streamline restaurant operations through automation and digital management.

1.4 Scope of the project

The scope of the Restaurant Table Booking System covers both customer-facing and administrative functionalities. It enables customers to create an account, log in, and conveniently book tables online by selecting their preferred date, time, number of guests, seating area, and ambience. The system also includes additional features such as advance payment, booking cancellation, booking history, and customer feedback submission. On the administrative side, the project provides a dedicated dashboard where restaurant staff can view and manage all reservations, update table statuses, monitor customer preferences, and review feedback for service improvement. The system focuses solely on table reservations and related interactions; it does not include food ordering, billing beyond advance payment, or inventory management. Overall, the project aims to enhance customer satisfaction and streamline restaurant operations through an organized and automated reservation platform.

CHAPTER 2
SYSTEM REQUIREMENTS SPECIFICATION

2.1 FUNCTIONAL REQUIREMENTS

2.1.1 User Management

The system allows users to register using their basic details and log in securely with their credentials. Two roles are supported: Customer and Admin. Customers can create and manage bookings, while Admins handle all system operations. The system also provides a secure logout option to end the user session.

2.1.2 Table Booking

Customers can book a table by selecting date, time, number of guests, and preferences such as seating (Window-side, Outdoor, AC/Non-AC), themes (Romantic, Family, Quiet Zone, Party), and special occasions. The system checks table availability and prevents double bookings.

2.1.3 Payment Processing

To confirm a reservation, customers must complete an advance payment through a simulated payment gateway. After successful payment, the booking is marked as "Paid" and stored in the system.

2.1.4 Booking Cancellation

Customers can cancel their reservation if needed, and cancellation charges are applied automatically as per restaurant policy. Users may also modify bookings before payment confirmation.

2.1.5 Booking History

Users can view all their past and upcoming reservations. Each booking entry shows details like date, time, table number, occasion, and payment status, helping users track their reservation history.

2.1.6 Feedback Submission

After completing a reservation, customers can give feedback by submitting a star rating and an optional comment. Admins can view all feedback to monitor service quality and identify improvement areas.

2.1.7 Admin Dashboard

Administrators can view all bookings, update their status, assign table numbers, manage cancellations, and review customer feedback. This ensures smooth restaurant operations.

2.1.8 Security

User passwords are stored securely using hashing, and sessions are protected to prevent unauthorized access. Only authenticated users can view or manage bookings.

2.2 Non Functional Requirements

1. Performance Requirements

The system should load pages quickly and respond to user actions within a few seconds. Booking operations, such as checking table availability or confirming payments, must be processed efficiently to avoid delays. The system should support multiple users booking tables at the same time without slowing down.

2. Reliability Requirements

The system must run consistently without crashes, ensuring that booking data is stored safely and never lost. It must handle unexpected issues gracefully, such as server downtime or failed payments, by showing proper error messages and preserving user data.

3. Usability Requirements

The interface should be simple, attractive, and easy to navigate for all users, including non-technical customers. Buttons, menus, and forms must be clear and user-friendly. The system should work effectively on laptops, tablets, and mobile devices.

4. Security Requirements

User data, including passwords and payment details, must be protected using secure encryption techniques. Only authorized users should access their respective dashboards, and session management must prevent unauthorized access. Admin functions should be strictly restricted to administrators.

5. Scalability Requirements

The system must be designed to support future expansion, such as adding more tables, additional restaurant branches, or new features. It should handle increasing user traffic without performance issues.

6. Availability Requirements

The booking system should be available for use at all times, especially during peak dining hours. Scheduled maintenance should be minimal and announced in advance to avoid inconvenience to users.

7. Maintainability Requirements

The system should have a clean and modular code structure so that developers can easily update features, fix bugs, or modify functionality in the future. Documentation must be clear and detailed for maintenance purposes.

8. Portability Requirements

The system should run smoothly on different browsers such as Chrome, Firefox, and Edge, and it must be deployable on various server environments with minimal changes.

2.3 Hardware Requirements

2.3.1 Server-Side Hardware Requirements

The Restaurant Table Booking System requires a server machine with a minimum 8 GB RAM, quad-core processor, and 50 GB of available disk space to efficiently run the database, web server, and backend services.

2.3.2 Client-Side Hardware Requirements

End users (customers and admins) can access the system using any device with basic configurations such as at least 2 GB RAM, a dual-core processor, and 1 GB of free storage. The system runs on standard web browsers, so laptops, desktops, tablets, and smartphones are all supported as long as they have a functional internet connection.

2.3.3 Network Requirements

A stable internet connection with at least 5 Mbps bandwidth is required for customers to use the system smoothly. The server environment requires a reliable high-speed connection to ensure fast response times and uninterrupted service during peak booking hours.

2.4 Software requirements

The Restaurant Table Booking System can be developed and executed on either a Windows or Linux operating system, making it flexible for different users. A modern web browser such as Google Chrome or Mozilla Firefox is required for accessing and testing the application. The system also relies on a MySQL database for storing user, booking, and feedback information, while a code editor like VS Code or Sublime Text is used for writing and managing the project files. Additionally, tools like XAMPP or WAMP server are essential for running the backend and database locally during development. The project uses a combination of programming languages and technologies, including HTML, CSS, and either PHP or Python, to build the frontend interface and backend logic.

2.5 User Characteristics

Users may have basic computer knowledge. They should know how to operate a web browser. Admin users require slightly higher technical skills. Staff or managers must understand booking/management workflows. Customers should be able to easily navigate the system without training. All users should understand simple forms and buttons.

2.6 Constraints

The system has a few limitations. It needs a good internet connection to work properly and show real-time table availability. It must follow privacy rules to keep customer data safe and can only run on supported browsers. The system also depends on the server's capacity, so too many users at the same time may slow it down. All bookings, changes, and payments must follow the restaurant's rules.

CHAPTER 3
ANALYSIS AND DESIGN

3.1 Use Case Diagram

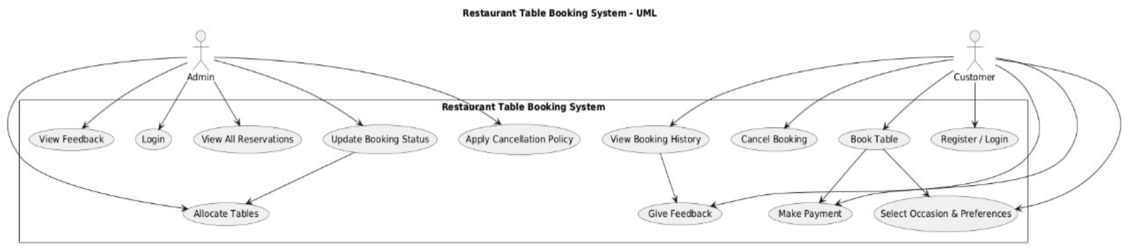


Figure 3.1 Use Case Diagram

3.1.1 Use Case Diagram Description

The Use Case Model of the Restaurant Table Booking System shows how customers and admins interact with the system. Customers can register, log in, book tables by selecting date, time, seating, ambience, and occasion, make payments, cancel bookings, view history, and give feedback. Admins can log in, view and manage all reservations, update booking status, allocate tables, apply cancellation rules, and review customer feedback. The system handles data storage, confirmations, and smooth booking operations, providing an organized experience for both users.

3.2 Class Diagram

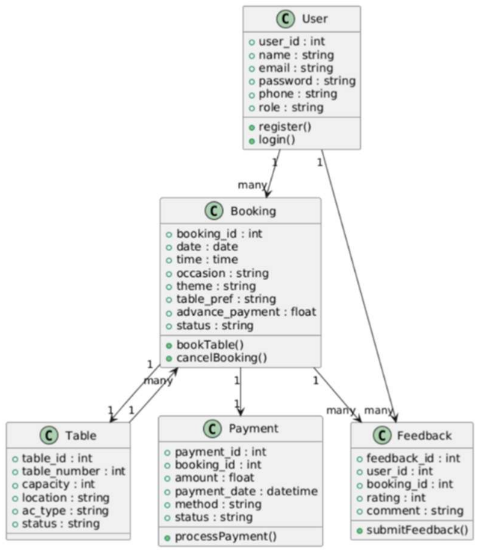


Figure 3.1 Class Diagram

3.2.1 Class Diagram Description

The class diagram defines the main objects of the Restaurant Table Booking System. The User class stores user details and is linked to multiple bookings. The Booking class holds reservation info and connects to one table, one payment, and optional feedback. The Table class stores table details and can have many bookings over time. The Payment class records the payment for each booking. The Feedback class stores user ratings and comments, linked to a specific booking.

3.3 Activity Diagram

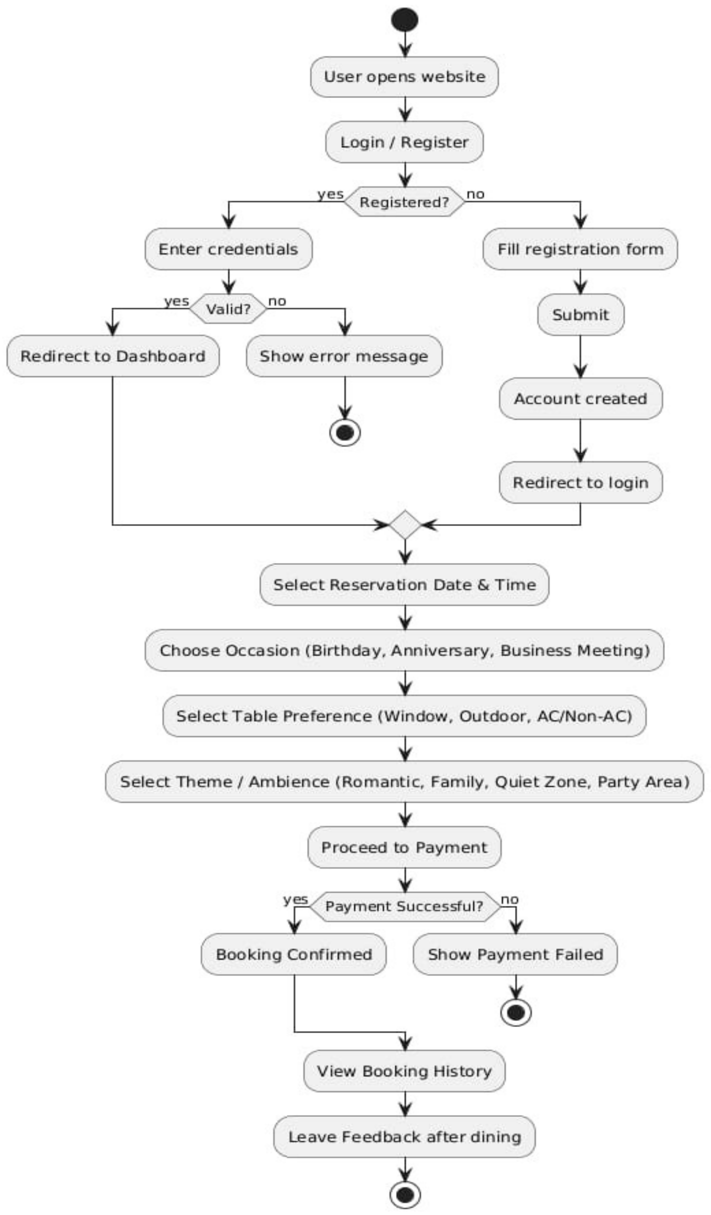


Figure 3.3 Activity Diagram

3.3.1 Activity Diagram Description

The activity diagram shows the customer workflow in the Restaurant Table Booking System. The user first opens the site and either registers or logs in. After a successful login, they select the reservation date, time, occasion, table type, and ambience. Then they proceed to payment. If the payment is successful, the booking is confirmed; if not, a payment failure message is shown.

3.4 Sequence Diagram

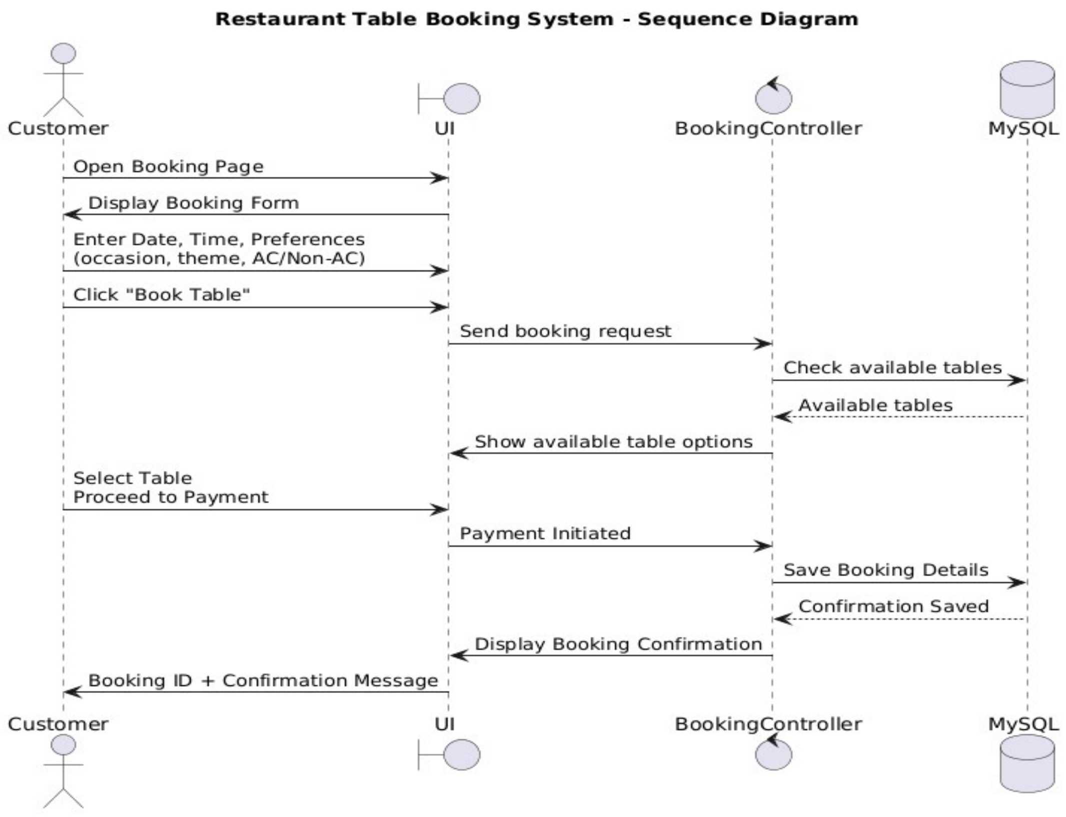

Restaurant Table Booking System - Sequence Diagram
Customer
UI
BookingController
MySQL
Open Booking Page
Display Booking Form
Enter Date, Time, Preferences (occasion, theme, AC/Non-AC)
Click "Book Table"
Send booking request
Check available tables
Available tables
Show available table options
Select Table Proceed to Payment
Payment Initiated
Save Booking Details
Confirmation Saved
Display Booking Confirmation
Booking ID + Confirmation Message

Figure 3.4 Sequence Diagram

3.4.1 Sequence Diagram Description

The sequence diagram shows how a customer books a table through the system. The customer enters booking details in the UI, which sends the request to the BookingController. The controller checks table availability in the MySQL database and returns the results to the UI. The customer selects a table and proceeds to payment. After payment, the controller saves the booking in the database. Once stored successfully, the UI displays a confirmation message with the booking ID.

3.5 State Machine Diagram

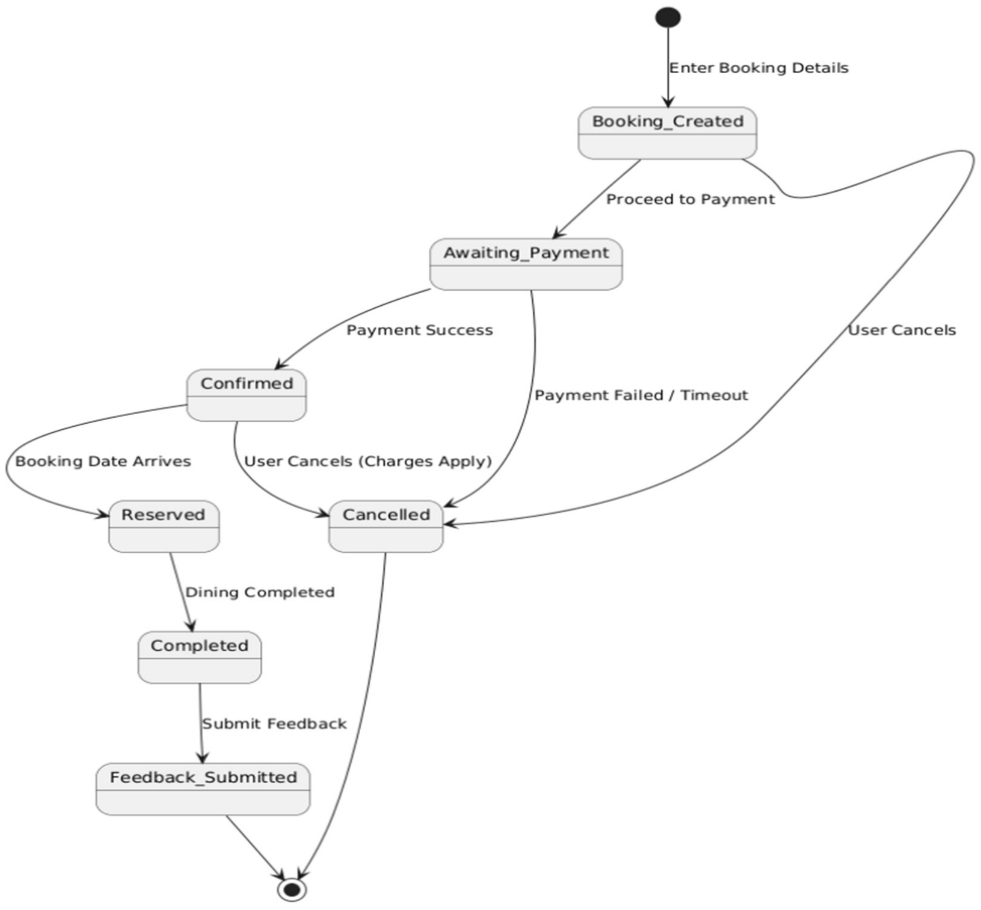


Figure 3.5 State Machine Diagram

3.5.1 State Machine Diagram Description

The state diagram shows how a restaurant table booking progresses through different stages. The process begins when the user enters booking details, creating the initial booking state. It then moves to an awaiting payment stage, where successful payment leads to a confirmed booking, while failed payment or user cancellation leads to a cancelled state. When the booking date arrives, confirmed bookings move to the reserved state, and after dining, they transition to completed. Finally, the user can submit feedback, reaching the last state in the booking lifecycle.

3.6 Component Diagram

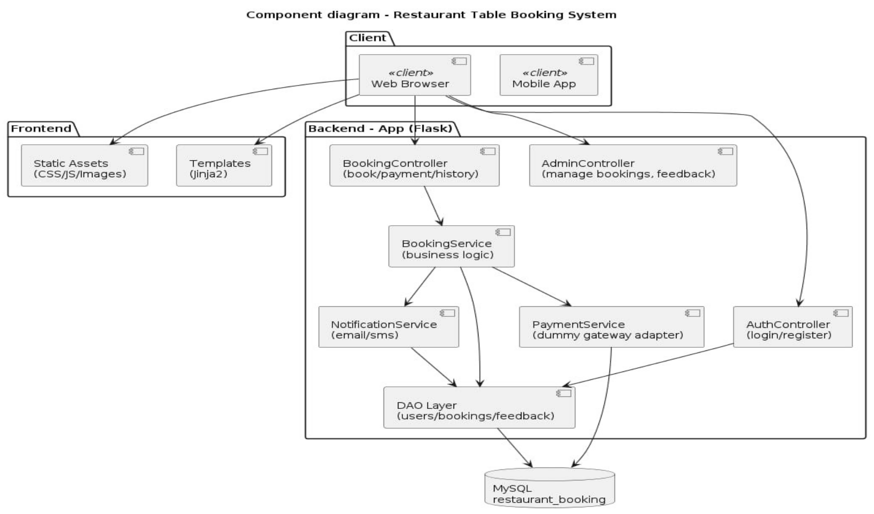

Component diagram - Restaurant Table Booking System
Client
«client» Web Browser
«client» Mobile App
Frontend
Static Assets (CSS/JS/Images)
Templates (jinja2)
Backend - App (Flask)
BookingController (book/payment/history)
AdminController (manage bookings, feedback)
BookingService (business logic)
NotificationService (email/sms)
PaymentService (dummy gateway adapter)
AuthController (login/register)
DAO Layer (users/bookings/feedback)
MySQL restaurant_booking

Figure 3.6 Component Diagram

3.6.1 Component Diagram Description

The component diagram shows that users access the system through a web browser or mobile app. The frontend uses HTML templates, CSS, and JavaScript, while the backend is a Flask application with controllers for authentication, booking, and admin tasks. These controllers use services for booking, payments, and notifications. The DAO layer handles all database operations and connects to the MySQL restaurant_booking database that stores user, booking, and feedback data.

3.7 Deployment Diagram

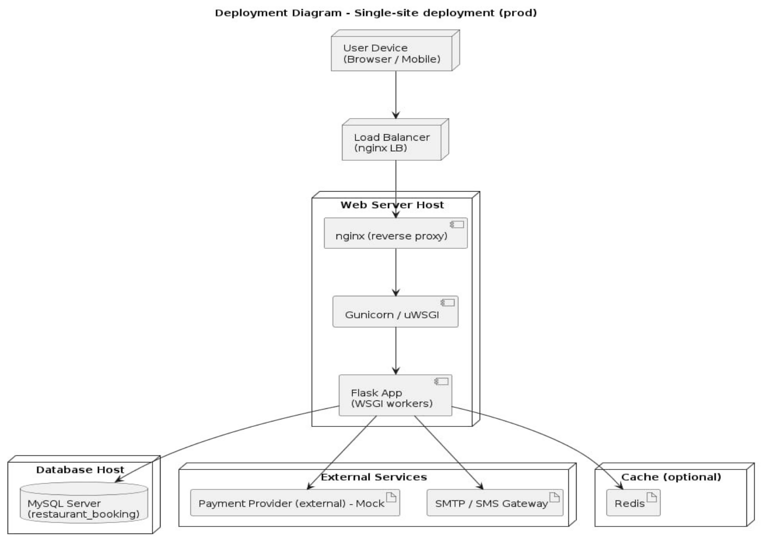


Figure 3.7 Deployment Diagram

3.7.1 Deployment Diagram Description

The deployment diagram shows that users access the system through browsers or mobile apps. Their requests go to an Nginx load balancer, which forwards them to the Flask app running on a WSGI server like Gunicorn. The Flask application connects to a MySQL database to store all booking, user, and feedback data. It may also use external services for payments and notifications, and an optional Redis cache can improve performance.

3.8 Package Diagram

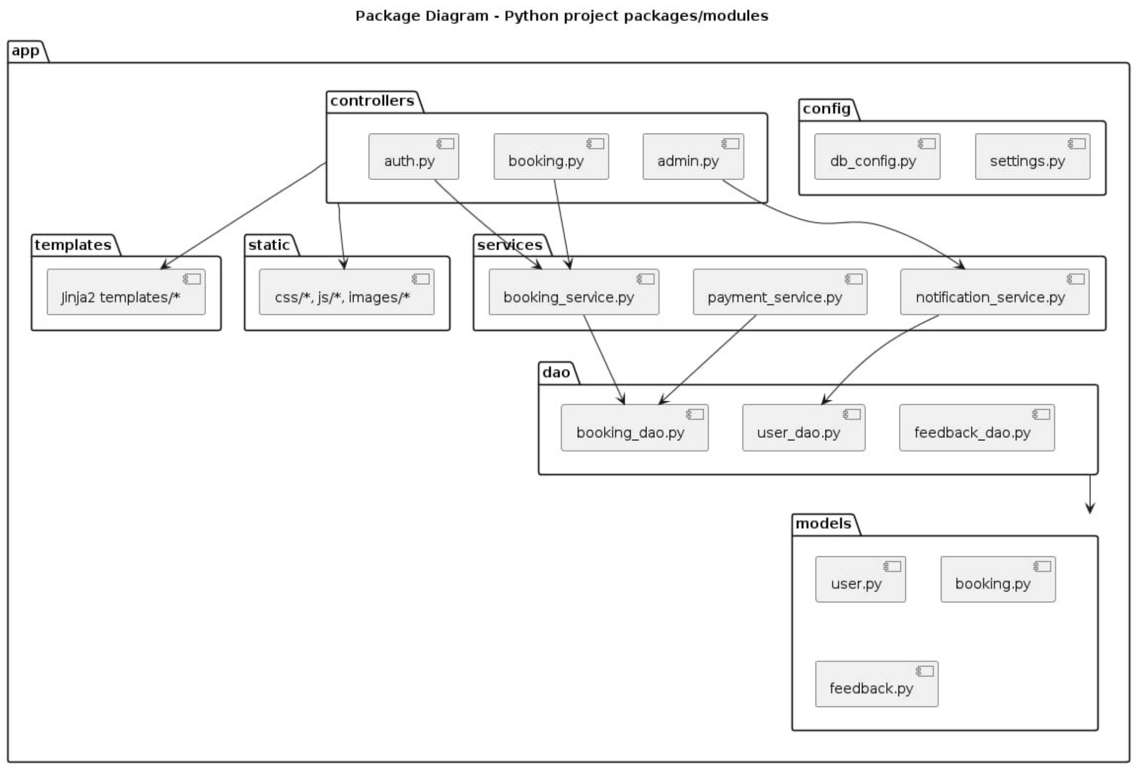


Figure 3.8 Package Diagram

3.8.1 Package Diagram Description

The package diagram groups the system into organized folders. Controlplers handle user actions (auth, booking, admin). Services contain business logic (booking, payment, notifications). The DAO package manages database operations (user, booking, feedback). Models define classes like User, Booking, and Feedback. The config package stores settings and DB configuration, while templates and static folders hold HTML, CSS, and images.

3.9 Collaboration Diagram

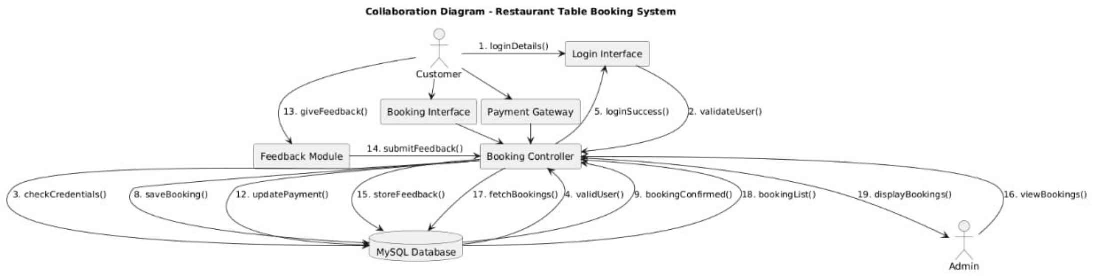


Figure 3.9 Collaboration Diagram

3.9.1 Collaboration Diagram Description

The collaboration diagram shows how the Customer, Admin, and system components interact to complete the restaurant table booking process. The customer logs in, books a table, makes payment, and submits feedback through system interfaces. The Booking Controller manages all requests by validating users, storing booking details, updating payment status, and saving feedback in the MySQL Database. The Admin views and manages bookings through the controller. Overall, the diagram highlights coordinated communication between users, system modules, and the database for efficient reservation management.

3.10 Design Patterns Used (GRASP, GoF)

The Restaurant Table Booking System uses GRASP principles to ensure clear responsibility distribution and a clean, maintainable architecture. The Controller pattern is applied through Flask route functions that manage user inputs, coordinate system actions, and connect the interface to the database. The Creator pattern is used when generating bookings, payments, and feedback entries, assigning object creation to the components that hold the necessary information. Information Expert ensures that modules with the relevant data handle associated operations, such as the database layer managing all CRUD tasks. The system also implements High Cohesion and Low Coupling by separating logic into well-focused modules—templates handle UI, routes handle logic, and the database module manages connections—making the system easier to update and debug. Protected Variations is also visible in the centralized DB configuration, modular templates, and replaceable payment simulation, allowing future upgrades without breaking core functionality.

The system also incorporates several GoF (Gang of Four) design patterns to improve flexibility, reusability, and long-term extensibility. The overall structure follows the MVC pattern, where Flask routes serve as Controllers, HTML templates act as Views, and MySQL tables operate as the Model. A Singleton-style approach is used in the centralized database configuration module that provides consistent connections throughout the application. The Factory Method pattern appears implicitly when the application creates new objects such as bookings, payments, and feedback entries through dedicated functions. The Strategy pattern is reflected in the payment process, where different payment methods can be added in the future without modifying existing booking logic. Additionally, the Template Method pattern is used for admin and customer pages, which share common logic but differ in presentation. Together, these GoF patterns make the system scalable, modular, and robust.

CHAPTER 4
IMPLEMENTATION

4.1 Module Description

4.1.1 User Module

This module handles all customer-related activities. New users can register by providing their basic details, and existing users can log in securely using their credentials. Once logged in, users can update their personal information, manage their profile, and explore all booking features. They can make table reservations by selecting their preferences, check their booking history, proceed with advance payment, and later provide feedback. The module ensures a smooth and user-friendly experience throughout the booking process.

4.1.2 Table Booking Module

The Table Booking Module manages the entire reservation process. Users can choose the desired date, time, and number of guests. They can also select special occasions such as Birthday, Anniversary, or Business Meeting, choose ambience themes like Romantic, Family, Quiet, or Party area, and decide seat preferences like Window-side, Outdoor, AC, or Non-AC. The system checks table availability and confirms the reservation. Once confirmed, all booking details are stored safely in the database for further processing.

4.1.3 Payment Module

This module handles secure advance payments for booked tables. It provides a simulated and easy-to-use payment page where customers can enter card details. The module verifies the payment information, processes the transaction, and updates the booking status to “Paid.” It ensures that every confirmed reservation is backed by a valid payment, helping the restaurant avoid last-minute cancellations or fake bookings.

4.1.4 Feedback Module

After completing their dining experience, users can share their opinions through the Feedback Module. They can give a star rating and write comments about food quality, ambience, and service. This information is stored in the database and is accessible to the admin. The feedback helps the restaurant understand customer satisfaction levels and make improvements to enhance service quality.

4.1.5 Admin Module

The Admin Module provides restaurant staff with complete control over system operations. Admins can securely log in to view all bookings, monitor ongoing reservations, and update booking statuses such as Confirmed, Paid, Cancelled, or Completed. They can also check customer preferences, plan table arrangements, handle special occasion requirements, and manage cancellations. Additionally, admins can read customer feedback to identify strengths and areas for improvement, ensuring better service and customer satisfaction.

4.2 Technology Description

The Restaurant Table Booking System is developed using a combination of modern and efficient technologies to ensure smooth performance, easy maintenance, and a user-friendly experience. The front-end is built using HTML and CSS, which provide a clean, attractive, and responsive interface for users to interact with the system. These technologies help in designing forms, buttons, booking pages, and a visually appealing layout.

The back-end is developed using Python (Flask Framework), which handles user requests, processes booking data, manages business logic, and interacts with the database. Flask is lightweight, fast, and suitable for building web applications with clear routing and modular structure.

The system uses MySQL as the database, which stores all essential data such as user details, table reservations, payments, feedback, and admin records. MySQL provides reliability, fast queries, and strong data integrity to ensure the system runs efficiently.

Overall, the combination of HTML/CSS for the UI, Python Flask for server-side processing, and MySQL for data management creates a secure, stable, and scalable platform for restaurant table booking and administration.

CHAPTER 5
TESTING

5.1 TESTING STRATEGY (TYPES OF TESTING)

Testing is an essential phase of the software development life cycle to ensure that the system works as intended and meets the requirements. The Restaurant Table Booking System is tested using a combination of manual and functional testing strategies. The testing focuses on detecting errors, verifying functionalities, ensuring usability, and confirming data integrity.

Types of Testing Performed

Unit Testing

Each module (login, booking, payment, feedback) is tested individually to ensure that it works correctly.

Integration Testing

Interaction between modules (e.g., booking module with payment module) is tested to confirm smooth data flow.

System Testing

The complete system is tested as a whole to ensure all functional and non-functional requirements are met.

User Acceptance Testing (UAT)

Real users test the system to check usability, navigation, and overall satisfaction.

Performance Testing

Page loading times, database response times, and simultaneous booking handling are tested.

Security Testing

Password encryption, access control, and data protection are verified.

5.2 SAMPLE TEST CASES

Test Case 1 – User Login Validation

Objective: Verify valid users can log in.

Input: Email = user@gmail.com, Password = user123

Expected Output: Redirect to user dashboard.

Status: Pass

Test Case 2 – Table Booking Functionality

Objective: Ensure users can book a table with correct details.

Input: Date, Time, Guests = 4, Occasion = "Birthday", Seating = "Window"

Expected Output: Booking saved and redirected to payment.

Status: Pass

Test Case 3 – Payment Confirmation

Objective: Ensure payment updates booking status.

Input: Payment for Booking ID

Expected Output: Booking status → "Paid"

Status: Pass

Test Case 4 – Booking Cancellation

Objective: Confirm a user can cancel a booking.

Action: Click "Cancel Booking"

Expected Output: Status changes to "Cancelled", charges applied.

Status: Pass

Test Case 5 – View Booking History

Objective: Ensure past bookings are displayed correctly.

Action: Open "My Booking History"

Expected Output: List of all previous bookings.

Status: Pass

Test Case 6 – Admin View All Bookings

Objective: Verify admin can see all bookings.

Action: Open /admin/bookings

Expected Output: Table showing all booking details and status.

Status: Pass

Test Case 7 – AC / Non-AC Selection

Objective: Validate environment preference.

Input: AC/Non-AC = “AC Section”

Expected Output: Saved in booking details.

Actual Result: Saved correctly.

Status: Pass

Test Case 8 – Feedback Submission

Objective: Verify that users can submit feedback after dining.

Input: Rating = 5 stars, Comment = “Excellent service!”

Expected Output: Feedback stored and displayed under “Feedback Page”.

Actual Result: Saved successfully.

Status: Pass

Test Case 9 – Admin Feedback Viewing

Objective: Ensure admin can see all feedback.

Action: Open /admin/feedback

Expected Output: Display list of all feedback.

Actual Result: Loaded successfully.

Status: Pass

5.3 TEST RESULTS

The testing of the Restaurant Table Booking System showed that all core modules performed successfully and without errors. The Registration Module accepted all valid user data while correctly rejecting invalid inputs. The Login Module verified both user and admin credentials, ensuring that only valid logins were allowed. The Booking Module handled reservations smoothly and prevented double booking of the same table. The Payment Module accurately updated booking statuses after payment simulation, confirming proper financial processing. The Booking Cancellation feature worked correctly by applying cancellation charges automatically. The Feedback Module saved user ratings and comments properly and displayed them when required. The Admin Dashboard functioned efficiently, allowing the admin to view all bookings and update their statuses without issues. Overall, all functionalities passed testing, demonstrating that the system is stable, reliable, and fully ready for deployment.

CHAPTER 6
CONCLUSION AND FUTURE ENHANCEMENT

6.1 CONCLUSION

The Restaurant Table Booking System provides an efficient, user-friendly, and automated solution for managing table reservations. By allowing customers to book tables online, select seating preferences, specify special occasions, and make advance payments, the system significantly reduces manual errors and waiting times. The admin dashboard enables restaurant staff to monitor reservations, update booking statuses, and view customer feedback, thereby improving operational efficiency and service quality. Throughout the development process, the system was tested extensively to ensure functionality, performance, and security. The results indicate that the system meets all functional and non-functional requirements. It enhances customer satisfaction, simplifies restaurant management, and provides a reliable, accessible platform for both users and staff.

6.2 FUTURE ENHANCEMENT

The Restaurant Table Booking System can be further improved in the future to make it more comprehensive, modern, and user-friendly. One possible enhancement is the integration of a real-time payment gateway, allowing customers to make secure online payments using credit/debit cards, UPI, or digital wallets. Automated SMS and email notifications can be added to confirm bookings, send reminders, or notify customers about cancellations. Advanced analytics for the admin can provide insights on booking patterns, customer preferences, peak hours, and revenue, helping in better decision-making. Additionally, integrating an online menu and food ordering system would allow customers to place orders along with their reservations. Developing dedicated mobile applications for Android and iOS would enhance accessibility and user experience. Features such as loyalty programs, discounts, and AI-based recommendations for seating or ambience based on customer history can also be implemented to provide a more personalized service. These future enhancements will further increase customer satisfaction and improve overall restaurant efficiency.

APPENDIX A – SOURCE CODE

SQL QUERY

```
CREATE DATABASE IF NOT EXISTS restaurant_booking;
USE restaurant_booking;
CREATE TABLE IF NOT EXISTS users (
   id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(150) NOT NULL,
   email VARCHAR(150) NOT NULL UNIQUE,
   password VARCHAR(255) NOT NULL,
   phone VARCHAR(30),
   role ENUM('admin','customer') DEFAULT 'customer',
   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
CREATE TABLE IF NOT EXISTS bookings (
   id INT AUTO_INCREMENT PRIMARY KEY,
   user_id INT,
   booking_date DATE,
   booking_time TIME,
   guests INT DEFAULT 2,
   occasion VARCHAR(100),
   preference VARCHAR(100),
   theme VARCHAR(100),
   section VARCHAR(100),
   status ENUM('Pending','Confirmed','Cancelled','Completed','Paid') DEFAULT
'Pending',
   total_amount DECIMAL(10,2) DEFAULT 0.00,
   payment_status ENUM('Unpaid','Paid','Refunded') DEFAULT 'Unpaid',
   cancellation_fee DECIMAL(10,2) DEFAULT 0.00,
   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE SET NULL
);
```

```
CREATE TABLE IF NOT EXISTS feedback (
  id INT AUTO_INCREMENT PRIMARY KEY,
  booking_id INT,
  user_id INT,
  rating TINYINT,
  comment TEXT,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (booking_id) REFERENCES bookings(id) ON DELETE
CASCADE,
  FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE SET NULL
);
CREATE TABLE IF NOT EXISTS transactions (
  id INT AUTO_INCREMENT PRIMARY KEY,
  booking_id INT,
  amount DECIMAL(10,2),
  method VARCHAR(50),
  status ENUM('Success','Failed','Refunded') DEFAULT 'Success',
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (booking_id) REFERENCES bookings(id) ON DELETE
CASCADE
);
INSERT IGNORE INTO users (name, email, password, phone, role)
VALUES ('Admin', 'admin@admin.com', 'admin123', '9999999999', 'admin');
```

DB_CONFIG.PY

```
import os
import mysql.connector
def get_db_connection():
  return mysql.connector.connect(
    host=os.getenv('DB_HOST', 'localhost'),
    user=os.getenv('DB_USER', 'root'),
    password=os.getenv('DB_PASS', 'root@123'),
    database=os.getenv('DB_NAME', 'restaurant_booking'),
    autocommit=True
  )
```

APP.PY

```
from flask import Flask, render_template, request, redirect, url_for, session, flash
from db_config import get_db_connection
from functools import wraps
from datetime import datetime
app = Flask(__name__)
app.secret_key = "replace_with_a_secret_key"
def login_required(role=None):
    def decorator(fn):
        @wraps(fn)
        def wrapper(*args, **kwargs):
            if 'user' not in session:
                flash('Please log in first.', 'warning')
                return redirect(url_for('login'))
            if role and session['user']['role'] != role:
                flash('Access denied.', 'danger')
                return redirect(url_for('index'))
            return fn(*args, **kwargs)
        return wrapper
    return decorator
# ---- Home / Landing ----
@app.route('/')
def index():
    return render_template('index.html')
# ---- Register ----
@app.route('/register', methods=['GET','POST'])
def register():
    if request.method == 'POST':
        name = request.form.get('name','').strip()
        email = request.form.get('email','').strip().lower()
        password = request.form.get('password','').strip()
        phone = request.form.get('phone','').strip()
        if not (name and email and password):
            flash('Please fill all required fields.', 'warning')
            return render_template('register.html')
        conn = get_db_connection()
        cur = conn.cursor()
        try:
            cur.execute("INSERT INTO users (name,email,password,phone,role)
VALUES (%s,%s,%s,%s,%s)",
                        (name, email, password, phone, 'customer'))
            flash('Registration successful. Please login.', 'success')
            return redirect(url_for('login'))
        except Exception as e:
            conn.rollback()
            flash('Email already registered or error occurred.', 'danger')
        finally:
            cur.close(); conn.close()
    return render_template('register.html')
# ---- Login ----
@app.route('/login', methods=['GET','POST'])
def login():
    if request.method == 'POST':
        email = request.form.get('email','').strip().lower()
        password = request.form.get('password','').strip()
        conn = get_db_connection()
        cur = conn.cursor(dictionary=True)
        cur.execute("SELECT * FROM users WHERE email=%s AND password=%s",
(email, password))
```

```
        user = cur.fetchone()
        cur.close(); conn.close()
        if user:
            session['user'] = {
                'id': user['id'],
                'name': user['name'],
                'email': user['email'],
                'role': user['role']
            }
            flash('Logged in successfully.', 'success')
            if user['role'] == 'admin':
                return redirect(url_for('admin_dashboard'))
            return redirect(url_for('booking'))
        flash('Invalid credentials.', 'danger')
    return render_template('login.html')
# ---- Logout ----
@app.route('/logout')
def logout():
    session.pop('user', None)
    flash('Logged out.', 'info')
    return redirect(url_for('index'))
# ---- Booking page ----
@app.route('/booking', methods=['GET','POST'])
@login_required(role='customer')
def booking():
    if request.method == 'POST':
        user_id = session['user']['id']
        booking_date = request.form.get('date')
        booking_time = request.form.get('time')
        guests = int(request.form.get('guests', 2))
        occasion = request.form.get('occasion','Other')
        preference = request.form.get('preference','Window Side')
        theme = request.form.get('theme','Family')
        section = request.form.get('section','AC')
        amount = float(request.form.get('amount', 500.00))
        # insert booking
        conn = get_db_connection()
        cur = conn.cursor()
        cur.execute("""
            INSERT INTO bookings (user_id, booking_date, booking_time, guests,
occasion, preference, theme, section, status, total_amount, payment_status)
            VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)
        """, (user_id, booking_date, booking_time, guests, occasion, preference, theme,
section, 'Pending', amount, 'Unpaid'))
        booking_id = cur.lastrowid
        cur.close(); conn.close()
        # store booking id in session for payment flow
        session['pending_booking_id'] = booking_id
        flash('Booking created. Proceed to payment to confirm.', 'info')
        return redirect(url_for('payment', booking_id=booking_id))
    # GET -> render
    return render_template('booking.html')
# ---- Payment simulation ----
@app.route('/payment/<int:booking_id>', methods=['GET','POST'])
@login_required(role='customer')
def payment(booking_id):
    # verify booking belongs to user and is unpaid/pending
    conn = get_db_connection()
    cur = conn.cursor(dictionary=True)
    cur.execute("SELECT * FROM bookings WHERE id=%s AND user_id=%s",
(booking_id, session['user']['id']))
    booking = cur.fetchone()
    cur.close(); conn.close()
    if not booking:
        flash('Invalid booking or permission denied.', 'danger')
        return redirect(url_for('booking'))
    if request.method == 'POST':
        # read mock card info (we do not validate security here)
        card_name = request.form.get('card_name','').strip()
        card_number = request.form.get('card_number','').strip()
        expiry = request.form.get('expiry','').strip()
        cvv = request.form.get('cvv','').strip()
        # mark as paid and confirmed
        conn = get_db_connection()
        cur = conn.cursor()
        cur.execute("UPDATE bookings SET payment_status=%s, status=%s WHERE
id=%s", ('Paid', 'Confirmed', booking_id))
        cur.execute("INSERT INTO transactions (booking_id, amount, method, status)
VALUES (%s,%s,%s,%s)",
                    (booking_id, booking['total_amount'], 'Card(Mock)', 'Success'))
        cur.close(); conn.close()
        flash('Payment successful. Booking confirmed!', 'success')
        # clear pending booking from session
        session.pop('pending_booking_id', None)
        return redirect(url_for('history'))
    return render_template('payment.html', booking=booking)
# ---- Booking history ----
@app.route('/history')
@login_required()
def history():
    user = session['user']
    conn = get_db_connection()
    cur = conn.cursor(dictionary=True)
    if user['role'] == 'customer':
        cur.execute("SELECT * FROM bookings WHERE user_id=%s ORDER BY
created_at DESC", (user['id'],))
        bookings = cur.fetchall()
        cur.close(); conn.close()
        return render_template('history.html', bookings=bookings)
    else:
        # admins get all bookings
        cur.execute("SELECT b.*, u.name AS customer_name FROM bookings b LEFT
JOIN users u ON b.user_id=u.id ORDER BY b.created_at DESC")
        bookings = cur.fetchall()
        cur.close(); conn.close()
        return render_template('admin_view_bookings.html', bookings=bookings)
# ---- Cancel booking (customer) ----
@app.route('/cancel/<int:booking_id>', methods=['POST'])
@login_required(role='customer')
def cancel_booking(booking_id):
    # basic cancellation policy: fixed fee 100
    cancellation_fee = 100.00
    conn = get_db_connection()
    cur = conn.cursor()
    # ensure belongs to user and not already cancelled
    cur.execute("SELECT * FROM bookings WHERE id=%s AND user_id=%s",
(booking_id, session['user']['id']))
    b = cur.fetchone()
    if not b:
        cur.close(); conn.close()
        flash('Booking not found.', 'danger')
        return redirect(url_for('history'))
    cur.execute("UPDATE bookings SET status=%s, cancellation_fee=%s WHERE
id=%s", ('Cancelled', cancellation_fee, booking_id))
```

```
    cur.execute("INSERT INTO transactions (booking_id, amount, method, status)
VALUES (%s,%s,%s,%s)",
                (booking_id, -cancellation_fee, 'CancellationFee', 'Success'))
    cur.close(); conn.close()
    flash(f'Booking {booking_id} cancelled. ₹{cancellation_fee} cancellation fee
applied.', 'info')
    return redirect(url_for('history'))
# ---- Feedback (customer) ----
@app.route('/feedback/<int:booking_id>', methods=['GET','POST'])
@login_required(role='customer')
def feedback(booking_id):
    # ensure booking belongs to user
    conn = get_db_connection()
    cur = conn.cursor(dictionary=True)
    cur.execute("SELECT * FROM bookings WHERE id=%s AND user_id=%s",
(booking_id, session['user']['id']))
    booking = cur.fetchone()
    if not booking:
        cur.close(); conn.close()
        flash('Invalid booking for feedback.', 'danger')
        return redirect(url_for('history'))
    if request.method == 'POST':
        rating = int(request.form.get('rating', 5))
        comment = request.form.get('comment','').strip()
        cur = conn.cursor()
        cur.execute("INSERT INTO feedback (booking_id, user_id, rating, comment)
VALUES (%s,%s,%s,%s)",
                    (booking_id, session['user']['id'], rating, comment))
        cur.close(); conn.close()
        flash('Thank you for your feedback!', 'success')
        return redirect(url_for('history'))
    cur.close(); conn.close()
    return render_template('feedback.html', booking=booking)
# ---- Admin dashboard (simple) ----
@app.route('/admin')
@login_required(role='admin')
def admin_dashboard():
    conn = get_db_connection()
    cur = conn.cursor(dictionary=True)
    # totals
    cur.execute("SELECT COUNT(*) AS total_bookings FROM bookings")
    totals = cur.fetchone()
    # recent bookings
    cur.execute("SELECT b.*, u.name AS customer_name FROM bookings b LEFT
JOIN users u ON b.user_id=u.id ORDER BY created_at DESC LIMIT 10")
    recent = cur.fetchall()
    cur.close(); conn.close()
    return render_template('admin_dashboard.html', totals=totals, recent=recent)
# ---- Admin view feedback ----
@app.route('/admin/feedback')
@login_required(role='admin')
def admin_feedback():
    conn = get_db_connection()
    cur = conn.cursor(dictionary=True)
    cur.execute("""SELECT f.*, u.name AS customer_name, b.booking_date
                FROM feedback f
                LEFT JOIN users u ON f.user_id = u.id
                LEFT JOIN bookings b ON f.booking_id = b.id
                ORDER BY f.created_at DESC""")
    feedbacks = cur.fetchall()
    cur.close(); conn.close()
    return render_template('admin_feedback.html', feedbacks=feedbacks)
```

```
# ---- Admin update booking status ----
@app.route('/admin/update_status', methods=['POST'])
@login_required(role='admin')
def admin_update_status():
    booking_id = int(request.form.get('booking_id', 0))
    new_status = request.form.get('status')
    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("UPDATE bookings SET status=%s WHERE id=%s", (new_status,
booking_id))
    cur.close(); conn.close()
    flash('Booking status updated.', 'success')
    return redirect(url_for('admin_dashboard'))
if __name__ == '__main__':
    app.run(debug=True)
```

LOGIN.HTML

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Login</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
  <div class="header">
    <div class="brand">CozyBistro</div>
  </div>
  <div class="container">
    <div class="form-card">
      <h2>Login</h2>
      <form method="post">
```

```
        <label class="label">Email</label>
        <input class="input" name="email" type="email" required>
        <label class="label">Password</label>
        <input class="input" name="password" type="password" required>
        <div style="margin-top:12px;">
          <button class="btn" type="submit">Login</button>
          <a class="btn secondary" href="{{ url_for('register') }}" style="margin-
left:10px;">Register</a>
        </div>
      </form>
    </div>
  </div>
</body>
</html>
```

REGISTER.HTML

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Register</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='style.css')
}}">
</head>
<body>
  <div class="header">
    <div class="brand">CozyBistro</div>
  </div>
  <div class="container">
    <div class="form-card">
```

```
      <h2>Create an account</h2>
      <form method="post">
        <label class="label">Full name</label>
        <input class="input" name="name" required>
        <label class="label">Email</label>
        <input class="input" name="email" type="email" required>
        <label class="label">Phone</label>
        <input class="input" name="phone" type="text">
        <label class="label">Password</label>
        <input class="input" name="password" type="password"
required>
        <div style="margin-top:12px;">
          <button class="btn" type="submit">Register</button>
          <a class="btn secondary" href="{{ url_for('login') }}"
style="margin-left:10px;">Already have account</a>
        </div>
      </form>
    </div>
  </div>
</body>
</html>
```

ADMIN_DASHBOARD.HTML

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Admin Dashboard</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
```

```
</head>
<body>
  <div class="header"><div class="brand">CozyBistro</div>
    <div><a href="{{ url_for('admin_feedback') }}">Feedback</a> | <a href="{{
url_for('logout') }}">Logout</a></div>
  </div>
  <div class="container">
    <h2>Admin Dashboard</h2>
    <div class="card-row">
      <div class="card">
        <h3>Total Bookings</h3>
        <p style="font-size:28px; font-weight:700;">{{ totals.total_bookings
}}</p>
      </div>
    </div>
    <h3>Recent Bookings</h3>
    <table class="table">
<thead><tr><th>ID</th><th>Customer</th><th>Date</th><th>Time</th><th>
Guests</th><th>Occasion</th><th>Status</th><th>Action</th></tr></thead>
      <tbody>
        {% for b in recent %}
        <tr>
          <td>{{ b.id }}</td>
          <td>{{ b.customer_name or 'N/A' }}</td>
          <td>{{ b.booking_date }}</td>
          <td>{{ b.booking_time }}</td>
          <td>{{ b.guests }}</td>
          <td>{{ b.occasion }}</td>
          <td>{{ b.status }}</td>
          <td>
            <form method="post" action="{{ url_for('admin_update_status') }}">
```

```
              <input type="hidden" name="booking_id" value="{{ b.id }}">
              <select name="status">
                <option {% if b.status=='Pending' %}selected{% endif
%}>Pending</option>
                <option {% if b.status=='Confirmed' %}selected{% endif
%}>Confirmed</option>
                <option {% if b.status=='Paid' %}selected{% endif %}>Paid</option>
                <option {% if b.status=='Cancelled' %}selected{% endif
%}>Cancelled</option>
                <option {% if b.status=='Completed' %}selected{% endif
%}>Completed</option>
              </select>
              <button class="btn" type="submit">Update</button>
            </form>
          </td>
        </tr>
        {% endfor %}
      </tbody>
    </table>
  </div>
</body>
</html>
```

BOOKING.HTML

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Book a Table</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
```

```
<div class="header">
  <div class="brand">CozyBistro</div>
  <div>
    <a href="{{ url_for('history') }}">My Bookings</a> |
    <a href="{{ url_for('logout') }}">Logout</a>
  </div>
</div>
<div class="container">
  <div class="form-card">
    <h2>Book Your Table</h2>
    <form method="post">
      <div class="form-row">
        <div style="flex:1;">
          <label class="label">Date</label>
          <input class="input" type="date" name="date" required>
        </div>
        <div style="width:160px;">
          <label class="label">Time</label>
          <input class="input" type="time" name="time" required>
        </div>
      </div>
      <label class="label">No. of Guests</label>
      <input class="input" type="number" name="guests" min="1" value="2">
      <label class="label">Occasion</label>
      <select class="input" name="occasion">
        <option>Birthday</option>
        <option>Anniversary</option>
        <option>Business Meeting</option>
        <option>Other</option>
      </select>
      <label class="label">Preferred Table</label>
```

```
        <select class="input" name="preference">
          <option>Window Side</option>
          <option>Outdoor</option>
          <option>AC Section</option>
          <option>Non-AC Section</option>
        </select>
        <label class="label">Theme / Ambience</label>
        <select class="input" name="theme">
          <option>Romantic</option>
          <option>Family</option>
          <option>Quiet Zone</option>
          <option>Party Area</option>
        </select>
        <label class="label">Section</label>
        <select class="input" name="section">
          <option>AC</option>
          <option>Non-AC</option>
        </select>
        <input type="hidden" name="amount" value="500">
        <div style="margin-top:12px;">
          <button class="btn" type="submit">Proceed to Payment (₹500)</button>
        </div>
      </form>
    </div>
  </div>
</body>
</html>
```

PAYMENT.HTML

```
<!doctype html>
<html>
```

```
<head>
  <meta charset="utf-8">
  <title>Payment</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
  <div class="header">
    <div class="brand">CozyBistro</div>
  </div>
  <div class="container">
    <div class="form-card">
      <h2>Secure Payment (Demo)</h2>
      <p>Booking ID: {{ booking.id }} &nbsp; | &nbsp; Amount: ₹{{
booking.total_amount }}</p>
      <form method="post">
        <label class="label">Card Holder Name</label>
        <input class="input" name="card_name" required>
        <label class="label">Card Number</label>
        <input class="input" name="card_number" maxlength="16" required
placeholder="XXXX XXXX XXXX XXXX">
        <div class="form-row">
          <div style="flex:1;">
            <label class="label">Expiry</label>
            <input class="input" type="month" name="expiry" required>
          </div>
          <div style="width:140px;">
            <label class="label">CVV</label>
            <input class="input" name="cvv" maxlength="4" required>
          </div>
        </div>
        <div style="margin-top:12px;">
```

```
          <button class="btn" type="submit">Pay Now</button>
        </div>
      </form>
    </div>
  </div>
</body>
</html>
```

FEEDBACK.HTML

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Feedback</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
  <div class="header"><div class="brand">CozyBistro</div></div>
  <div class="container">
    <div class="form-card">
      <h2>Feedback for Booking #{{ booking.id }}</h2>
      <p>Date: {{ booking.booking_date }} | Time: {{ booking.booking_time }}</p>
      <form method="post">
        <label class="label">Rating</label>
        <select class="input" name="rating" required>
          <option value="5">5 - Excellent</option>
          <option value="4">4 - Very Good</option>
          <option value="3">3 - Good</option>
          <option value="2">2 - Poor</option>
          <option value="1">1 - Very Poor</option>
        </select>
        <label class="label">Comments</label>
        <textarea class="input" name="comment" rows="4" required></textarea>
        <div style="margin-top:12px;"><button class="btn" type="submit">Submit
Feedback</button></div>
      </form>
    </div>
  </div>
</body>
</html>
```

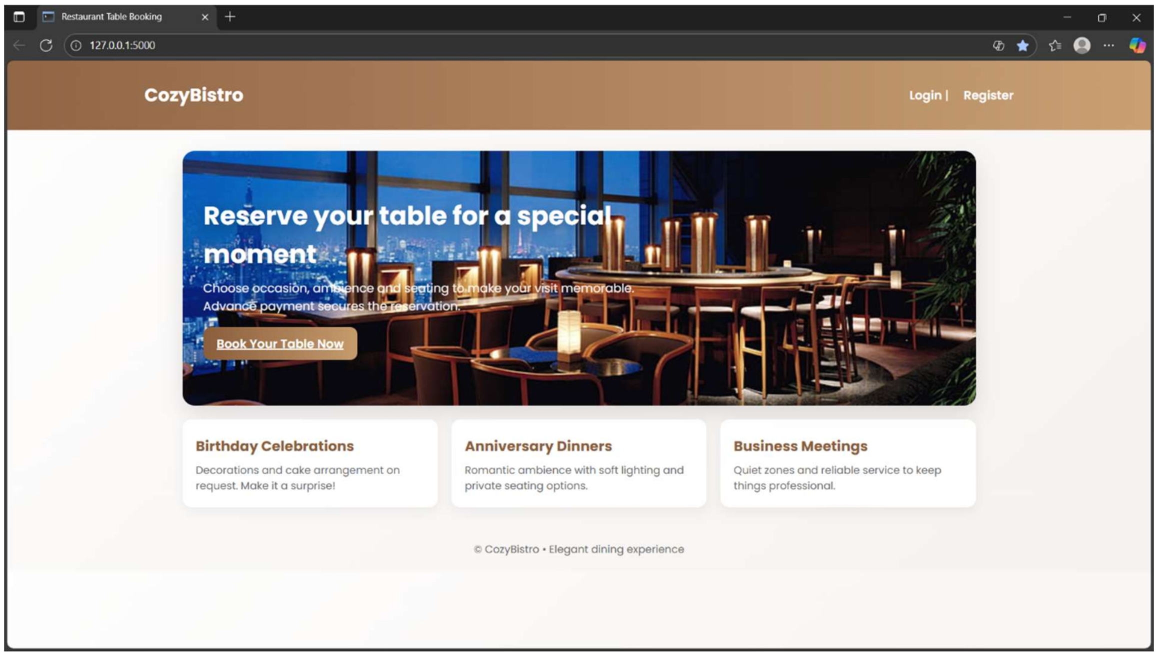
STYLES.CSS

```
/* static/style.css - Elegant indoor restaurant theme (light & warm) */
@import
url('https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;600;700&displ
ay=swap');

:root{
  --bg1: #f6f2ef;
  --accent: #8b5e3c;    /* warm brown */
  --accent-2: #c69c6d;  /* light gold */
  --muted: #6b6b6b;
  --card: rgba(255,255,255,0.9);
}
*{box-sizing:border-box}
body{
  margin:0;
  font-family: 'Poppins', sans-serif;
  background: linear-gradient(135deg, #fefefe 0%, #f6f2ef 100%);
  color:#333;
  -webkit-font-smoothing:antialiased;
}
.header{
  background: linear-gradient(90deg, rgba(139,94,60,0.95), rgba(198,156,109,0.95));
  color: #fff;
  padding: 28px 12%;
  display:flex;
  justify-content:space-between;
  align-items:center;
}
.header .brand { font-size:24px; font-weight:700; }
.header a { color:#fff; text-decoration:none; margin-left:16px; font-weight:600; }
.container{ max-width:1100px; margin:28px auto; padding:0 20px; }
.banner{
  background-image: url('/static/images/bg.jpg');
  background-size:cover;
  background-position:center;
  height:340px;
  border-radius:16px;
  display:flex;
  align-items:center;
  color:#fff;
  padding:28px;
  margin-bottom:18px;
  box-shadow: 0 12px 30px rgba(0,0,0,0.12);
}
.banner .left{ max-width:60%; }
.banner h1{ font-size:34px; margin:0 0 8px; text-shadow: 0 4px 16px
rgba(0,0,0,0.25); }
.banner p{ margin:0 0 16px; opacity:0.95; }
.card-row{ display:flex; gap:18px; margin-bottom:18px; flex-wrap:wrap; }
.card{
  background:var(--card);
  border-radius:12px;
  padding:18px;
  flex:1;
  min-width:220px;
  box-shadow: 0 6px 18px rgba(0,0,0,0.06);
}
.card h3{ margin:4px 0 8px; color:var(--accent); }
.card p{ margin:0; color:var(--muted); font-size:14px; }
.form-card{
  background:var(--card);
  padding:20px;
  border-radius:12px;
  box-shadow: 0 8px 24px rgba(0,0,0,0.06);
  max-width:700px;
  margin: 0 auto 24px;
}
.form-row{ display:flex; gap:12px; flex-wrap:wrap; }
.input, select, textarea{
  width:100%;
  padding:10px 12px;
  border-radius:8px;
  border:1px solid #e6e0da;
  background:#fff;
  font-size:15px;
}
.label{ font-weight:600; font-size:14px; margin-bottom:6px; display:block;
color:#333; }
.btn{
  display:inline-block;
  background: linear-gradient(90deg,var(--accent),var(--accent-2));
  color:#fff;
  padding:10px 18px;
  border-radius:10px;
  border:none;
  cursor:pointer;
  font-weight:600;
}
.btn.secondary{ background:#fff; color:var(--accent); border:1px solid #e6d7c7; }
.table{
  width:100%;
  border-collapse:collapse;
  background:#fff;
  border-radius:8px;
  overflow:hidden;
  box-shadow: 0 8px 20px rgba(0,0,0,0.04);
}
.table th, .table td{
  padding:12px 14px;
  border-bottom:1px solid #f1e9e1;
  text-align:left;
}
.table th { background:#faf6f3; color:var(--muted); font-weight:600; }
.footer{ text-align:center; padding:18px 0; color:var(--muted); font-size:14px; margin-
top:24px; }
a.link { color:var(--accent); text-decoration:none; font-weight:600; }
@media (max-width:800px){
  .banner{ height:240px; padding:18px; }
  .banner .left{ max-width:100%; }
  .card-row{ flex-direction:column; }
}
```

APPENDIX B – SCREENSHOTS

1. HOME PAGE



Reserve your table for a special moment

Choose occasion, ambience and seating to make your visit memorable. Advance payment secures the reservation.

Birthday Celebrations
Decorations and cake arrangement on request. Make it a surprise!

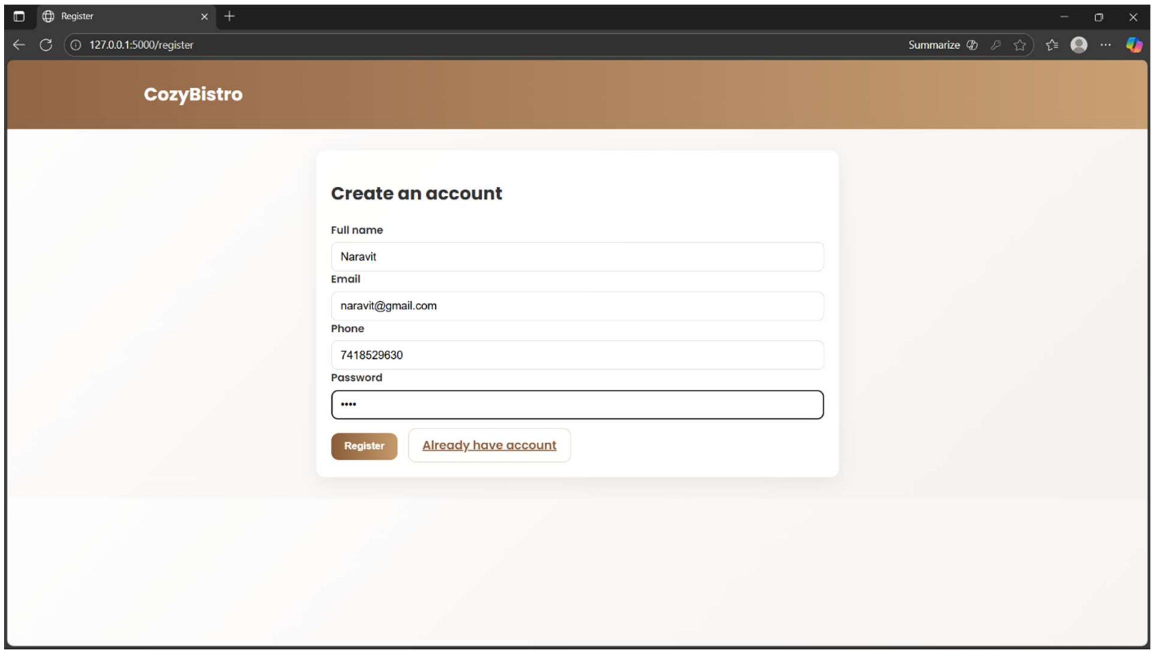
Anniversary Dinners
Romantic ambience with soft lighting and private seating options.

Business Meetings
Quiet zones and reliable service to keep things professional.

© CozyBistro • Elegant dining experience

Figure B.1 Home Page

2. REGISTRATION PAGE



Create an account

Full name: Naravit

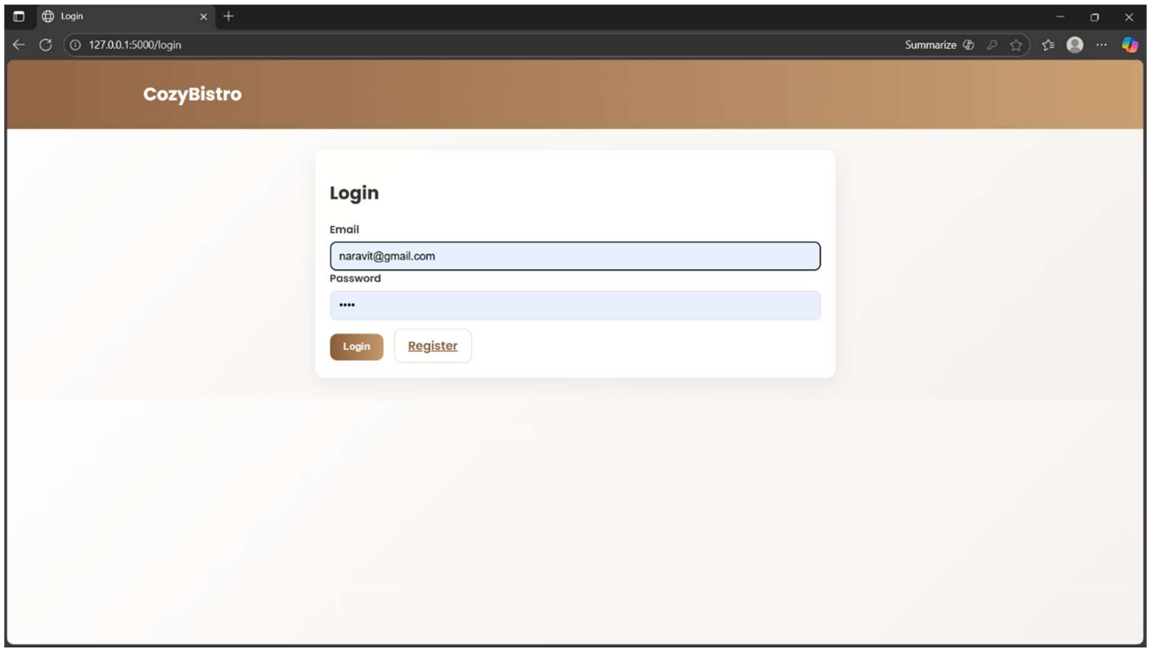
Email: naravit@gmail.com

Phone: 7418529630

Password: ••••

Figure B.2 Registration Page

3. LOGIN PAGE



Login

Email

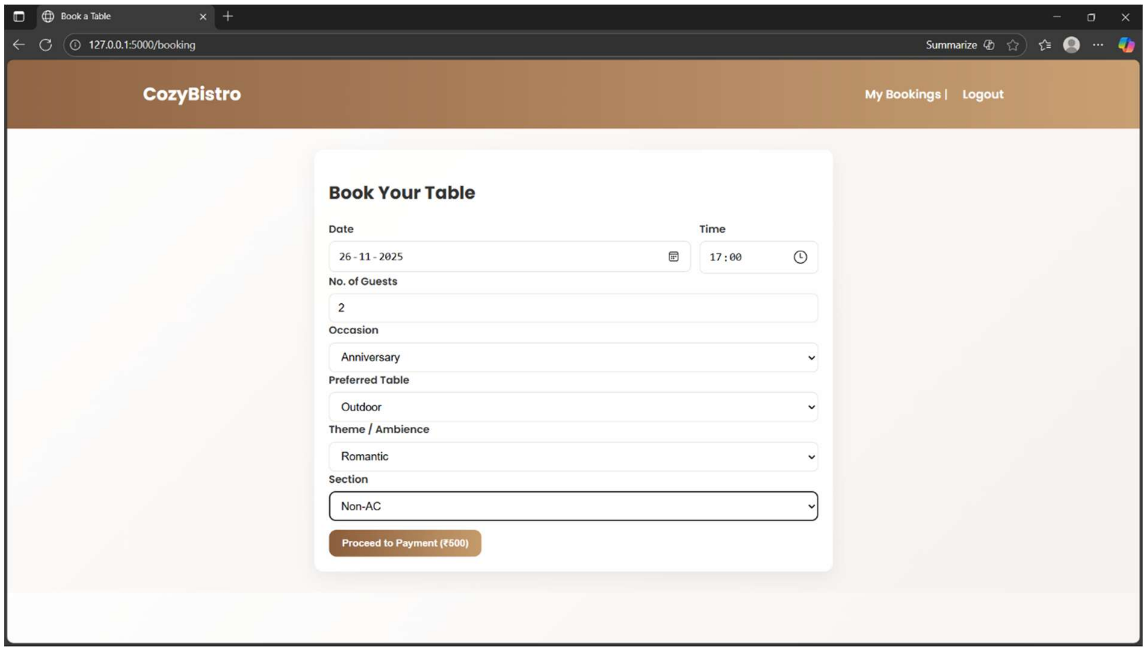
naravit@gmail.com

Password

••••

Figure B.3 Login Page

4. BOOKING PAGE



Book Your Table

Date: 26 - 11 - 2025

Time: 17 : 00

No. of Guests: 2

Occasion: Anniversary

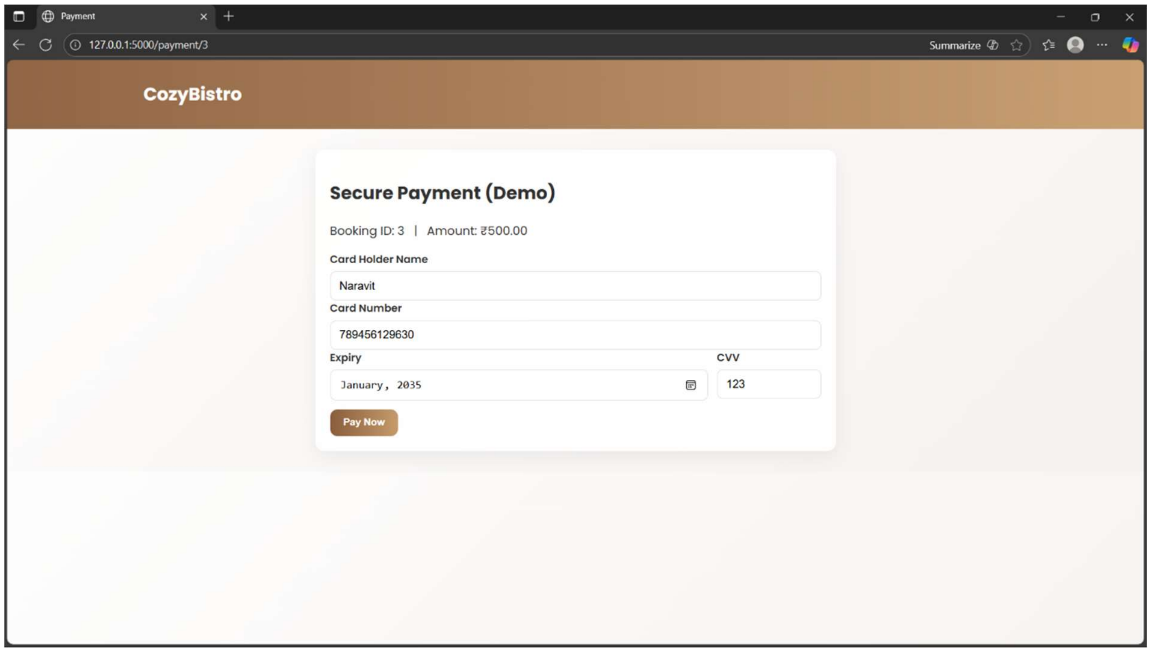
Preferred Table: Outdoor

Theme / Ambience: Romantic

Section: Non-AC

Figure B.4 Booking Page

5. PAYMENT PAGE



CozyBistro

Secure Payment (Demo)

Booking ID: 3 | Amount: ₹500.00

Card Holder Name: Naravit

Card Number: 789456129630

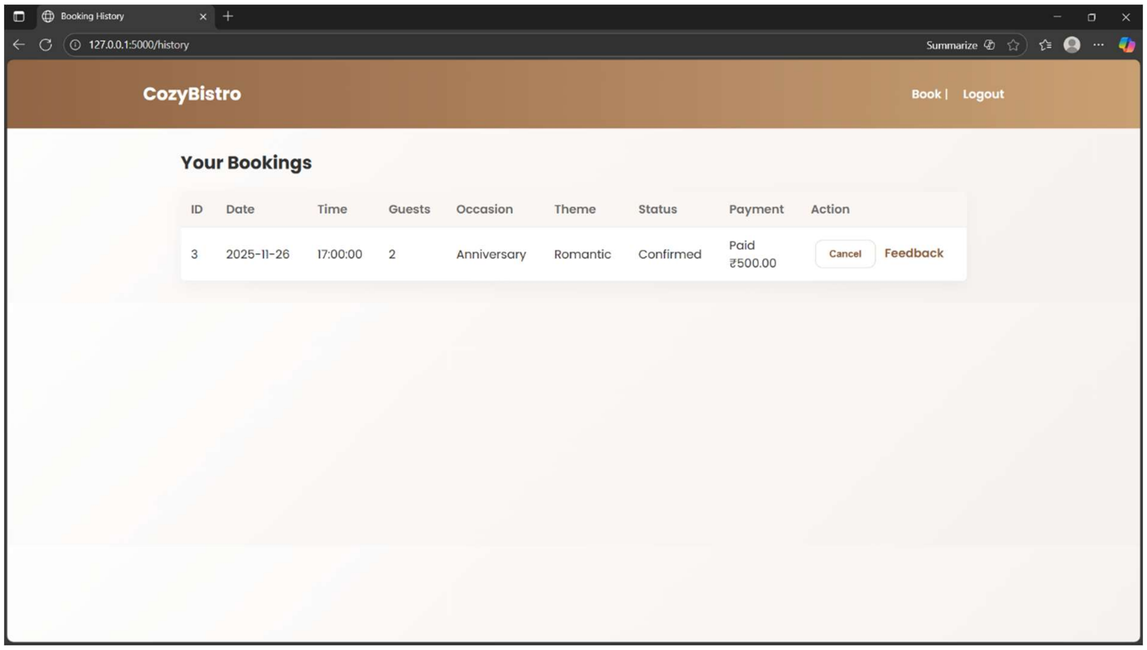
Expiry: January, 2035

CVV: 123

Pay Now

Figure B.4 Payment Page

6. HISTORY PAGE



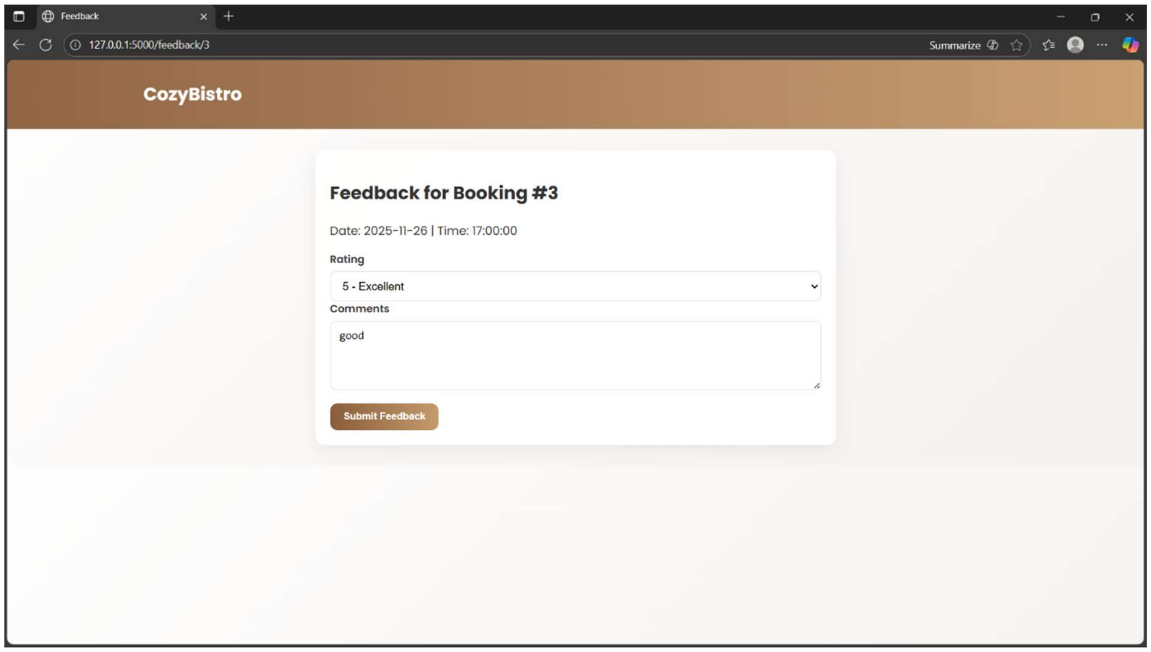
CozyBistro   Book | Logout

Your Bookings

ID	Date	Time	Guests	Occasion	Theme	Status	Payment	Action
3	2025-11-26	17:00:00	2	Anniversary	Romantic	Confirmed	Paid ₹500.00	Cancel Feedback

Figure B.5 History Page

7. CUSTOMER FEEDBACK PAGE



Feedback for Booking #3

Date: 2025-11-26 | Time: 17:00:00

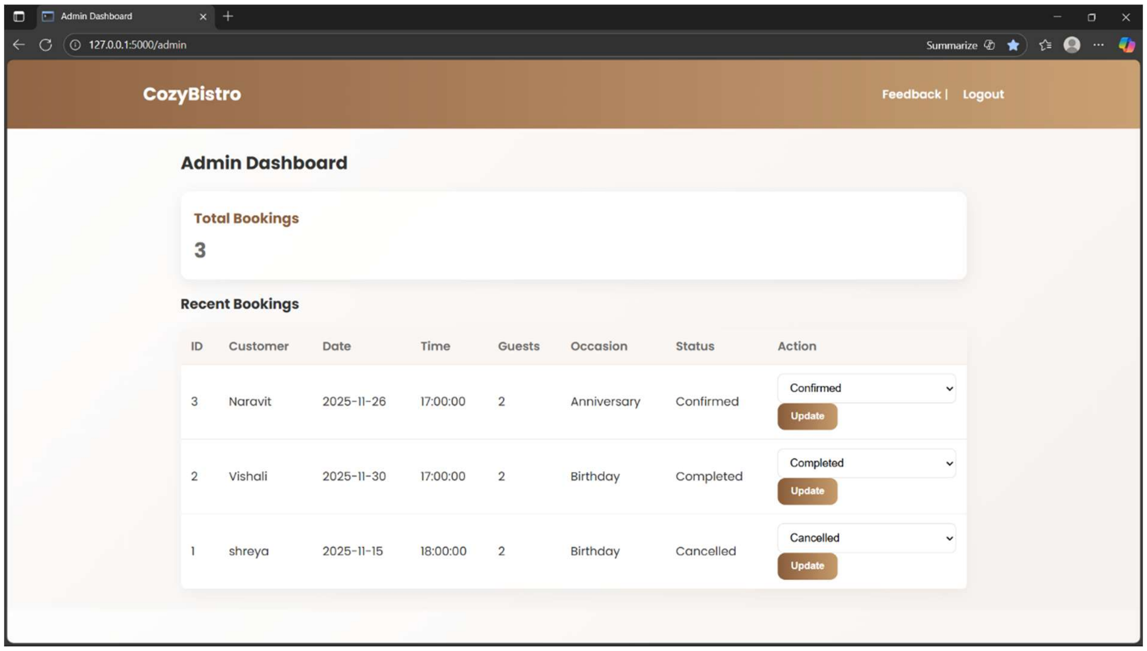
Rating: 5 - Excellent

Comments: good

Submit Feedback

Figure B.7 Customer Feedback Page

8. ADMIN DASHBOARD



Admin Dashboard

Total Bookings

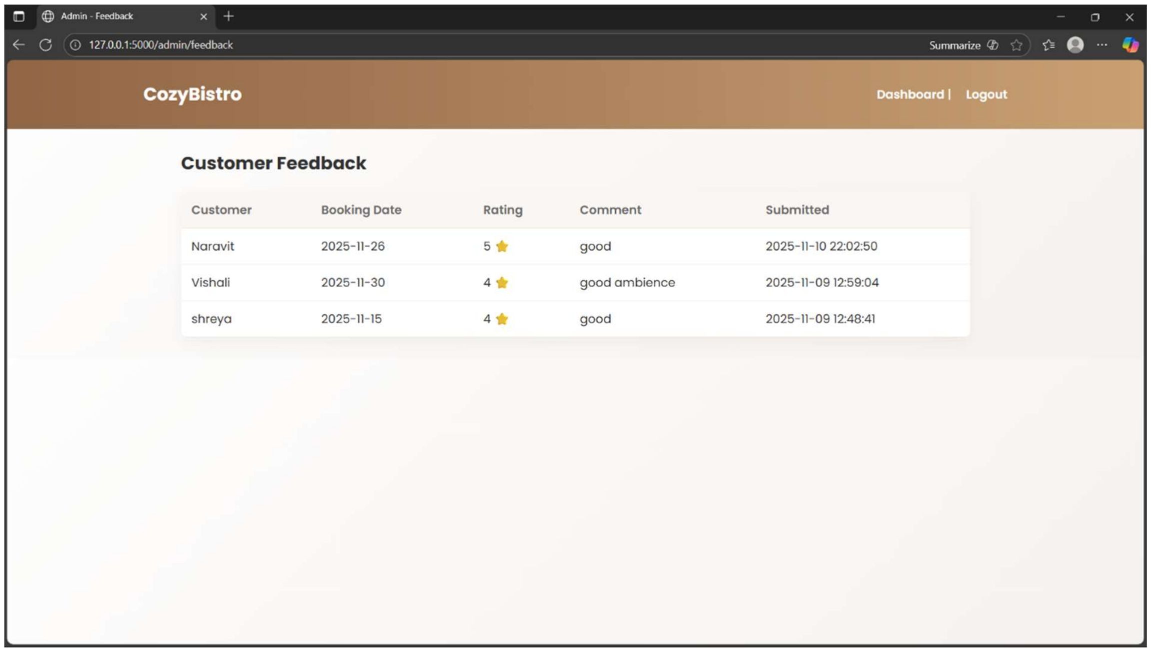
3

Recent Bookings

ID	Customer	Date	Time	Guests	Occasion	Status	Action
3	Naravit	2025-11-26	17:00:00	2	Anniversary	Confirmed	Confirmed Update
2	Vishali	2025-11-30	17:00:00	2	Birthday	Completed	Completed Update
1	shreya	2025-11-15	18:00:00	2	Birthday	Cancelled	Cancelled Update

Figure B.8 Admin Dashboard

9. ADMIN FEEDBACK PAGE



Customer Feedback

Customer	Booking Date	Rating	Comment	Submitted
Naravit	2025-11-26	5 ★	good	2025-11-10 22:02:50
Vishali	2025-11-30	4 ★	good ambience	2025-11-09 12:59:04
shreya	2025-11-15	4 ★	good	2025-11-09 12:48:41

Figure B.9 Admin Feedback Page

REFERENCES

REFERENCE BOOKS

1. Luke Welling, Laura Thomson (2016), *PHP and MySQL Web Development*, Addison-Wesley, 5th Edition, pp. 1–1008.
2. Mark Lutz (2013), *Learning Python*, O'Reilly Media, 5th Edition, pp. 1–1600.
3. Wesley J. Chun (2010), *Core Python Programming*, Prentice Hall, 2nd Edition, pp. 1–1120.

REFERENCE LINKS

4. CodeWithHarry: https://www.youtube.com/c/CodeWithHarry
5. Flask Documentation: https://flask.palletsprojects.com/
6. FreeCodeCamp: https://www.youtube.com/c/Freecodecamp
7. MySQL Official Documentation: https://dev.mysql.com/doc/
8. ProgrammingKnowledge: https://www.youtube.com/c/ProgrammingKnowledge
9. W3Schools HTML/CSS Tutorials: https://www.w3schools.com/