RESORT MANAGEMENT SYSTEM

MINI PROJECT REPORT

Submitted by

J JACINTH MANUEL (9517202209039)

S MANOJ KUMAR (9517202209068)

R RAGAVAN (9517202209092)

in

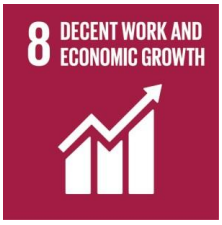
23AD553 –MINI PROJECT III :SOFTWARE ENGINEERING FOR AI APPLICATIONS

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

MEPCO SCHLENK ENGINEERING COLLEGE

SIVAKASI

November 2025



MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI

AUTONOMOUS

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

BONAFIDE CERTIFICATE

This is to certify that it is the bonafide work of JACINTH MANUEL J Reg. No. (9517202309039), MANOJ KUMAR S Reg. No. (9517202309068), RAGAVAN R Reg. No. (9517202309092) for the mini project titled "RESORT MANAGEMENT SYSTEM" in 23AD553 - MINI PROJECT III - SOFTWARE ENGINEERING FOR AI APPLICATIONS during the fifth semester July 2025 - November 2025 under my supervision.

SIGNATURE

Ms.A.Rakshana Malya, B.Tech, ME
Assistant Professor,
AI&DS Department,
Mepco Schlenk Engg. College, Sivakasi

SIGNATURE

Dr. J. Angela Jennifa Sujana, M.E.,Ph.D
Professor & Head,
AI&DS Department,
Mepco Schlenk Engg. College, Sivakasi.

ACKNOWLEDGEMENT

First and foremost we praise and thank **"The Almighty",** the lord of all creations, who by his abundant grace has sustained us and helped us to work on this project successfully.

We really find unique pleasure and immense gratitude in thanking our respected management members**,** who is the backbone of our college.

A deep bouquet of thanks to respected Principal **Dr.S.Arivazhagan M.E.,Ph.D.,** for having provided the facilities required for our mini project.

We sincerely thank our Head of the Department **Dr. J. Angela Jennifa SujanaM.E.,Ph.D.,** Associate Professor(SG) & Head, Department of Artificial Intelligence and Data Science, for her guidance and support throughout the mini project .

We also thank our guide **Ms. A. Rakshana Malya** Assistant Professor, Department of Artificial Intelligence and Data Science for their valuable guidance and it is great privilege to express our gratitude to them.

We extremely thank our project coordinator **Mrs. M. Revatheeswari** Assistant Professor of Artificial Intelligence and Data Science, who inspired us and supported us throughout the mini project.

We extend our heartfelt thanks and profound gratitude to all the faculty members of Artificial Intelligence and Data Science department for their kind help during our mini project work.

We also thank our parents and our friends who had been providing us with constant support during the course of the mini project work.

ABSTRACT

The Resort Management System (RMS) project aims to digitalize and streamline the process of booking, managing, and maintaining resorts by providing a centralized, interactive, and transparent platform for users and administrators. The system's core functionality revolves around user convenience and operational efficiency—allowing customers to search resorts, make bookings, order food, manage payments through a virtual wallet, and generate invoices, while enabling administrators to oversee resort data, manage users, and track overall performance.

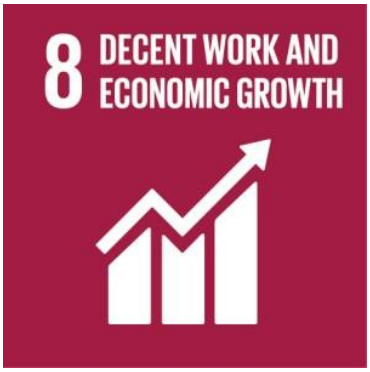
The application architecture follows the MERN (MongoDB, Express.js, React, Node.js) stack to ensure scalability, modularity, and responsiveness. Data Flow Diagrams (DFDs) are used to illustrate process flows—such as user registration, authentication, resort listing, booking confirmation, staff assignment, food ordering, checkout, and PDF invoice generation—while Entity Relationship Diagrams (ERDs) define structured relationships among entities like USER, RESORT, BOOKING, STAFF, FOOD, and INVOICE.

The administrator dashboard provides intuitive tools for adding, updating, and deleting resorts, viewing user transactions, updating wallet balances, and analyzing booking statistics. The system employs JWT-based authentication and bcrypt hashing for secure user login and registration, ensuring strong data protection. Additionally, a chatbot module "Ask RMS Bot " has been integrated to guide users with booking-related FAQs, food menu queries, and general assistance.

Comprehensive black box and white box testing validated all functional modules, including booking verification, payment deduction, and automated PDF invoice generation, ensuring a reliable, user-centric experience. The system enhances transparency in resort operations, simplifies administrative control, and delivers a professional and automated digital booking workflow. Future advancements include email notifications, AI-powered recommendation systems, and mobile app integration to improve accessibility and personalization.

Table of Contents

SDG Mapping

	SDG	GOAL	RELEVANCE TO PROJECT
	SDG 8	Decent Work and Economic Growth	RMS creates employment for resort staff and supports digital tourism.
	SDG 12	Responsible Consumption and Production	RMS reduces paper use through digital bookings and billing, promoting sustainable resource management.
	SDG 17	Partnerships for the Goals	RMS encourages partnerships between resorts, local travel agencies, and tourism boards through its digital ecosystem

List of Tables

List Of Figures

Abbreviations

S No.	Abbreviation	Description
1	RMS	Resort Management System
2	SDLC	Software Development Life Cycle
3	SRS	Software Requirement Specification
4	HTTPS	Hyper Text transfer Protocol
5	DFD	Data Flow Diagram
6	API	Application Programming Interface
7	JSON	JavaScript Object Notation
8	SDG	Sustainable Development Goals
9	UAT	User Acceptance Testing
10	FAQ	Frequently Asked Questions

CHAPTER 1
INTRODUCTION

1.1 Waterfall Model for Resort Management System

1.1.1 Model Chosen

For the Resort Management System (RMS), the Waterfall Software Development Life Cycle (SDLC) model has been chosen because it provides a structured and sequential approach to development. The RMS project includes various well-defined modules such as room booking, billing, staff management, and an AI-driven chatbot. These requirements are clearly documented in the SRS and are not expected to undergo frequent changes during development. The Waterfall model is ideal in such cases because it follows a linear phase-by-phase process starting from requirement gathering, design, implementation, testing, and deployment.

1.1.2 Reason

We selected the Waterfall model for several reasons that align with the needs of the (RMS) project:

- **Thorough Documentation:** Given the importance of data security and accuracy, in billing and user authentication, Waterfall's emphasis on detailed documentation at each stage helped us define clear requirements, design specifications, and test plans. This documentation supports traceability and reduces the risk of errors.
- **Reliability and Stability:** Since the RMS handles sensitive customer data and financial transactions, Waterfall's comprehensive testing at each phase helped us ensure that the final system is robust, secure, and meets all operational requirements without unexpected failures.
- **Predictable Timelines and Resource Management:** The sequential nature of Waterfall it easier to estimate project timelines, allocate resources efficiently, and manage costs—factors critical for academic and professional project settings where deadlines and budgets must be controlled.
- **Minimized Scope Changes**: By finalizing requirements and designs upfront, Waterfall helped avoid frequent changes during development, thus reducing complexity and preventing delays in project delivery.

  The Waterfall methodology, with its clear stage-wise progression, aligned well with the project:

- **Requirement Analysis and Design Completed Early:** This allowed precise understanding of resort management workflows security needs, enabling a well-structured system design.
- **Focused Testing at Each Phase:** Testing was integrated after each development stage, ensuring that modules like billing and secure login worked flawlessly before integration.
- **Stable System Delivery:** The Waterfall model ensured delivery of a thoroughly tested, reliable system on schedule, fulfilling the resort's operational demands.
- **Simplified Project Management:** With fixed phases and milestones, tracking progress and managing the project was straightforward, aiding successful completion within time and budget.

In conclusion, adopting the Waterfall model for the Resort Management System facilitated a disciplined, well-documented development process focused on reliability, security, and predictable delivery. This approach ensured the system met the resort's critical operational requirements efficiently and effectively.

Software Requirements Specification

for

RESORT MANAGEMENT SYSTEM

Version 1.0 approved

Prepared by Development Team

J.JACINTH MANUEL

S.MANOJ KUMAR

R.RAGAVAN

20th July 2025

CHAPTER 2

SOFTWARE REQUIREMENT SPECIFICATION

2.1 Purpose

The purpose of this Software Requirements Specification (SRS) document is to provide a comprehensive overview of the functional and non-functional requirements for the Resort Management System (RMS). This system is designed to enhance the operational efficiency of resorts by automating various processes such as reservations, guest management, billing, and service delivery.

2.1.2 Document Conventions

This document follows standard **IEEE SRS** formatting guidelines. Key conventions include:

- **Bold text** for section headings
- *Italic text* for emphasis
- Monospace text for code or commands
- Numbered and bulleted lists for clarity
- Priority levels explicitly mentioned for each requirement

2.1.3 Intended Audience and Reading Suggestions

The primary audience for this document includes:

- **Developers:** Responsible for system implementation
- **Project Managers:** Oversee project execution
- **Municipality Officers:** End-users managing grievances
- **Testers:** Verify system functionality and compliance
- **System Administrators:** Maintain and support the system

Suggested reading sequence:

1. **Introduction** – Overview of the system

2. **Overall Description** – Understanding functionality
3. **System Features** – Detailed feature breakdown
4. **External Interface Requirements** – Understanding integrations
5. **Nonfunctional Requirements** – Performance and security aspects

2.1.4 Product Scope

The Resort Management System is designed to automate and manage various aspects of resort operations, including room booking, customer management, payment processing, housekeeping, and feedback handling. The system aims to provide a seamless experience for both guests and staff, reducing manual effort and minimizing errors. The scope includes the development of a user-friendly interface and robust reporting features for management oversight.

References

- **IEEE SRS Standard** (ISO/IEC/IEEE 29148:2018)
- System Design Documents (Use Case, Sequence, and DFD Diagrams)

2.2 Overall Description

2.2.1 Product Perspective

The **Resort Management System** is envisioned as a standalone application that can be deployed as either a desktop or web application. It will consist of several internal modules, such as booking, inventory management, and billing, as well as external modules that leverage AI technologies, including a chatbot for customer support and a room recommendation engine.

The system will be designed to integrate with existing resort management tools and databases, ensuring a smooth transition and minimal disruption to current operations.

2.2.2 Product Functions

The core functions of the system include:

- User Registration & Login
- Booking Management
- Check-In & Food Orders
- Chatbot for Customer Support

- Staff Management
- Billing Process & Invoice flow
- Admin Dashboard & Reports
- Feedback and Ratings

2.2.3 User Classes and Characteristics

- **Admin:** Has full access to all system functionalities, including user management, reporting, and system settings.
- **Receptionist:** Responsible for handling guest check-ins and check-outs, managing room bookings, and providing customer service.
- **Customer:** Guests who can access the booking portal to make reservations and manage their profiles.
- **Support Staff:** Personnel who have access to housekeeping and transport modules to ensure guest needs are met.

2.2.4 Operating Environment

- **Platform:** Web-based, accessible via desktop and mobile browsers.
- **Operating Systems:** Windows, macOS, Linux, Android, iOS.
- **Technologies Used:** Cloud-based infrastructure, RESTful APIs for integration.

2.2.5 Design and Implementation Constraints

- The system must be designed to function offline with limited features, ensuring that essential operations can continue without internet connectivity.
- A mobile application version is considered for future development, allowing users to access the system on-the-go.

2.2.6 User Documentation

- **User Manuals:** Detailed guides for each user class, outlining functionalities and procedures.
- **Online Help:** Contextual help within the application to assist users in real-time.
- **Training Materials:** Resources for onboarding staff and ensuring they are familiar with the system's capabilities.

2.2.7 Assumptions and Dependencies

- Users have access to **internet-enabled devices**.

- An active internet connection is required for the chatbot functionality.
- The system will be deployed in a **secured cloud environment**.

2.3 External Interface Requirements

2.3.1 User Interfaces

The system provides a **web-based** user interface accessible via desktop and mobile devices. It features:

- **Dashboard for Users:** Submit and track grievances.
- **Dashboard for Booking**: Resorts are displayed with details.
- **Dashboard for Admin:** Shows the stats about resort bookings.
- **Notifications:** Real-time updates via emails.
- **Help & Support:** In-app Chatbot for answering FAQs and contact options.

2.3.2 Hardware Interfaces

- **Supported Devices:** Desktop computers, smartphones, tablets.
- **Connectivity:** Works over the internet via browsers.
- **Server Infrastructure:** Cloud-hosted with backup storage.

2.3.3 Software Interfaces

- **Operating Systems:** Compatible with Windows, macOS, Linux, Android, iOS.
- **Databases:** MongoDB for structured complaint storage.
- **APIs:** RESTful APIs for third-party integrations.
- **Authentication:** OAuth-based user authentication.

2.3.4 Communications Interfaces

- **Web Browsing:** HTTPS for secure access.
- **Email/SMS:** Notifications for booking updates.
- **Data Exchange:** JSON-based communication between client and server.

\

2.4. System Features

2.4.1 Secure Login System

2.4.1.1 Description and Priority

- **High Priority**: Users can login with their credentials or register themselves.

2.4.1.2 Stimulus/Response Sequences

1. User can perform booking.
2. A unauthorized user cannot perform booking.
3. User receives updates about check-in/check-out.

2.4.1.3 Functional Requirements

- REQ-1: User Registration & Login.
- REQ-2: Resort Browsing & Sorting.

2.4.2 Facility and Service Management

2.4.2.1 Description and Priority

- **High Priority**: Staffs are assigned to users for service management.

2.4.2.2 Stimulus/Response Sequences

1. User can avail the facilities after bookings.
2. Users can contact the chatbot for any queries.

2.4.2.3 Functional Requirements

- REQ-3: Booking Management.
- REQ-4: Food Ordering System.

2.5. Other Nonfunctional Requirements

2.5.1 Performance Requirements

- The system must support at least **500 concurrent users**.
- The response time for bookings submission should not exceed **3 seconds**.
- Booking status updates should be reflected in real-time with a **delay of no more than 5 seconds**.
- The booking portal must be available **24/7 with 99.9% uptime**.

2.5.2 Safety Requirements

- The system must ensure **secure access control** to prevent unauthorized modifications.
- Regular **data backups** should be conducted **daily** to prevent loss of critical complaint records.
- The system should provide **role-based access restrictions** to prevent unauthorized data exposure.
- Disaster recovery mechanisms must be in place to restore operations within **30 minutes** of a system failure.

2.5.3 Security Requirements

- All **user authentication** must be handled via **OAuth 2.0** or multi-factor authentication (MFA).
- Data transmission should be **encrypted using TLS 1.3** to prevent data interception.
- User passwords must be stored using **bcryptjs hashing** with a high iteration count.
- Complaint data should only be accessible to authorized personnel.
- The system must log all activities for **audit tracking and compliance verification**.

2.5.4 Software Quality Attributes

- **Availability:** The system should maintain an uptime of **99.9%**.
- **Scalability:** The platform must support an increasing number of complaints without performance degradation.
- **Usability:** The interface must be designed with **accessibility standards (WCAG 2.1)** for ease of use.
- **Maintainability:** The system should be modular and allow **future enhancements without major rewrites**.
- **Portability:** The portal should be accessible on **desktop, tablets, and mobile devices**.
- **Interoperability:** The system should allow **API integration with third-party applications**.

2.5.5 Business Rules

- Only **registered users** can perform bookings..
- Citizens can provide **feedback and ratings**.
- Only **authorized personnel** can modify resorts and assign tasks.
- **Data privacy laws and local regulations** must be adhered to in data handling.

2.6. Other Requirements

- The RMS will utilize a NoSQL database, such as MongoDB.
- The system should be capable of handling concurrent requests from multiple users without noticeable delays.

Appendix A: Glossary

- **RMS (Resort Management System):** The web-based application for Resort Management.
- **User:** A citizen who submits a complaint.
- **Admin:** A official responsible for assigning and monitoring resorts.
- **External Assistance:** Third-party specialists handling escalated grievances.
- **API (Application Programming Interface):** A set of functions allowing the system to communicate with other applications.
- **OAuth:** A secure authentication protocol used for user login and verification.
- **HTTPS (Hypertext Transfer Protocol Secure):** A protocol for secure communication over a computer network.
- **JSON (JavaScript Object Notation):** A lightweight data-interchange format used for communication between system components.

CHAPTER 3

SOFTWARE DESIGN DOCUMENT

3.1 Requirements

This section outlines the estimated effort for the key development modules of the Resort Management System. The estimates provide a ballpark figure for the initial frontend development phase, as described in the project documentation.

3.3.1 Estimates

S.NO	Description	Hrs. Est.
1	User Registration and Authentication (JWT + bcrypt)	10
2	Resort Browsing & Filtering (Cost/Rating-based Sorting)	20
3	Booking & Availability Management (Multi-room, Prevent Overlap)	25
4	Food Ordering & Wallet Deduction Workflow	15
5	Staff Auto-assignment & Management Module	10
6	PDF Invoice Generation (with Professional Layout)	10
7	Review & Rating System (Star-based Feedback)	10
8	Chatbot Integration, Database Setup & Optimization (MongoDB Collections & Indexes)	10
8	Admin Dashboard (User Wallet Update, Resort CRUD, Stats)	10
9	Deployment & Testing	20
	TOTAL	**140**

Table 3.1 Estimates

3.3.2 Traceability Matrix

The table below cross-references the SRS requirements with the corresponding SDD modules to ensure complete coverage

SRS Requirement	SDD Module
Req 1- User Registration & Login	5.1.1 (Authentication Module)
Req 2 - Resort Browsing & Sorting	5.1.2 (Resort Listing & Filter Module)
Req 3 - Booking Management	5.1.3 (Booking Engine)
Req 4 - Food Ordering System	5.1.4 (Food Order Module)
Req 5 - Checkout & Invoice Generation	5.1.5 (Invoice & Billing System)
Req 6 - Review & Rating Submission	5.1.6 (Feedback Module)
Req 7 - Chatbot Assistant	5.1.7 (AI Chatbot Interface)
Req 8 - Admin Dashboard	5.1.8 (Admin Control Panel)
Req 9 - Staff Assignment & Management	5.2.1 (Staff Management System)
Req 10 - Database Setup & Optimization	5.2.2 (Deployment Plan)
Req 11- Deployment & Testing	5.3.1(Deployment Plan & Test Automation)

Table 3.2 Traceability matrix

3.2 System Architecture

The system architecture of the Urban Grievance Portal follows a modular and layered design. It consists of the following components:

- **Frontend Layer:** User Interface built using React.js for accessibility and responsiveness.
- **Backend Layer:** Powered by Node.js and Express.js, exposing a RESTful API that handles authentication.
- **Database Layer: I**mplemented using MongoDB, storing structured collections such as users, resorts, bookings, orders, wallets, staff, and reviews.
- **Authentication:** OAuth 2.0-based user authentication using JWT.
- **Payment & Wallet System: Integrated wallet balance management for quick payments**

and refund.

3.3 Data Dictionary

The data dictionary includes a brief description of each element used in this module.

Field	Notes	Type
USER_ID	Unique identifier for each registered user	ObjectId
NAME	Full name of the user	VARCHAR
PASSWORD	Encrypted password using bcrypt	VARCHAR
EMAIL	User email for contact	VARCHAR
ROLE	Defines user type (User / Admin / Staff)	VARCHAR
RESORT_ID	Unique identifier for each resort	ObjectId
RESORT_NAME	Name of the resort	VARCHAR
BOOKING_ID	Unique booking reference number	ObjectId
CHECKIN_DATE	Date of check-in	DATE
CHECKOUT_DATE	Date of check-out	DATE
TOTAL_AMOUNT	Total payable amount for the booking	DECIMAL
WALLET _BALANCE	Current balance in user's wallet	DECIMAL
FOOD_ORDER_ID	Unique ID for food orders	ObjectId
ORDER_ITEMS	List of ordered food items	ARRAY
STATUS	Current state (Pending / Confirmed / Completed / Cancelled)	VARCHAR
REVIEW_RATING	Rating given by user (1–5 stars)	INT
CREATED_AT	Timestamp of record creation	TIMESTAMP
UPDATED_AT	Timestamp of last modification	TIMESTAMP

Table 3.3 Data Dictionary

3.4 ANALYSIS MODELS

3.4.1 USECASE DIAGRAM

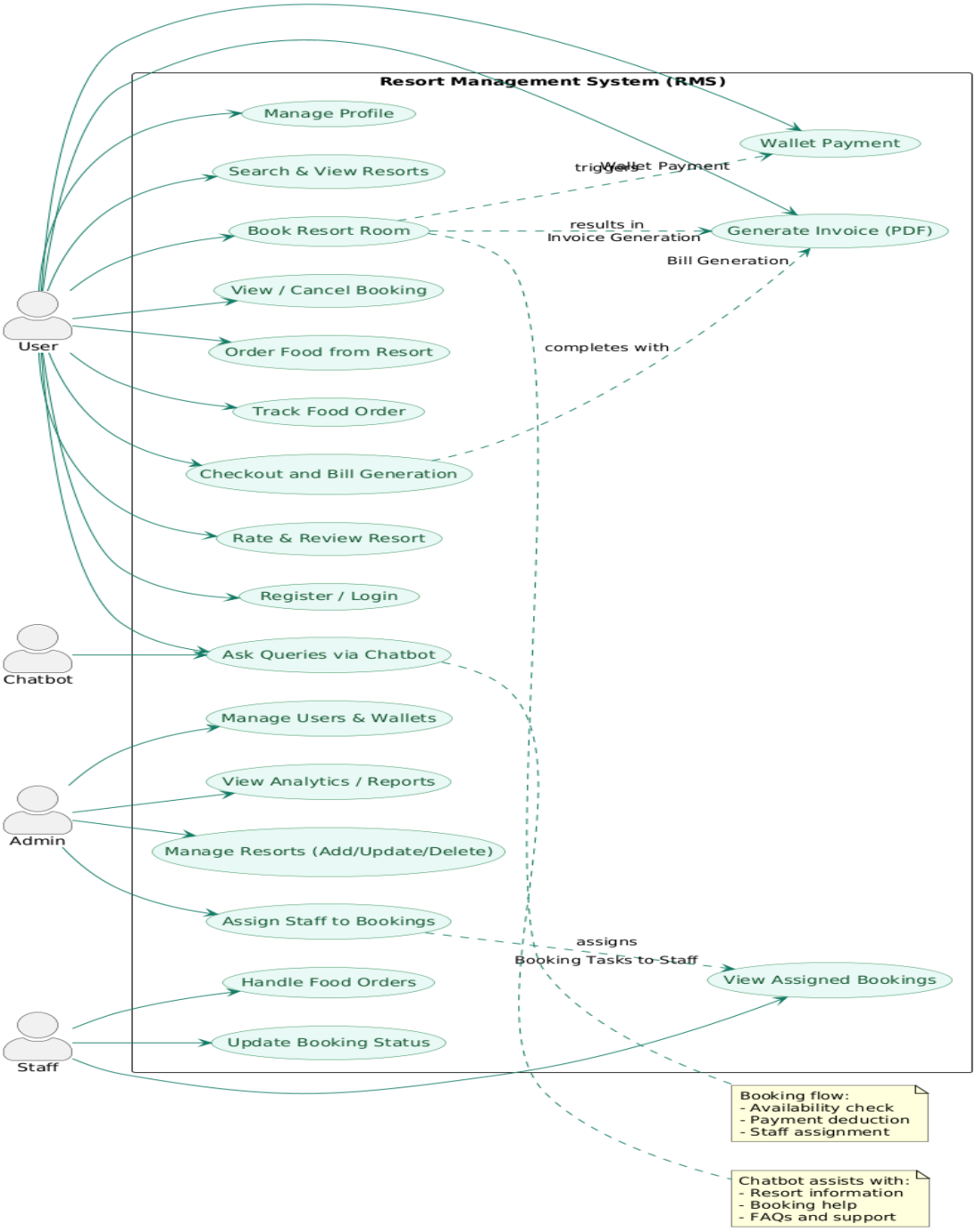


Figure 3.4.1: UseCase Diagram of Resort Management System

3.4.2 ACTIVITY DIAGRAM

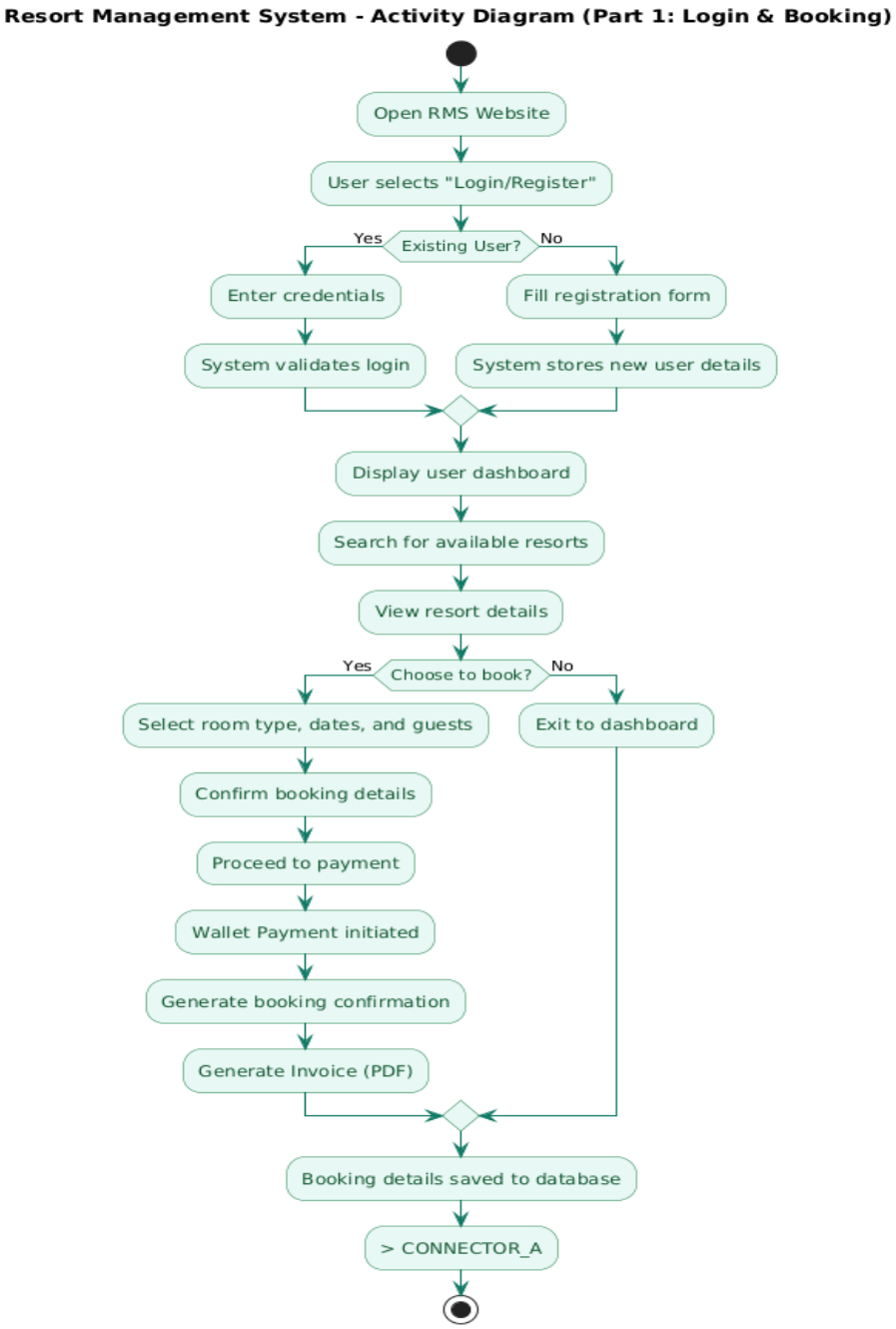


Figure 3.4.2: Activity Diagram of Login & Booking

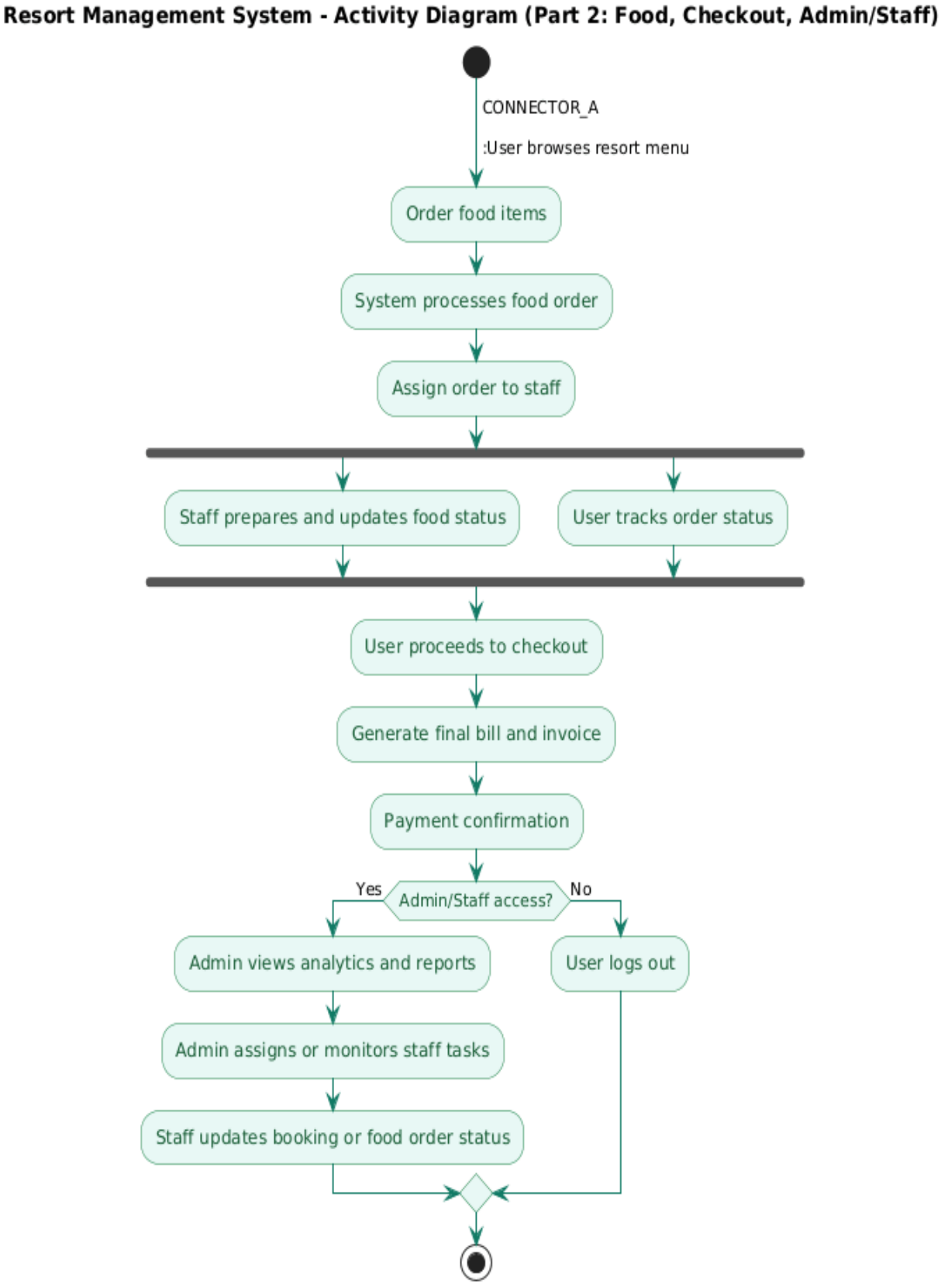


Figure 3.4.3: Activity Diagram of Food Order& Checkout

3.4.3 SEQUENCE-DIAGRAM

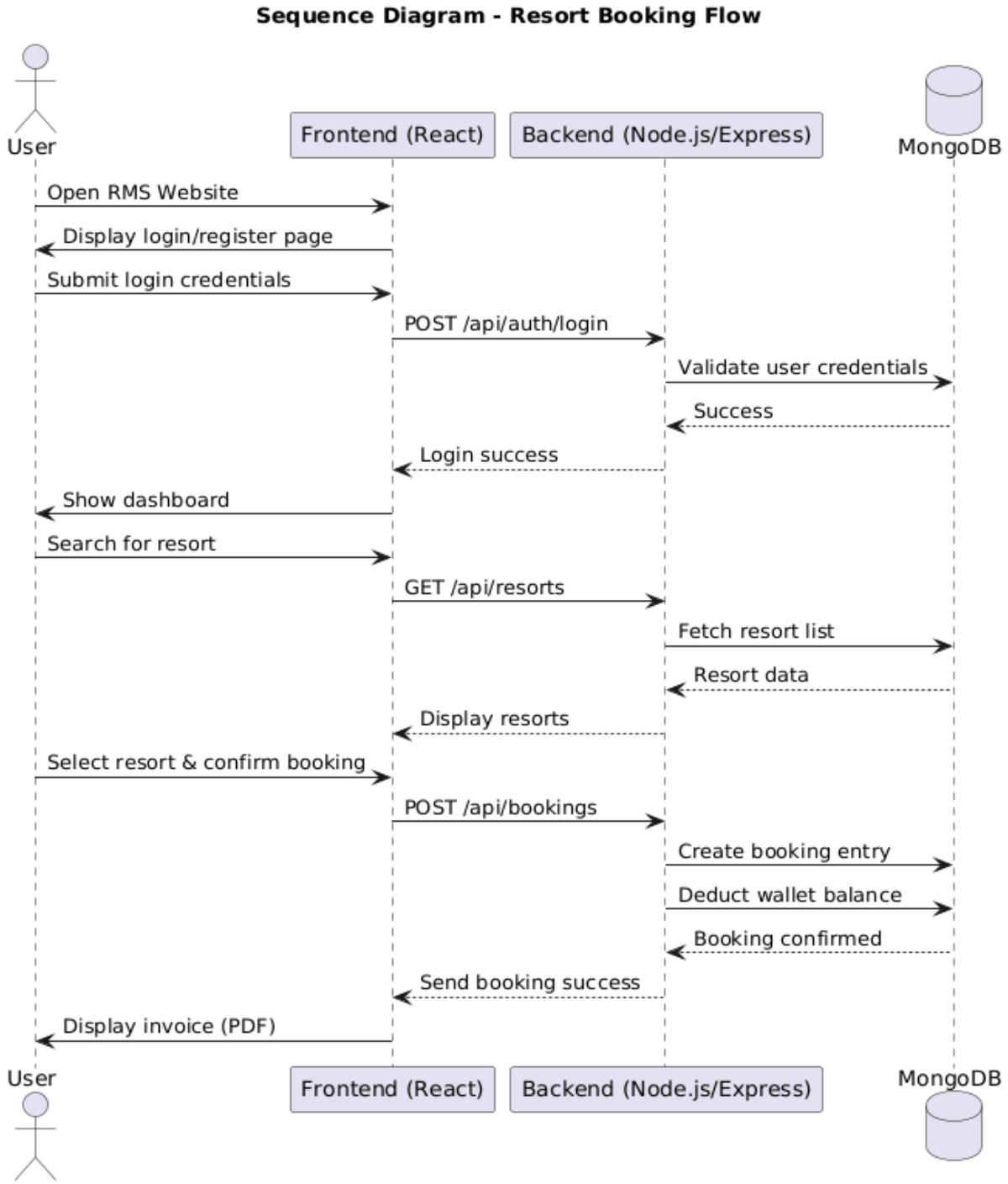


Figure .3.4.4: Sequence Diagram Of Resort Management System

3.4.4 ER-DIAGRAM

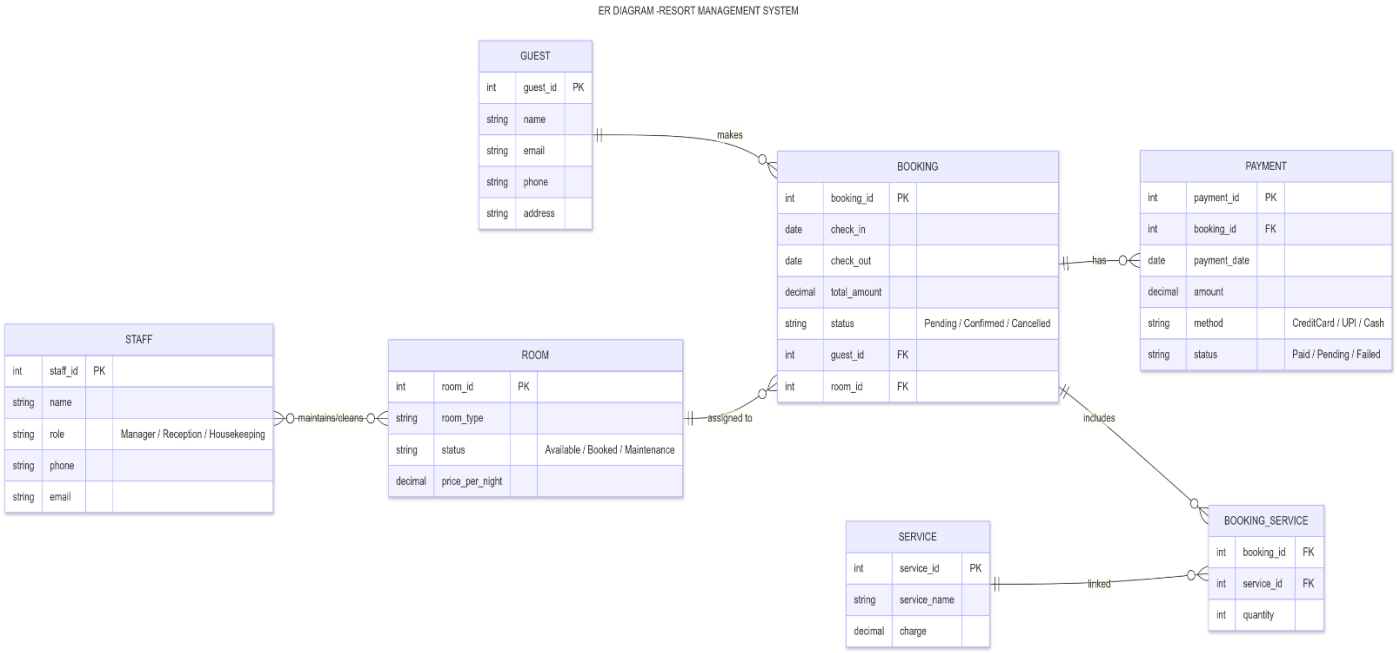


Figure 3.4.5: ER Diagram Of Resort Management System

3.4.5 CLASS DIAGRAM

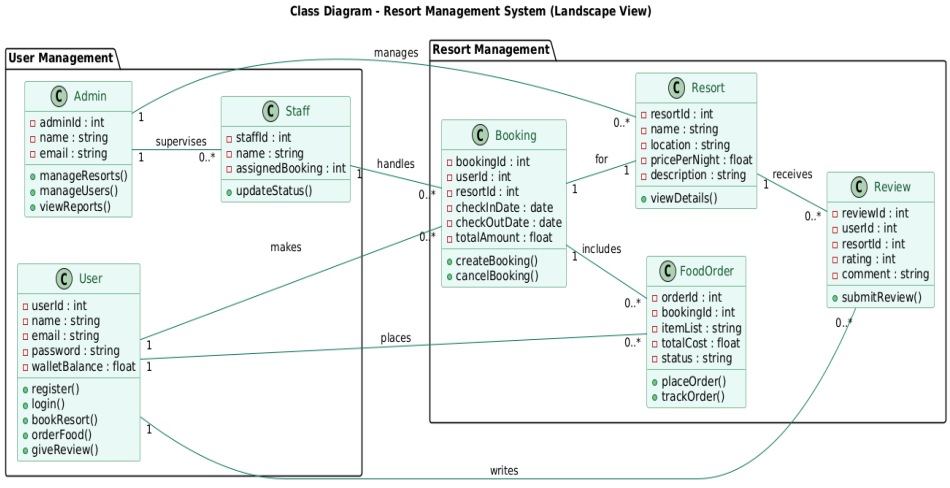


Figure 3.4.6: Class Diagram Of Resort Management System

3.4.6 LOW LEVEL DESIGN

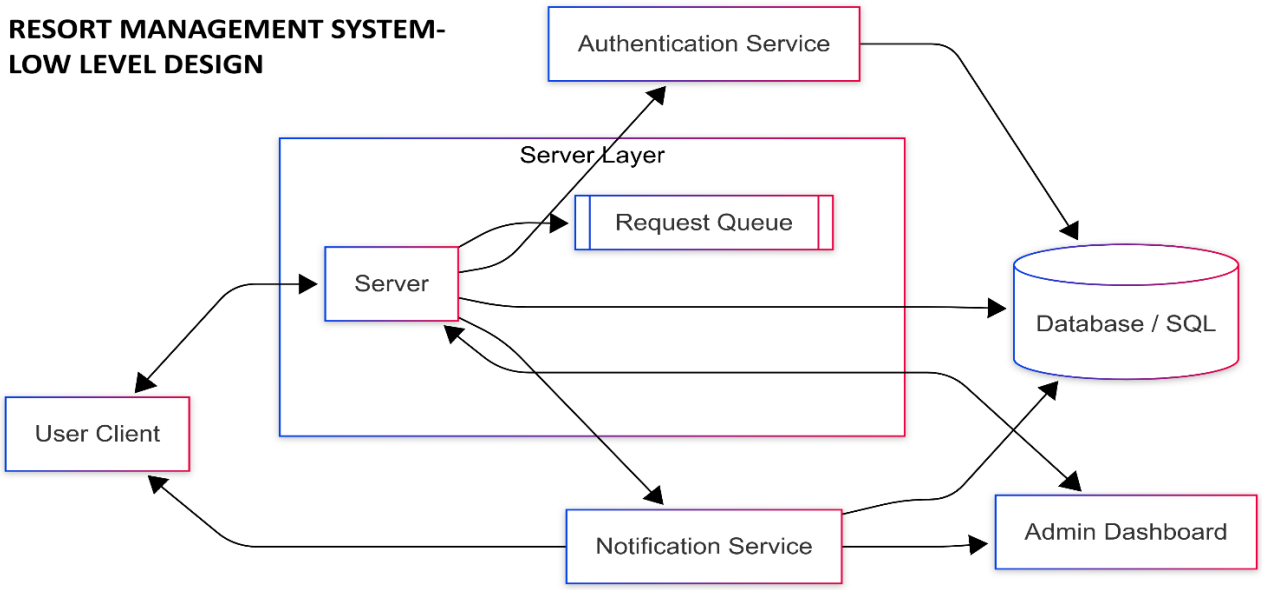


Figure 3.4.7:low level Design Of Resort Management System

3.4.7 HIGH LEVEL DESIGN

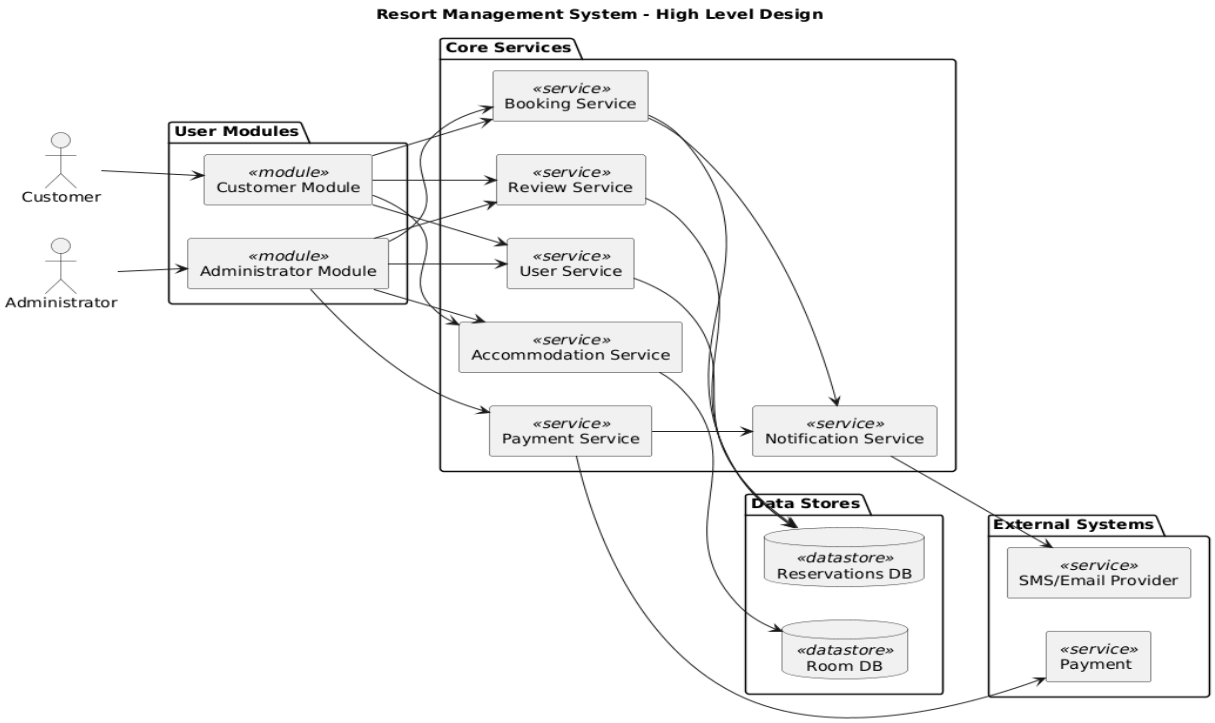


Figure 3.4.8: low level Design Of Resort Management System

3.4.8 DATA FLOW (LEVEL 0) DIAGRAM

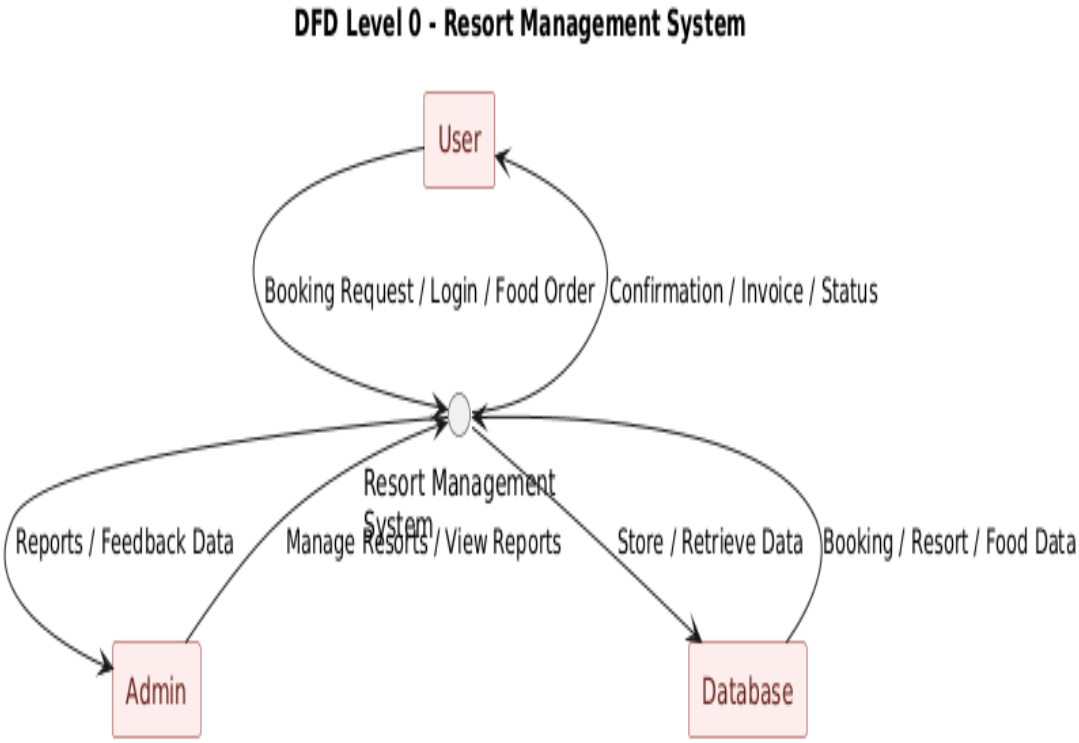


Figure 3.4.8:Data Flow Level 0 Of Resort Management System

3.4.9 DATA FLOW DIAGRAM (LEVEL 1) DIAGRAM

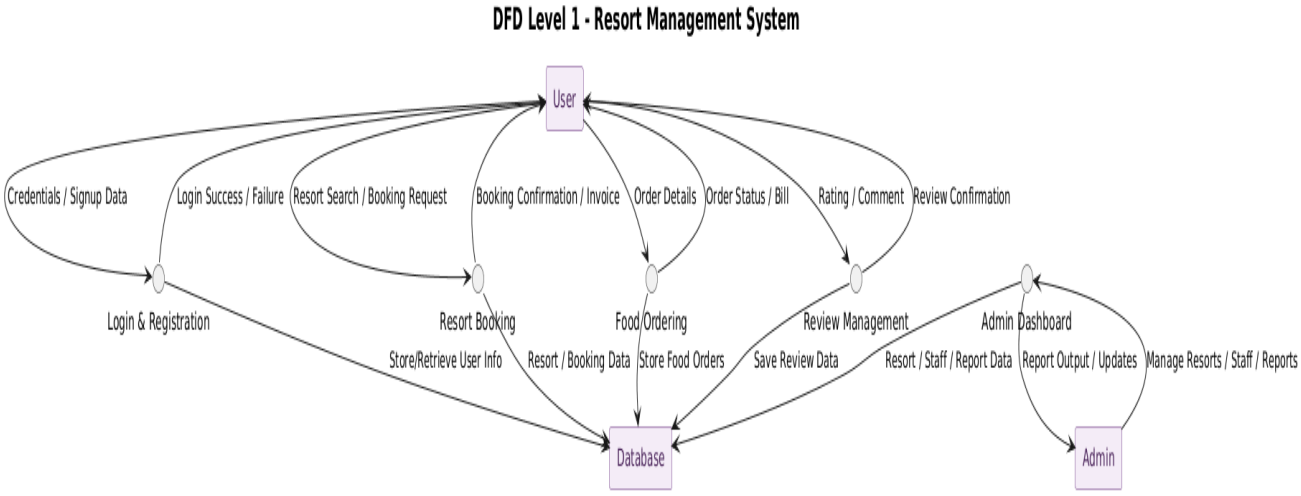


Figure 3.4.8:Data Flow Level 1 Of Resort Management System

3.4.10 DATA FLOW DIAGRAM (LEVEL 2) DIAGRAM

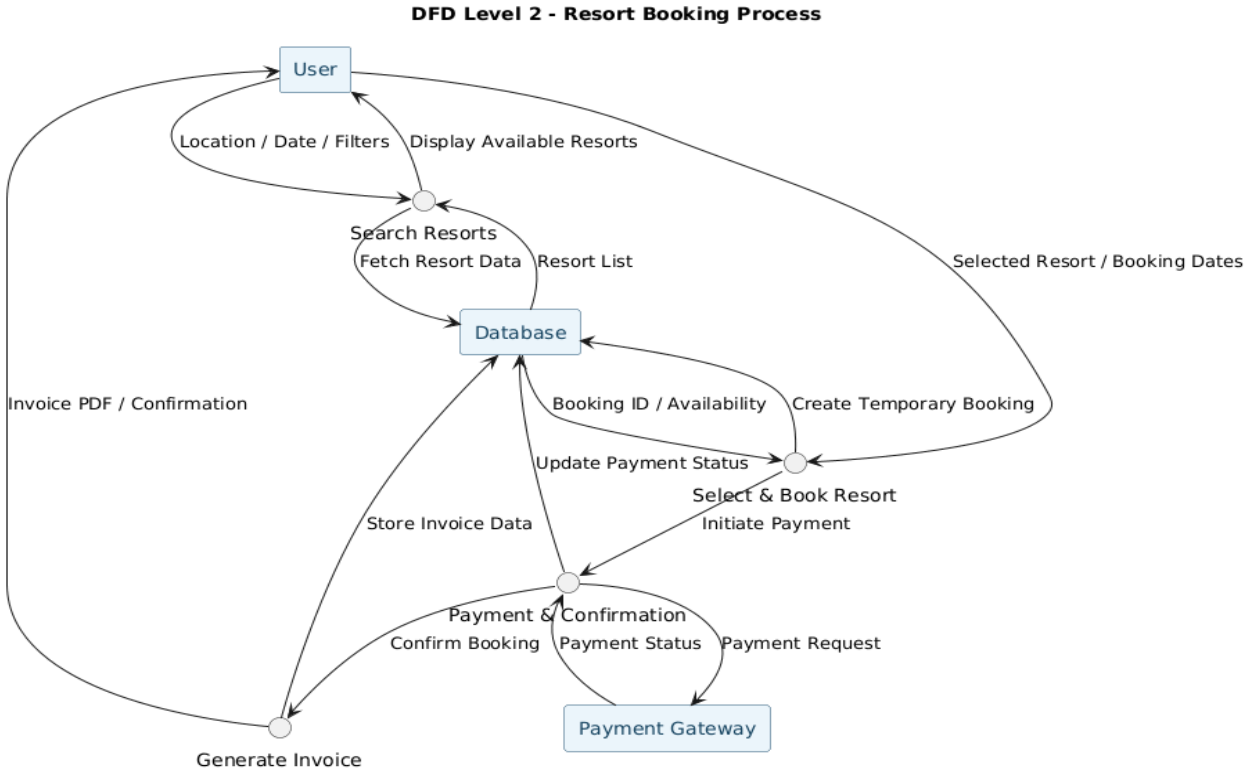


Figure 3.4.8:Data Flow Level 2 Of Resort Management System

3.4.11 COMPONENT DIAGRAM

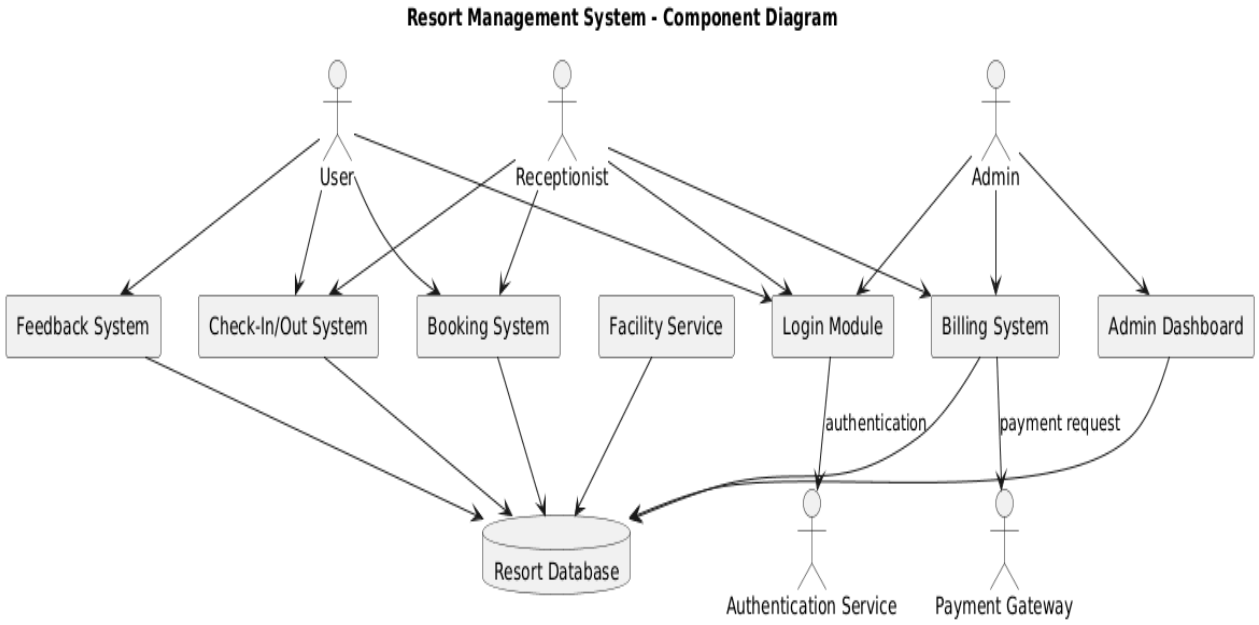


Figure 3.4.9: Class Diagram Of Resort Management System

3.4.12 DEPLOYMENT DIAGRAM

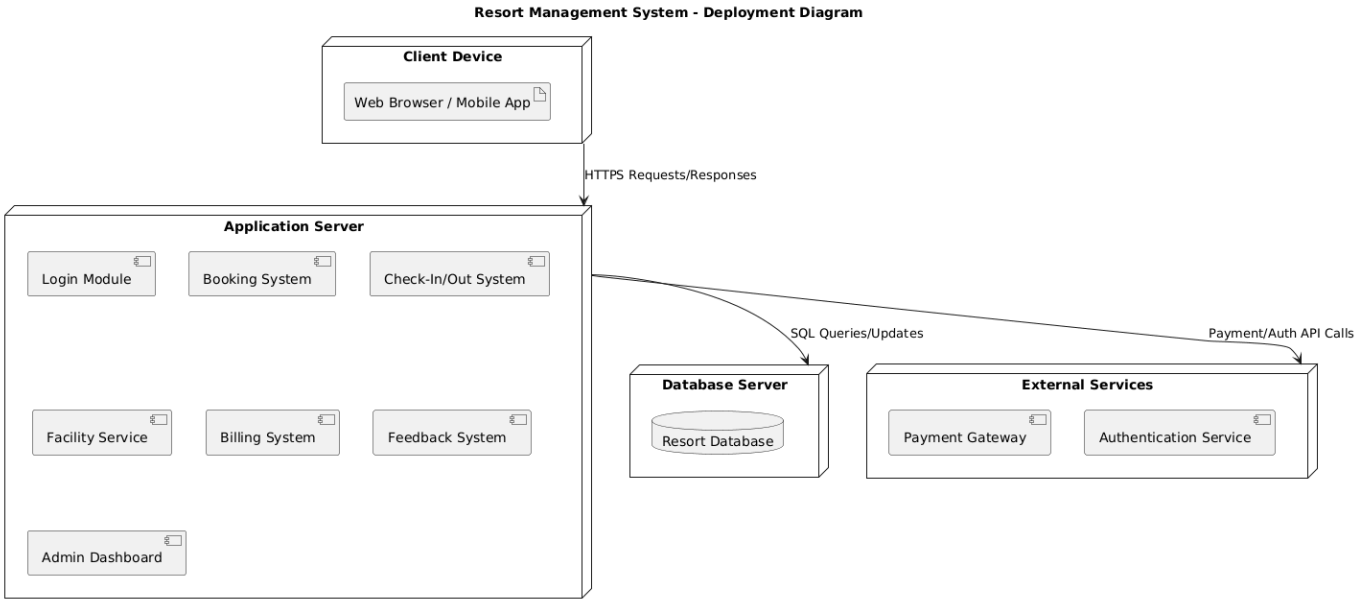


Figure 3.4.10: Deployment Diagram of Resort Management System

3.4.13 DEPLOYMENT DIAGRAM

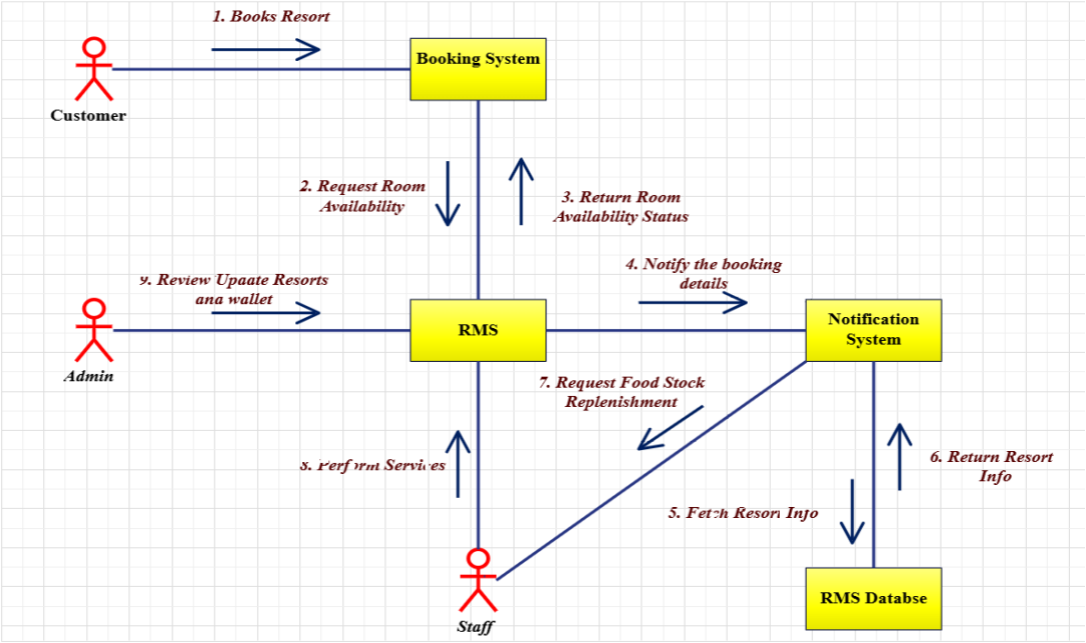


Figure 3.4.10: Collaboration Diagram of Resort Management System

CHAPTER 4

TESTING AND INTEGRATION

4.1 Testing Strategy

A comprehensive testing strategy ensured reliability and usability within the 3-month timeline.

4.1.1 Unit Testing

- **Purpose**: Verified individual components and isolated modules of the Resort Management System.

.

- **Tools**:
    - **JUnit** and **Mockito** for Spring Boot: Tested service and repository layers (e.g,ResortService, BookingService, UserRepository, BookingRepository).
    - **Jest** with **React Testing Library** for React: Tested UI components (e.g., login form, resort search component, booking confirmation modal).
- **Coverage**: Achieved 75% code coverage, focusing on core features like grievance submission.
- **Example**: Tested the submitGrievance API endpoint for database persistence.

4.1.2 Integration Testing

- **Purpose**: Validated component interactions.
- **Tools**:
    - **Spring Boot Test** with an H2 database for API endpoints (e.g., /issue/submit).
    - **Cypress** for end-to-end React frontend testing.
- **Focus**: Ensured seamless integration, such as grievance submission from React to Spring Boot.
- **Example**: Tested the createBooking API method to ensure correct database persistence and wallet balance deduction..

4.1.3 System Testing

- **Purpose**: Confirmed overall functionality and performance.
- **Approach**:
    - Tested end-to-end workflows (e.g., user login, searching for resorts, selecting a resort, confirming a booking, and viewing the invoice).
    - Used **JMeter** to verify 200 concurrent users with a response time under 2 seconds.

- **Example**: Simulated 50 users simultaneously searching for resorts and attempting to book, ensuring the system remained stable and responsive under expected load..

4.1.4 User Acceptance Testing (UAT)

- **Purpose**: Ensured that the Resort Management System met the stakeholders' requirements and was user-friendly.
- **Approach**:
    - Conducted UAT with 5–10 users (potential customers) in the final week.
    - Focused on usability (e.g., accurate resort search results, correct booking confirmation) and functionality
- **Feedback**: Implemented minor UI tweaks (e.g., clearer button labels for "Book Now," improved navigation on the dashboard) pre-launch.

4.2 Cyclomatic Complexity Analysis

Cyclomatic complexity analysis was conducted to measure the complexity of the code and guide testing efforts within the 3-month timeline.

4.2.1 Purpose

- Assessed the number of linearly independent paths in the codebase to identify complex modules requiring thorough testing.
- Used to optimize unit and integration testing by focusing on high-complexity areas.

4.2.2 Methodology

- **Calculation**: Cyclomatic complexity (V(G)) = E - N + 2P, where:
    - E = number of edges in the control flow graph
    - N = number of nodes
    - P = number of connected components (typically 1 for a single method)
- **Tools**: Utilized **SonarQube** and **CodeClimate** to analyze Spring Boot services (e.g., GrievanceService) and React components.
- **Threshold**: A complexity score above 10 was considered high, indicating a need for additional test cases.

4.2.3 Findings

- **Spring Boot**: The createBooking method within BookingService, which involves multiple conditional checks (e.g., user authentication, resort availability, wallet balance validation, transaction processing), had a complexity of 12. This suggested 12 distinct test paths were necessary to cover all scenarios, including success, insufficient balance, unavailable resort, etc..

- **React**: The resort search filter component, with conditional rendering based on multiple search criteria and state management, had a complexity of 8, which was manageable with existing Jest tests.

4.2.4 Benefits

- Identified critical code sections, reducing the risk of bugs in complex workflows.
- Supported the 3-member team's testing efforts by prioritizing resources on high-complexity areas within the tight timeline.

4.3 Applicability of Black-Box and White-Box Testing

4.3.1 Black-Box Testing

Black-box testing, focusing on inputs and outputs without code insight, was highly applicable:

- **Relevance**:
    - **User-Centric Features**: Validated functionalities like grievance submission and report generation from a user perspective, critical for public users, municipal officers, and field agents.
    - **UAT and System Testing**: Applied to ensure the system met requirements (e.g., successfully logging in, finding specific resorts, completing a booking) as experienced by stakeholders.
    - **API Testing**: Tested Spring Boot APIs (e.g., /api/auth/login, /api/resorts,/api/bookings) by sending requests and verifying responses, focusing on functionality.
- **Example**: In UAT, users would navigate the React frontend to log in, search for a resort, select it, confirm the booking, and verify the displayed invoice, all without knowledge of the underlying backend logic.

4.3.2 White-Box Testing

White-box testing, leveraging internal code knowledge, was equally applicable:

- **Relevance**:
    - **Unit Testing**: The team used white-box testing with JUnit and Jest to test Spring Boot services (e.g., if-else conditions in BookingService for balance checks, authentication logic in AuthService) and React components, ensuring code paths were covered.
    - **Integration Testing**: Applied to debug interactions between React and Spring Boot, such as ensuring fetch calls mapped correctly to APIs.
    - **Optimization**: Within the 3-month timeline, white-box testing identified performance bottlenecks, like optimizing database queries in Spring Boot.

- **Example**: Wrote a JUnit test for the createBooking method to explicitly execute all error-handling branches (e.g., insufficient wallet balance, resort not found, invalid user), requiring detailed knowledge of its internal logic**.**

4.3.3 Combined Approach

Both approaches complemented each other:

- **Black-Box in UAT/System Testing**: Ensured functional and usability requirements from an external perspective.
- **White-Box in Unit/Integration Testing**: Ensured code-level correctness and integration, vital for the Spring Boot-React architecture.
- **Balance**: The tight 3-month schedule benefited from white-box testing for efficiency and black-box testing for user validation, ensuring a robust system.

4.4 Test Cases

Scenario 1:Authentication Module

Test Case ID	Test Description	Input	Expected Output	Result
TC01	Login with valid admin credentials	admin@resort.com / admin123	Dashboard loads successfully	**Pass**
TC02	Login with invalid password	admin@resort.com / wrongpass	Error: Invalid credentials	**Pass**
TC03	Register new user	user@demo.com / user123	Registration success message	**Pass**
TC04	Login with newly registered user	user@demo.com / user123	Redirect to Home	**Paas**
TC05	Access dashboard without token	No Authorization header	Access denied / 401	**Pass**

Table 4.1: Testing of Authentication module

Scenario 2:Resort Listing Module

Test Case ID	Test Description	Input	Expected Output	Result
TC01	View all resorts	GET /api/resorts	List of resorts displayed	**Pass**
TC02	Sort resorts by price (High to Low)	Click sort by price	Sorted by descending price	**Pass**
TC03	Filter resorts by city	Enter 'Chennai'	Resorts from Chennai displayed	**Pass**
TC04	Image loading check	Load resort image	Image loads from /uploads/resorts/	**Paas**
TC05	View resort details	Click on resort card	Details page displayed	**Pass**

Table 4.2: Testing of Resort Listing module

Scenario 3:Booking Module

Test Case ID	Test Description	Input	Expected Output	Result
TC01	Book a valid resort	City, Check-in, Check-out, Guests	Booking confirmed message	**Pass**
TC02	Book resort with invalid dates	Check-out < Check-in	Sorted by descending price	**Pass**
TC03	Check staff assignment	Book resort	Staff assigned automatically	**Pass**
TC04	View My Bookings list	GET /api/bookings/my	List of user's bookings	**Paas**

Table 4.3: Testing of Booking module

Scenario 4:Food Order Module

Test Case ID	Test Description	Input	Expected Output	Result
TC01	Fetch food menu	GET /api/food	List of food items	**Pass**
TC02	Add food item to booking	POST food ID	Order success message	**Pass**
TC03	Add invalid food ID	POST invalid ID	Error: Food not found	**Pass**
TC04	Remove food from cart	Click remove button	Item removed	**Paas**
TC05	Total calculation check	Add 3 items	Sum of 3 item prices	**Pass**

Table 4.4: Testing of Food Order module

Scenario 5:Checkout Module

Test Case ID	Test Description	Input	Expected Output	Result
TC01	Checkout with sufficient wallet	**Wallet >= total price**	Checkout complete	**Pass**
TC02	Checkout with insufficient wallet	Wallet < total price	Error: Insufficient balance	**Pass**
TC03	Invoice PDF generation	Click 'Download Invoice'	Invoice saved in /exports/	**Pass**
TC04	Wallet deduction validation	Before/after booking	Wallet reduced by booking amount	**Paas**
TC05	Invoice format verification	Open PDF	Shows GST, details, logo	**Pass**

Table 4.5: Testing of Checkout module

Scenario 6:Chatbot Module

Test Case ID	Test Description	Input	Expected Output	Result
TC01	Ask booking question	**How to book?**	Bot gives booking steps	**Pass**
TC02	Ask about refund policy	What is refund policy?	Bot replies with info	**Pass**
TC03	Ask invalid question	**asdasd?**	Bot gives default reply	**Pass**
TC04	UI visibility	**Click chat icon**	Chat window toggles	**Paas**
TC05	Close chat window	Click close button	Chat window hides	**Pass**

Table 4.6: Testing of Food Order module

Scenario 7:Admin Module

Test Case ID	Test Description	Input	Expected Output	Result
TC01	Add new resort	POST resort data	Resort added successfully	**Pass**
TC02	Update resort details	PUT /api/resorts/:id	Resort updated	**Pass**
TC03	Delete resort	DELETE /api/resorts/:id	Resort deleted	**Pass**
TC04	View booking stats	GET /api/admin/stats	Stats displayed	**Paas**
TC05	Update user wallet	POST /api/admin/wallet	Wallet updated successfully	**Pass**

Table 4.7: Testing of Admin module

CHAPTER 5

PROJECT METRICS

5.1 Effort Estimation Using COCOMO Model

The Constructive Cost Model (COCOMO) estimated effort, development time, and staffing for the urban grievance project.

5.1.1 Project Classification

Classified as a **Semi-Detached project** due to moderate complexity, including user registration/login, resort booking, food order management, payment processing, admin management, and invoice generation requiring coordination between backend (Node.js/Express), frontend (React), and a database (MongoDB).

COCOMO Parameters

Using the Intermediate COCOMO model for a Semi-Detached project:

- **Effort (E) = a × (KLOC)^b × EAF**
    - a = 3.0 (coefficient for Semi-Detached)
    - b = 1.12 (exponent for Semi-Detached)
    - KLOC = 6
    - EAF (Effort Adjustment Factor) = 1.10 (moderate tool support , good team cohesion)
- **Development Time (D) = c × (E)^d**
    - c = 2.5
    - d = 0.35

5.1.2 Calculations

1. **Effort Calculation**:
    - E = 3.0 × (6)^1.12 × 1.10
    - E = 3.0 × 7.11 × 1.10 ≈ 23.46 person-months
2. **Development Time**:
    - D = 2.5 × (23.46)^0.35
    - D = 2.5 × 2.45 ≈ 6.13 months

3. **Staffing**:
    - Average staff = 23.46 / 6.13 ≈ 3.83 ≈ 4 people

5.1.3 Validation with Real Data

The estimate (6.13 months, 4 people) differs from the actual 3 months and 3-member team, indicating high productivity. Adjusting EAF to 0.9 to reflect efficiency:

- E = 3.0 × 7.11 × 0.85 ≈ 18.14 person-months
- D = 2.5 × (18.14)^0.35 ≈ 2.5 × 2.32 ≈ 5.80 months
- Staffing = 18.14 / 5.80 ≈ 3.13 ≈ 3 people, aligning with reality.

The adjusted COCOMO estimate suggests **17.39 person-months**, **5.58 months**, and a team of **3 developers**, reflecting efficient execution within the 3-month timeline.

5.2 Function Points (FP) Estimation

Function Points (FP) analysis estimated the project's size based on functionality.

5.2.1 FP Components

The system's functionalities include:

1. **External Inputs (EI)**: 5 inputs (e.g., User Login, Register, Resort Booking, Food Order, Payment, Review Submission), rated as average complexity (4 FP each).
    - 6× 4 = 24 FP
2. **External Outputs (EO)**: 3 outputs (e.g., Booking Confirmation, Invoice Generation, Admin Reports, Notifications), rated as average complexity (5 FP each).
    - 4 × 5 = 20 FP
3. **External Inquiries (EQ)**: 4 inquiries (e.g., View Available Resorts, Booking History, Order Status, Resort Details), rated as average complexity (4 FP each).
    - 4 × 4 = 16 FP
4. **Internal Logical Files (ILF)**: 3 logical files (e.g., User, Resort, Booking, FoodOrder collections), rated as average complexity (10 FP each).
    - 4 × 10 = 40 FP
5. **External Interface Files (EIF)**: 1 interface (e.g., Payment Gateway / Email Notification Service), rated as simple complexity (5 FP).
    - 1 × 5 = 5 FP

Unadjusted Function Points (UFP) = 24 + 20 + 16 + 40 + 5 = 105 FP

5.2.2 Complexity Adjustment

With a Value Adjustment Factor (VAF) score of 35 (moderate influence),

VAF = 0.65 + (0.01 × 35) = 1.0.

Adjusted Function Points (FP) = 105 × 1.0 = 105 FP

5.2.3 Effort Estimation Using FP

Assuming a productivity rate of 15 FP per person-month (adjusted for the 3-month completion):

- **Effort** = 105 FP / 15 FP per person-month ≈ 7
- person-months
- **Development Time**: With 3 developers, Time = 7 / 3 ≈ 2.33 months

The FP analysis estimates **86 Function Points**, requiring **7 person-months** and 2**.3 months** with a 3-person team, aligning closely with the actual 3-month timeline.

CHAPTER 6

OUTPUTS

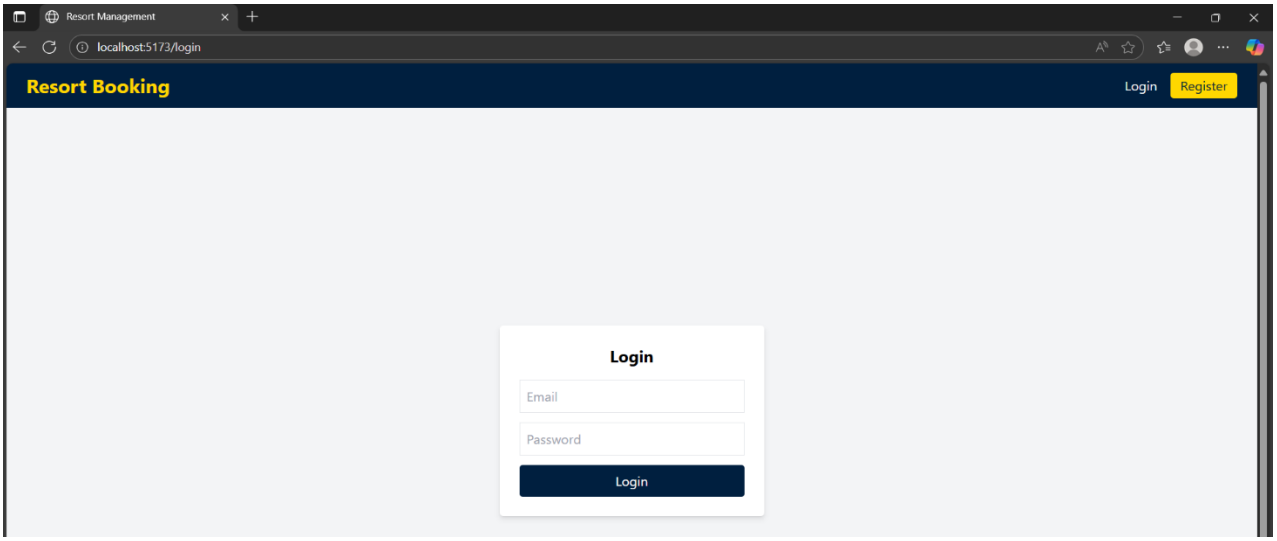


Figure 6.1: Login Page

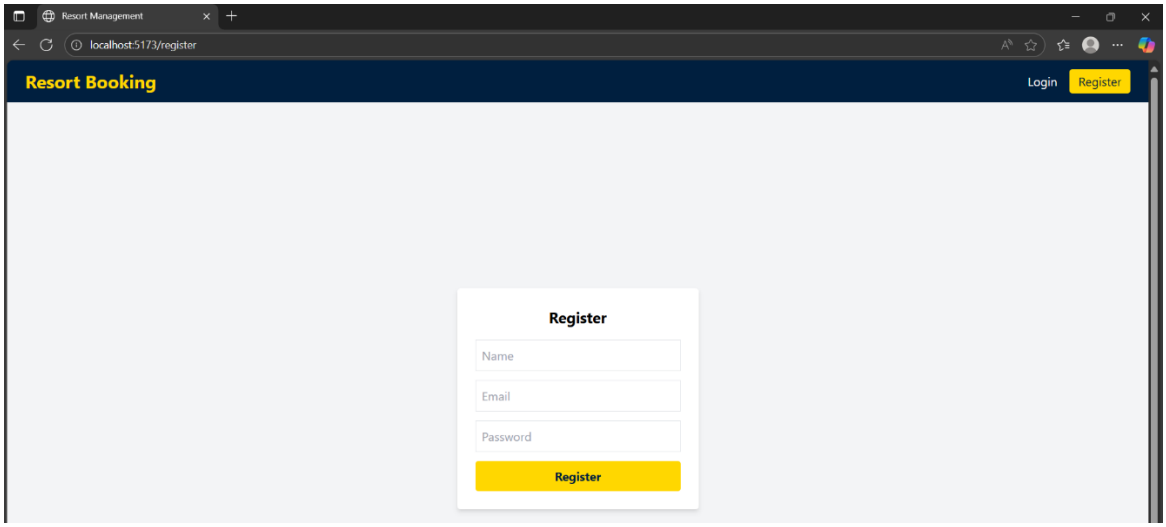


Figure 6.2: Registeration Page

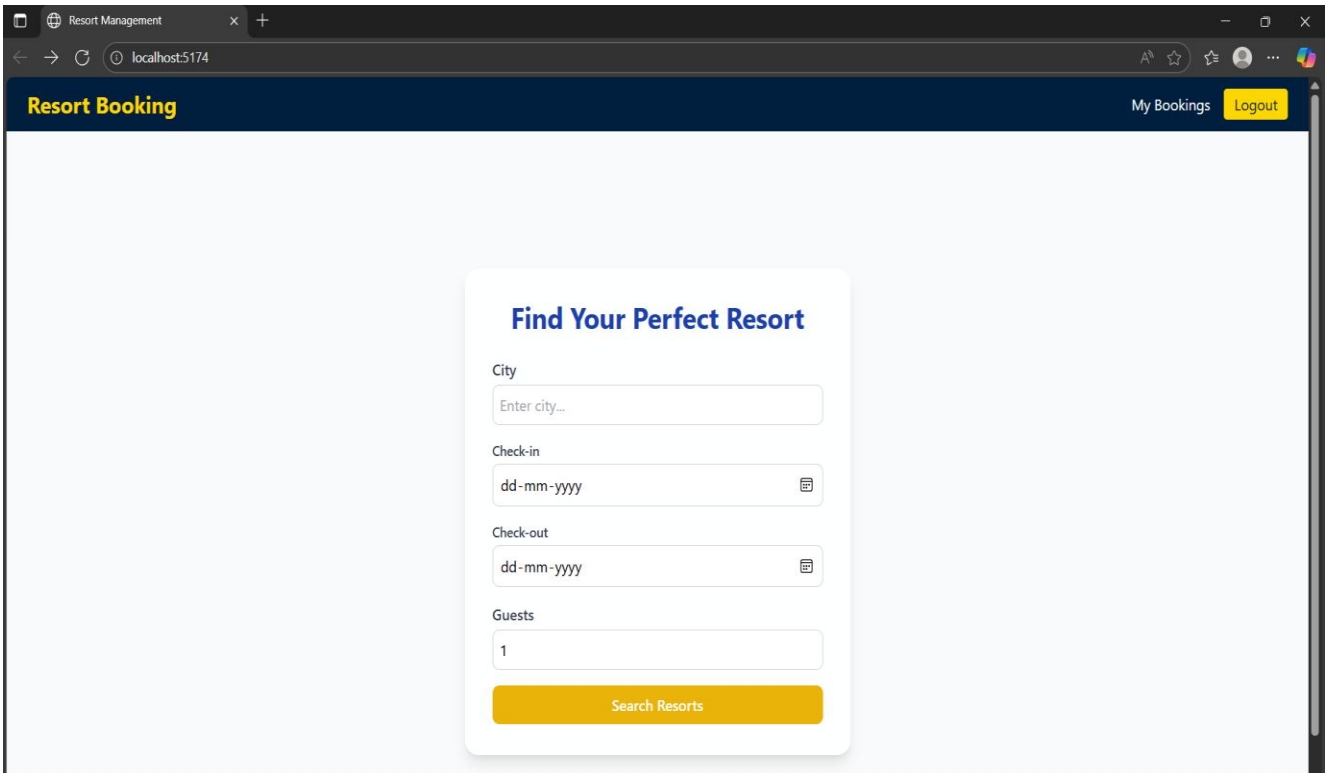


Figure 6:3: Home Page

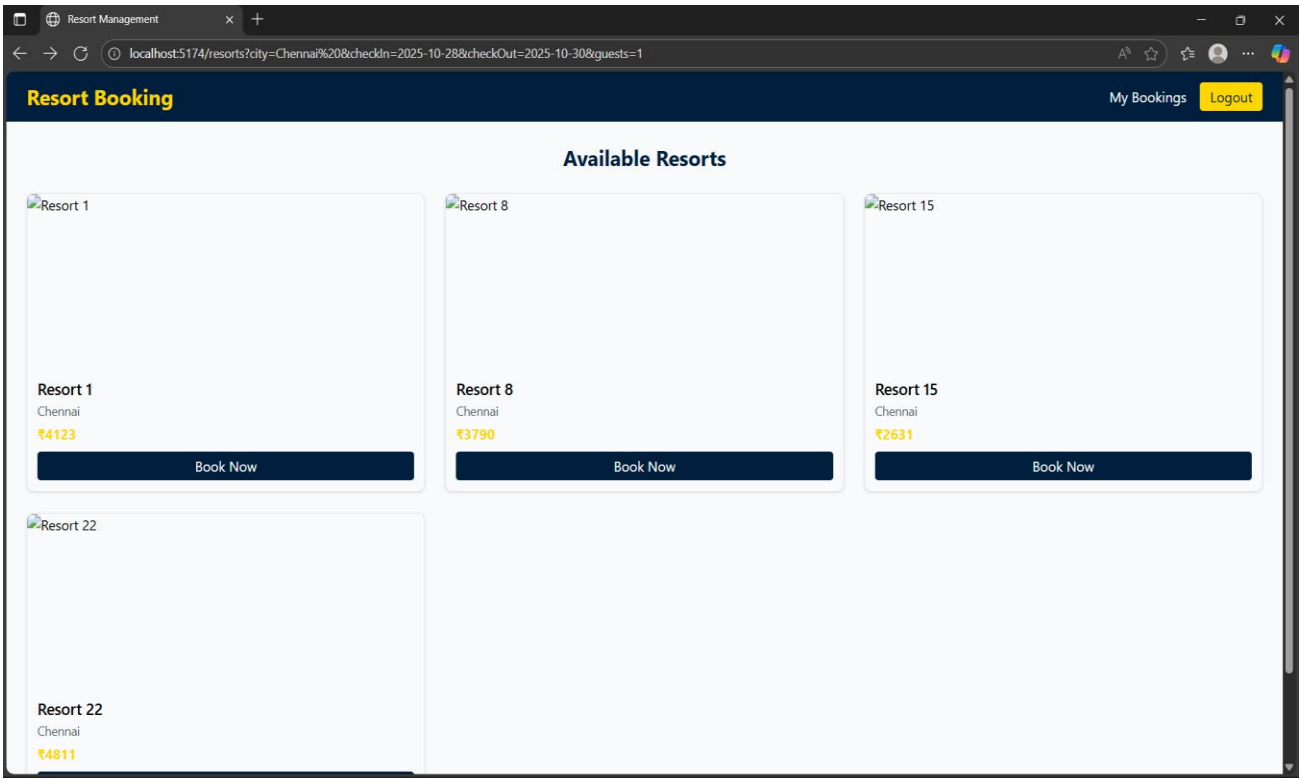


Figure 6.4:Resort Listing Page

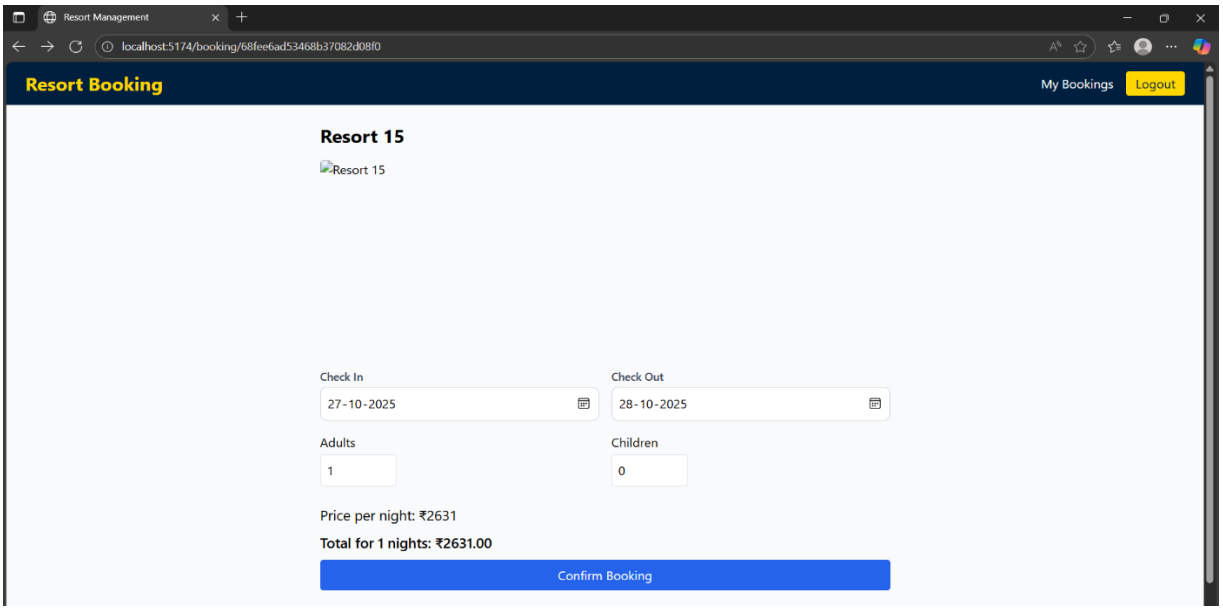


Figure 6.5: Resort Booking Page

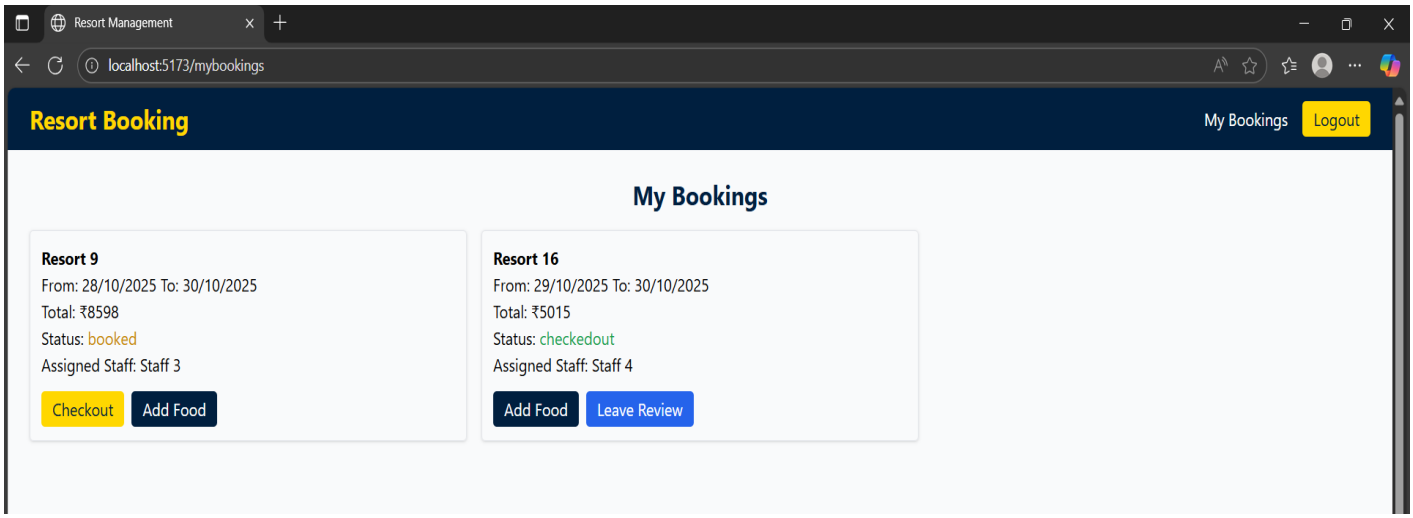


Figure 6.6:My Bookings Page

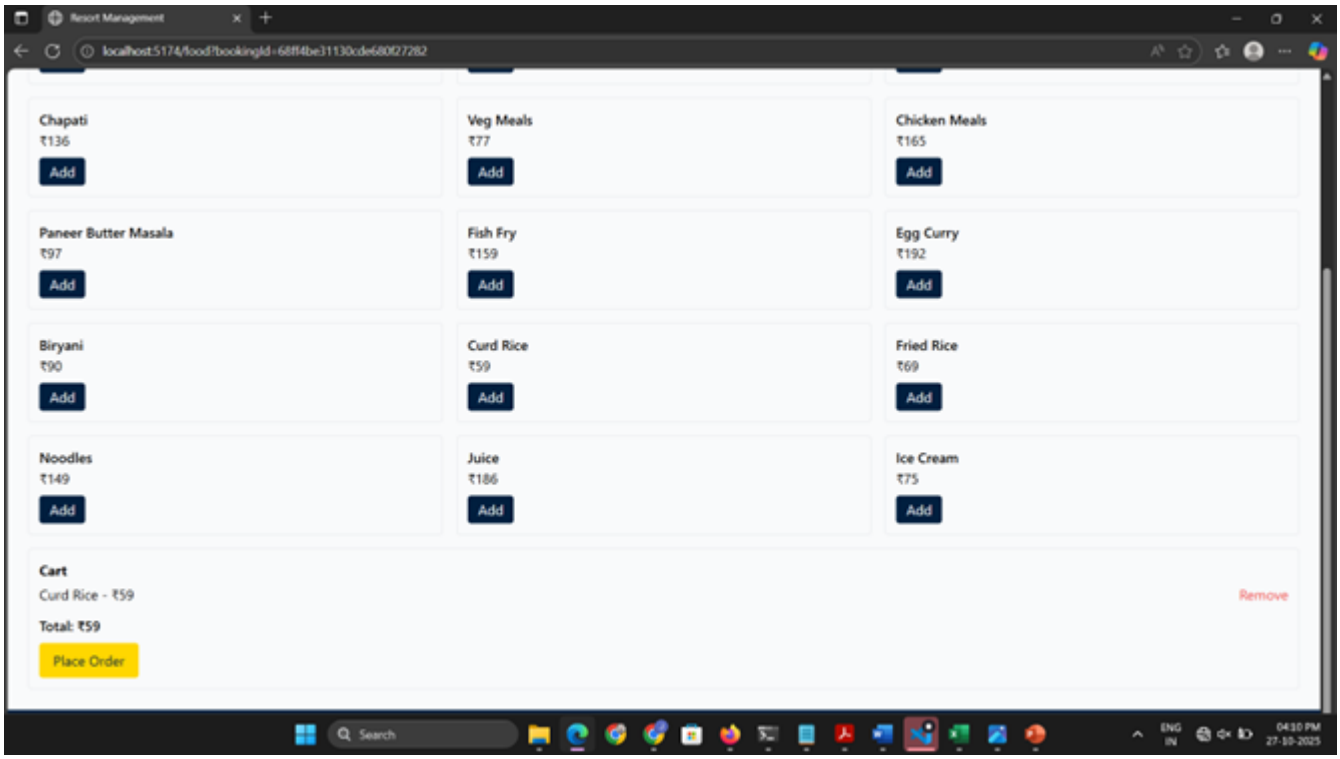


Figure 6.6: Food Ordering Page

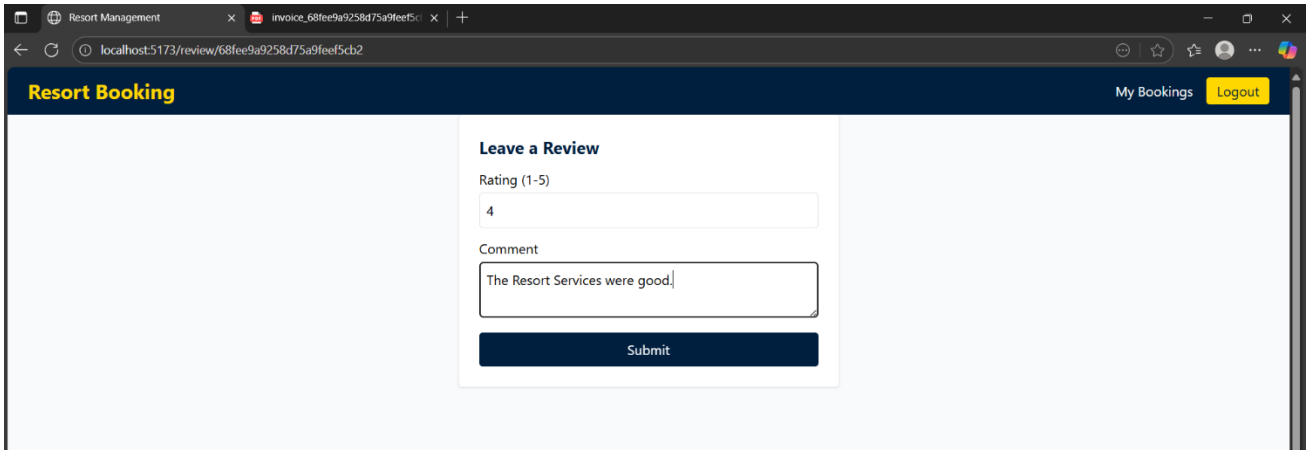


Figure .6.8: Review & Rating Submission Page

Resort Management - Invoice

Invoice ID: invoice_68fed68f5296cd633db1065e.pdf
Booking ID: 68fed68f5296cd633db1065e
User: NANDHAGOPAL V (nandhagopalv2005_bai27@mepcoeng.ac.in)
Resort: Resort 8 - Chennai
Check In: 28/10/2025, 5:30:00 am
Check Out: 30/10/2025, 5:30:00 am
Assigned Staff: Staff 1

Charges:
Accommodation & extras: ¹4786

Total Paid: ¹4786

`Figure 6.9: Generated Invoice after checkout

Figure No.6.10.Reopening Issue

Figure No.6.12.Reassigning new Field Agent

Figure No.6.10.Reopening Issue

CHAPTER 7
IMPLEMENTATION

BACKEND :

Server.js

```
import express from "express";
import dotenv from "dotenv";
import cors from "cors";
import connectDB from "./config/db.js";
import authRoutes from "./routes/authRoutes.js";
import resortRoutes from "./routes/resortRoutes.js";
import bookingRoutes from "./routes/bookingRoutes.js";
import adminRoutes from "./routes/adminRoutes.js";
import foodRoutes from "./routes/foodRoutes.js";
import path from "path";
dotenv.config();
connectDB();
const app = express();
app.use(cors());
app.use(express.json());
app.use("/images", express.static(path.join(process.cwd(), "images")));
app.use("/exports", express.static(path.join(process.cwd(), "backend", "exports")));
app.use("/api/auth", authRoutes);
app.use("/api/resorts", resortRoutes);
app.use("/api/bookings", bookingRoutes);
app.use("/api/admin", adminRoutes);
app.use("/api/food", foodRoutes);
app.get("/", (req, res) => res.send("Resort backend is running"));
const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Server running on ${PORT}`));
```

bookingRoutes.js

```
import express from "express";
import Booking from "../models/Booking.js";
import Resort from "../models/Resort.js";
import Staff from "../models/Staff.js";
import User from "../models/User.js";
import Food from "../models/Food.js";
import PDFDocument from "pdfkit";
import fs from "fs";
import path from "path";
import { protect } from "../middleware/auth.js";

const router = express.Router();

// Create booking
router.post("/", protect, async (req, res) => {
try {
const { resortId, checkIn, checkOut, adults = 1, children = 0 } = req.body;
const resort = await Resort.findById(resortId);
if (!resort) return res.status(404).json({ message: "Resort not found" });

const nights = Math.max(
1,
Math.ceil((new Date(checkOut) - new Date(checkIn)) / (1000 * 60 * 60 * 24))
);
const persons = Number(adults) + Number(children);
const multiplier = Math.max(1, persons / 2);
const totalPrice = Math.round(resort.price * nights * multiplier);

// assign first unassigned staff
```

```
const staff = await Staff.findOneAndUpdate({ assigned: false }, { assigned: true }, { new: true });

const booking = await Booking.create({
user: req.user._id,
resort: resort._id,
staff: staff ? staff._id : null,
checkIn,
checkOut,
guests: persons,
totalPrice,
status: "booked",
foodOrders: []
});

// populate for response
const populated = await Booking.findById(booking._id).populate("resort staff");
res.json({ message: "Booked (pending checkout)", booking: populated });
} catch (err) {
console.error(err);
res.status(500).json({ message: err.message });
}
});

// Get user's bookings
router.get("/my", protect, async (req, res) => {
try {
const bookings = await Booking.find({ user: req.user._id }).populate("resort staff");
res.json(bookings);
} catch (err) {
res.status(500).json({ message: err.message });
}
});
```

```
// Checkout: deduct from wallet, create PDF invoice, return invoice path
router.post("/:id/checkout", protect, async (req, res) => {
try {
const booking = await Booking.findById(req.params.id).populate
("user resort staff foodOrders");
if (!booking) return res.status(404).json({ message:
"Booking not found" });
if (booking.user._id.toString() !== req.user._id.toString())
return res.status(403).json({ message: "Forbidden" });
if (booking.status === "checkedout") return res.status(400).json({ message:
"Already checked out" });

const user = await User.findById(req.user._id);
// total price may include added food orders
const amount = booking.totalPrice;
if (user.wallet < amount) return res.status(400).json({ message:
 "Insufficient wallet balance" });

user.wallet -= amount;
await user.save();

booking.status = "checkedout";
await booking.save();

// create exports dir
const exportsDir = path.join(process.cwd(), "backend", "exports");
if (!fs.existsSync(exportsDir)) fs.mkdirSync(exportsDir, { recursive: true });
const filename = `invoice_${booking._id}.pdf`;
const filepath = path.join(exportsDir, filename);

const doc = new PDFDocument();
```

```
doc.pipe(fs.createWriteStream(filepath));

doc.fontSize(18).text("Resort Management - Invoice", { align: "center" });
doc.moveDown();
doc.fontSize(12).text(`Invoice ID: ${filename}`);
doc.text(`Booking ID: ${booking._id}`);
doc.text(`User: ${user.name} (${user.email})`);
doc.text(`Resort: ${booking.resort.name} - ${booking.resort.city}`);
doc.text(`Check In: ${new Date(booking.checkIn).toLocaleString()}`);
doc.text(`Check Out: ${new Date(booking.checkOut).toLocaleString()}`);
doc.text(`Assigned Staff: ${booking.staff ? booking.staff.name : "Not assigned"}`);
doc.moveDown();

doc.text("Charges:");
doc.text(`  Accommodation & extras: ₹${booking.totalPrice}`, { indent: 10 });
if (booking.foodOrders && booking.foodOrders.length > 0) {
doc.text("  Food Items:", { indent: 10 });
booking.foodOrders.forEach(f => {
doc.text(`    - ${f.name} : ₹${f.price}`, { indent: 20 });
});
}

doc.moveDown();
doc.fontSize(14).text(`Total Paid: ₹${amount}`, { align: "right" });
doc.end();

// send invoice path (served by /exports static in server.js)
res.json({ message: "Checkout complete", invoice: `/exports/${filename}`,
wallet: user.wallet });
} catch (err) {
console.error(err);
res.status(500).json({ message: err.message });
```

```
}
});

// Add food items to booking -> increases booking.totalPrice and stores foodOrders
router.post("/:id/food", protect, async (req, res) => {
try {
const { items } = req.body; // array of food ids
const booking = await Booking.findById(req.params.id);
if (!booking) return res.status(404).json({ message: "Booking not found" });
if (booking.user.toString() !== req.user._id.toString())
return res.status(403).json({ message: "Forbidden" });

const foods = await Food.find({ _id: { $in: items } });
if (!foods || foods.length === 0) return res.status(400).json({ message:
 "No valid food items" });

// append food refs
booking.foodOrders = booking.foodOrders.concat(foods.map(f => f._id));

// increase totalPrice
const foodTotal = foods.reduce((s, f) => s + f.price, 0);
booking.totalPrice = (booking.totalPrice || 0) + foodTotal;

await booking.save();
const populated = await Booking.findById(booking._id).populate("resort staff
 foodOrders");
res.json({ message: "Food added to booking", booking: populated });
} catch (err) {
console.error(err);
res.status(500).json({ message: err.message });
}
});
```

```
// Add a review for a booking (simple embed)
router.post("/:id/review", protect, async (req, res) => {
try {
const { rating = 5, comment = "" } = req.body;
const booking = await Booking.findById(req.params.id);
if (!booking) return res.status(404).json({ message: "Booking not found" });
if (booking.user.toString() !== req.user._id.toString()) return res.status(403).json({ message:
 "Forbidden" });

// store a small review object on booking
booking.review = { rating: Number(rating), comment, date: new Date() };
await booking.save();
res.json({ message: "Review saved", review: booking.review });
} catch (err) {
console.error(err);
res.status(500).json({ message: err.message });
}
});

export default router;
```

authRoots.js

```
import express from "express";
import User from "../models/User.js";
import Booking from "../models/Booking.js";
import Resort from "../models/Resort.js";
import { protect, adminOnly } from "../middleware/auth.js";
const router = express.Router();
router.get("/stats", protect, adminOnly, async (req, res) => {
try {
```

```
const users = await User.countDocuments();
const resorts = await Resort.countDocuments();
const bookings = await Booking.countDocuments();
res.json({ users, resorts, bookings });
} catch (err) {
res.status(500).json({ message: err.message });
}
});
router.post("/wallet/:id", protect, adminOnly, async (req, res) => {
try {
const { amount } = req.body;
const user = await User.findById(req.params.id);
user.wallet += Number(amount);
await user.save();
res.json({ message: "Wallet updated", wallet: user.wallet });
} catch (err) {
res.status(500).json({ message: err.message });
}
});
export default router;
```

foodRoutes.js

```
import express from "express";
import Food from "../models/Food.js";
const router = express.Router();
router.get("/", async (req, res) => {
try {
const items = await Food.find();
res.json(items);
} catch (err) {
res.status(500).json({ message: err.message });
```

```
}
});
export default router;
```

resortRoutes.js

```
import express from "express";
import Resort from "../models/Resort.js";
const router = express.Router();
router.get("/", async (req, res) => {
try {
const { city } = req.query;
const query = city ? { city: new RegExp(city, "i") } : {};
const resorts = await Resort.find(query);
res.json(resorts);
} catch (err) {
res.status(500).json({ message: err.message });
}
});
router.get("/:id", async (req, res) => {
try {
const resort = await Resort.findById(req.params.id);
if (!resort) return res.status(404).json({ message: "Resort not found" });
res.json(resort);
} catch (err) {
res.status(500).json({ message: err.message });
}
});
router.post("/", async (req, res) => {
try {
const r = await Resort.create(req.body);
res.json(r);
```

```
} catch (err) {
res.status(500).json({ message: err.message });
}
});
router.delete("/:id", async (req, res) => {
try {
await Resort.findByIdAndDelete(req.params.id);
res.json({ message: "Deleted" });
} catch (err) {
res.status(500).json({ message: err.message });
}
});
export default router;
```

Booking.js

```
import mongoose from "mongoose";
const bookingSchema = new mongoose.Schema({
user: { type: mongoose.Schema.Types.ObjectId, ref: "User" },
resort: { type: mongoose.Schema.Types.ObjectId, ref: "Resort" },
staff: { type: mongoose.Schema.Types.ObjectId, ref: "Staff" },
checkIn: Date,
checkOut: Date,
guests: Number,
totalPrice: Number,
status: { type: String, default: "booked" },
foodOrders: [{ type: mongoose.Schema.Types.ObjectId, ref: "Food" }],
review: { rating: Number, comment: String, date: Date }
});
export default mongoose.model("Booking", bookingSchema);
```

Food.js

```
import mongoose from "mongoose";
const foodSchema = new mongoose.Schema({
name: String,
price: Number,
type: String
});
export default mongoose.model("Food", foodSchema);
```

Resort.js

```
import mongoose from "mongoose";
const resortSchema = new mongoose.Schema({
name: String,
city: String,
description: String,
price: Number,
image: String,
availableRooms: { type: Number, default: 5 },
rating: { type: Number, default: 4.5 }
});
export default mongoose.model("Resort", resortSchema);
```

Staff.js

```
import mongoose from "mongoose";
const staffSchema = new mongoose.Schema({
name: String,
assigned: { type: Boolean, default: false }
```

```
});
export default mongoose.model("Staff", staffSchema);
```

User.js

```
import mongoose from "mongoose";
import bcrypt from "bcryptjs";
const userSchema = new mongoose.Schema({
name: String,
email: { type: String, unique: true },
password: String,
role: { type: String, enum: ["user","admin","staff"], default: "user" },
wallet: { type: Number, default: 5000 }
});
userSchema.pre("save", async function(next){
if (!this.isModified("password")) return next();
const salt = await bcrypt.genSalt(10);
this.password = await bcrypt.hash(this.password, salt);
next();
});
userSchema.methods.comparePassword = function(candidate){
return bcrypt.compare(candidate, this.password);
};
export default mongoose.model("User", userSchema);
```

auth.js

```
import jwt from "jsonwebtoken";
import User from "../models/User.js";
export const protect = async (req, res, next) => {
try {
```

```
const header = req.headers.authorization;
if (!header) return res.status(401).json({ message: "No token provided" });
const token = header.split(" ")[1];
const decoded = jwt.verify(token, process.env.JWT_SECRET);
const user = await User.findById(decoded.id).select("-password");
if (!user) return res.status(401).json({ message: "User not found" });
req.user = user;
next();
} catch (err) {
return res.status(401).json({ message: "Not authorized" });
}
};
export const adminOnly = (req, res, next) => {
if (!req.user || req.user.role !== "admin") {
return res.status(403).json({ message: "Admin only" });
}
next();
};
```

DatePicker.js

```
import React, { useState } from "react";
const DatePicker = ({ label, value, onChange, min }) => {
const [date, setDate] = useState(value || "");
const handleChange = (e) => { setDate(e.target.value); onChange(e.target.value); };
return (
<div className="flex flex-col">
<label className="text-sm font-semibold mb-1 text-gray-700">{label}</label>
<input type="date" value={date} min={min} onChange={handleChange}
className="border border-gray-300 rounded-lg p-2 w-full focus:ring-2
 focus:ring-yellow-500 focus:outline-none" />
```

```
</div>
);
};
export default DatePicker;
```

Navbar.js

```
import React from 'react';
import { Link, useNavigate } from 'react-router-dom';
export default function Navbar(){
const navigate = useNavigate();
const user = JSON.parse(localStorage.getItem('user')||'null');
const logout = ()=>{ localStorage.clear(); navigate('/login'); };
return (
<nav className="bg-primary text-white flex justify-between px-6 py-3 items-center
 shadow">
<Link to="/" className="text-2xl font-bold text-accent">Resort Booking</Link>
<div className="flex gap-4 items-center">
{user ? (
<>
{user.role==='admin' && <Link to="/admin">Admin</Link>}
<Link to="/mybookings">My Bookings</Link>
<button onClick={logout} className="bg-accent text-primary px-3 py-1
rounded">Logout</button>
</>
) : (
<>
<Link to="/login">Login</Link>
<Link to="/register" className="bg-accent text-primary px-3 py-1 rounded">Register</Link>
</>
)}
```

```
</div>
</nav>
);
}
```

ProtectedRoute.jsx

```
import React from 'react';
import { Navigate } from 'react-router-dom';
export default function ProtectedRoute({children, adminOnly=false}){
const token = localStorage.getItem('token');
const user = JSON.parse(localStorage.getItem('user')||'null');
if(!token) return <Navigate to="/login" replace />;
if(adminOnly && user?.role!=='admin') return <Navigate to="/" replace />;
return children;
}
```

ResortCard.js

```
import React from "react";
import { Link } from "react-router-dom";
export default function ResortCard({ resort }){
return (
<div className="border rounded-lg shadow hover:shadow-lg transition overflow-hidden">
<img src={resort.image} alt={resort.name} className="h-48 w-full object-cover" />
<div className="p-3">
<h3 className="text-lg font-semibold">{resort.name}</h3>
<p className="text-sm text-gray-600">{resort.city}</p>
<p className="mt-1 text-accent font-bold">₹{resort.price}</p>
<Link to={`/booking/${resort._id}`} className="block bg-primary text-white
```

```
 text-center mt-2 py-1 rounded">Book Now</Link>
</div>
</div>
);
}
```

AdminDashboard.jsx

```
import React, { useEffect, useState } from "react";
import axios from "axios";
export default function AdminDashboard(){
const [stats,setStats]=useState(null);
const token = localStorage.getItem("token");
useEffect(()=>{ axios.get('/api/admin/stats',{ headers:{ Authorization:`Bearer $
{token}` } }).then(res=>setStats(res.data)).catch(()=>{}); },[]);
return (<div className="p-6"><h1 className="text-2xl font-bold mb-4
text-center text-primary">Admin Dashboard</h1>{stats? (<div className=
"grid grid-cols-1
md:grid-cols-3 gap-4"><div className="p-4 border rounded"><h3 className="font-
bold">Users</h3><p>{stats.users}</p></div><div className="p-4
 border rounded"><h3 className="font-bold">Resorts</h3><p>{stats.resorts}
</p></div><div className="p-4 border rounded"><h3 className="font-
bold">Bookings</h3><p>{stats.bookings}</p></div></div>) : <p>Loading...
</p>}</div>);
}
```

Booking.jsx

```
import React, { useEffect, useState } from "react";
import axios from "axios";
import { useParams, useNavigate } from "react-router-dom";
import DatePicker from "../components/DatePicker";
export default function Booking(){
```

```
const { id } = useParams();
const [resort, setResort] = useState(null);
const [checkIn, setCheckIn] = useState("");
const [checkOut, setCheckOut] = useState("");
const [adults, setAdults] = useState(2);
const [children, setChildren] = useState(0);
const [loading, setLoading] = useState(false);
const navigate = useNavigate();

const token = localStorage.getItem("token");
useEffect(()=>{ axios.get(`/api/resorts/${id}`).
then(res=>setResort(res.data))
.catch(err=>console.error(err)); },[id]);
const handleBooking=async()=>{ if(!token)
return navigate('/login'); if(!checkIn||!checkOut)
return alert('Select dates'); setLoading(true); try{ await
 axios.post('/api/bookings',{ resortId: id, checkIn, checkOut,
adults, children }, { headers: { Authorization: `Bearer ${token}`
 } }); setLoading(false); alert('Booked (pending checkout)');
navigate('/mybookings'); }catch(err){ setLoading(false);
alert(err.response?.data?.message||'Booking failed'); } };
if(!resort) return <div className="p-6">
Loading resort details...</div>;
return (<div className="p-6 max-w-3xl mx-auto">
<h2 className="text-2xl font-bold mb-4">{resort.name}
</h2><img src={resort.image} alt={resort.name}
className="w-full h-64 object-cover rounded mb-4" />
<p className="mb-2">{resort.description}</p><div
className="grid grid-cols-1 md:grid-cols-2 gap-4">
<DatePicker label="Check In" value={checkIn}
onChange={setCheckIn} min={new Date().toISOString().split('T')[0]} />
<DatePicker label="Check Out" value={checkOut}
```

```
 onChange={setCheckOut} min={checkIn ||
new Date().toISOString().split('T')[0]}
 /><div><label className="block mb-1">Adults
</label><input type="number" min="1" value={adults}
onChange={e=>setAdults(Number(e.target.value))}
className="border p-2 rounded w-24" /></div><div>
<label className="block mb-1">Children</label>
<input type="number" min="0" value={children}
onChange={e=>setChildren(Number(e.target.value))}
 className="border p-2 rounded w-24"
 /></div></div><div className="mt-4"><p
className="text-lg font-semibold">Price
per night: ₹{resort.price}</p><button
onClick={handleBooking} disabled={loading}
className="bg-primary text-white px-4 py-2 mt-3
rounded">{loading?'Booking...':'Confirm Booking'}
</button></div></div>);
}
```

Checkout.jsx

```
import React, { useEffect, useState } from "react";
import axios from "axios";
export default function Checkout(){
const [bookings,setBookings]=useState([]);
const token = localStorage.getItem("token");
useEffect(()=>{ axios.get('/api/bookings/my',{ headers:
{ Authorization:`Bearer ${token}` } }).then(res=>
setBookings(res.data)).catch(()=>{}); },[]);
const handleCheckout=async(id)=>{ try{ const res=await
axios.post(`/api/bookings/${id}/checkout`,{},
{ headers:{ Authorization:`Bearer ${token}` } });
```

```
alert(res.data.message); if(res.data.invoice){
 window.open(`http://localhost:5000${res.data.invoice}
`,'_blank'); } const updated = await axios.get('/api/bookings/my',
{ headers:{ Authorization:`Bearer ${token}` } }); setBookings(updated.data);
 }catch(err){ alert(err.response?.data?.message||'Checkout failed'); } };
return (<div className="p-6"><h2 className="text-2xl
font-bold text-primary mb-4">Pending Checkouts</h2>
{bookings.length===0 && <p>No bookings found.</p>}
{bookings.map(b=>(<div key={b._id} className="border p-3 mb-3 rounded"><h3
className="font-semibold">{b.resort?.name}</h3><p>From: {new
Date(b.checkIn).toLocaleDateString()} To: {new
Date(b.checkOut).toLocaleDateString()}</p><p>Total: ₹{b.totalPrice}</p>{b.status!=='checkedout'
&& <button onClick={()=>handleCheckout(b._id)} className="bg-accent text-primary px-3 py-1
rounded mt-2">Checkout & Generate Invoice</button>}{b.status==='checkedout' && <p
className="text-green-600">Checked out</p>}</div>))}</div>);
}
```

FoodOrder.jsx

```
// frontend/src/pages/FoodOrder.jsx
import React, { useEffect, useState } from "react";
import axios from "axios";
import { useLocation } from "react-router-dom";

export default function FoodOrder(){
const [menu,setMenu]=useState([]);
const [cart,setCart]=useState([]);
const [bookingId,setBookingId]=useState("");
const [myBookings,setMyBookings]=useState([]);
const token = localStorage.getItem("token");
const location = useLocation();
```

```
useEffect(()=>{
// fetch food menu
axios.get('/api/food').then(res=>setMenu(res.data)).catch(()=>{});
// fetch user's bookings for dropdown
axios.get('/api/bookings/my', { headers: { Authorization: `Bearer ${token}` } })
.then(res => {
setMyBookings(res.data);
// if bookingId passed via query param, preselect it
const q = new URLSearchParams(location.search);
const b = q.get('bookingId');
if (b) setBookingId(b);
}).catch(()=>{});
},[]);

const add=(item)=>setCart(prev=>[...prev,item]);
const remove=(i)=>setCart(prev=>prev.filter((_,idx)=>idx!==i));

const placeOrder=async()=>{
if(!bookingId) return alert('Select a booking to attach this order to');
try{
const ids = cart.map(c=>c._id);
const res = await axios.post(`/api/bookings/$
{bookingId}/food`, { items: ids }, { headers: { Authorization: `Bearer ${token}` }});
alert("Food added to booking");
// optionally navigate to MyBookings or refresh data
setCart([]);
} catch(err){
alert(err.response?.data?.message || 'Order failed');
}
};
```

```
return (
<div className="p-6">
<h2 className="text-2xl font-bold mb-4 text-primary">Food Menu</h2>

<div className="mb-4">
<label className="block font-semibold mb-1">Attach to Booking</label>
{myBookings.length > 0 ? (
<select value={bookingId} onChange=
{e=>setBookingId(e.target.value)} className="border p-2 rounded w-full">
<option value="">-- choose booking --</option>
{myBookings.map(b=>(
<option key={b._id} value={b._id}>
{b.resort?.name} ({new Date(b.checkIn).toLocaleDateString()} - {new
Date(b.checkOut).toLocaleDateString()})
</option>
))}
</select>
) : (
<p>No bookings found. Make a booking first.</p>
)}
</div>

<div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-4">
{menu.map(item=>(
<div key={item._id} className="border p-3 rounded">
<h3 className="font-semibold">{item.name}</h3>
<p>₹{item.price}</p>
<button onClick={()=>add(item)} className="bg-primary text-white mt-2 px-3 py-1
rounded">Add</button>
</div>
))}
</div>
```

```
{cart.length > 0 && (
<div className="mt-4 p-3 border rounded bg-gray-50">
<h4 className="font-bold">Cart</h4>
{cart.map((c,i)=>(
<div key={i} className="flex justify-between">
<p>{c.name} - ₹{c.price}</p>
<button onClick={()=>remove(i)} className="text-red-500">Remove</button>
</div>
))}
<p className="mt-2 font-semibold">Total: ₹{cart.reduce((s,i)=>s+i.price,0)}</p>
<button onClick={placeOrder} className="bg-accent text-primary px-4 py-2
 rounded mt-2">Place Order</button>
</div>
)}
</div>
);
}
```

Home.jsx

```
import React, { useState } from "react";
import { useNavigate } from "react-router-dom";
import DatePicker from "../components/DatePicker";
export default function Home(){
const [city, setCity] = useState("");
const [checkIn, setCheckIn] = useState("");
const [checkOut, setCheckOut] = useState("");
const [guests, setGuests] = useState(1);
const navigate = useNavigate();
const handleSearch = ()=> {
```

```
if(!city||!checkIn||!checkOut){ alert("Please fill all fields!"); return; }
navigate(`/resorts?city=${city}&checkIn=${checkIn}&checkOut=$
{checkOut}&guests=${guests}`);
};
return (
<div className="flex flex-col items-center justify-center
min-h-screen bg-cover bg-center" style={{ backgroundImage:
 "url('/images/background.jpeg')" }}>
<div className="bg-white bg-opacity-90 p-8 rounded-2xl
shadow-lg max-w-md w-full text-center">
<h1 className="text-3xl font-bold text-blue-800 mb-6">Find Your Perfect Resort</h1>
<div className="flex flex-col gap-4 text-left">
<div>
<label className="block font-semibold text-gray-700 mb-1">City</label>
<input type="text" placeholder="Enter city..." value={city} onChange=
{e=>setCity(e.target.value)} className="border border-gray-300 rounded-lg
 p-2 w-full focus:ring-2 focus:ring-yellow-500 focus:outline-none" />
</div>
<DatePicker label="Check-in" value={checkIn} onChange={setCheckIn}
 min={new Date().toISOString().split("T")[0]} />
<DatePicker label="Check-out" value={checkOut} onChange={setCheckOut}
min={checkIn || new Date().toISOString().split("T")[0]} />
<div>
<label className="block font-semibold text-gray-700 mb-1">Guests</label>
<input type="number" min="1" value={guests} onChange={e=>
setGuests(e.target.value)} className="border border-gray-300 rounded-lg p-2
 w-full focus:ring-2 focus:ring-yellow-500 focus:outline-none" />
</div>
<button onClick={handleSearch} className="bg-yellow-500 text-white
font-semibold py-2 px-4 rounded-lg hover:bg-yellow-600 transition">
Search Resorts</button>
</div>
```

```
</div>
</div>
);
}
```

Login.jsx

```
import React, { useState } from "react";
import axios from "axios";
import { useNavigate } from "react-router-dom";
export default function Login(){
const [email,setEmail]=useState(""); const [password,setPassword]=useState("");
const navigate = useNavigate();
const handleLogin=async(e)=>{ e.preventDefault(); try{ const res=await
axios.post("/api/auth/login",{email,password});
 localStorage.setItem("token",res.data.token);
localStorage.setItem("user",JSON.stringify(res.data.user)); navigate("/"); }
catch{ alert("Invalid credentials"); } };
return (
<div className="flex justify-center items-center h-screen bg-gray-100">
<form onSubmit={handleLogin} className="bg-white p-6
rounded shadow-md w-80">
<h2 className="text-xl font-bold text-center mb-4">Login</h2>
<input className="w-full border p-2 mb-3" type="email"
placeholder="Email" value={email} onChange=
{e=>setEmail(e.target.value)} required />
<input className="w-full border p-2 mb-3" type="password"
 placeholder="Password" value={password} onChange=
{e=>setPassword(e.target.value)} required />
<button className="bg-primary text-white w-full py-2 rounded">
Login</button>
</form>
```

```
</div>
);
}
```

MyBokking.jsx

```
import React, { useEffect, useState } from "react";
import axios from "axios";
import { useNavigate } from "react-router-dom";

export default function MyBookings(){
const [bookings,setBookings] = useState([]);
const token = localStorage.getItem("token");
const navigate = useNavigate();

const fetchBookings = async () => {
try {
const res = await axios.get('/api/bookings/my', { headers:
{ Authorization: `Bearer ${token}` }});
setBookings(res.data);
} catch (err) {
setBookings([]);
}
};

useEffect(()=>{ fetchBookings(); },[]);

const handleCheckout = async (id) => {
try {
const res = await axios.post(`/api/bookings/${id}/checkout`, {},
{ headers: { Authorization: `Bearer ${token}` }});
```

```
alert(res.data.message);
if (res.data.invoice) {
window.open(`http://localhost:5000${res.data.invoice}`, "_blank");
}
fetchBookings();
} catch (err) {
alert(err.response?.data?.message || "Checkout failed");
}
};

const handleAddFood = (bookingId) => {
navigate(`/food?bookingId=${bookingId}`);
};

const handleReview = (bookingId) => {
navigate(`/review/${bookingId}`);
};

return (
<div className="p-6">
<h1 className="text-2xl font-bold mb-4 text-primary text-center">My Bookings</h1>
{bookings.length === 0 ? <p>No bookings found.</p> :
<div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-4">
{bookings.map(b => (
<div key={b._id} className="border p-3 rounded shadow">
<h3 className="font-bold">{b.resort?.name}</h3>
<p>From: {new Date(b.checkIn).toLocaleDateString()} To: {new
Date(b.checkOut).toLocaleDateString()}</p>
<p>Total: ₹{b.totalPrice}</p>
<p>Status: <span className={b.status===
'checkedout' ? 'text-green-600' : 'text-yellow-600'}>{b.status}</span></p>
<p>Assigned Staff: {b.staff ? b.staff.name : "Not assigned"}</p>
```

```
<div className="flex gap-2 mt-3">
{b.status !== "checkedout" && (
<button onClick={() => handleCheckout(b._id)}
className="bg-accent text-primary px-3 py-1 rounded">
Checkout
</button>
)}

<button onClick={() => handleAddFood(b._id)}
className="bg-primary text-white px-3 py-1 rounded">
Add Food
</button>

{b.status === "checkedout" && (
<button onClick={() => handleReview(b._id)}
className="bg-blue-600 text-white px-3 py-1 rounded">
Leave Review
</button>
)}
</div>
</div>
))}
</div>
}
</div>
);
}
```

Register.jsx

```
import React, { useState } from "react";
import axios from "axios";
import { useNavigate } from "react-router-dom";
export default function Register(){
const [form,setForm]=useState({name:",email:",password:"}); const navigate=useNavigate();
const handleChange=e=>setForm({...form,[e.target.name]:e.target.value});
const handleRegister=async(e)=>{ e.preventDefault(); try{ await axios.post("/api/auth/register",form);
alert("Registered"); navigate("/login"); }catch{ alert("Registration failed"); } };
return (
<div className="flex justify-center items-center h-screen bg-gray-100">
<form onSubmit={handleRegister} className="bg-white p-6 rounded shadow-md w-80">
<h2 className="text-xl font-bold text-center mb-4">Register</h2>
<input name="name" className="w-full border p-2 mb-3" placeholder="Name"
onChange={handleChange} required />
<input name="email" className="w-full border p-2 mb-3" placeholder="Email"
onChange={handleChange} required />
<input name="password" type="password" className="w-full border p-2 mb-3" placeholder="Password"
onChange={handleChange} required />
<button className="bg-accent text-primary w-full py-2 rounded font-bold">Register</button>
</form>
</div>
);
}
```

Resirt.jsx

```
import React, { useEffect, useState } from "react";
import axios from "axios";
import ResortCard from "../components/ResortCard";
import { useLocation } from "react-router-dom";
```

```
export default function Resorts(){
const [resorts,setResorts]=useState([]); const { search } =
useLocation();
useEffect(()=>{ const params=new URLSearchParams(search);
const city=params.get('city'); axios.get(`/api/resorts${city?`?city=
${city}`:"}`).then(r=>

setResorts(r.data)).catch(()=>setResorts([])); },[search]);
return (<div className="p-6"><h1 className="text-2xl font-bold
mb-6 text-center text-primary">Available Resorts</h1><div
className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-6">
{resorts.map(r=><ResortCard key={r._id} resort={r}/>)}</div></div>);
}
```

Revies.jsx

```
import React, { useState } from "react";
import axios from "axios";
export default function Review(){
const [rating,setRating]=useState(5); const [comment,setComment]=useState("");
const token = localStorage.getItem("token");
const submitReview=async(bookingId)=>{ try{ await
 axios.post(`/api/bookings/${bookingId}/review`,
{ rating, comment }, { headers:{ Authorization:`Bearer $
{token}` } }); alert('Review submitted'); setComment(''); }catch{ alert('Failed to submit review'); } };
return (<div className="p-6 max-w-md mx-auto bg-white shadow rounded"><h2 className="text-xl font-
bold mb-3 text-primary">Leave a Review</h2><label className=
"block mb-1">Rating (1-5)</label><input type="number"
min="1" max="5" value={rating} onChange=
{e=>setRating(e.target.value)} className=
"border p-2 w-full mb-3 rounded"/><label className="block mb-1">
Comment</label><textarea value={comment} onChange={e=>
```

```
setComment(e.target.value)} className=
"border p-2 w-full mb-3 rounded"/><button onClick={()=>
submitReview(prompt('Enter booking id to attach review:'))}
className="bg-primary text-white w-full py-2 rounded">Submit</button></div>);
}
```

App.jsx

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import Navbar from './components/Navbar';
import Footer from './components/Footer';
import Home from './pages/Home';
import Login from './pages/Login';
import Register from './pages/Register';
import Resorts from './pages/Resorts';
import Booking from './pages/Booking';
import Checkout from './pages/Checkout';
import MyBookings from './pages/MyBookings';
import FoodOrder from './pages/FoodOrder';
import Review from './pages/Review';
import AdminDashboard from './pages/AdminDashboard';
import ProtectedRoute from './components/ProtectedRoute';

export default function App(){
return (
<Router>
<Navbar />
<div className="min-h-screen bg-gray-50">
<Routes>
<Route path="/" element={<Home/>}/>
```

```
<Route path="/login" element={<Login/>}/>
<Route path="/register" element={<Register/>}/>
<Route path="/resorts" element={<Resorts/>}/>
<Route path="/booking/:id" element={<ProtectedRoute><Booking/></ProtectedRoute>}/>
<Route path="/mybookings" element={<ProtectedRoute><MyBookings/></ProtectedRoute>}/>
<Route path="/food" element={<ProtectedRoute><FoodOrder/></ProtectedRoute>}/>
<Route path="/checkout" element={<ProtectedRoute><Checkout/></ProtectedRoute>}/>
<Route path="/review/:id" element={<ProtectedRoute><Review/></ProtectedRoute>}/>
<Route path="/admin" element={<ProtectedRoute
adminOnly={true}><AdminDashboard/></ProtectedRoute>}/>
</Routes>
</div>
<Footer />
</Router>
);
}
```

Index.css

```
@tailwind base;
@tailwind components;
@tailwind utilities;
body { font-family: ui-sans-serif, system-ui, -apple-system,
 "Segoe UI", Roboto, "Helvetica Neue", Arial; }
```

main.jsx

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App'
import './index.css'
ReactDOM.createRoot(document.getElementById('root')).render(
```

```
<React.StrictMode><App /></React.StrictMode>
)
```

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width,initial-scale=1.0" />
<title>Resort Management</title>
</head>
<body>
<div id="root"></div>
<script type="module" src="/src/main.jsx"></script>
</body>
</html>
```

Seed.js

```
import mongoose from "mongoose";
import dotenv from "dotenv";
import bcrypt from "bcryptjs";
import User from "./models/User.js";
import Staff from "./models/Staff.js";
import Food from "./models/Food.js";
import Resort from "./models/Resort.js";
import connectDB from "./config/db.js";
import fs from "fs";
import path from "path";
import { fileURLToPath } from "url";
dotenv.config();
await connectDB();
const seed = async () => {
```

```
try {
await User.deleteMany();
await Staff.deleteMany();
await Food.deleteMany();
await Resort.deleteMany();
const admin = new User({
name: "Admin",
email: "admin@resort.com",
password: await bcrypt.hash("admin123", 10),
role: "admin",
wallet: 5000
});
const user = new User({
name: "Regular User",
email: "user@resort.com",
password: await bcrypt.hash("user123", 10),
role: "user",
wallet: 5000
});
await admin.save();
await user.save();
const staffMembers = Array.from({ length: 10 }).map((_, i) =>
 ({ name: `Staff ${i + 1}` }));
await Staff.insertMany(staffMembers);
const foodNames = [
"Idly", "Dosa", "Parotta", "Chapati", "Veg Meals", "Chicken Meals",
"Paneer Butter Masala", "Fish Fry", "Egg Curry", "Biryani",
"Curd Rice", "Fried Rice", "Noodles", "Juice", "Ice Cream"
];
const foodItems = foodNames.map(name => ({ name, price:
Math.floor(Math.random() * 150) + 50, type: "food" }));
await Food.insertMany(foodItems);
```

```
const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);
const imageDir = path.join(__dirname, "..", "images");
if (!fs.existsSync(imageDir)) {
console.warn("No images/ directory found at project root.
Create it and put your resort .jpeg/.jpg/.png files there.");
} else {
const allFiles = fs.readdirSync(imageDir);
const imageFiles = allFiles.filter(f => /\.(jpe?g|png|webp|gif)$/i.test(f));
const cities = ["Chennai", "Madurai", "Ooty", "Kodaikanal", "Coimbatore",
"Munnar", "Trivandrum"];
const resorts = imageFiles.map((file, i) => ({
name: `Resort ${i + 1}`,
city: cities[i % cities.length],
description: "Luxury stay with all amenities",
price: Math.floor(Math.random() * 3000) + 2000,
image: `/images/${file}`,
availableRooms: Math.floor(Math.random() * 5) + 3,
rating: (Math.random() * 1.5 + 3.5).toFixed(1)
}));
await Resort.insertMany(resorts);
console.log(`Seeded ${resorts.length} resorts from images/`);
}
console.log("✅ Seed data inserted successfully");
process.exit(0);
} catch (err) {
console.error(err);
process.exit(1);
}
};
seed();
```

CHAPTER 8

CONCLUSION

The Resort Management System (RMS) project has evolved into a comprehensive, user-centric framework designed to digitalize and simplify the complete resort booking and management lifecycle. Through the development of detailed system models, including Data Flow Diagrams (DFDs), Entity Relationship Diagrams (ERDs), and UML representations, the project effectively captures the flow of data and processes across modules such as User Registration, Resort Browsing, Booking, Food Ordering, Payment, Invoice Generation, and Feedback Submission.The ERD integrates key entities such as USER, RESORT, BOOKING, FOOD_ORDER, STAFF, PAYMENT, and INVOICE, ensuring relational consistency and supporting secure, real-time data transactions. The addition of wallet-based payment functionality enhances transaction transparency and minimizes third-party dependency, while the automated Staff Assignment module optimizes internal operations and workload distribution.

The Admin Dashboard, equipped with modules for Resort Management, Staff Control, Wallet Monitoring, and Analytical Reporting, provides administrators with a comprehensive view of the system's performance and user engagement. Features such as resort filtering, booking confirmation, live order tracking, and auto-generated PDF invoices contribute to a seamless and efficient user experience. The inclusion of a Chatbot Support System further enhances user interaction, offering instant assistance and information retrieval.Comprehensive testing strategies, including black-box and white-box testing, were employed to validate system reliability, usability, and performance. Test cases focused on critical workflows—such as user authentication, booking validation, payment deduction, and invoice generation—ensured the functional integrity of each component. Integration testing confirmed smooth communication between the frontend (React + Vite) and backend (Node.js + MongoDB) layers, while User Acceptance Testing verified that the application aligns with real-world expectations of resort users and administrators.

Overall, the Resort Management System demonstrates a well-structured, scalable, and efficient solution for modernizing the resort management process. It successfully integrates technological innovation with user-oriented design, ensuring a balance between automation, accuracy, and accessibility. The project establishes a solid foundation for future enhancements—such as integrating third-party payment gateways, AI-driven recommendations, and real-time analytics—paving the way for a smart, sustainable, and digitally empowered tourism ecosystem.

CHAPTER 9
REFERENCES

[1] R. S. Pressman, Software Engineering: A Practitioner's Approach, 8th ed. New York, NY, USA: McGraw-Hill Education, 2014.

[2] I. Sommerville, Software Engineering, 10th ed. Boston, MA, USA: Pearson Education, 2015.

[3] IEEE Standards Association, IEEE 829-1998 – Standard for Software and System Test Documentation. Piscataway, NJ, USA: IEEE, 1998.

[4] W3C, "HTML5 Specification," World Wide Web Consortium (W3C). [Online]. Available: https://www.w3.org/TR/html5/

[5] ERDPlus, "ERDPlus – Online Entity-Relationship Diagram Tool." [Online]. Available: https://erdplus.com/

[6] Postman, "Postman API Testing Tool." [Online]. Available: https://www.postman.com/

[7] React, "React – A JavaScript Library for Building User Interfaces." [Online]. Available: https://reactjs.org/

[8] Express.js, "Express – Fast, Unopinionated, Minimalist Web Framework for Node.js." [Online]. Available: https://expressjs.com/

[9] Multer, "Multer – Middleware for Handling multipart/form-data." GitHub Repository. [Online]. Available: https://github.com/expressjs/multer