Hotel Booking System

Japhta Kubeka(4323640)
Ditebogo Motshabi(4366345)
Thilitshi Mudzungwane(4335400)
Uhone Mukondeleli(4366051)
Kgothatso Dlamini(4326970)
Phungo Genuwine(4334893)

The Software Syndicate

ABSTRACT

This report provides a detailed look at the requirements analysis and the user modeling for a proposed Hotel Room Booking System. The goal of the system is to make it easy for guests to find, compare, and book hotel rooms online, while also giving hotel staff efficient tools to manage room availability, reservations, and customer information [1]. By examining the needs of both guests and staff, we have created user models, mental models, identified both functional and non-functional requirements, and illustrated the system's structure using UML use case and class diagrams. This analysis forms the foundational prototype for the subsequent design and implementation phases of the project.

1 INTRODUCTION

In today's digital economy, online booking systems have become important for enhancing user's experience and operational and operational efficiency [1]. Traditional methods of booking hotel rooms such as walk-ins, phone calls, often lead to limited access to information, miscommunication or even delays. Our web-based hotel booking system addresses these limitations by providing realtime access to room availability, room information, booking process while also streamlining hotel staff operations [2]. This project focuses on developing this system to reduce administration workload and ensure reliability. The focus of this stage is to understand and analyze how different users will perceive and interact with the system and the requirements that will guide its design and implementation.

2 Models

2.1 User Model

Figure 1 below shows the User Model for the Hotel Booking Website, which includes two main user types: Customer and Admin. Customer can register, log in, search for available rooms, book rooms, make payments, view their booking history, cancel booking and check out. While admin on the other hand are responsible for handling check-in process and managing errors.

The system supports customer with features like password verification and sending receipts once they are done making a payment. This model highlights the core tasks and interactions from each user type withing the system [1].

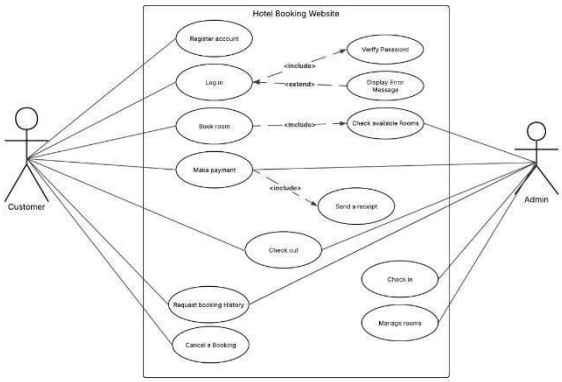

Figure 1: User Model for Customer and Admin

2.2 Mental Model

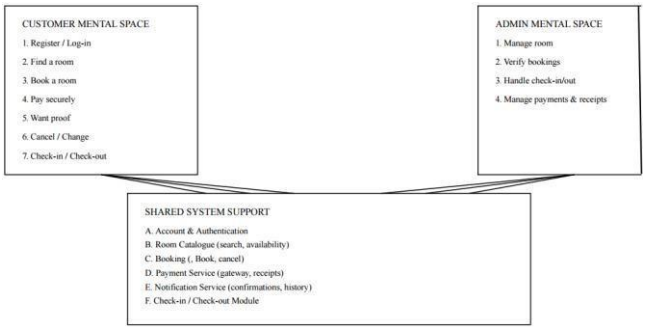

Figure 2: Customer and Admin Mental Space with Shared System Support

Figure 2 illustrates the mental model of both end-user (customer) and administrator withing the hotel booking system, together with the shared system components that support their interactions. This model helps to understand the activities of each user typer and the services the system must provide to facilitate them efficiently [2].

2.2.1 Customer Mental Space.

This represents the series of actions a typical guest would perform when using the system. It starts with account creation if they do not have an account already, or log-in if they already have an account, followed by searching for room that is available and that meets their preferences. After finding an option (room) of their choice, the customer can book the room and make payment to secure the room. The model also accounts for proof of booking by sending then a receipt via email, the ability to cancel or modify reservations and finally check-in and check-out once upon arrival and departure [1].

2.2.2 Admin Mental Space.

The admin mental space focuses on managing hotel's operations through the system. Admins can handle check-in, check-out, verify incoming bookings, manage room availability and details, and also manage payments and receipts. This ensures organized operation and accurate tracking of resources [1].

2.2.3 Shared System Support.

This system provides shared modules that bridge between both user perspectives. This includes account and authentication services to ensure user access to their data, room catalogue for searching availability, booking module for creating, canceling, or modifying reservations. This mental model clearly outlines how customer and admin interact with the system, ensuring a seamless and secure hotel booking experience [2].

2.3 Functionality Model

Table 1: Functionality

FUNCTIONALITY	DESCRIPTION
Register	Customer creates their account
Log in and log out	Customers can log into their account and log out as they want
Search Available	Customers can view types of rooms, prices and the availability of the rooms
Book Room	Customers select the room that they like or prefer and confirm their bookings.
Make Payment	When confirming their booking, then the customers can make a payment.
Cancel Booking	Customers can cancel their booking before check-in
View Booking History	Customer can access past reservation
Manage Rooms	Admin can generate revenue reports and bookings
Send Receipt	The systems automatically send receipt through email

2.4 Behavioral Model

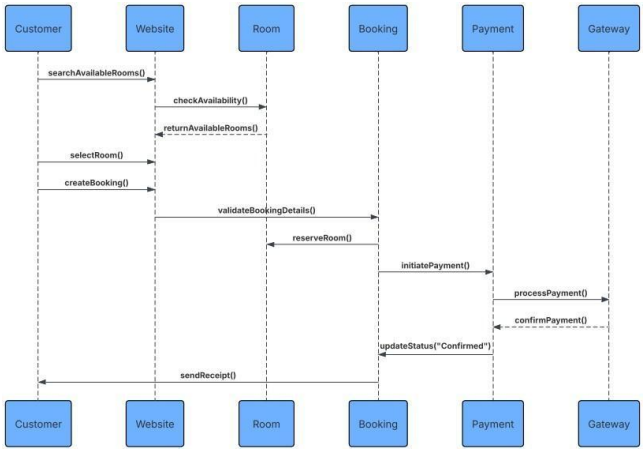

Figure 3: Behavioral Model

2.5 Requirement Analysis

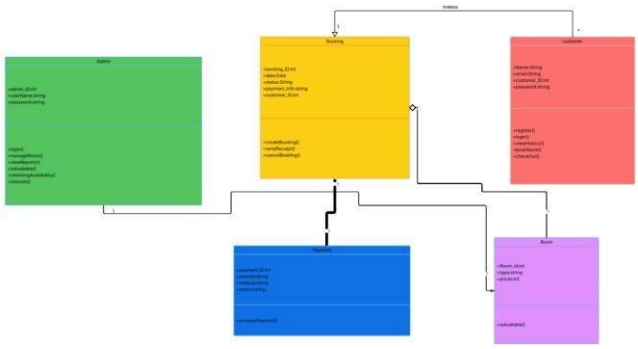

Figure 4: Requirement Analysis

Figure 4 illustrates the foundational requirement analysis for the system, outlining the core entities, their key attributes, and essential operations. This analysis identifies several major components, including Admin, Booking, Customer, and Payment, which forms the backbone of the hotel booking system's architecture. Each component is defined by its specific data structure. For instance, customer entity is composed of attributes such as customer ID, name, and password while Booking_Data includes date, status, and payment information. This diagram also captures the functional requirements methods such as Logiin(), createBooking(), register(), and processPayment(). This requirement analysis turns user needs and business rules into a clear plan, showing the data that needs to be managed and how they will be managed together with the main functions the system must perform to achieve its intended goal [3].

2.6 Design Model

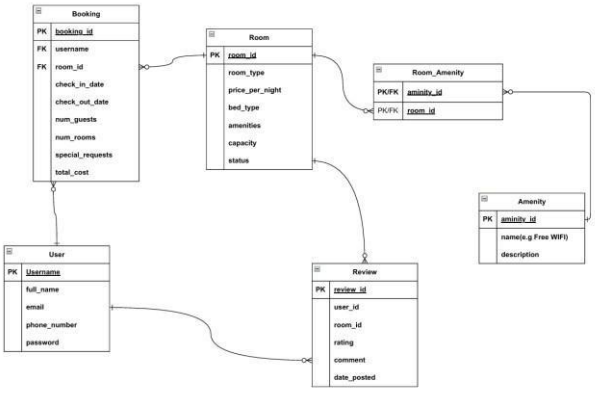

Figure 5 : Entity–Relationship Diagram (ERD) of the Hotel Reservation and Management System, illustrating the core entities (User, Room, Booking, Review, Amenity) and their relationships.

The hotel reservation and management application incorporates user authentication, room browsing and booking, guest reviews, and an administrative dashboard. It utilizes a relational database for persistence, with an Entity-Relationship Diagram (ERD) modeling the core data structures [4]. The ERD includes key entities such as User, which has attributes like username (primary key), full_name, email, phone_number, username, and password; Room, with attributes including room_id, room_type (e.g., Standard, Deluxe, Executive Suite), price_per_night, bed_type, amenities (stored as JSON or in a related table), capacity, and status (available or booked); Booking, featuring booking_id (primary key), username (foreign key), room_id (foreign key), check_in_date, check_out_date, num_guests, num_rooms, special_requests, status (reserved, checked_in, or checked_out), and total_cost; Review, with review_id (primary key), username (foreign key), room_id (foreign key), rating, comment, and date_posted; and Amenity, including amenity_id (primary key), name (e.g., Free WiFi, Jacuzzi), and description [5].

Relationships in the ERD

User has a one to many relationship with Booking, allowing a single user to initiate multiple bookings; a Room has a one-to-many relationship with Booking, enabling a single room to be associated with multiple bookings as long as dates do not overlap; a User has a one to many relationship with Review, permitting a single user to submit multiple reviews; a Room has a one to many relationship with Review, allowing a single room to receive multiple reviews; and a Room has a many to many relationship with Amenity, managed through a junction table like Room_Amenity with room_id and amenity_id as foreign keys. This ERD supports efficient queries, such as checking room availability via date overlap verification, calculating occupancy rates, and generating operational reports, while maintaining data integrity through foreign key constraints and triggers to prevent overlapping bookings [4].

2.7 Class Design

The system employs object-oriented principles in its class design, modeled with classes that encapsulate entities and behaviors, assuming implementation in a language like Java [6]. The User class includes private attributes such as username, fullName, email, phoneNumber and password (hashed) ,along with methods like a constructor, register() to insert into the database, login() to authenticate the user, and getBookings() to retrieve bookings from the database. The Room class features private attributes including roomId, roomType, pricePerNight, bedType, amenities (as a list), capacity, and available (boolean), with methods such as a constructor, isAvailable() to query for date overlaps, and updateStatus() to update the database. The Booking class has private attributes like bookingId, user (object), room (object), checkIn, checkOut, numGuests, numRooms, specialRequests, status, and totalCost, including methods like a constructor that computes cost, confirm() to update status and notify the user, calculateCost() to return the total based on nights, price, and rooms, and cancel() to update status and release the room. The Review class includes private attributes such as reviewId, user (object), room (object), rating, comment, and datePosted, with

methods like a constructor and post() to insert into the database [6].

2.8 Architectural Design

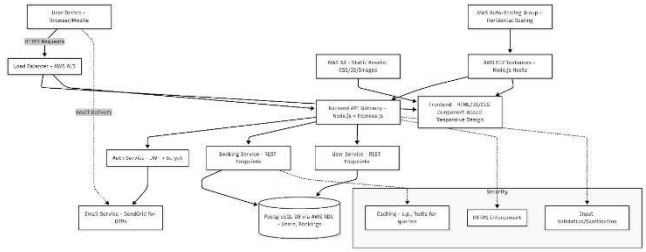

Figure 6: Architectural Design of the Hotel Reservation and Management System.

Architecturally, the system adopts a RESTful microservices pattern with a component-based frontend to enhance modularity and scalability [7]. Key components include a frontend built with HTML, JavaScript, and CSS for styling (featuring brown buttons, brown backgrounds, and responsive design aligned with mockups); a backend using Node.js with Express.js for API routing; a PostgreSQL database for storing user data, bookings, and related entities; JWT for authentication and session management, paired with bcrypt for password hashing; SendGrid for email services like sending OTPs during password resets; and AWS hosting infrastructure, including EC2 for compute, S3 for static assets, and RDS for PostgreSQL. This setup supports secure protocols like HTTPS, load balancing for up to 1,000 concurrent users initially, and modular development [6].

Non-functional requirements emphasize performance with page load times under 1.5 seconds, achieved via optimized queries, caching, and minified assets; scalability to handle 1,000 concurrent users initially through horizontal scaling with AWS auto-scaling groups; security measures including HTTPS enforcement, JWT authentication, input validation and sanitization to prevent injections, and bcrypt hashing; usability through responsive design for all devices and compliance with WCAG 2.1 for accessibility (e.g., ARIA labels and keyboard navigation); and consistency with a uniform colour scheme (brown buttons, brown backgrounds) and typography across all pages, as per mockups [7].

2.9 Interface Design

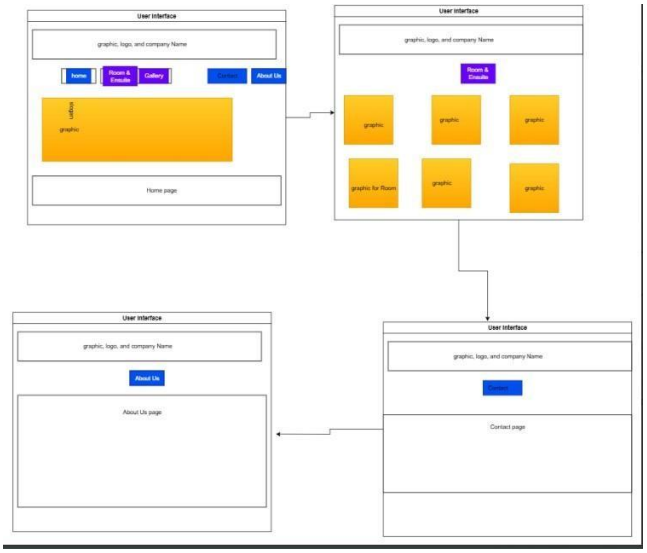

Figure 7: Interface Design of the Hotel Reservation and Management System.

Encompasses both APIs and UIs. API interfaces are RESTful endpoints for system interactions, such as POST /api/auth/signup to register a new user (with body parameters like full_names, email, phone_number, username, and password), POST /api/auth/signin to authenticate and return a JWT (body: username, password), POST /api/auth/reset-password to send an OTP via email (body: email), POST /api/auth/verify-otp to verify the OTP and reset the password (body: email, otp_code, new_password), GET /api/rooms checkIn=yyyy-mm-dd&checkOut=yyyy-mm-dd to retrieve available rooms, and POST /api/bookings to create a booking. Data exchange occurs in JSON format with robust error handling via HTTP status codes [6]. The user interface design follows usability principles, with a continuous brown-and-white colour scheme to imply elegance. The homepage features a header with the “Syndicate hotel” logo top left, a brown “Log in” button and a white “Sign Up” button both top right, a hero section with a full-width hotel room image (e.g., with double bed) and overlay text (“Find your perfect stay”, “Rooms designed for your comfort…”), a search bar for check-in, check-out, guests, and rooms, and a brown, rounded “Book now” button, as well as a footer with quick links, contact information, and social media links [1].

The authentication suite comprises a login page with form fields for username and password, a “Show Password” checkbox, a “Sign In” button, and links to “Forgot Password?” and “Sign Up”; a sign-up page with fields for full names, email, phone number, username, password, and confirm password, a “Show Password” checkbox, a “Sign Up” button, and a link to “Sign In”; and a reset password page with an email input, instructional text (“Enter your email to reset your password, an OTP will be sent to your email. OTP will expire in 3 minutes”), and a “Send Email” button. The booking flow in the initial phase includes an availability grid and

guest form, to be developed in later sprints, while the admin backend for future implementation will manage rooms, bookings, and content. Navigation features links to Home, Rooms & Ensuites, Gallery, Contact, and About Us, with forms including placeholders (e.g., date formats), card-based displays for rooms and reviews, and filters for bookings [2].

2.10 Component-Level Design

•

```
//Login Component:
function login(username, password):
    user = queryDatabase("SELECT * FROM users WHERE username = ?", username)
    if user exists and verifyHash(password, user.password):
        createSession(user)
        if user.role == "admin":
            redirect("/admin.html")
        else:
            redirect("/homepage.html")
        return true
    else:
        displayError("Invalid credentials")
        return false

//Room Search Component:
function searchRooms(checkIn, checkOut, guests, rooms):
    validateDates(checkIn, checkOut)  // Ensure checkIn < checkOut
    availableRooms = queryDatabase("SELECT * FROM rooms r WHERE r.room_id NOT IN
                                    (SELECT b.room_id FROM bookings b WHERE
                                     b.check_in < ? AND b.check_out > ? /* handle overlaps */)", checkOut, checkIn)
    applyFilters(guests, rooms, amenities)
    sortBy("recommended")  // e.g., rating DESC
    renderAsCards(availableRooms)  // With images and prices
    return availableRooms

//Booking Confirmation Component:
function confirmBooking(bookingDetails):
    if not isLoggedIn():
        redirectToLogin()
    totalCost = calculateNights(checkOut, checkIn) * room.pricePerNight * numRooms
    insertIntoDatabase(bookingDetails)
    updateRoomStatus("booked", dates)
    sendConfirmationEmail(user.email, bookingDetails)
    redirectToPayment()

//Admin Dashboard Component:
function loadDashboard():
    authorizeAdmin()
    totalGuests = queryDatabase("SELECT COUNT(*) FROM bookings WHERE status = 'checked_in'")
    occupancyRate = (countBookedRooms() / totalRooms) * 100
    checkInsToday = queryDatabase("SELECT COUNT(*) FROM bookings WHERE check_in = CURRENT_DATE")
    revenue = sum(queryDatabase("SELECT total_cost FROM bookings WHERE date >= WEEK_START"))
    renderStatsCards(totalGuests, occupancyRate, checkInsToday, revenue)
    loadGuestTable(withPagination)
```

Figure 8: Pseudocode for Component-Level Design of the Hotel Reservation and Management System, illustrating the key functional components such as Login, Room Search, Booking Confirmation, and Admin Dashboard, along with their interactions.

The login component involves a function that queries the database for a user by username, verifies the hashed password, creates a session if valid, redirects to admin or homepage based on role, and displays an error otherwise. The room search component validates dates, queries the database for available rooms excluding those with overlapping bookings, applies filters for guests, rooms, and amenities, sorts by recommendation (e.g., rating descending), and renders results as cards with images and prices. The booking confirmation component checks if the user is logged in, calculates total cost based on nights, price, and rooms, inserts details into the database, updates room status, sends a confirmation email, and redirects to payment. The admin dashboard component authorizes the admin, queries for stats like total guests, occupancy rate, checkins today, and weekly revenue, renders them as stats cards, and loads a paginated guest table [7].

2.11 User interface

There are number of interfaces which will face by the users and admins to manage or get their desired requirements. Home Page, About Page, Our Gallery Page, View Rooms, Contact Us page, Reviews, Signup, Login, Reservation Form, Booking Success Message,Admin Dashboard, Guest management.

2.11.1 Home Page.

A home page is generally the main page a visitor navigating to a website from a web search engine will see, and it may also serve as a landing page to attract visitors. The home page is used to facilitate navigation to other pages on the site by providing links to prioritized and recent articles and pages.

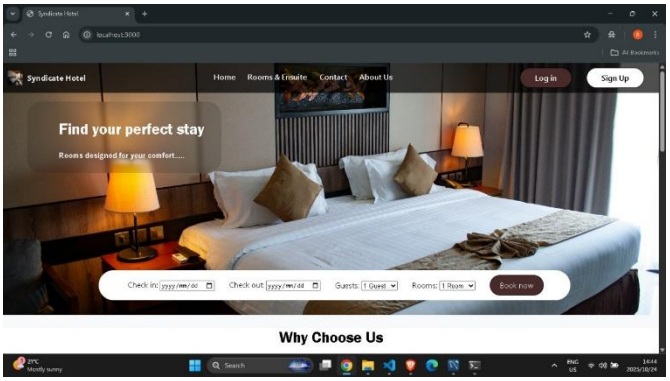

Figure 9: Home Page

2.11.2 About Page.

Here the user can see all the services provided by the Syndicate hotel.

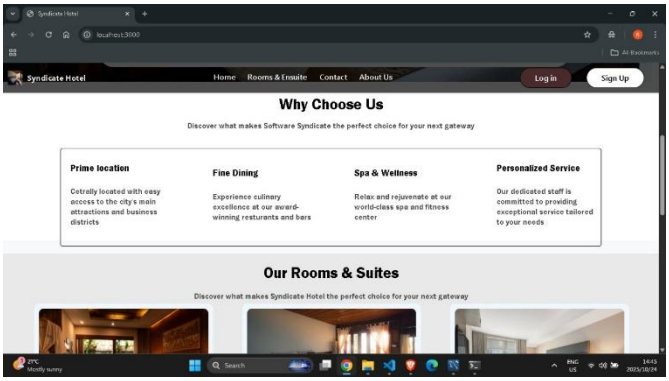

Figure 10: About Page

2.11.4 View Rooms.

Here users can search the desired Room which is available.

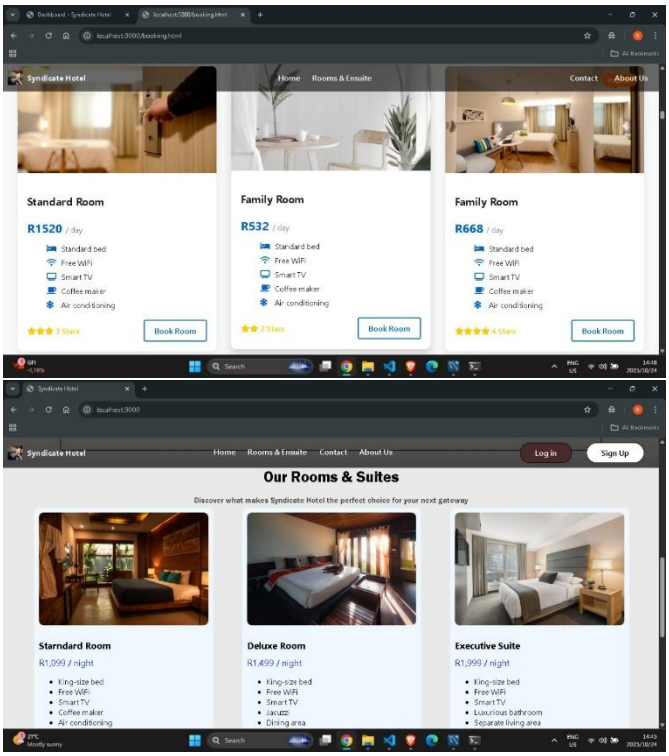

Figure 11: View Rooms

2.11.5 Contact Us page.

Here the user can see the location of the Hotel and all the contact information (Email, Address, Phone NO) which are provided by the hotel.

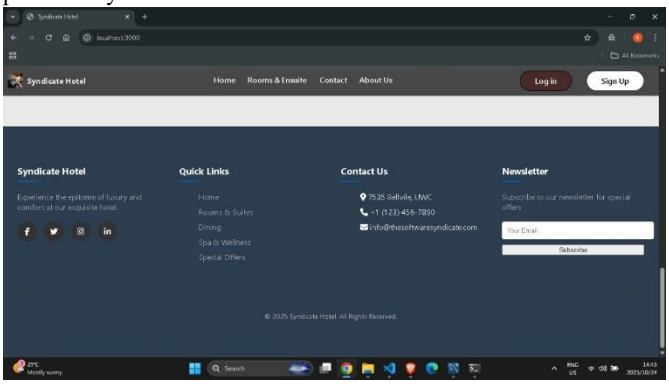

Figure 12: Contact Us page

2.11.6 Reviews.

Here User can see reviews

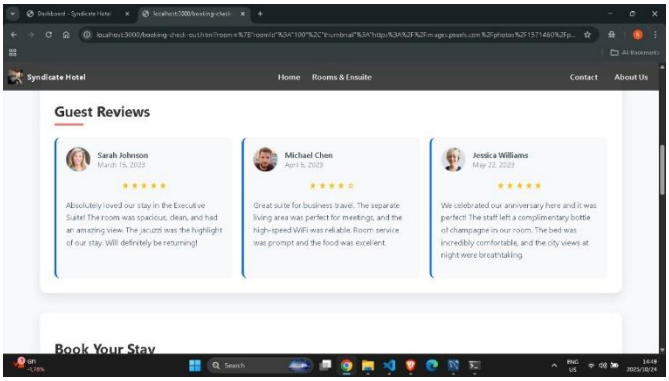

Figure 13: Reviews

2.11.7 Signup page.

User can sign up.

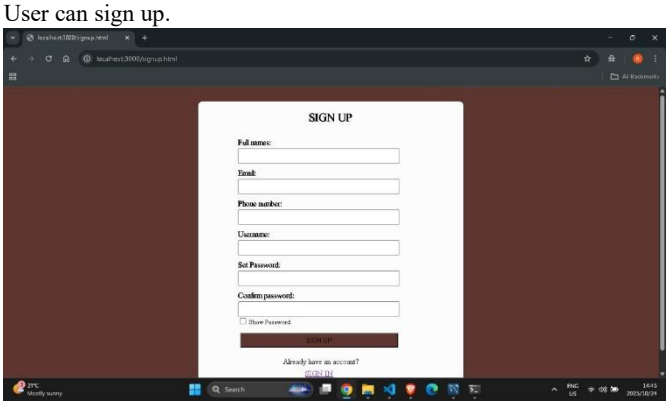

Figure 14: Signup page

2.11.8 Login page.

User can log in if they already have an account.

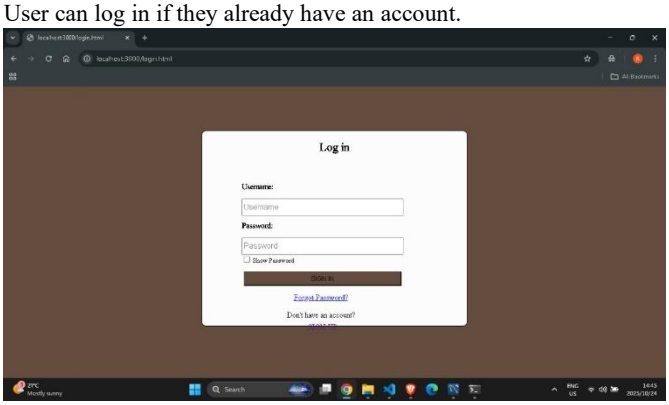

Figure 15: Login page

2.11.9 Reservation Form

Here the users can submit their personal information and reservation information once he get access to the system.

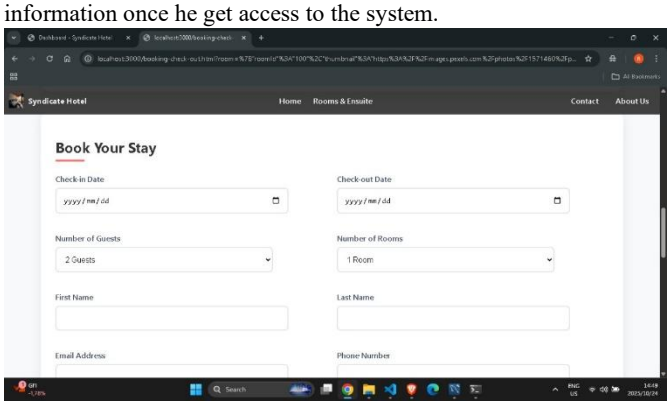

Figure 16: Reservation Form

2.11.10 Payment and Booking Success Message.

When a user submit their booking application successfully it will display a message (Your Booking application has been sent).

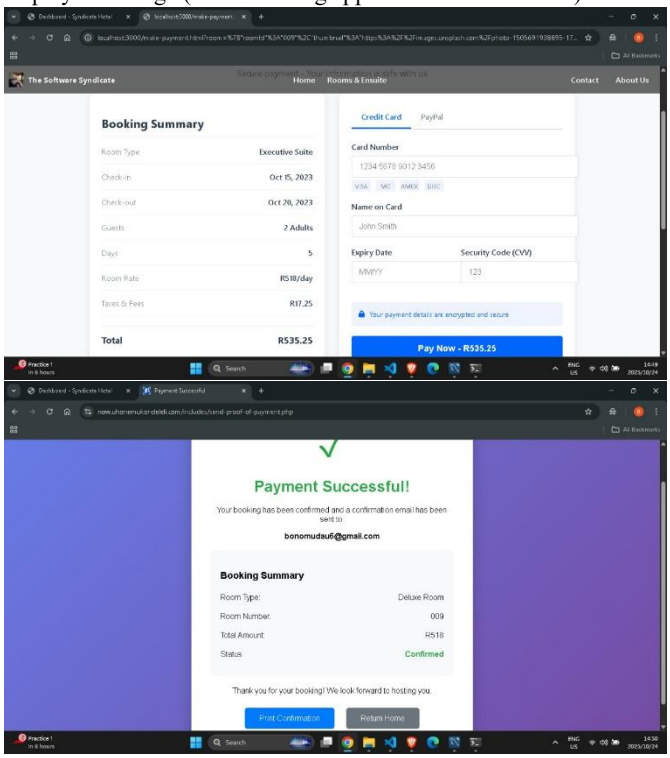

Figure 17: Payment and Booking Success Message

2.11.11 Admin Dashboard.

In this interface Admin can see all the functionality of the system at one.

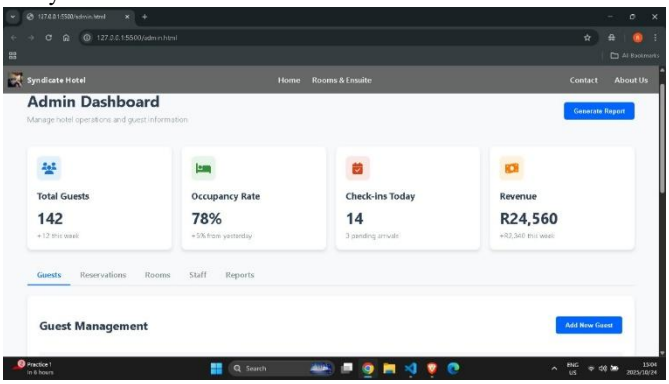

Figure 18: Admin Dashboard

2.11.12 Guest management.

Here the admin can see all the Payment details.

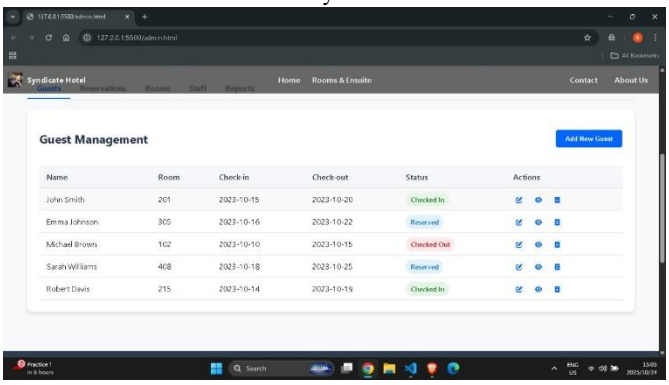

Figure 19: Guest management

Design principles adhere to established models, including a user model that reflects mental models where guests expect intuitive booking flows similar to industry standards and admins anticipate analytical dashboards; a design model that maps to the ERD and class structures; an implementation model combining frontend technologies with backend logic, following golden rules like consistency, feedback, and error prevention; and a manifested model ensuring high usability with minimal clicks for tasks, accessibility features, and responsive design [2] .

3 Software Engineering Aspects

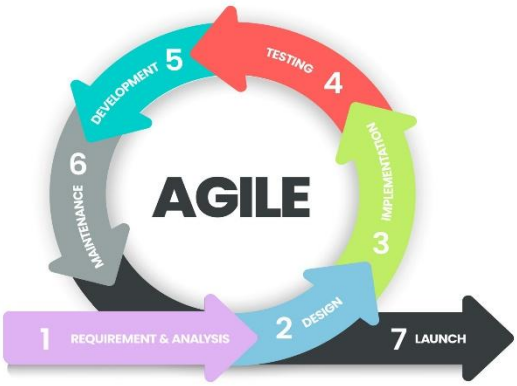

Figure 20: Agile Model

3.1 Process Model

We used Agile Process model to develop our prototype. This allowed us to quickly build a core set of features such as e.g. room listing, search, a basic form and then gradually add more functionalities such as user account, payment simulation based on feedback and priority. This approach was crucial for managing changing requirements and delivering a functional prototype [6]. During implementation, the Agile process was extended into three structured sprints:

- **Sprint 1:** Development of the database and user authentication (registration and login).
- **Sprint 2:** Core booking flow, including room search, booking, and cancellation features.
- **Sprint 3:** Admin dashboard implementation and testing refinements.

Daily stand-ups via Discord and bi-weekly retrospectives supported team communication and adaptability. For example, an OTP-based password reset feature was introduced in response to security feedback. This approach ensured responsiveness to change, improved collaboration, and enhanced overall software quality [6].

3.2 Principles

We applied modularity by designing front-end architecture that is prepared for future back-end integration. We separate our code into distinct modules for UI, Logic and mock data services. This later by simply replacing the mock data modules with real API calls, ensuring our prototype is scalable and maintainable [7].

We applied abstraction and information hiding to simplify complex operations and manage code complexity. For instance, a function like validateBookingForm() encapsulates the detailed logic of checking data conflicts, verifying input formats, and displaying error messages. The main application code simply calls this single function, remaining completely unaware of its internal steps. This design hides the implementation details, demonstrating information hiding and provides a clean, abstracted interface for form validation, making the code easier to understand, use, and modify later [6].
Reusability was prioritized through shared utilities, such as the calculateCost() function, which supports both the booking system and administrative revenue reports. These principles ensured the prototype remained scalable, adaptable, and efficient for future enhancements.

3.3 Methods

Our project used fundamental Software engineering techniques throughout. For Requirement Analysis we began by identifying core user stories such as the need to search for rooms and complete a booking. During the design phase, we created wireframes to plan the user experience and architected a modular front end to ensure scalability. Program Construction was executed using HTML, CSS and Java Script. Testing was conducted continuously through manual checks and peer reviews to features and ensure quality Throughout the prototype, the code was written with Support in mind, featuring clear documentation and a structure that simplifies future debugging and expansion [7].

3.4 Goals

Paragraph contents of 2.1 are typed here. You may not need all the paragraph divisions used below. They are shown here in case you need to – so see how these are formatted. The core functionality of our prototype was to simulate the available rooms, selecting dates, and completing a reservation. This primary functional goal was successfully achieved, delivering an interactive and representative user experience. Usability, particularly understandability, was a fundamental design objective. We focused on creating an intuitive interface with clear navigation and minimalistic forms to ensure users could complete the booking process effortlessly and without confusion. Finally, maintainability was a guiding principle in our development process. By adhering to a modular design and writing clean, well-documented code, we ensured that the codebase remains easy to update, debug, and extend for future developers
[1].

3.5 Umbrella Activities

Software Quality Assurance (SQA) was maintained through. continuous manual testing of all user interface components. and functional workflows, such as form validation and navigation flows. This was complemented by peer code. reviews, where team members systematically examined each other. other's work to identify errors and enforce coding standards. Furthermore, informal technical reviews were held at key. milestones, such as integrating new features like search or booking modules. These sessions ensured that each component met our technical and

design criteria and integrated seamlessly with the existing system architecture [6].

4 Prototype demonstration

This section provides a guided overview of the functional. prototype, as demonstrated in the accompanying video. The video showcases the core user journey and validates the successful implementation of our key software engineering goals [2].

5 IMPLEMENTATIONS

The implementation focused on creating a web-based application that realizes the core functionalities identified in the user model, mental model, and functionality model. This included user authentication, room search and booking, payment simulation, and basic admin dashboard features. Given the project's scope as a prototype, we prioritized a fullstack web application using modern, accessible technologies to ensure rapid development and alignment with the architectural design (RESTful APIs, modular frontend, and relational database).

5.1 Hardware and Software Tools Used

5.1.1 Hardware.

Development Machines each team member used personal laptops with the following specifications: The system specifications include an Intel Core i3 processor with at least eight cores to ensure efficient compilation and testing performance. It is equipped with 8 GB of DDR4 RAM, which can be upgraded to 16 GB to better handle database simulations and browser development tools. Storage is provided by a fast 512 GB NVMe SSD, enabling quick boot times and efficient file input/output operations during coding sessions. For graphics performance, the setup utilizes either integrated Intel Iris Xe or NVIDIA GeForce RTX 3050, ensuring smooth rendering of user interface prototypes in browsers. The operating systems used include Windows 11 Pro for most team members and macOS Ventura for one member performing crossplatform development and testing using Xcode.

Testing Devices for usability testing, we utilized: The testing setup also includes mobile devices such as the Samsung A03 and iPhone 14 running iOS 17 to verify responsive design and ensure compatibility across different platforms. A stable Wi-Fi connection with a speed of 100 Mbps is available to support seamless online collaboration, software updates, and testing of web-based applications.

5.1.2 software tools.

The development environment was structured to ensure efficiency, collaboration, and scalability throughout the project. Visual Studio Code (VS Code) was chosen as the primary Integrated Development Environment (IDE) due to its lightweight performance, extensive extensions ecosystem, and cross-platform compatibility. Key extensions included Live Server for real-time frontend previews, Prettier and ESLint for code formatting and linting, Thunder Client for API testing (as a simpler alternative to Postman), and the PostgreSQL extension for executing database queries directly within the editor.

For the frontend, HTML5, CSS3 (featuring a brown-themed design), and vanilla JavaScript ES6+ were utilized to maintain a lightweight and fundamental prototype without relying on heavy frameworks like React. On the backend, Node.js with Express.js was implemented to handle RESTful API endpoints, while PostgreSQL served as the relational database, hosted locally through Docker for streamlined setup and environment isolation. Security and authentication were managed using bcrypt for password hashing and jsonwebtoken (JWT) for token-based authentication. Additionally, Nodemailer was integrated with a sandbox SendGrid account to handle email receipts, limited to 100 test emails per day. Version control and team collaboration were facilitated through Git and GitHub, with feature-specific branches (e.g., feature/bookingmodule) and pull requests used for peer review and quality assurance. pgAdmin supported database management, enabling visual schema design and query execution in line with the Entity Relationship Diagram (ERD) presented in Figure 5.

For testing and deployment, Jest was employed for backend unit testing, while Docker ensured consistency by containerizing both the application and database. The project was deployed on Heroku's free tier for demonstration purposes, with environment variables securely managing sensitive credentials. Finally, Figma was used to refine the UI mockups (as shown in Figure 7), and Draw.io, integrated within VS Code, assisted in updating UML diagrams throughout iterative development stages.

5.2 Implementation Details

Github link: https://github.com/hotel-bookingdevs/Syndicate-Hotel

The prototype was implemented in phases that aligned with Agile sprints, each lasting two weeks. Key components were developed incrementally, beginning with the database setup. Using the EntityRelationship Diagram (Figure 5) as a guide, tables were created in PostgreSQL to represent core entities including User, Room, Booking, Review, and Amenity. The database schema was implemented through SQL scripts executed via pgAdmin, ensuring a structured foundation for subsequent features such as booking workflows and user interactions.

6 Testing

To verify that the Hotel Booking System aligns with user needs, identifies any glitches, and offers a solid user experience, we implemented black box testing as our key testing method. This approach was selected because it evaluates the system's behavior from a user's viewpoint, without delving into the code's internals, making it perfect for checking functional elements like booking processes and non-functional aspects like speed and ease of use, as specified in our user models and requirements analysis.

6.1 Black-Box Testing Approach

Black box testing designs tests based on the system's specs and anticipated outputs, drawing from functional features (e.g., sign-up, room hunting, booking, payment, cancellation) and non-functional ones (e.g., user-friendliness, efficiency, security). The objective was to ensure everything operates correctly for users and admins, pinpoint functional flaws, and assess how intuitive and responsive the system is.

6.2 Designing Test Cases

6.2.1 User Registration (Functional).

Tested with standard inputs like name, email, phone, username, password. Expected: successful account creation, data saved, redirect to login. It passed. For errors, trying an existing username showed the proper message. Initially, duplicates emails were allowed, but we fixed it by enforcing unique emails in the database.

6.2.2 Room Search (Functional).

Used inputs like check-in 2025-11-01, check-out 2025-11-03, 2 guests, 1 room. Expected: list of available rooms with details. It passed, displaying correct options. Invalid dates triggered errors correctly. Early bug with overlapping bookings caused inconsistent results; fixed by optimizing the date-check queries.

6.2.3 Booking Confirmation and Payment (Functional).

Selected a room, confirmed, simulated payment. Expected: booking saved, room marked booked, confirmation email sent via SendGrid. Passed, including blocking unavailable rooms. Bug with special characters in requests breaking payment; resolved with input sanitization.

6.2.4 Usability (Non-Functional).

Navigated the full flow on desktop (Windows 11, Chrome) and mobile (iPhone 14, Safari). Expected: smooth, responsive, 3-5 clicks to book. Passed intuitive, completed in 4 clicks. Accessibility checked with screen readers. Mobile button misalignment fixed via CSS adjustments.

6.2.5 Performance (Non-Functional).

Simulated 50 concurrent searches with Thunder Client. Expected: load under 1.5 seconds. Passed at 1.2 seconds average after optimizations like caching. Slow queries initially; added indexes to speed up.

6.3 Executing Tests and Tools Used

Tests ran iteratively in Agile sprints. Manual testing on devices like Samsung A03, iPhone 14. Thunder Client for API checks (e.g., booking endpoints). Jest for backend units like cost calculations. Peer reviews helped spot issues. Documented in Google Sheets with inputs, expectations, results, fixes.

6.4 Results and Quality Insights

Testing validated core functions and non-functions, fixing bugs like email duplicates and UI glitches for better reliability. Usability confirmed minimal clicks and intuitiveness. Highlighted future needs, like auto-testing for larger scales.

7 Reflection on the SE Process

We applied software engineering elements process, principles, methods, goals, umbrella activities, and tools throughout the Hotel Booking System development. Here's our reflection on the Agile model we chose, why it fit, and the honest highs and lows from our group, The Software Syndicate.

7.1 Why We Picked Agile

Agile was chosen for its adaptability to changing requirements and iterative style, suiting our prototype project. Unlike Waterfall's rigidity, Agile let us tackle priorities like room search and booking early, then refine based on feedback within our tight schedule. It handled additions like OTP resets well, with sprints and stand-ups promoting teamwork.

7.2 Positive Experiences with Agile

7.2.1 Flexibility and Adaptability.

Agile let us pivot fast. In Sprint 2, user feedback flagged unclear booking form errors, so we enhanced validateBookingForm() in Sprint 3 for better messages, showcasing Agile’s strength in handling change.

7.2.2 Team Collaboration

Daily Discord stand-ups kept everyone aligned, sharing progress and ideas like adopting SendGrid for email receipts, quickly integrated after discussion.

7.2.3 Steady Progress

Simulated Splitting work into sprints (database, booking flow, admin dashboard) ensured momentum. Completing authentication in Sprint 1 set a strong foundation.

7.2.4 Feedback-Driven Growth.

Simulated weekly reviews refined processes, like better Git branching to reduce conflicts.

7.3 The Challenges

7.3.1 Time Management.

Agile’s flexibility led to scope creep, like over-polishing the brown-themed UI, delaying cancellation features. Stricter prioritization is needed next time.

7.3.2 Uneven Workloads.

Skill gaps meant some took on more (e.g., PostgreSQL setup), causing frustration in Sprint 1. Peer mentoring helped, but better task balancing is key.

7.3.3 Manual Testing Limits.

Time constraints meant heavy reliance on manual testing. It worked but was slow; tools like Selenium could’ve boosted efficiency.

7.3.4 Meeting Overload.

Daily stand-ups sometimes felt repetitive when progress was light. Adjusting frequency for quieter phases could help.

7.4 SE Principles, Methods, Goals, and Umbrella Activities

We leaned on principles like modularity (separating UI/logic for scalability), abstraction (hiding complexity in functions like validateBookingForm()), and reusability (shared calculateCost() across modules). Methods included user story analysis, wireframing,

and iterative coding with HTML, CSS, JavaScript, and Node.js, aligning with user needs. Our goals functionality, usability, maintainability were met, as tests confirmed smooth navigation and booking. Umbrella activities like Software Quality Assurance (peer reviews, continuous testing) ensured high-quality outputs, with sprint reviews catching integration issues early.

7.5 Tools and Their Impact

Tools like VSCode, Git, PostgreSQL, Figma were essentials VSCode for efficient coding, Git for control. Docker had a learning curve, delaying setup; Figma aided design but had sync issues. Key lesson: ensure team training.

7.6 Overall Takeaway

Agile matched our needs for fast prototyping and feedback, resulting in a functional prototype. Modular design and iterations helped meet requirements. Challenges like workloads highlight areas for better planning and tools. Overall, the SE framework supported collaboration, quality, and a user-aligned deliverable.

8 CONCLUSIONS

In conclusion, the hotel booking system provides solutions that enhance user experience for customers and efficiency for administrators. This service enables customers to easily search for available rooms, book their preferred room, complete secure payments, and manage their reservations. Overall, the system achieves its goal of simplifying hotel bookings and delivering seamless booking experience [1].

References

[1] OTRAMS, "The ultimate guide to hotel reservation system," OTRAMS, Oct 2022. [Online]. Available: https://www.otrams.com/ultimate-guide-to-hotelreservationsystem/.

[2] A. Xu, "Hotel reservation system," ByteByteGo, [Online]. Available: https://bytebytego.com/courses/systemdesigninterview/hotel-reservation-system.

[3] GeeksforGeeks, "How to design a database for booking and reservation systems," GeeksforGeeks, Jul 2025.. [Online]. Available: https://www.geeksforgeeks.org/dbms/howtodesign-a-database-for-booking-and-reservation-systems/.

[4] GeeksforGeeks, "How to design ER diagrams for booking and reservation systems," GeeksforGeeks, [Online]. Available: https://www.geeksforgeeks.org/dbms/how-to-designerdiagrams-for-booking-and-reservation-systems/.

[5] J. Sandoval, "Data model for a hotel management system," Vertabelo Blog, Sep 2023. [Online]. Available: https://vertabelo.com/blog/data-model-for-hotelmanagementsystem/.

[6] NexSoftSys, "Design backend system of an online hotel booking app using Java microservices," NexSoftSys, 2024. [Online]. Available: https://www.nexsoftsys.com/articles/how-to-designbackendsystem-of-an-online-hotel-booking-app-usingjava.html.

[7] A. Corso, "Software architecture - Hotel reservation booking system," austincorso.com, 23 Jan 2020.. [Online]. Available: https://austincorso.com/2020/01/23/hotel-reservationbookingsystem.html.