TRIBHUVAN UNIVERSITY

INSTITUTE OF SCIENCE AND TECHNOLOGY

Project Report On

Vacation Home Rental System

Under the Supervision of

Er. Ganga Subba

Submitted by:

Sachin Maharjan (26422/077)

Sarvajit Khadka (26430/077)

Sushank Gurung (26437/077)

A project report submitted to the Department of Computer Science and Information Technology in partial fulfillment of the requirements for the degree of Bachelors of Science in Computer Science and Information Technology.

Submitted to:

Tribhuvan University

Institute of Science and Technology

February, 2025

SAGARMATHA COLLEGE OF SCIENCE AND TECHNOLOGY

AFFILIATE TO TRIBHUVAN UNIVERSITY

SUPERVISOR'S RECOMMENDATION

I hereby recommend that the project entitled **"Vacation Home Rental System"** prepared and submitted by Sachin Maharjan, Sarvajit Khadka and Sushank Gurung in partial fulfillment of the requirements for the degree of Bachelor of Science (B. Sc) in Computer Science and Information and Technology awarded by Tribhuvan University, has been completed under my supervision. I recommend the same for the acceptance by Tribhuvan University.

Er. Ganga Subba

Associate Professor

Supervisor

SAGARMATHA COLLEGE OF SCIENCE AND TECHNOLOGY

AFFILIATE TO TRIBHUVAN UNIVERSITY

CERTIFICATE OF APPROVAL

This is to certify that this project prepared and submitted by Sachin Maharjan, Sarvajit Khadka and Sushank Gurung entitled **"Vacation Home Rental System"** in partial fulfillment of the requirements for the degree of Bachelor of Science (B. Sc) in Computer Science and Information Technology has been well studied. In our opinion, it is satisfactory in the scope and quality as a project for the required degree.

Er. Ganga Subba
Associate Professor
Supervisor
Sagarmatha College of Science & Technology

Er. Pratik Timalsena
Head of Department
Sagarmatha College of Science & Technology

External Examiner
Tribhuvan University

ACKNOWLEDGEMENT

Foremost, we would like to express our sincere gratitude towards our project supervisor **Er. Ganga Subba**, Associate Professor , Department of Computer Science and Information Technology, Sagarmatha College of Science and Technology, for her insightful advice, motivating suggestion, invaluable guidance, help and support in successful progress of this project.

In addition to our mentor, we would like to give a special gratitude to our HOD Mr. Pratik Timalsena, Mr. Shambhu Pariyar Department of Computer Science and Information Technology, whose contribution in stimulating suggestions and encouragement that helped us to coordinate the project and the staffs of Computer Science and Information Technology Department of Sagarmatha College of Science and Technology.

Sachin Maharjan (26422/077)
Sarvajit Khadka (26430/077)
Sushank Gurung (26437/077)

COPYRIGHT ©

DEPARTMENTAL ACCEPTANCE

The project work entitled **"Vacation Home Rental System"**, submitted by **Sachin Maharjan, Sarvajit Khadka, Sushank Gurung** in partial fulfillment of the requirement for the award of the degree of **"Bachelor of Computer Science and Information Technology"** has been accepted as a bonafide record of work independently carried out by team in the department.

Er. Pratik Timalsena
Head of Department
Department of Computer Science and Information Technology,
Sagarmatha College of Science and Technology,
Tribhuvan University Affiliate,
Sanepa, Lalitpur,
Nepal.

ABSTRACT

The **Vacation Home Rental System** is a platform designed to improve accessibility, offer personalized experiences, and foster trust in the home-sharing market. Focused on vacation homes in Nepal, it caters to foreign travelers, young adults, and families by enabling users to seamlessly act as both hosts and guests. The system features secure registration, a hybrid recommendation algorithm combining content-based and collaborative filtering, and advanced search options based on user preferences such as guest capacity and amenities. Administrators can manage listings, approve or reject properties, delete users, and monitor business sales. JWT is used for secure password handling, with an email OTP feature for password recovery. Built on PostgreSQL, React, and Express with Node.js, and utilizing AWS S3 for cloud image storage, the platform ensures performance, reliability, and scalability.

Keywords: Hosts, Guests, Listing, Booking, Recommendation, Nepal

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

API	Application Programming Interface
AWS	Amazon Web Services
CSS	Cascading Style Sheet
HTML	Hyper Text Markup Language
JS	JavaScript
JSON	JavaScript Object Notation
JWT	JSON Web Token
PERN	Postgres, Express, React, Node.js
S3	Simple Storage Service
UI	User Interface
UX	User Experience
VHR	Vacation Home Rentals

CHAPTER 1

INTRODUCTION

1.1 Background

Vacation rentals are private accommodations, such as apartments, houses, or cottages, rented out to travelers for short stays. These rentals offer more privacy, space, and often access to kitchens and amenities compared to traditional hotels. The vacation rental market has grown due to increasing demand for personalized travel experiences and the rise of home-sharing platforms.

The Vacation Home Rental System aims to meet this demand by providing an easy-to-use platform for both hosts and guests. The system will offer responsive design, secure payment processing, and a robust backend, making it simple for hosts to manage listings and for guests to book stays.

Nepal is becoming a popular travel destination, and vacation rentals are growing in importance here. With its rich culture and stunning landscapes, Nepal offers tourists a chance to stay in unique properties, supporting local communities and promoting sustainable tourism. The system's goal is to provide travelers with an authentic and memorable experience that connects them with Nepal's heritage and local communities.

1.2 Problem Statement

Traditional hotel bookings often lack personalization, space, and local experiences, which many travelers now seek. With the rise of online platforms, there is an increasing demand for a reliable and user-friendly vacation rental system that provides unique and cost-effective accommodations. In Nepal, where tourism is flourishing, vacation rentals are becoming a preferred option. However, the absence of a seamless platform connecting travelers with local hosts creates challenges. Guests struggle with limited accessibility to vacation homes due to scattered options, while homeowners face difficulties in managing

bookings, availability, and customer interactions. Additionally, existing platforms often fail to address user experience gaps, such as transparency, trust, and ease of use.

1.3 Objectives

The objectives of this project are:

- To create a user-friendly platform where guests can easily discover, book, and pay for vacation homes securely, while enabling hosts to efficiently list and manage their properties.

1.4 Scope and Limitation

1.4.1 Scope

The vacation home rental system is a web-based platform designed to cater to the needs of travelers and property owners in Nepal. It focuses on short-term vacation rentals, emphasizing cultural integration by promoting local experiences, events, and activities alongside property listings. The system serves two primary user groups: property owners, including individuals and businesses managing homes, apartments, cottages, and unique stays, who can list their properties and manage bookings; and travelers, both domestic and international, seeking convenient and personalized accommodations.

1.4.2 Limitations

T he vacation home rental system has certain limitations. It relies on stable internet access for bookings and communication, which may be a challenge in remote areas. Maintaining consistent quality standards across all listed properties can be difficult due to variations in host service levels. Pricing may also lack consistency as hosts set their own rates, which are only reviewed upon request. Additionally, the system supports a limited number of payment gateways, which may not accommodate all regional payment preferences.

1.5 Development Methodology

The Waterfall model was chosen for our vacation home rental system because it provides a structured, sequential approach ideal for projects with clear and stable requirements. With well-defined roles, payment integrations, and workflows, the model ensures systematic progress, thorough documentation, and simplified project management. Its linear nature guarantees each phase requirements, design, implementation, and testing is completed before moving forward, ensuring predictability and quality assurance.

- Requirement Analysis

  This stage involves gathering and documenting all the functional and non-functional requirements of the vacation home rental system, such as user roles, payment integrations, and admin approvals.

- Planning

  Here, the project timeline, resources, and development strategies are outlined, ensuring that each phase is properly scheduled and aligned with the system's objectives.

- System Design

  The architecture of the system is created, including database design, user interface layouts, and system workflows to ensure seamless interaction among components.

- Implementation

  The system is developed by writing code and integrating components like payment gateways, role-based access controls, and property management features.

- Testing

  This phase involves verifying and validating the system to ensure it meets the requirements, is free of defects, and performs efficiently under various scenarios.

- Maintenance

  After deployment, the system is monitored and updated to fix issues, accommodate new requirements, and ensure its smooth functioning over time.

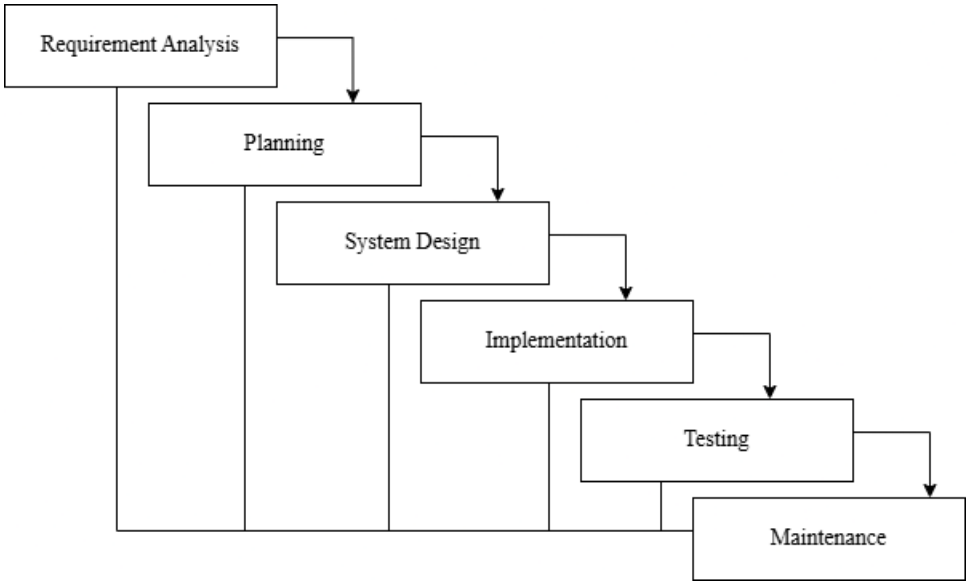


Figure 1.1: Waterfall Methodology

1.6 Report Organization

Chapter 1: Introduction

This chapter provides an overview of the project, including a general introduction, the problem statement, objectives, scope, limitations, and the methodology adopted during development.

Chapter 2: Background Study and Literature Review

This chapter highlights the background study related to the project and reviews existing systems in the vacation rental domain. It discusses their working concepts, features, advantages, and drawbacks to identify gaps addressed by this project.

Chapter 3: System Analysis

This chapter details the system analysis performed for the project. It includes functional and non-functional requirements, feasibility analysis, and other aspects critical for system development. Additionally, it covers design aspects such as the Data Flow Diagram, and Entity-Relationship Diagram, illustrating the system's structure and functionality.

Chapter 4: System Design

This chapter describes the design process for the system, including detailed architectural diagrams, database schema design, and the algorithms used to develop various modules of the Vacation Home Rental System.

Chapter 5: Implementation and Testing

This chapter explains the implementation process, including the tools, frameworks, programming languages, and database platforms utilized. It also details the testing phase, including test cases, results, and the analysis of system outputs to ensure reliability and functionality.

Chapter 6: Conclusion and Future Enhancements

This chapter summarizes the outcomes of the project, highlighting key achievements and lessons learned during the development process. It also discusses potential future enhancements to improve the system's functionality and scalability.

CHAPTER 2

BACKGROUND STUDY AND LITERATURE REVIEW

2.1 Background Study

Vacation rentals are private accommodations that are rented out to travelers on a short-term basis. Usually they are fully furnished apartments, condos, houses, cottages, or lodges, where guests can use the kitchen, amenities, and outdoor space (if any). Vacation rentals are seen as alternatives to traditional hotel stays, giving travelers more privacy, space, and often the feeling of a local lifestyle.

The most popular vacation rental websites are Airbnb, Vrbo and Booking.com.Vacation rental software helps manage vacation rental properties. Similar to hotel property management systems, it automates main operations such as reservation scheduling, booking processing, and communication, allows users to synchronize availability across distribution channels, supports revenue management, marketing, reporting, and so on. Some of the popular software providers are Guesty, Hostaway, and Lodgify [1].

Vacation rentals have become a popular alternative to traditional hotels. This form of travel promises a unique, personalized experience many travelers desire. As the demand for such travel experiences grows, so does the need for efficient booking, searching, and listing software. For entrepreneurs, the market holds great promise.

Advanced technology allows for the fast development of vacation rental software solutions, simplified property management, and marketing options, making vacation rentals a great startup idea. Property owners rely on vacation rental software for connectivity and easy management of routine operations.

These user-friendly platforms connect property managers with potential guests, ensuring a smooth and fast search and booking process. This software can streamline the rental processes, connect travelers with businesses, and unlock the full market potential, presenting opportunities for growth and success[2].

2.2 Literature Review

An occasional recreational vacation is a necessity for many people. It provides a perfect opportunity for the body and the mind to get much-needed rest after weeks, months, or years of daunting tasks and a break from routine. It also gives people morale as they perform their usual tasks afterward. Unfortunately, many people are unable to afford a vacation not only internationally but also locally due to the high costs involved. This makes many people prefer spending their holidays with extended families, by, for instance, traveling to their rural homes as opposed to taking a vacation. To boost the tourism sector in our country that is being promoted by initiatives such as 'Tembea Kenya', we should encourage domestic tourism [3].

Another challenge is the experience in hotels that some people do not like that would entirely cause them to opt to spend their holidays differently, for example, the lack of privacy in the shared accommodation facility, the limited space in hotels, the numerous restrictions, the level of cleanliness in the shared facility especially during the COVID-19 crisis, etcetera. The aim of this project was to solve the problem by coming up with a technological means of enabling people to make reservations for vacation homes with each other such that they mutually benefit from the program thus eliminating the fee for renting out the house [3].

Based upon a hedonic regression analysis of home sales in the City of South Lake Tahoe (CA), a vacation home rental (VHR) sells for more than a similar non-VHR. In addition, the presence of VHRs within a quarter-mile radius of a home sale also raises its value. However, the overall effect on the value of all home sale values from the presence of VHRs in South Lake Tahoe is negative. VHRs benefit owners and neighbors near them, but overall reduce the city's residential property values. These findings offer a logical basis for the heated discussions often observed over planning activities that restrict the presence of VHRs and/or attempt to mitigate their neighborhood effects. The paper concludes with implications for planning policy directed at VHRs [4].

2.3 Study of Existing Systems

1. Airbnb

Airbnb is the most popular vacation rental platform with over 6 million listings worldwide. It has more than 4 million hosts who have earned a collective $150 billion. There is no fee to list a property, though a service fee between 3-15% applies when a booking occurs. Airbnb Plus and the Superhost badge help hosts stand out in a crowded marketplace. The average U.S. host earns $13,800 annually [5].

2. Vrbo

Vrbo (Vacation Rental by Owner) focuses on family-friendly holiday homes. Hosts pay an annual fee of $499 or an 8% per-booking commission. Vrbo lists over 2 million properties and is popular for kid- and pet-friendly accommodations. Adding amenities and offering concierge services can help you succeed on the platform [5].

3. Booking.com

Founded in 1996, Booking.com is a major travel platform with over 28 million listings. While there's no cost to list a property, hosts are charged a 10-25% commission per booking, depending on location. It has 1.5 million nights booked daily[5].

4. Agoda

Agoda, owned by Booking.com, is Asia's largest online travel agency. Founded in 2005, Agoda offers listings in 38 languages across 200 countries. Although it's best known for hotel bookings, it's also a good option for vacation rentals[5].

5. TripAdvisor

TripAdvisor, founded in 2000, is a well-known platform for vacation rentals and travel reviews. It owns sites like FlipKey and HolidayLettings, automatically giving your listing wider exposure. The service fee is 3%, with guest fees between 8-16%[5].

6. FlipKey

FlipKey is linked to TripAdvisor, providing additional visibility across TripAdvisor's network. It charges a 3% service fee per booking, making it a cost-effective platform for vacation rentals[5].

CHAPTER 3

SYSTEM ANALYSIS

3.1 System Analysis

To develop a successful and user-friendly vacation rental platform, Vacation Home Rental System requires a comprehensive analysis of both functional and non-functional requirements. These requirements ensure that the system meets the needs of all users, including property owners, travelers, and administrators.

3.1.1 Requirement Analysis

3.1.1.1 Functional Requirements

A functional requirement describes what a system or application must do. It defines the specific behaviors, tasks, or functions the system should perform, such as user authentication, data processing, or generating reports. These requirements focus on how the system will meet the needs of its users and stakeholders.

1. User Registration and Authentication

   - Users must be able to create accounts, log in securely, and manage their profiles.
   - Users must be able to change password if they forget it through OTP email.

2. Property Listing Management

   - Hosts should be able to list properties with detailed descriptions, images, pricing, and availability.
   - Options for editing, updating, and removing listings.

3. Property Listing Management

- Hosts should be able to list properties with detailed descriptions, images, pricing, and availability.
- Options for editing, updating, and removing listings.

4. Search and Filtering

- Advanced search functionality, including filters for location, price range, property type, and guests allowed.
- Map-based search for enhanced user experience.

5. Booking and Payment Processing

- Secure and straightforward booking process with real-time availability checks.
- Integration with multiple payment gateways to support various payment methods.

6. Review and Rating System

- Users should be able to leave reviews and ratings for properties and hosts.

7. Communication System

- In-app messaging system for hosts and guests to communicate securely.
- Notifications and alerts for booking confirmations, changes, and reminders.

Use Case Diagram

A Use Case Diagram is a visual representation that depicts the interactions between users (actors) and a system. It outlines the functional requirements of a system by showing various use cases, which are the actions or services that the system provides to fulfill user needs.

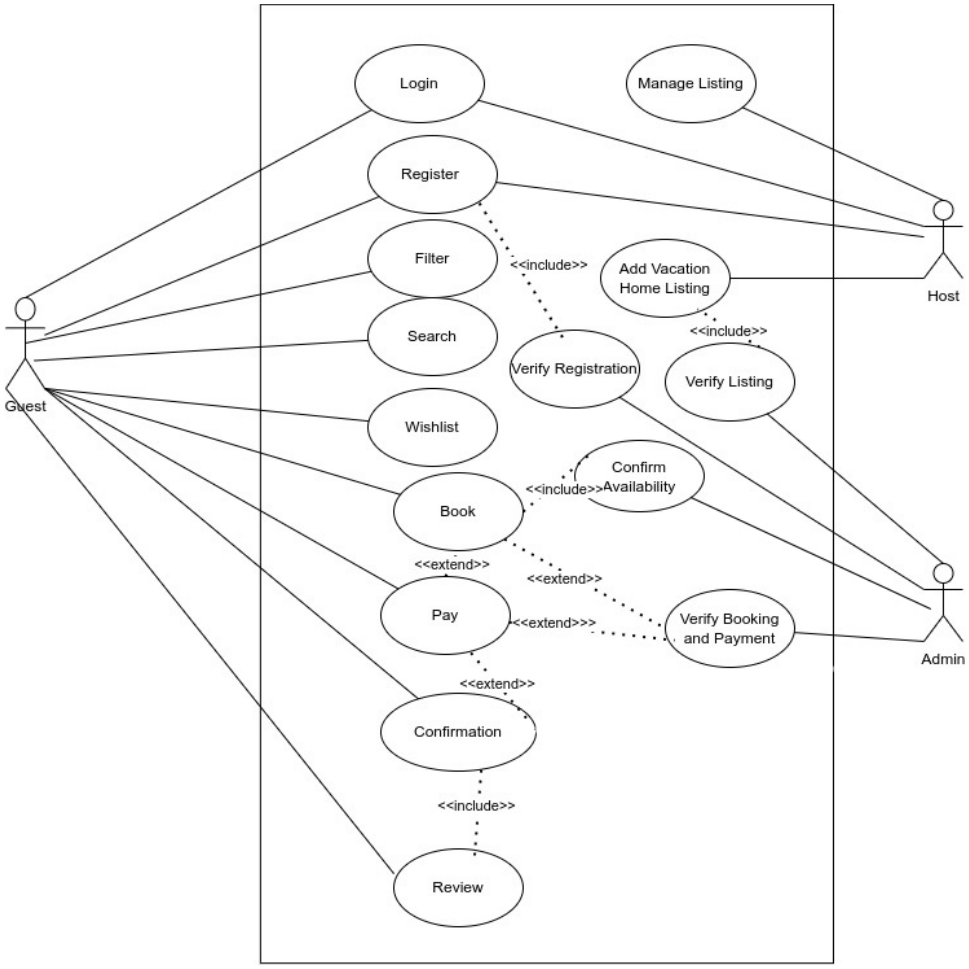


Figure 3.1: Usecase Diagram

3.1.1.2 Non Functional Requirements

A nonfunctional requirement refers to the attributes or qualities that a system must have, rather than the specific behaviors it must perform. These requirements focus on how well the system performs its tasks, such as performance, security, usability, reliability, and scalability. For example, a nonfunctional requirement might specify that a website should load in less than 2 seconds or that the system should handle up to 1,000 users simultaneously.

1. Scalability: The platform must support a growing number of users, listings, and transactions without performance degradation.

2. Security: Robust security measures to protect user data, including encryption, secure authentication, and regular security audits.

3. Performance: The system should load quickly and handle high traffic during peak times, ensuring a smooth user experience.

4. Usability: Intuitive and user-friendly interface designed for all user types, including those with limited technical skills.

5. Reliability: The platform should ensure high availability with minimal downtime and provide backups for data protection.

6. Hardware Requirements:

- Laptop, desktop, or smartphone with 4 GB RAM.
- Screen resolution of 1366x768 or higher.
- Intel Core i3 processor or better.
- Internet connection with 1Mbps speed.

7. Software Requirements:

- Web browser: Chrome, Firefox, Safari, or Edge.
- Mobile OS: Android 8.0 or higher, or iOS 12+.
- PayPal account for payments.
- Browser with JavaScript and cookies enabled.

3.1.2 Feasibility Study

A feasibility study is an assessment that evaluates whether a project or idea is practical and achievable. It looks at different factors like cost, resources, technology, and time to determine if the project can be successfully completed. It helps to identify potential problems and risks before starting the project, ensuring that it's worth pursuing.

3.1.2.1 Technical

Vacation Home Rental System will be built using modern and widely adopted web technologies, like React.js for the front-end, Express.js for the back-end, and a robust

database like PostgreSQL. These technologies are well-supported, scalable, and capable of handling the platform's requirements, such as secure transactions, real-time booking, and user-friendly interfaces. The development team's familiarity with these tools ensures that the technical challenges can be effectively managed, making the system technically feasible.

3.1.2.2 Operational

Vacation Home Rental System is designed to meet the needs of property owners and travelers by providing an intuitive platform for listing, searching, and booking vacation rentals. The user interface will be easy to navigate, and the features like secure payment processing and in-app communication align with user expectations. Given the clear demand for such services in the vacation rental market and the alignment with current industry trends, the platform is operationally feasible. It will be readily accepted by users who are familiar with similar services but are seeking a more diverse and reliable option.

3.1.2.3 Schedule

The development timeline for Vacation Home Rental System is realistic, with key milestones such as design, development, testing, and deployment planned over a structured period. Given the project scope and the development team's expertise, the estimated time frame allows for the completion of essential features and thorough testing before launch. With adequate resources and a clear project plan, Vacation Home Rental System's schedule is feasible, ensuring the platform can be delivered on time.

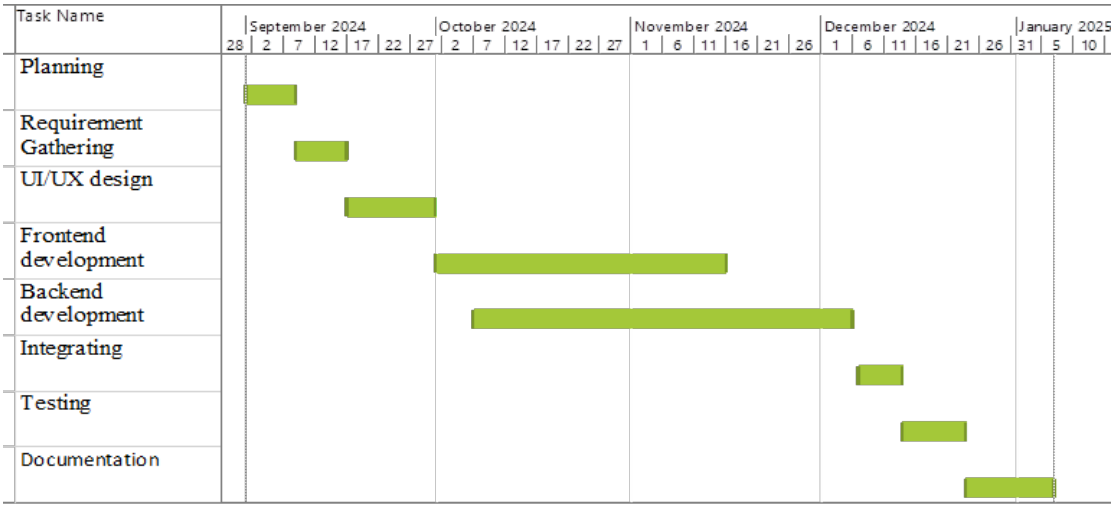


Figure 3.2: Gantt Chart

3.1.3 Analysis

The feasibility study for the Vacation Home Rental System assesses the practicality of the project in three key areas: technical, operational, and schedule. Technically, the system will be built using popular and scalable technologies like React.js, Express.js, and PostgreSQL, ensuring that the platform can handle secure transactions and real-time bookings. Operationally, the system will meet the needs of property owners and travelers with a user-friendly interface, secure payment processing, and communication features, making it a viable option in the vacation rental market. The development timeline is realistic, with key milestones and thorough testing planned, ensuring the system will be delivered on time. Overall, the project is technically, operationally, and schedule-wise feasible.

3.1.3.1 Data modeling using ER Diagrams

An Entity-Relationship (ER) diagram is a visual representation used to model the structure of a database. It illustrates the relationships between entities (e.g., people, objects, or concepts) and their attributes. ER diagrams use symbols like rectangles to represent entities, diamonds for relationships, and ovals for attributes, helping to design and communicate the data structure clearly before actual implementation.

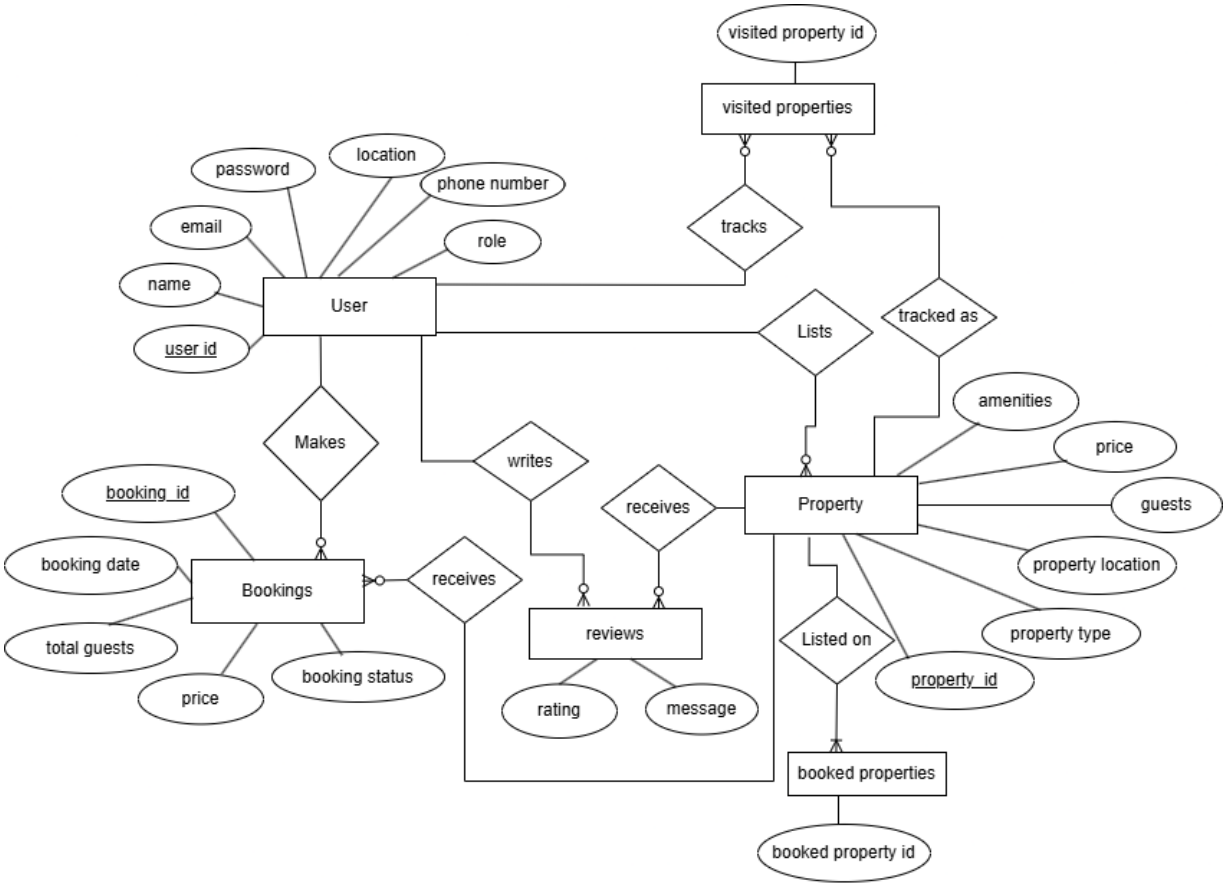


Figure 3.3: ER-Diagram

3.1.3.2 Process modeling using DFD

The process modeling of the vacation home rental system is done using Data Flow Diagrams (DFD). DFDs show how data moves through the system and the interactions between users, hosts, admins, and different system components. It illustrates processes like registration, property listing, booking, payment, and reviews, along with the data inputs, outputs, and storage. DFDs help in understanding the system's functions and identifying areas for improvement.

1. Level 0 DFD

Level 0 DFD (Context Diagram) for Vacation Rental System represents the highest-level view showing the entire system as a single process. It illustrates how the system interacts with external entities (Host, User/Guest, and Admin) without showing any internal processes or data stores. It's essentially a bird's eye view showing data flowing in and out of the system boundary.

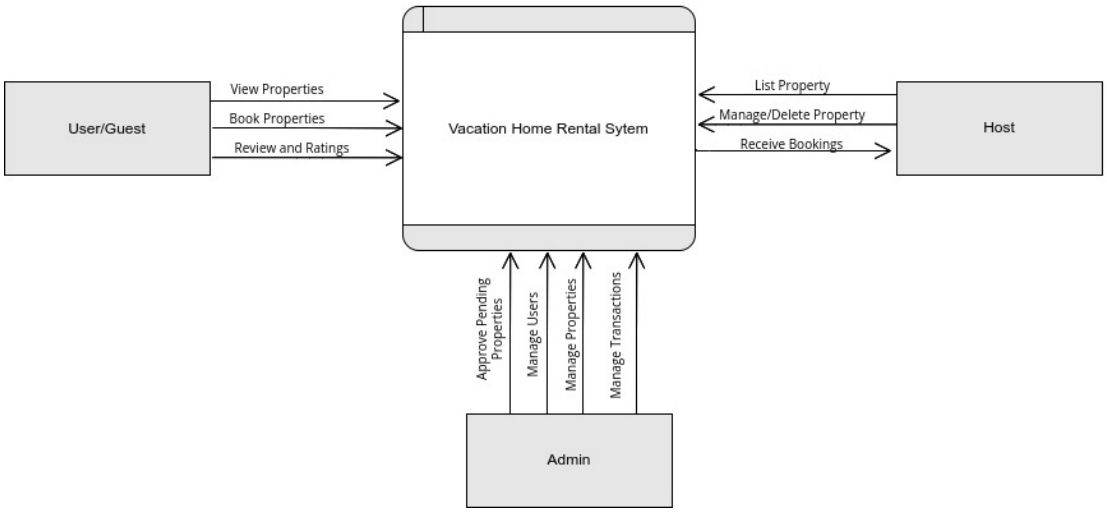


Figure 3.4: Level 0 DFD

2. Level 1 DFD of User

Level 1 DFD for User/Guest breaks down their interactions into detailed processes including property search, booking management, and review systems. Users can register/login, search for properties based on preferences, make bookings, manage their reservations, write reviews, and communicate with hosts. They interact with multiple data stores including user details, property listings, bookings, reviews, messages, wishlists, visited properties, and preferences, making this the most data-store-intensive user type in the system.

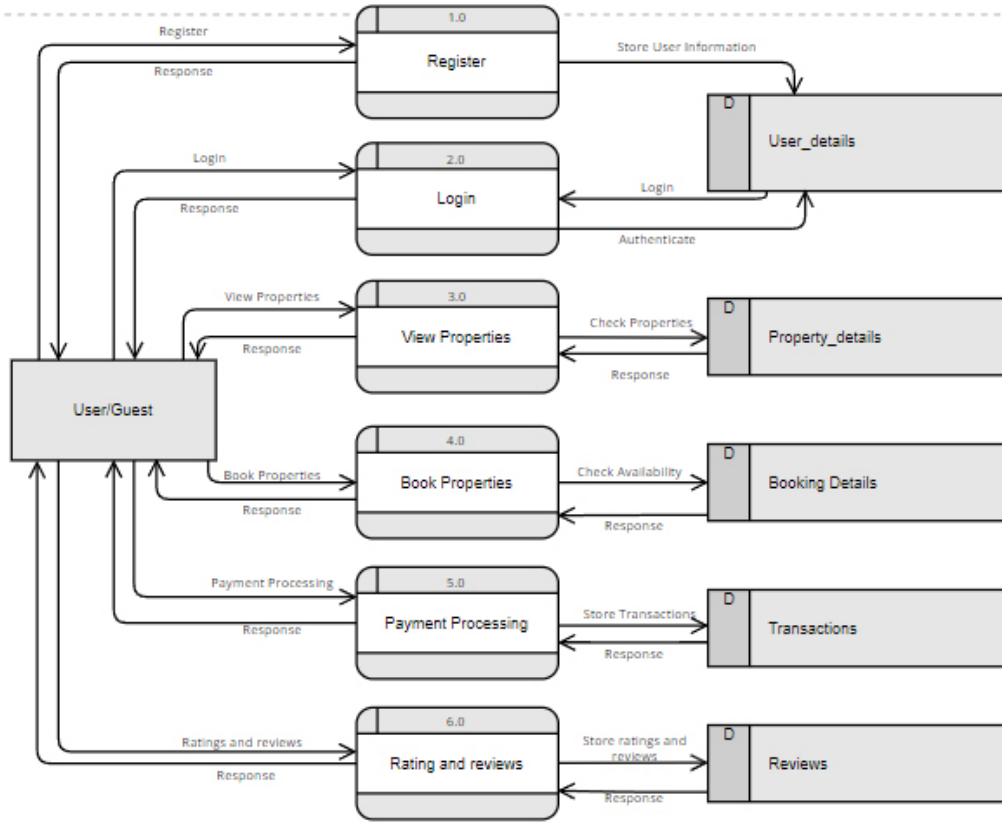


Figure 3.5: Level 1 DFD of User

2. Level 1 DFD of Host

Level 1 DFD for Host focuses on property management and guest interaction processes. Hosts can list new properties (which go through an approval process), manage existing listings, handle booking requests, communicate with guests through messages, and monitor property reviews. They primarily interact with data stores related to property listings (both pending and approved), bookings, messages, and reviews. The host's processes are centered around property management and guest service rather than the discovery and booking processes that characterize the user/guest DFD.

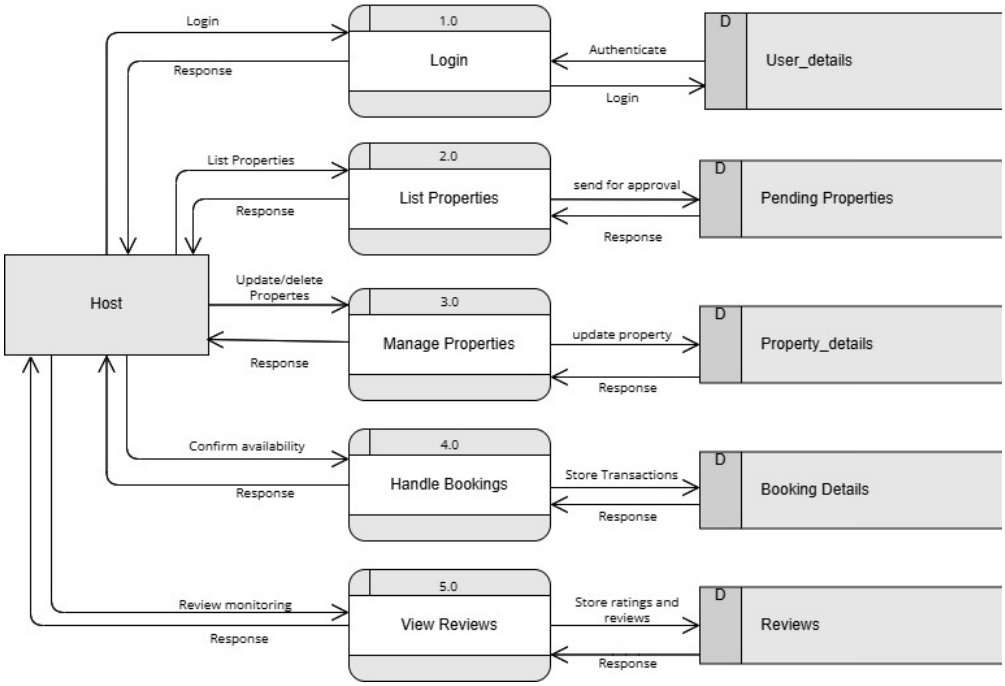


Figure 3.6: Level 1 DFD of Host

CHAPTER 4

SYSTEM DESIGN

4.1 Design

System design defines the architecture, components, modules, interfaces, and data for the Vacation Home Rental System to meet the specified requirements. It involves translating user needs into a detailed blueprint that guides the development and implementation of the system. The system design covers both the high-level structure of the system, such as its overall flow and user interactions, and the low-level details, including the implementation of specific modules like property listing, booking, payment processing, and user management. This ensures a well-organized, efficient, and scalable solution for the vacation home rental platform.

4.1.1 Database Design

In the process of developing the Vacation Home Rental System, designing a comprehensive database schema is of paramount importance. This process involves accurately structuring the database, including tables, columns, relationships, constraints, and other essential elements. A well-structured schema ensures data integrity, performance, and scalability, serving as the architectural backbone of the system.

The database integrated into our system is PostgreSQL, a powerful and open-source object-relational database. PostgreSQL is widely recognized for its advanced indexing, complex query handling, and extensibility. It supports various features such as JSON, full-text search, and foreign data wrappers, making it an ideal choice for a scalable and dynamic system like the Vacation Home Rental System.

The database schema has been designed to include key entities such as users, properties, reservations, payments, and reviews. Each entity is represented as a table with attributes and relationships, utilizing primary and foreign keys to ensure data consistency and

referential integrity. Constraints such as unique, not-null, and check constraints are applied to enforce rules and maintain data quality.

Upon transitioning from the conceptual Entity-Relationship (ER) diagram to the physical database schema, additional tables have been introduced to enhance functionality and optimize data management. These additions include:

- **pending_property_listing_detail**: This table was added to manage properties that are awaiting approval before being listed in the system. It ensures that only verified properties are made available to users.
- **admin_host_messages**: This table facilitates communication between administrators and property hosts, allowing them to exchange messages regarding property listings, verification status, and other concerns.
- **wishlist**: Not present in the ER diagram, this table enables users to save and revisit properties they are interested in, enhancing the user experience.
- **preferences**: This table stores user preferences, such as preferred property types, property region and price ranges, allowing for personalized property recommendations.

These additions ensure that the database schema is not only aligned with the conceptual ER model but also refined to enhance system functionality, data retrieval efficiency, and user experience. The resulting schema effectively supports the operations of the Vacation Home Rental System, maintaining data integrity and performance.

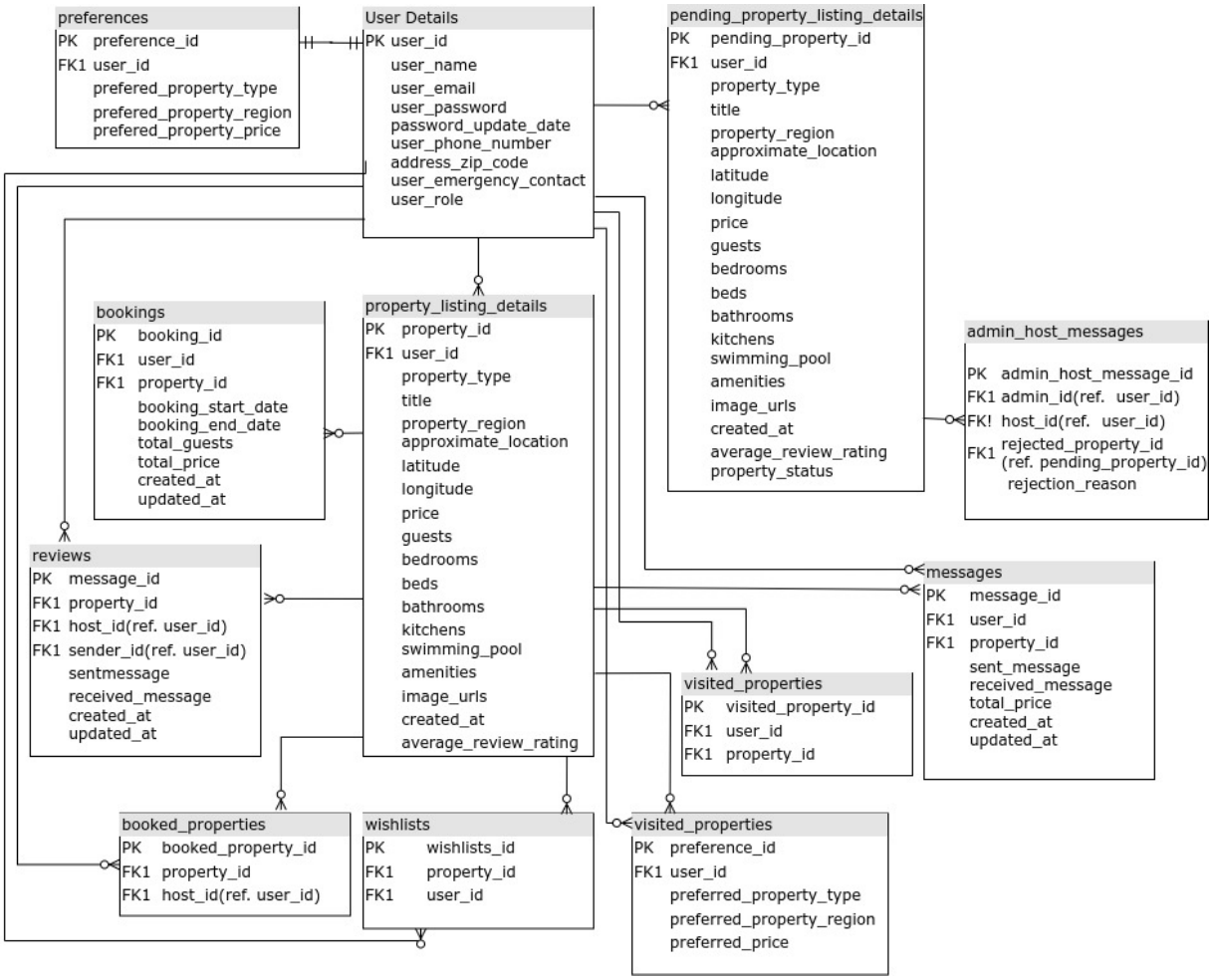


Figure 4.1: Database Schema Design

4.1.2 System Design

The vacation home rental system is designed to provide a seamless and user-friendly experience. The user interface is built using React JS, Material UI, and CSS, ensuring a visually appealing design and intuitive navigation. The authentication and authorization module handles tasks like user registration, login, and password management, using JWT for token-based authentication and Bcrypt for secure password hashing. The server-side application is powered by Node.js and Express.js, efficiently managing the backend processes and business logic. PostgreSQL serves as the primary database for structured data storage, while AWS S3 Bucket is utilized for storing property images and other files, ensuring scalability and reliability.

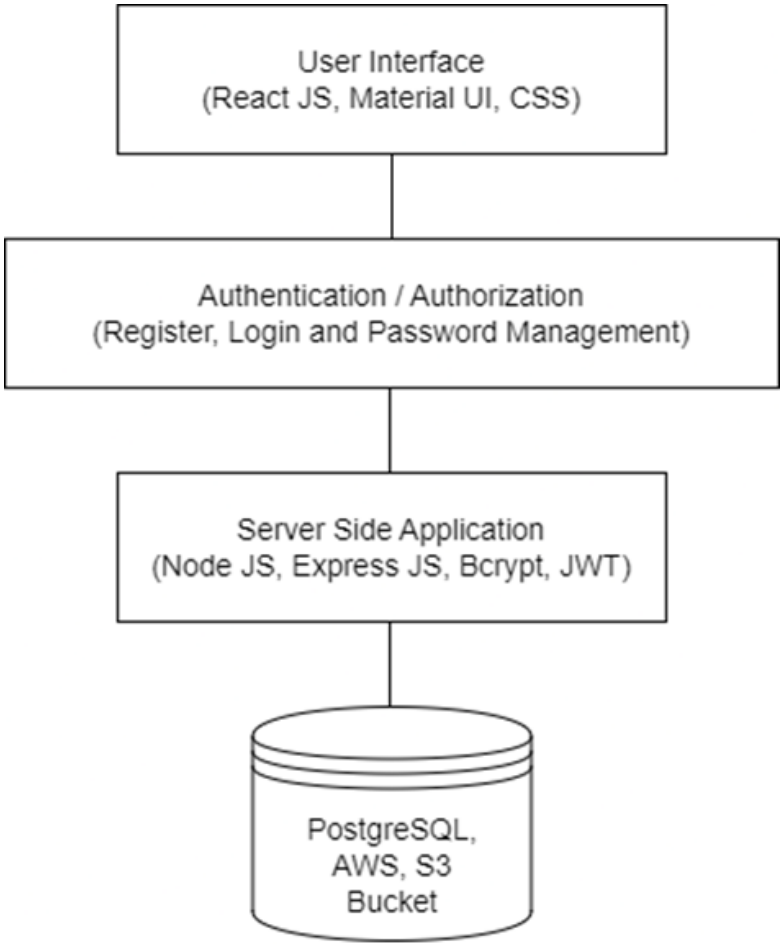


Figure 4.2: System Design

4.1.3 Flowchart

The flowchart of vacation home rental system illustrates the process flow among its primary users: Guests, Hosts, and Admins. Guests can search for properties, view details, make bookings, and complete payments. Hosts can register, list properties, and submit them for admin approval. Admins review property listings, approve or decline them, and oversee the platform's operations. The flowchart visually connects these actions, showing how data moves between users and the system to ensure seamless functionality.

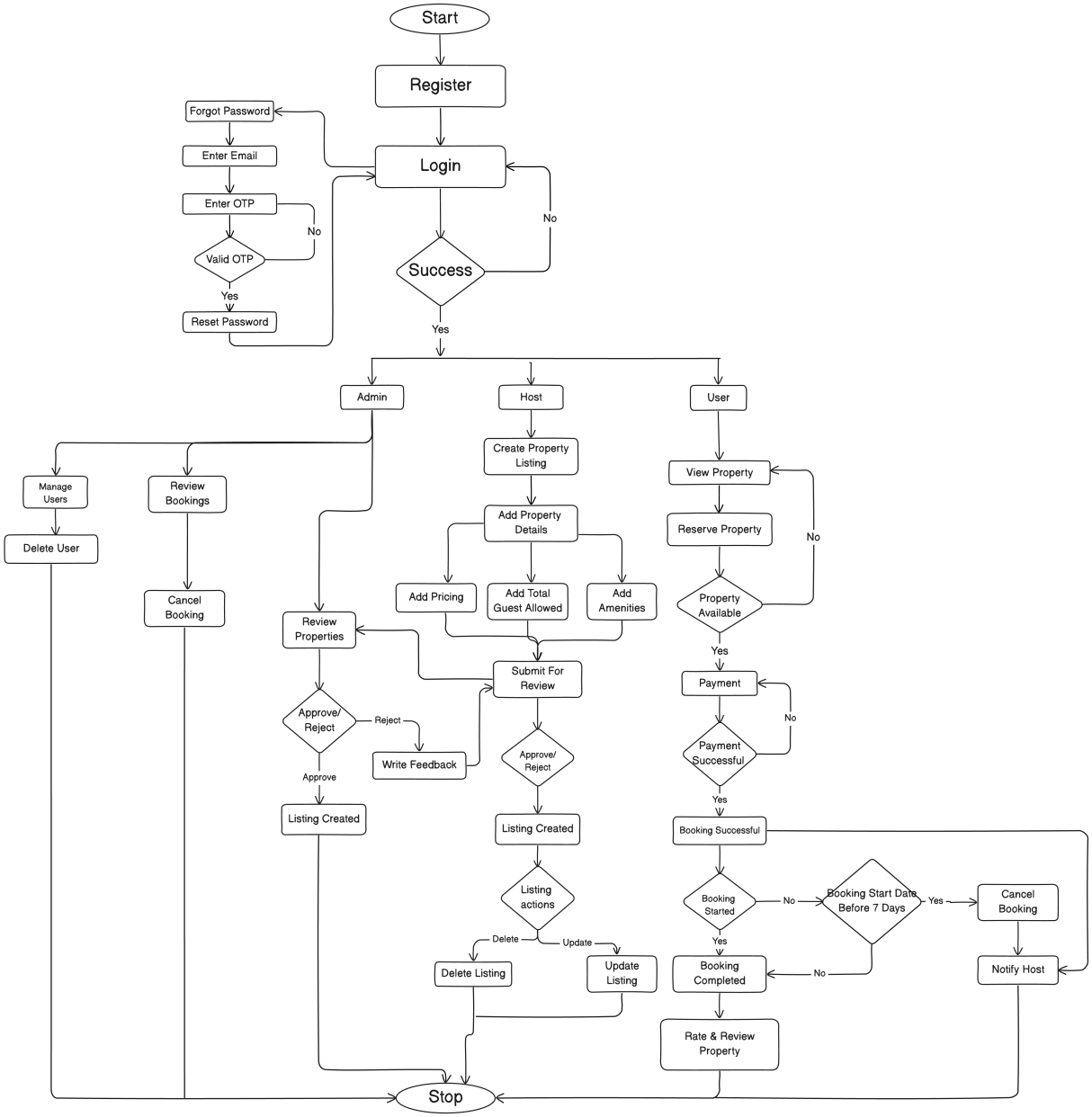


Figure 4.3: System Flowchart

4.1.4 Form Design

A form is an important part of a system as it is used to gather data from users in an organized manner. It serves as a bridge between users and the system, allowing them to input necessary information, such as credentials, personal details, or preferences.

- In the Guest Side

    1. Form from Login and Signup

2. Form for Updating profile/ password information

- In the Admin Side

1. Form for Login
2. Form for updating password

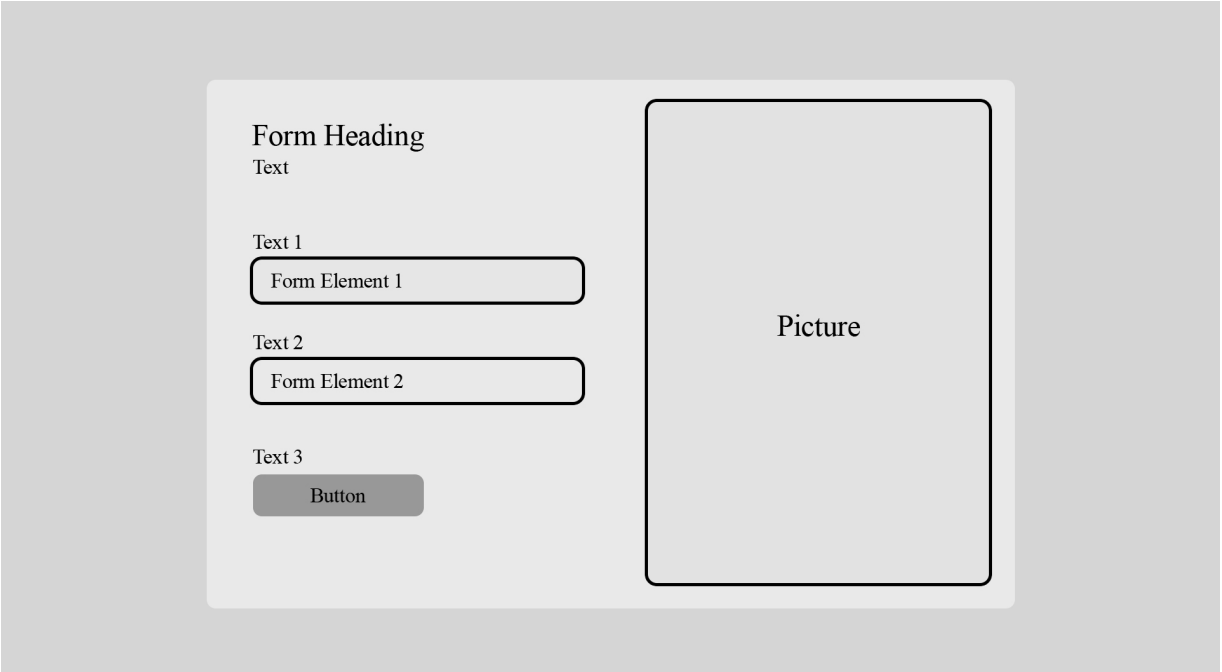


Figure 4.4: Form Design

4.1.5 Interface Design

The UI of Vacation Home Rental System is developed using React library,Vanilla CSS, Material UI and Tailwind CSS. The overall interface of the application is meant to be as satisfactory,interactive and user-friendly as possible. The system encompasses essential pages such as Signup, Login, Home, Account Setting, Dashboard, Property Details, Create Listing and Notifications.

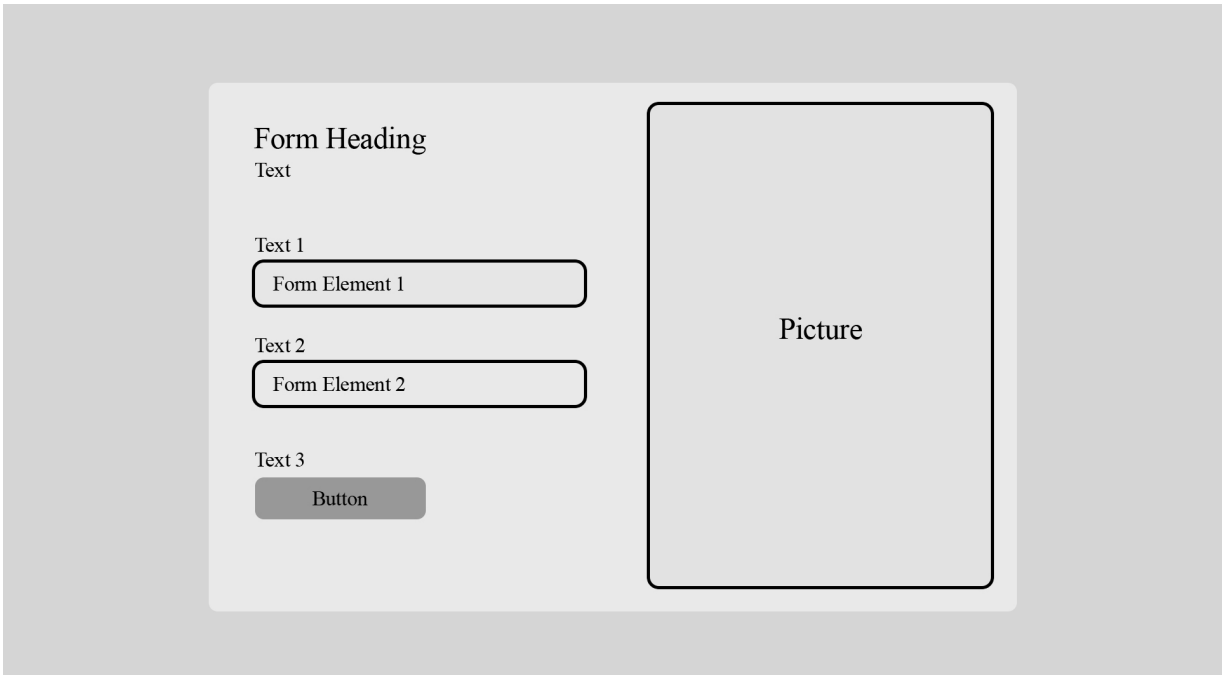


Figure 4.5: Login and Signup Page

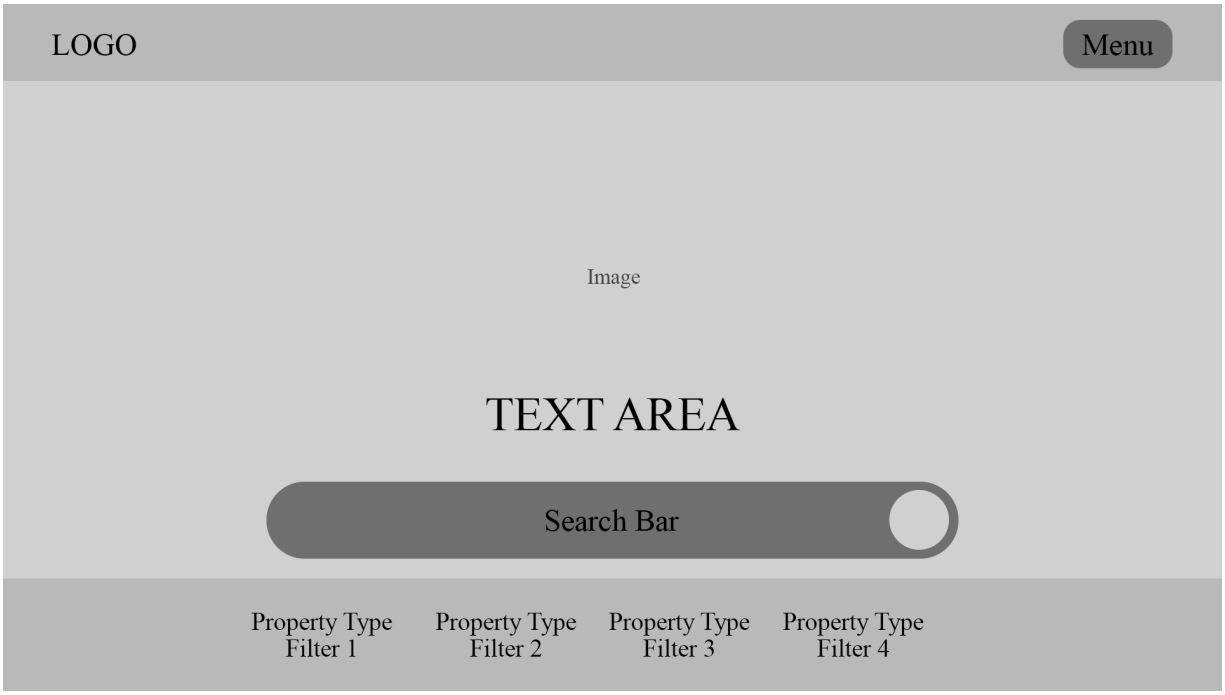


Figure 4.6: Interface for Landing Page

Figure 4.7: Interface for Home Page

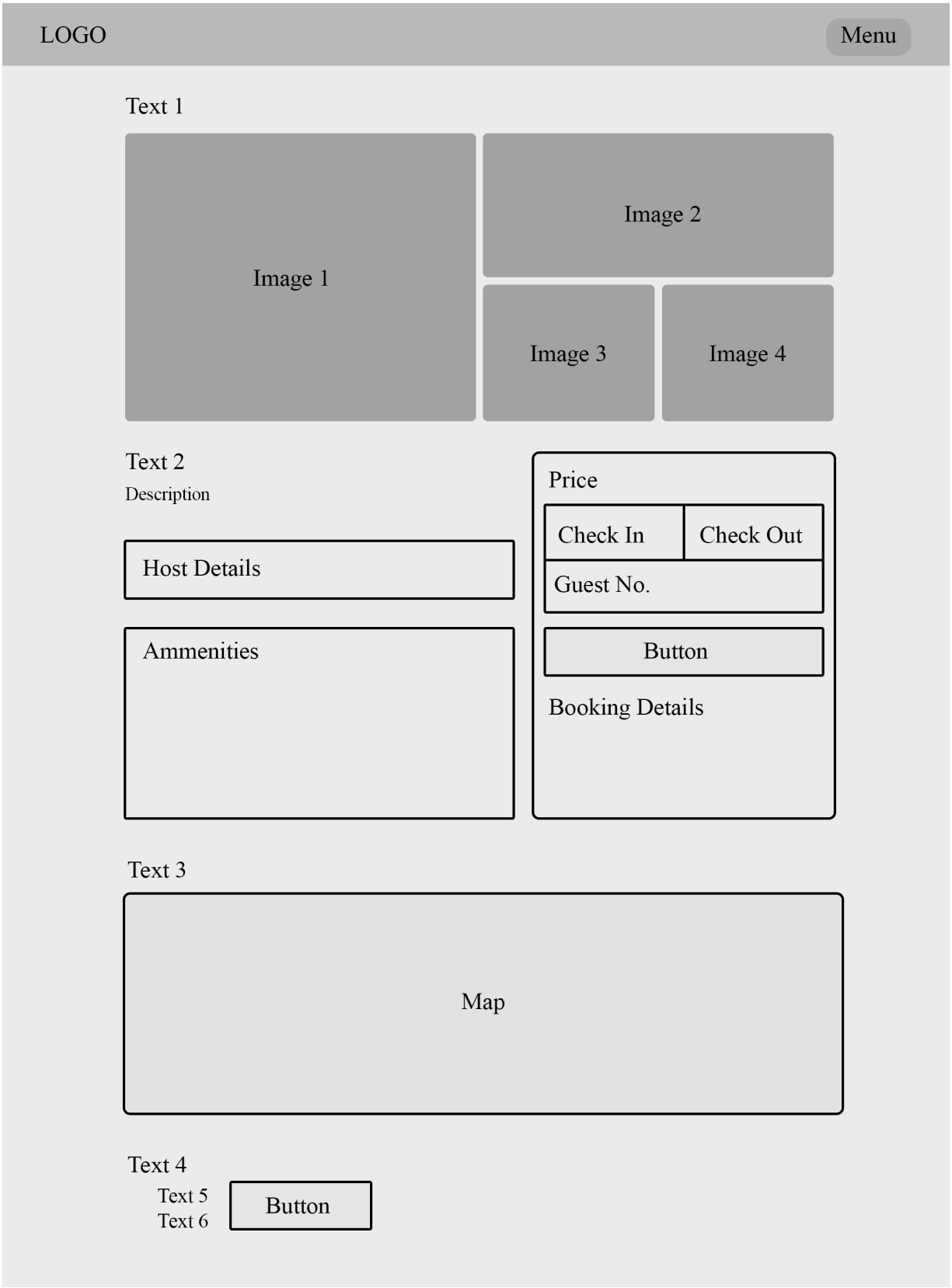


Figure 4.8: Interface for Property Details

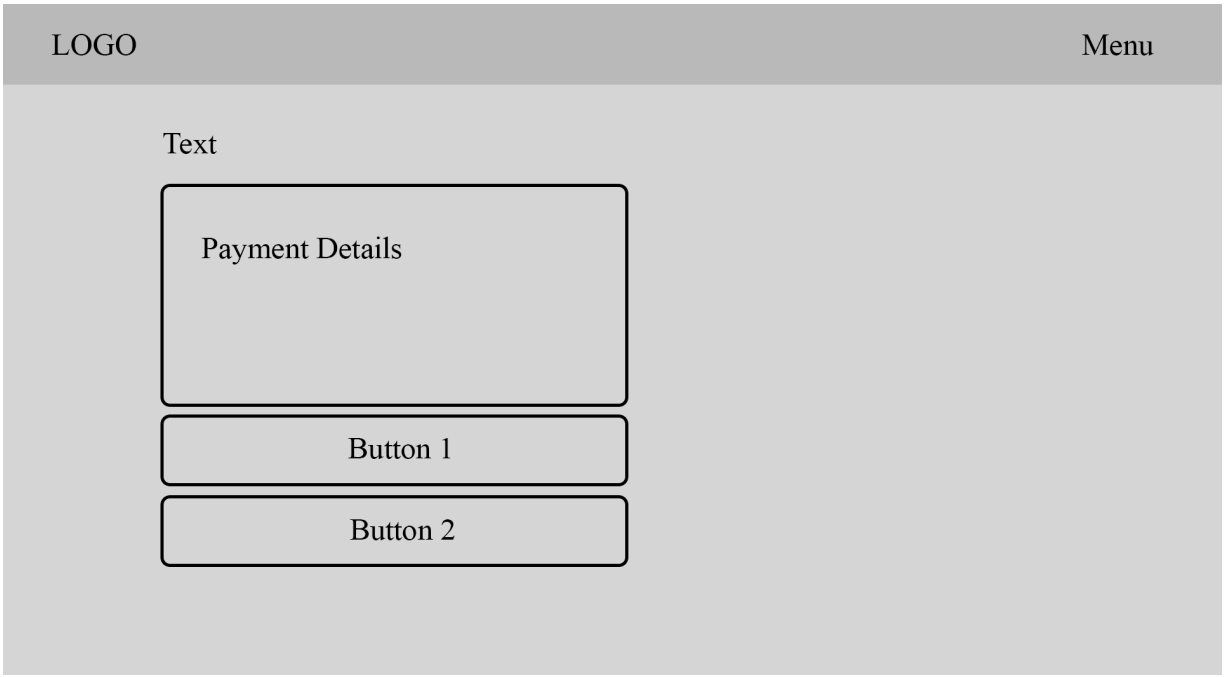


Figure 4.9: Interface for Payment Gateway

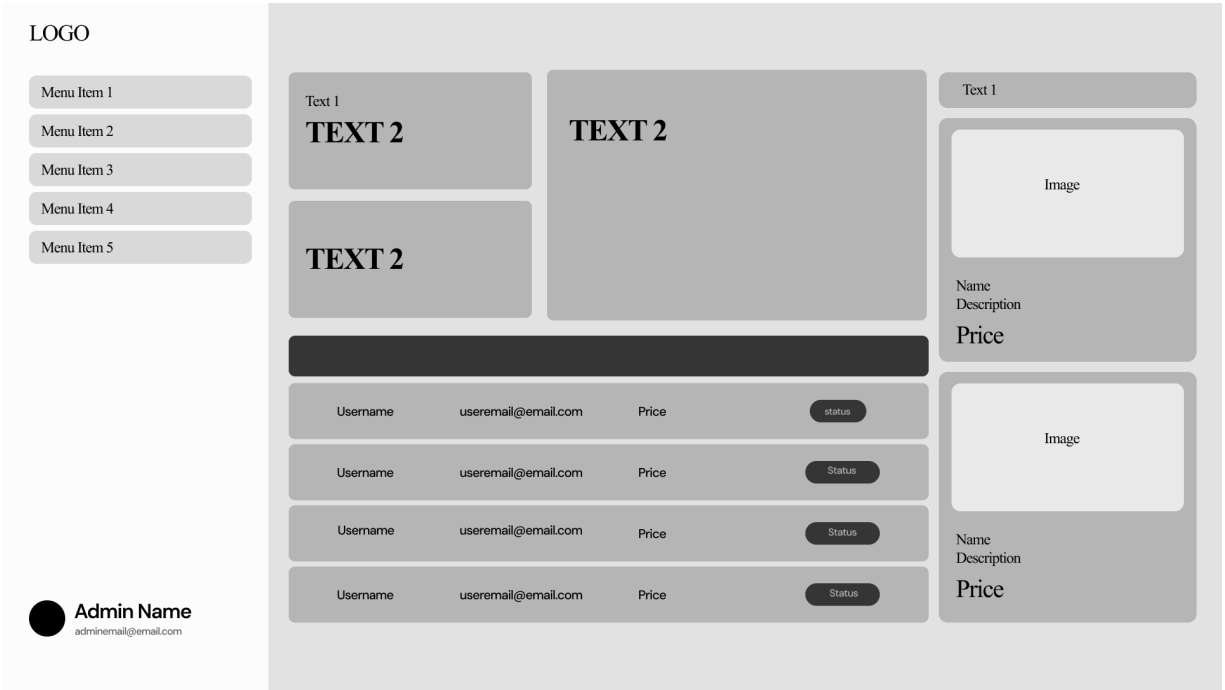


Figure 4.10: Interface for Admin Dashboard

4.2 Algorithm Details

The system implements algorithms such as recommendation algorithm, filtering algorithm. This is for users to attain a catered recommendation and for users to search as per their needs.

4.2.1 Recommendation System Components

The property recommendation system combines two main types of recommendation algorithms: Content-Based Filtering and Collaborative Filtering. It computes a hybrid score for each property, incorporating both the user's explicit preferences and the behaviors of similar users.

4.2.1.1 Implementation Rationale

Hybrid Recommendation Approach

- Content-Based Filtering is used to match the properties with the user's explicit preferences (price, region, property type). This allows for personalization based on the individual's preferences.
- Collaborative Filtering leverages the ratings of similar users to identify hidden preferences the system might not explicitly know, improving recommendations through collective user behavior.

AWS Integration for Image URLs:

The system generates signed URLs for property images using AWS S3, ensuring secure and time-limited access to the images. This functionality supports up to 5 images per property, with a default expiry of 3600 seconds.

Dynamic User History Weighting:

The system dynamically adjusts the importance of content and collaborative scores based on the user's review history. Logarithmic scaling ensures fair weighting, with a maximum limit of 1 for history influence. For example:

- User with 1 review might have a history weight of 0.15.
- User with 10 reviews might have a history weight of 0.52.
- User with 100 reviews might have a history weight of 1.

Scalability and Flexibility:

The modular nature of the algorithm allows for easy adjustments to scoring weights, thresholds, and configurations, making it adaptable for future growth and improvements.

4.2.1.2 Algorithm Steps

Step 1: User Preference Collection

Objective: Fetch the user's stored preferences to personalize property recommendations.

```
SELECT
prefered_property_type,
prefered_property_region,
prefered_price
FROM preferences
WHERE user_id = $1
```

- Retrieve preferred property type, region, and price from the database.
- If preferences are unavailable, return an error or prompt the user to set preferences.

Step 2: Property Filtering

Objective: Identify and retrieve properties not yet visited by the user and also excluding those with active bookings.

```
SELECT p.*,
COALESCE((SELECT AVG(rating) FROM reviews
WHERE property_id = p.property_id), 0)
AS average_rating
FROM property_listing_details p
```

```
WHERE p.property_id NOT IN (
SELECT DISTINCT property_id
FROM reviews
WHERE user_id = $1
)
AND p.property_id NOT IN (
SELECT DISTINCT property_id
FROM bookings
WHERE booking_status = 'Booked'
AND booking_end_date >= CURRENT_DATE
)
LIMIT $2
```

- Exclude properties reviewed or currently booked by the user.
- Limit the results to the maximum recommendation count.

Step 3: Score Calculation

A. Content-Based Score Calculation

1. Price Similarity Score:

$$\text{priceScore} = \max(0, 1 - \frac{|\text{price} - \text{prefPrice}|}{\text{PRICE\_THRESHOLD}}) \tag{4.1}$$

   where PRICE_THRESHOLD = Rs.10,000

2. Region Matching Score:

$$\text{regionScore} = \begin{cases} 1 & \text{if region matches preference} \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

3. Property Type Matching Score:

$$\text{typeScore} = \begin{cases} 1 & \text{if type matches preference} \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

4. Total Content-Based Score:

$$\text{contentScore} = \text{priceScore} \cdot w_{\text{price}} + \text{regionScore} \cdot w_{\text{region}} + \text{typeScore} \cdot w_{\text{type}} \quad (4.4)$$

where $w_{\text{price}} = 0.4$, $w_{\text{region}} = 0.3$, and $w_{\text{type}} = 0.3$.

B. Collaborative Score Calculation

1. Similar Users Identification:

```
SELECT r2.user_id, CORR(r1.rating, r2.rating) as similarity
FROM reviews r1
JOIN reviews r2 ON r1.property_id = r2.property_id
WHERE r1.user_id = $1 AND r2.user_id != $1
GROUP BY r2.user_id
HAVING COUNT(*) >= 3 AND CORR(r1.rating, r2.rating) >= $2
```

Users with Pearson correlation above the threshold
(SIMILAR_USERS_THRESHOLD = 0.3) are considered similar.

2. Weighted Rating Calculation:

$$\text{weightedRating} = \frac{\sum(\text{rating} \cdot \text{similarity})}{\sum(\text{similarity})} \quad (4.5)$$

Step 4: Hybrid Score Calculation

Combine the content-based and collaborative scores to create a final score.

$$\text{finalScore} = \text{contentScore} \cdot w_{\text{content}} + \text{collabScore} \cdot w_{\text{history}} \quad (4.6)$$

where w_{content} and w_{history} are dynamically adjusted based on user history.

In Summary The dynamic weighting system operates as follows:

1. User History Weight Calculation

The system uses logarithmic scaling to calculate the base user history weight. This approach ensures the weight stays bounded within reasonable limits and dampens the impact of large numbers of reviews.

The formula for calculating the user history weight is:

$$\text{userHistoryWeight} = \min\left(\frac{\log_{10}(\text{reviewCount} + 1)}{2}, 1\right) \tag{4.7}$$

For example:

- 1 review: $\frac{\log_{10}(2)}{2} \approx 0.15$
- 10 reviews: $\frac{\log_{10}(11)}{2} \approx 0.52$
- 100 reviews: $\frac{\log_{10}(101)}{2} \approx 1.0$ (capped at 1)

The $\log_{10}$ function provides a natural dampening effect, while dividing by 2 scales the result to a reasonable range. Adding 1 to reviewCount ensures the function is defined even for users with zero reviews.

2. Weight Application

This base weight is then used to calculate the final dynamic weights:

1. Adjusted History Weight: A minimum threshold ensures user history is always considered:

$$w_{\text{history}} = \max(\text{userHistoryWeight}, 0.2) \tag{4.8}$$

2. Content Weight: Complementary weight to adjusted history weight:

$$w_{\text{content}} = \max(0, \min(1, 1 - w_{\text{history}})) \tag{4.9}$$

3. Final Score: Combines content and collaborative scores using dynamic weights:

$$\text{finalScore} = \text{contentScore} \cdot w_{\text{content}} + \text{collabScore} \cdot w_{\text{history}} \tag{4.10}$$

Step 5: Result Processing

1. Image URL Generation:
    - Generate signed URLs for up to 5 images per property using AWS S3.
2. Sorting and Limiting:
    - Sort properties by hybrid score and limit to the top recommendations.

4.2.2 Property Filtering System

The property filtering system implements a multi-stage filtering approach to refine property listings based on user-specified criteria and real-time availability. This system ensures that users receive relevant results that match their requirements while maintaining performance at scale.

4.2.2.1 Implementation Rationale

Multi-Stage Filtering Approach:

The filtering process is divided into distinct stages: - Primary filtering based on essential criteria (region, guest capacity) - Availability filtering based on booking dates - Secondary filtering based on user preferences (price range, property type) - Distance-based filtering when location data is available

Real-time Availability Management:

The system implements real-time availability checks to prevent booking conflicts and ensure that displayed properties are actually bookable for the requested dates.

Location-Aware Filtering:

When geographic coordinates are provided, the system incorporates distance calculations using the Haversine formula to enable location-based sorting and filtering.

Scalability Considerations:

The algorithm is designed with database optimization in mind, using indexed fields and efficient query patterns to maintain performance with large datasets.

4.2.2.2 Algorithm Steps

Step 1: Primary Property Filtering

Objective: Filter properties based on fundamental criteria.

```
SELECT *
FROM property_listing_details
WHERE guests >= $1
[AND property_region = $2]
```

Key components:

- Guest capacity validation
- Optional region filtering
- Uses indexed columns for efficient querying

Step 2: Availability Verification

Objective: Check property availability for requested dates.

```
SELECT booking_start_date, booking_end_date
FROM bookings
WHERE property_id = $1
```

Availability logic:

$$isAvailable = \forall booking \in bookings : (endDate \leq booking.start \vee startDate \geq booking.end) \tag{4.11}$$

Step 3: Price Range Filtering

Objective: Filter properties within specified price boundaries.

Key components:

- Minimum price threshold: $minPrice$

- Maximum price threshold: $maxPrice$
- Price range validation:

$$minPrice \leq property.price \leq maxPrice \tag{4.12}$$

Step 4: Property Type Filtering

Objective: Filter properties based on type classification.

```
AND property_type = $propertyType
WHERE propertyType != 'Any'
```

Step 5: Distance Calculation

Objective: Calculate and sort properties based on distance when location is provided.

Distance calculation using Haversine formula:

$$d = 2R \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \tag{4.13}$$

where:

- R is Earth's radius,
- ϕ_1, ϕ_2 are latitudes (in radians),
- λ_1, λ_2 are longitudes (in radians).

Step 6: Result Processing

1. Image Processing:
    - Generate signed URLs for property images
    - Validate image URL existence
2. Result Sorting:
    - Primary sort by distance (if location provided)

- Secondary sort by rating
- Handle null distance values

3. Response Formatting:
    - Include property details
    - Add calculated distances
    - Include sorting criteria in response

Error Handling:

The system implements comprehensive error handling:

- Database query failures
- Invalid coordinate values
- Missing required parameters
- Image URL generation failures

Performance Optimization:

The Key optimization strategies are:

- Indexed database fields for frequent query parameters
- Efficient query construction using parameterized queries
- Parallel processing for image URL generation
- Early filtering to reduce dataset size before expensive operations

CHAPTER 5

IMPLEMENTATION AND TESTING

5.1 Implementation

System implementation is a key phase in the development of our project, where the system design is transformed into a working and operational solution. This phase ensures that the system meets the required quality standards and functions as intended. Throughout the project, we followed the iterative development methodology, which involves repeated cycles of planning, requirement analysis, system design, development, and testing. By using this approach, we were able to refine and improve the system incrementally, ensuring its reliability and effectiveness in meeting the project's goals.

5.1.1 Tools Used

Frontend Technologies

- HTML: HTML (HyperText Markup Language) is the standard language used to structure the content of web pages. It defines elements like headings, paragraphs, links, images, and forms, allowing browsers to display web content correctly. HTML serves as the backbone of any website's structure.
- CSS: CSS (Cascading Style Sheets) is used to style and layout web pages. It controls the appearance of HTML elements, such as fonts, colors, spacing, and positioning. CSS ensures that web pages are visually appealing and responsive across various devices.
- JavaScript: JavaScript is a programming language that adds interactivity and dynamic behavior to web pages. It enables features like form validation, animations, and data fetching, making websites more interactive and responsive to user actions.

JavaScript Frameworks and Libraries

- React: React is a JavaScript library used to build user interfaces, especially single-page applications. It allows developers to create reusable UI components that update efficiently, making applications faster and easier to maintain.
- React Router DOM: React Router DOM is a library used in React applications to handle routing. It enables navigation between different components or pages within a single-page application (SPA), enhancing user experience by loading content dynamically without reloading the page.

UI Frameworks and Libraries

- Tailwind CSS: Tailwind CSS is a utility-first CSS framework that provides pre-defined classes to style elements quickly. It allows developers to create responsive and customizable designs without writing custom CSS, streamlining the design process.
- Material UI: Material UI is a popular React UI framework that implements Google's Material Design principles. It provides pre-built components like buttons, forms, and navigation bars, allowing developers to create consistent, modern, and accessible user interfaces.

Backend

- Node.js: Node.js is a JavaScript runtime environment that allows developers to run JavaScript on the server side. It is known for its speed and scalability, making it suitable for building real-time applications and APIs.
- Express.js: Express.js is a minimal and flexible Node.js web application framework. It simplifies the process of handling HTTP requests, routing, and middleware, making backend development faster and more efficient.

Database

- PostgreSQL: PostgreSQL is an open-source, relational database management system (RDBMS). It supports complex queries, transactions, and provides strong data integrity, making it ideal for storing and managing structured data in web applications.

- S3 Bucket: Amazon S3 (Simple Storage Service) is a cloud storage service provided by AWS. It allows the storage of large amounts of data, such as images and videos, with high scalability, security, and availability.

Version Control

- Git: Git is a distributed version control system that helps developers track changes in code and collaborate with others. It allows developers to manage and synchronize code versions, facilitating team collaboration, rollback of changes, and branch management during the development process. Git is essential for maintaining code quality and managing large codebases efficiently.

5.2 Implementation Details of Modules

5.2.1 User Registration and Authentication Module

```
router.post("/", async (req, res) => {
  const { username, email, password } = req.body;
  if (!username || !email || !password) {
    return res.status(400).json("Please fill all fields ");
  }
  try {
    const checkIfUserExists = await pool.query(
      "SELECT * from user_details WHERE user_name=$1 OR user_email=$2",
      [username, email]
    );
    if (checkIfUserExists.rows.length > 0) {
      return res.status(400).json("User with same username or email already exists");
    }
    const saltRounds = 10;
    const hashedPassword = await bcrypt.hash(password, saltRounds);
    const newUser = await pool.query(
      "INSERT INTO user_details (user_name,user_email,user_password) values ($1,$2,$3)
RETURNING   user_id",[username, email, hashedPassword]);
    res.status(201).json({ message: "Account created" });
  } catch (error) {
    res.status(500).json({
      error: error.message,
    });
  }
});
```

Figure 5.1: User Registration Module

Purpose:

- Provides user registration by storing user details securely.
- Prevents duplicate usernames or emails.

Implementation:

- Accepts username, email, and password from the frontend.
- Checks if the username or email already exists in the database.
- If not, hashes the password and stores the user data.
- Generates a JWT token upon successful registration.

Registration Flow:

1. User Initiates Registration: Provides username, email, and password, then submits the registration form.
2. Input Validation: Verifies all required fields, checks email format, and validates password requirements.
3. Duplicate Check: Checks for existing usernames and verifies email uniqueness in the database.
4. Account Creation: Hashes the password securely, stores user details, and completes the account creation process.

5.2.2 Login and Authentication Module

```
router.post("/", async (req, res) => {
  const { email, password, isAdmin } = req.body;
  if (!email || !password) {
    return res.status(400).json("Please fill all fields");
  }
  try {
    const result = await pool.query("SELECT * FROM user_details WHERE user_email=$1",[email]);
    if (result.rows.length === 0) {
      return res.status(400).json("Your account doesn't exist");
    }
    const user = result.rows[0];
    const checkIfPasswordIsCorrect = await bcrypt.compare(password,user.user_password);
    if (!checkIfPasswordIsCorrect) {
      return res.status(400).json("Your email or password is incorrect");
    }
    if (user.user_role === "admin" && !isAdmin) {
      return res.status(403).json("Admins must select the 'Are you an admin?' checkbox");
    }
    if (user.user_role !== "admin" && isAdmin) {
      return res.status(403).json("You are not authorized as an admin");
    }
    const token = jwtGenerator(user.user_id);
    res.status(200).json({message: "Logged in",token: token});
  } catch (error) {
    res.status(500).json("Server error");
  }
});
```

Figure 5.2: User Login Module

Purpose:

- Authenticates users by verifying email and password.
- Ensures proper admin login if required.

Implementation:

- Accepts email and password from the frontend.
- Validates the email against the database.
- Compares the provided password with the stored hashed password.
- Generates a JWT token for valid users.

Login Flow:

1. User Initiates Login: Enters email and password, and indicates admin status if applicable.
2. Credential Verification: Locates the user by email, compares the password hash, and verifies the user role.
3. Admin Authentication: Checks the admin checkbox if applicable, verifies admin privileges, and applies admin-specific rules.
4. Session Creation: Generates a JWT token, sets token expiration, and returns the authentication response.

5.2.3 JWT Generator Utility

```
function jwtGenerator(user_id) {
  const payload = {
    user: user_id,
  };
  return jwt.sign(payload, process.env.jwtSecretKey, {
    expiresIn: "10d",
  });
}
```

Figure 5.3: JWT Generator Module

Purpose:

- Creates a secure JWT token for authenticated users.
- Sets token expiration to 10 days.

Implementation:

- Accepts the user ID and creates a payload.
- Signs the token using a secret key with an expiry time.
- Returns the token for authentication use.

JWT Token Flow:

1. Token Generation: Creates a payload with the user ID, adds an expiration timestamp, signs it with a secret key, and sends the token to the frontend.
2. Token Usage: Includes the token in the Authorization header and checks its expiration status.
3. Token Renewal: Checks for approaching expiration and maintains session continuity.

5.2.4 Property Listing and Management Module

Property Listing Creation

```
const storage = multer.memoryStorage();
const upload = multer({ storage });
router.post("/",authenticateToken,upload.array("propertyImages", 5), async (req, res) => {
    const { propertyType, location, amenities, propertyRegion } = req.body;
    const images = req.files;
    const { latitude, longitude } = location;
    const details = JSON.parse(req.body.details);
    const userId = req.userId.id;
    try {
      const uploadPromises = images.map(async (image) => {
        const imageKey = `${userId}/${uuidv4()}-${image.originalname}`;
        const uploadParams = {
          Bucket: bucketName,
          Key: imageKey,
          Body: image.buffer,
          ContentType: image.mimetype,
        };
        const command = new PutObjectCommand(uploadParams);
        await s3.send(command);
        return imageKey;
      });
      const uploadedImages = await Promise.all(uploadPromises);
      const query = `INSERT INTO pending_property_listing_details (user_id, property_type,
title,approximate_location, latitude,longitude, price, guests, bedrooms, beds, bathrooms,
kitchens, swimming_pool, amenities, image_urls,property_region) VALUES ($1, $2, $3,$4, $5, $6,
$7, $8, $9, $10, $11, $12, $13,$14, $15,$16) RETURNING pending_property_id;`;
     const values = [userId,propertyType,details.Title,details.Location,latitude,longitude,
        details.Price,details.Guests,details.Bedrooms,details.Beds,details.Bathrooms,
        details.Kitchens,details["Swimming Pool"] || null,JSON.stringify(amenities),
        uploadedImages,propertyRegion];
     const result = await pool.query(query, values);
     res.json({message: "Listing created successfully",listingId: result.rows[0].id,
        uploadedImages,});
    } catch (error) {
      res.status(500).json({
        message: "An error occurred while processing the listing",
        error: error.message,
      });}});
```

Figure 5.4: Property Listing Module

Purpose:

- Allows users to create property listings with detailed information (Title,Price,Total Guest allowed etc).
- Uploads and stores multiple property images securely in the AWS S3 Bucket.

Implementation:

- Accepts property details (type, location, amenities, region) and up to 5 images.
- Uploads images to an S3 bucket and stores image URLs in the database.
- Inserts property data into the `pending_property_listing_details` table.
- Returns the listing ID and uploaded image URLs upon success.

Process Flow:

1. Property Owner Initiates Listing: Fills out the property details form, uploads up to 5 property images, and sets pricing and guest capacity.
2. Image Processing: Uploads images to AWS S3, generates URLs for each image, and stores image metadata.
3. Database Storage: Saves property details to the pending property listing table, links image URLs to the property, and associates the owner ID with the listing.
4. Confirmation: Returns a success message, provides the listing ID, and marks the property as pending admin approval.

5.2.5 Property Listing Deletion Module

```
router.delete("/", authenticateToken, async (req, res) => {
  const userId = req.userId.id;
  const { id } = req.body;
  try {
    if (!userId || !id) {
      return res.status(400).json({ message: "Invalid request" });
    }
    const result = await pool.query("SELECT image_urls FROM property_listing_details WHERE
id=$1 AND user_id=$2",[id, userId]);
    const rows = result.rows;
    if (rows.length === 0) {
      return res.status(404).json({ message: "Listing not found" });
    }
    const imageUrls = rows[0].image_urls;
    const deleteImagePromises = imageUrls.map(async (imageKey) => {
      const deleteParams = {
        Bucket: bucketName,
        Key: imageKey,
      };
      const command = new DeleteObjectCommand(deleteParams);
      return s3.send(command);
    });
    await Promise.all(deleteImagePromises);
    await pool.query(
      "DELETE FROM property_listing_details WHERE id=$1 AND user_id=$2",[id, userId]
    );
    res.status(200).json({ message: "Listing deleted successfully" });
  } catch (error) {
    res.status(500).json({
      errorMessage: error.message,
    });
  });
```

Figure 5.5: Property Listing Deletion Module

Purpose:

- Enables users to delete their property listings.
- Ensures related images are also removed from storage.

Implementation:

- Deletes associated images from the S3 bucket.
- Removes the property listing from the database.
- Returns a confirmation message upon successful deletion.

Process Flow:

1. Owner Requests Deletion: Selects the property to delete and confirms the deletion request.
2. Validation: Verifies owner permissions, checks the property booking status, and ensures no active bookings.
3. Resource Cleanup: Deletes images from the S3 bucket, removes database entries, and clears associated data.
4. Confirmation: Sends a success message, updates the owner's listing count, and logs the deletion event.

5.2.6 Property Listing Update

```
router.put("/", authenticateToken, async (req, res) => {
  const userId = req.userId.id;
  const {id,title,price,propertyType,approximateLocation,guests,beds,bedrooms,bathrooms,
    kitchens,propertyRegion,} = req.body;
  if (!id || !title || !price || !propertyType) {
    return res.status(400).json({ message: "Missing required fields" });
  }
  try {
    const checkQuery =
      "SELECT * FROM property_listing_details WHERE id = $1 AND user_id = $2";
    const checkValues = [id, userId];
    const checkResult = await pool.query(checkQuery, checkValues);
    if (checkResult.rowCount === 0) {
      return res.status(403).json({ message: "Unauthorized to update this listing" });
    }
    const updateListingQuery = `UPDATE property_listing_details SET title = $1, price = $2,
property_type = $3, approximate_location = $4,guests = $5, beds = $6, bedrooms = $7, bathrooms
= $8, kitchens = $9 , property_region=$10 WHERE property_id = $11`;
    const values =[title,price,propertyType,approximateLocation,guests,beds,bedrooms,
bathrooms,kitchens,propertyRegion,id];
    const updateListingData = await pool.query(updateListingQuery, values);
    if (updateListingData.rowCount === 0) {
      return res.status(404).json({ message: "Listing not found or no changes made" });
    }
    res.status(200).json({ message: "Listing updated successfully" });
  } catch (error) {
    res.status(500).json({ message: `Internal server error: ${error.message}` });
  }
});
```

Figure 5.6: Property Listing Update Module

Purpose:

- Allows users to update property listing details.
- Provides flexibility to modify property information.

Implementation:

- Validates user ownership of the listing before updating.
- Updates fields like title, price, property type, location, and region in the database.
- Returns a success message if the update is completed.

Process Flow:

1. Owner Initiates Update: Selects the property to modify, updates relevant fields, and submits the changes.
2. Validation: Verifies owner permissions, validates the new data, and checks the property status.
3. Database Update: Applies the changes to the database and updates the timestamp of the update.
4. Confirmation: Returns a success message, reflects changes immediately, and notifies the host of the changes.

5.2.7 Payment and Booking Management Module

Payment

```
router.post("/create-order", async (req, res) => {
  const { cart } = req.body;
  try {
    const orderResponse = await createOrder(cart);
    res.status(200).json(orderResponse.jsonResponse);
  } catch (error) {
    res.status(500).json({ error: "Failed to create order", details: error.message });
  }
});

router.post("/capture-order", async (req, res) => {
  const { orderID } = req.body;
  if (!orderID) {
    return res.status(400).json({ error: "Order ID is required" });
  }
  try {
    const captureResponse = await captureOrder(orderID);
    res.status(200).json(captureResponse.jsonResponse);
  } catch (error) {
    res.status(500).json({ error: "Failed to capture payment", details: error.message });
  }
});
```

Figure 5.7: Payment and Booking Management Module

Purpose:

- Handles PayPal payment integration for property bookings
- Creates and captures payment orders securely
- Ensures successful payment before booking confirmation

Implementation:

- Creates PayPal order with property details and amount
- Captures payment after user confirmation
- Handles payment success and failure scenarios
- Validates order ID and payment status

5.2.8 Booking Creation

```
router.post("/", authenticateToken, async (req, res) => {
  const {bookingStartDate,bookingEndDate,totalGuests,totalStay,bookedPropertyId,totalCost,
    bookingStatus,} = req.body;
  const bookersId = req.userId.id;
  try {if (bookersId && bookedPropertyId) {
      const insertBookingData = await pool.query("INSERT INTO bookings (user_id,property_id,
      booking_start_date,booking_end_date,total_guests,total_price,booking_status) VALUES
($1,$2,$3,$4,$5,$6,$7)",[bookersId,bookedPropertyId,bookingStartDate,bookingEndDate,
totalGuests,totalCost,bookingStatus,]);
      res.status(200).json({ message: "Booking successful!" });
    } else {
      res.status(400).json({ message: "Missing booker or property ID." });
    }} catch (error) {console.error(error.message);
    res.status(500).json({ message: `Server error  : ${error.message}` });
  }});
```

Figure 5.8: Booking Creation Module

Purpose:

- Creates booking records after successful payment
- Stores booking details including dates, guests, and costs
- Links booking to specific user and property

Implementation:

- Validates booking data (dates, guests, property availability)

- Uses authentication token to verify user identity
- Inserts booking record into database with payment status
- Returns booking confirmation to user

5.2.9 Payment and Booking Module Flow

Process Flow:

1. User initiates booking: Selects property and dates, confirms the number of guests and total cost.
2. Payment Processing: Frontend calls /create-order with cart details, PayPal order created with the total amount, and the user completes payment on the PayPal interface.
3. Payment Capture: Frontend calls /capture-order with orderID, payment captured and verified, and success/failure response returned.
4. Booking Creation: Booking details sent to the booking endpoint, new booking record created in the database, and booking confirmation returned to the user.

5.3 Review and Rating Module

5.3.1 Submit and Update Review

```
router.post("/:propertyId", authenticateToken, async (req, res) => {
  const { propertyId } = req.params;
  const userId = req.userId.id;
  const { rating, message } = req.body;
  if (!userId || !propertyId) {
    return res.status(400).json({ message: "User ID or Property ID not found" });
  }
  if (!rating || !message) {
    return res.status(400).json({ message: "Rating and message are required" });
  }
  try {
    const checkIfAlreadyReviewed = await pool.query(
      "SELECT * FROM reviews WHERE user_id=$1 AND property_id=$2",
      [userId, propertyId]);
    let result;
    if (checkIfAlreadyReviewed.rows.length > 0) {
      result = await pool.query("UPDATE reviews SET rating=$1, review_message=$2,
review_date_time=NOW() WHERE user_id=$3 AND property_id=$4",
      [rating, message, userId, propertyId]);
      const getRatings = await pool.query(
        "SELECT rating FROM reviews WHERE property_id = $1",[propertyId]);
      const allRatings = getRatings.rows.map((row) => row.rating);
      const averageRating =allRatings.reduce((sum, rating) => sum + rating, 0) /
          allRatings.length || 0;
      const insertRatingData = await pool.query("UPDATE property_listing_details SET
averate_review_rating=$1 WHERE property_id=$2",[averageRating, propertyId]);
    } else {
result = await pool.query("INSERT INTO reviews (user_id, property_id, rating, review_message)
VALUES ($1, $2, $3, $4)",[userId, propertyId, rating, message]);
      const getRatings = await pool.query("SELECT rating FROM reviews WHERE property_id = $1",
        [propertyId]);
      const allRatings = getRatings.rows.map((row) => row.rating);
      const averageRating =
        allRatings.reduce((sum, rating) => sum + rating, 0) /
          allRatings.length || 0;
      const insertRatingData = await pool.query(
        "UPDATE property_listing_details SET averate_review_rating=$1 WHERE property_id=$2",
        [averageRating, propertyId]
      );}
    if (result.rowCount > 0) {
      res.status(200).json({ message: "Review submitted successfully" });
    } else {
      res.status(400).json({ message: "Error submitting review" });
    }
  } catch (error) {
    res.status(500).json({ message: "Internal server error" });
  }
});
```

Figure 5.9: Review and Rating Module

Purpose:

- Allows users to submit reviews and ratings for properties they have stayed in.

- Enables users to update their existing reviews and ratings.
- Automatically calculates and updates the average rating for each property.

Implementation:

- Validates if the user and property IDs exist.
- Checks if the user has already reviewed the property.
- Updates existing reviews or inserts new reviews accordingly.
- Calculates the average rating and updates it in the property listing.

Process Flow:

1. User Submits Review: Provides a rating and review message, and the system checks if a review already exists for this user and property.
2. Review Handling: If a review exists, it updates the rating and review message, along with the timestamp; if no review exists, it inserts a new review record into the database.
3. Average Rating Calculation: Retrieves all ratings for the property, calculates the average rating, and updates the property's average rating in the `property_listing_details` table.
4. Confirmation: Returns a success message upon successful submission or update, or handles errors with appropriate responses.

5.3.2 Property Recommendation Module

```
router.post('/recommendations', authenticateToken, async (req, res) => {
    const userId = req.userId.id;

    try {
        // 1. Get user preferences and history weight
        const [preferences, userHistoryWeight] = await Promise.all([
            getUserPreferences(userId),
            getUserHistoryWeight(userId)
        ]);

        if (!preferences) {
            return res.status(404).json({
                error: 'No preferences found',
                message: 'Please set your preferences first'
            });
        }

        // 2. Get unviewed properties
        const properties = await getUnviewedProperties(userId, SCORE_WEIGHTS.MAX_RECOMMENDATIONS);

        // 3. Calculate scores and get signed URLs for each property
        const recommendedProperties = await Promise.all(
            properties.map(async (property) => {
                const [collaborativeScore, signedImageUrls] = await Promise.all([
                    getCollaborativeScore(userId, property.property_id),
                    generateSignedUrls((property.image_urls || []).slice(0, 5))
                ]);

                const contentScore = calculateContentScore(property, preferences);
                const hybridScore = calculateHybridScore(
                    contentScore,
                    collaborativeScore,
                    userHistoryWeight
                );

                return {
                    ...property,
                    image_urls: signedImageUrls,
                    content_score: contentScore,
                    collaborative_score: collaborativeScore,
                    hybrid_score: hybridScore
                };
            })
        );

        // 4. Sort by hybrid score and return results
        const sortedRecommendations = recommendedProperties
            .sort((a, b) => b.hybrid_score - a.hybrid_score)
            .slice(0, SCORE_WEIGHTS.MAX_RECOMMENDATIONS);

        res.json({
            recommendedProperties: sortedRecommendations,
            meta: {
                total: sortedRecommendations.length,
                userHistoryWeight,
                generatedAt: new Date().toISOString()
            }
        });
    } catch (error) {
        console.error('Error in recommendations:', error);
        res.status(500).json({
            error: 'Internal server error',
            message: 'Unable to generate recommendations'
        });
    }
});
```

Figure 5.10: Property Recommendation Code

Purpose:

- Provides personalized property recommendations for users based on preferences, history, and collaborative filtering.
- Enhances user experience by suggesting properties relevant to their interests.

Implementation:

- Retrieves user preferences such as preferred property type, region, and price range from the database.
- Calculates a user history weight based on the number of reviews submitted, applying logarithmic scaling.
- Identifies unviewed properties by filtering out those already reviewed or booked by the user.
- Computes a collaborative score using similarity-based collaborative filtering, leveraging Pearson correlation among user ratings.
- Determines a content-based score by evaluating property attributes against user preferences.
- Merges content and collaborative scores using a weighted hybrid approach influenced by user history.

Process Flow:

1. Retrieve User Preferences: Fetches the user's preferred property type, region, and price from the database.
2. Calculate User History Weight: Determines user engagement level based on past reviews, scaling the weight logarithmically.
3. Fetch Unviewed Properties: Retrieves properties that the user has not reviewed or booked and computes their average rating.
4. Compute Collaborative Score: Identifies similar users based on rating patterns and derives a weighted property score.

5. Calculate Content Score: Assesses how well a property matches user preferences and assigns a weighted score.
6. Determine Hybrid Score: Combines content and collaborative scores, adjusting the weight dynamically based on user engagement.
7. Return Recommended Properties: Sorts properties by their hybrid scores and selects the top recommendations.

5.4 Testing

Testing is a vital phase in software development that ensures the system operates as intended and meets user requirements. It involves different types of testing, such as unit testing, which focuses on verifying individual components or modules for correctness, and system testing, which evaluates the entire system's functionality and performance as an integrated whole. These testing methods help identify defects, ensure reliability, and improve overall software quality, reducing risks before deployment.

5.4.1 Test Cases for Unit Testing

Unit testing for user login ensures that the authentication process functions correctly. It tests the validation of user inputs, like checking if the username and password match the expected format, and verifies that the credentials are correctly compared against the stored data. Invalid inputs should trigger appropriate error messages. By isolating the login logic with mock dependencies (e.g., database calls or password hashing), unit tests help ensure the reliability and security of the login process.

1. Unit Test for Signup

Table 5.1: Test Cases for User Signup

Test ID	Test Cases	Test Input	Expected Outcome	Actual Outcome	Test Result
T01	Empty user input	null	Registration Failed	Registration Failed	Pass
T02	New User registration	UserName: XYZ, Email: xyz@gmail.com, Password: abcd, Confirm Password: abcd	Registration success	Registration success	Pass
T03	Existing User registration	UserName: XYZ, Email: xyz@gmail.com, Password: abcd, Confirm Password: abcd	Registration failed	Registration failed	Pass
T04	Empty username field	UserName: null, Email: xyz@gmail.com, Password: abcd, Confirm Password: abcd	Registration Failed	Registration Failed	Pass
T05	Empty email field	UserName: test, Email: null, Password: abcd, Confirm Password: abcd	Registration Failed	Registration Failed	Pass
T06	Empty Password field	UserName: test, Email: test@gmail.com, Password: null, Confirm Password: null	Registration Failed	Registration Failed	Pass
T07	Password Mismatch	UserName: test, Email: test@gmail.com, Password: abcd, Confirm Password: abcd123	Registration Failed	Registration Failed	Pass

2. Unit Test for Login

Table 5.2: Test Cases for User Login

Test ID	**Test Cases**	**Test Input**	**Expected Outcome**	**Actual Outcome**	**Test Result**
T08	Empty user input	null	Login Failed	Login Failed	Pass
T09	Incorrect Email	Email: 123@gmail.com Password: abc	Login Failed	Login Failed	Pass
T10	Incorrect Password	Email: test@gmail.com Password: a	Login Failed	Login Failed	Pass
T11	Correct credentials	Email: user@gmail.com Password: abc	Login success	Login success	Pass

2. Unit Test for Property Search Functionality

Table 5.3: Test Cases for Property Search Functionality

Test ID	Test Cases	Test Input	Expected Outcome	Actual Outcome	Test Result
T12	Search by Region	Region: Bagmati	Properties on bagmati region should be displayed	Properties on bagmati region is displayed	Pass
T13	Search by Date	Check in:2025-04-1, Check out:2025-04-4	Properties available on given date should be displayed	Properties available on given date is displayed	Pass
T14	Search by Total Guest Capacity	Total guests: 12	Properties that accommodate 12 guests should be displayed	Properties that accommodate 12 guests is displayed	Pass
T15	Search by Region and Date Range	Region:Pokhara Check-in: 2025-04-01 Check-out: 2025-04-10	Properties in Pokhara available during the date range should be displayed	Properties in Pokhara available during the date range is displayed	Pass
T16	Search by Date Range and Number of Guests	Check-in: 2025-05-01, Check-out: 2025-05-05, Guests: 3	Properties available for the dates with 3 guests should be displayed	Properties available for the dates with 3 guests is be displayed	Pass
T17	Search by Region, Date, and Guests	Region: Lumbini, Check-in: 2025-06-01, Check-out: 2025-06-07, Guests: 5	Properties in Lumbini for 5 guests during the date range should be displayed	Properties in Lumbini for 5 guests during the date range is displayed	Pass
T18	Search by Property Type	Property Type: House	Properties with type "House" should be displayed	Properties with type "House" is displayed	Pass
T19	Search by Distance	Distance: Nearest	Properties closest to the location should displayed	Properties closest to the location is displayed	Pass
T20	Search by Price	Price: 12000	Properties with price less than or equal to 12000 should be displayed	Properties with price less than or equal to 12000 is displayed	Pass
T21	Search by Rating	Rating: 5	Properties with rating 5 should be displayed	Properties with rating 5 is displayed	Pass
T22	Search by Property Type, Distance, Price, and Rating	Property Type: House, Distance: Nearest, Price: 12000, Rating: 5	Properties that are houses, nearest, priced at 12000, and rated 5 should be displayed	Properties that are houses, nearest, priced at 12000, and rated 5 is displayed	Pass

3. Unit testing for property listing

Table 5.4: Image Upload and Property Type Selection Tests

Test ID	Test Cases	Test Input	Expected Outcome	Actual Outcome	Test Result
T21	Submit without inserting any images	null	Form submission should fail	Form submission failed	Pass
T22	Submit by Uploading less than 5 images	3 images uploaded	Form submission should fail	Form submission failed	Pass
T23	Upload 5 Images	5 images uploaded	5 image stored and displayed	5 image stored and displayed	Pass
T24	Attempt to upload more than 5 images	Try to add 6th image	Add image button hidden	Add image button hidden	Pass
T25	Remove and Image	Click remove button on an image	Image removed and slot available for new image upload	Image removed and slot available for new image upload	Pass
T26	Upload invalid file type	Upload .pdf file	File should not be accepted	File input rejected	Pass
T27	Submit without selecting property types	null	Form submission should fail	Form submission failed	Pass
T28	Select property type	Property Type: House	Property type should be stored in state	Selected Property type stored and displayed	Pass
T29	Change property Selection	Property Type: Tent	Previous state should be overridden and displayed	Previous state overridden and displayed	Pass

Table 5.5: Amenities, Region, Location and Form Submission Tests

Test ID	Test Cases	Test Input	Expected Outcome	Actual Outcome	Test Result
T30	Submit without Selecting amenities	null	Form submission should fail	Form submission failed	Pass
T31	Select multiple amenities	Amenities: Wifi,TV AC,BBQ grill	Amenity should be stored in state and displayed	Amenities stored in state and displayed	Pass
T32	Deselect amenity	Click selected amenity	Amenity should be removed from selection and state	Amenity removed from selection and state	Pass
T33	Submit without selecting region	Null	Form submission should fail	Form submission failed	Pass
T34	Select region	Region: Bagmati	Region Should be Stored in state and displayed	Region Stored in state and displayed	Pass
T35	Change region	Region: Lumbini	Region Should be overridden and displayed	Region overridden and displayed	Pass
T36	Submit without Selecting Location on Map	null	Form submission should fail	Form submission failed	Pass
T37	Submit Selecting location on Map	Latitude: 24.000012 Longitude: 80.123	Coordinates should be stored in state and location should be displayed on map	Coordinates stored in state and location displayed on map	Pass
T38	Submit Complete Listing Form	All required fields filled correctly	Listing Should be created successfully	Listing Created successfully	Pass
T39	Submit while already submitting	Click submit button multiple times	Only one submission should be processed	Only one submission is processed	Pass
T40	Submit without Auth Token	Remove Token from LocalStorage	User should be redirected to login page	User redirected to login page	Pass

5.4.2 Test Cases for System Testing

To test the proper functioning of the system as a whole, we assume a test case as follows: A new user(User4) is created with the required credentials, then the user books(visits) three properties and these properties are assumed to be reviewed after the booking_end_date of all three bookings have been passed. Before creating User4, we have registered users with their own set of visited and reviewed properties, as shown in Table 5.6 below.

Table 5.6: Reviews Table before inserting User4

S.N.	User_id	Property_id	Rating
1	1	2	5
2	1	4	4
3	1	5	3
4	2	2	5
5	2	4	4
6	2	5	3
7	2	6	5
8	3	2	1
9	3	6	1
10	3	4	1
11	3	5	1
12	1	7	3
13	1	8	4
14	2	8	5
15	3	7	4
16	3	10	5

We can also take required property details of all 10 properties with required parameters like property_id, property_region and price in 5.7.

Table 5.7: Property List

Property ID	Property Name	Property Region	Property Type	Price
1	Sunrise Villa	Koshi	House	10000.00
2	Bagmati Grand	Bagmati	Hotel	12000.00
4	Mountain Retreat	Gandaki	Cottage	15000.00
5	Lumbini Heights	Lumbini	Apartment	17000.00
6	Ocean Breeze	Sudurpaschim	Tent	18000.00
7	Madhesh Residency	Madhesh	House	20000.00
8	Karnali Inn	Karnali	Hotel	20998.00
9	Riverside Haven	Koshi	Cottage	21998.00
10	Urban Skyline	Bagmati	Apartment	26000.00

After creating User4, the user logs into the system. We confirm that User4 does not receive any recommendations because no preferences are set and there are no prior bookings or ratings. The user then sets the following preferences:

- **preferred_property_type** = House
- **preferred_property_region** = Madhesh
- **preferred_price** = Rs.20,000

ext, bookings and reviews are created for User4. After rating the booked properties, the updated Reviews Table is shown in Table 5.8. User4 then receives recommendations on their homepage.

Table 5.8: Reviews Table after inserting User4

S.N.	User_id	Property_id	Rating
1	1	2	5
2	1	4	4
3	1	5	3
4	2	2	5
5	2	4	4
6	2	5	3
7	2	6	5
8	3	2	1
9	3	6	1
10	3	4	1
11	3	5	1
12	1	7	3
13	1	8	4
14	2	8	5
15	3	7	4
16	3	10	5
17	4	2	4
18	4	4	3
19	4	5	2
20	4	8	2
21	4	10	5

The Hybrid Score for User4 is calculated then we receive output as follows for each property in order

Table 5.9: Content, Collaborative, and Hybrid Scores of Recommended Properties

Property ID	Content Score	Collaborative Score	Hybrid Score
7	1.00	0.7	0.88
6	0.32	0.49	0.39
9	0.32	0.00	0.20
1	0.30	0	0.18

5.5 Result Analysis

The Recommendation received by a particular user is based on the properties they have set as their preference and by the pattern of their previous bookings. In the above test case, if User4 is to rate no properties then the scores value changes as follows:

Hybrid Score with 5 properties rated by User4: The collaboration filtering will influence 39% of the Hybrid score, and content based filtering will influence 61% of the Hybrid score.Let us assume the properties preference is same as before (i.e. House,Madhesh,20000). Here, the following properties will be recommended as user has 5 properties previously booked and reviewed.

Table 5.10: Scores with 5 properties reviewed

Property ID	Content Score	Collaborative Score	Hybrid Score
7	1.00	0.7	0.88
6	0.32	0.49	0.39
9	0.32	0	0.20
1	0.30	0	0.18

Hybrid Score with no properties rated by User4:
The collaboration filtering falls back to an adjustment where collaborative filtering will only influence 20% of the Hybrid score, and content based filtering will influence 80% of the Hybrid score.Let us assume the properties preference is same as before (i.e.

House,Madhesh,20000). Here, all the properties will be recommended as user has no properties previously booked but current bookings of other users will be discarded if present.

Table 5.11: Scores without properties reviewed

Property ID	Content Score	Collaborative Score	Hybrid Score
7	1.00	0.7	0.94
8	0.36	0.9	0.47
6	0.32	0.6	0.38
10	0.16	1	0.33
5	0.28	0.46	0.32
4	0.2	0.6	0.28
9	0.32	0	0.26
1	0.3	0	0.24
2	0.08	0.73	0.21

Hybrid Score with two properties rated by User4:

The collaboration filtering falls back to an adjustment where collaborative filtering will only influence 24% of the Hybrid score, and content based filtering will influence 76% of the Hybrid score.Let us assume the properties preference is same as before (i.e. House,Madhesh,20000). Here, all the properties except 2 and 4 will be recommended as user has 2 and 4 booked and reviewed.

Table 5.12: Scores with 2 properties reviewed

Property ID	Content Score	Collaborative Score	Hybrid Score
7	1.00	0.7	0.93
8	0.36	0.9	0.49
6	0.32	0.6	0.39
10	0.16	1	0.36
5	0.28	0.46	0.32
9	0.32	0	0.24
1	0.3	0	0.23

Hybrid Score with three properties rated by User4:

The collaboration filtering falls back to an adjustment where collaborative filtering will only influence 30% of the Hybrid score, and content based filtering will influence 70% of the Hybrid score.Let us assume the properties preference is same as before (i.e. House,Madhesh,20000). Here, all the properties except 2, 4 and 5 will be recommended as user has 2,4 and 5 booked and reviewed.

Table 5.13: Scores with 3 properties reviewed

Property ID	**Content Score**	**Collaborative Score**	**Hybrid Score**
7	1.00	0.6	0.88
8	0.36	0.9	0.52
6	0.32	1	0.52
9	0.32	0	0.22
1	0.3	0	0.21
10	0.16	0	0.11

CHAPTER 6

CONCLUSION AND FUTURE RECOMMENDATIONS

6.1 Conclusion

The vacation home rental system developed successfully addresses the need for an efficient, user-friendly platform for both property hosts and guests. Through its seamless user interface and robust backend, the system allows users to register, search for vacation properties, book accommodations, and leave reviews. Hosts can manage property listings, handle bookings, and communicate with guests, while administrators can monitor and manage user activities. The system was built using modern web technologies and adheres to best practices in design and development, ensuring reliability and scalability. Overall, the system provides a complete solution for vacation home rentals, offering convenience, security, and flexibility for all users.

6.2 Future Recommendations

While the current system provides a solid foundation, several enhancements can be made to further improve its functionality and user experience:

1. Mobile Application: Developing a mobile application for both iOS and Android to allow users to access the platform on-the-go.
2. Dynamic Pricing: Implementing more advanced pricing model based on specific amenities, ratings, and guest preferences.
3. Real-time Messaging: Adding a real-time chat feature for hosts and guests to communicate instantly.
4. Geographic Expansion: Introducing new locations to the system for international usage.

5. Payment Gateway Integration: Expanding payment options to include more international payment gateways for greater flexibility.

These improvements would not only enhance the user experience but also broaden the system's reach and functionality in the competitive vacation rental market.

REFERENCES

[1] Altexsoft. “Vacation Rentals.” 2022. Retrieved from Altexsoft, 2022. URL: https://www.altexsoft.com/glossary/vacation-rentals/.

[2] A. Shalimov. “Vacation Rentals.” 2023. Retrieved from Eastern Peak, 2023. URL: https://easternpeak.com/blog/vacation-rental-management-software/.

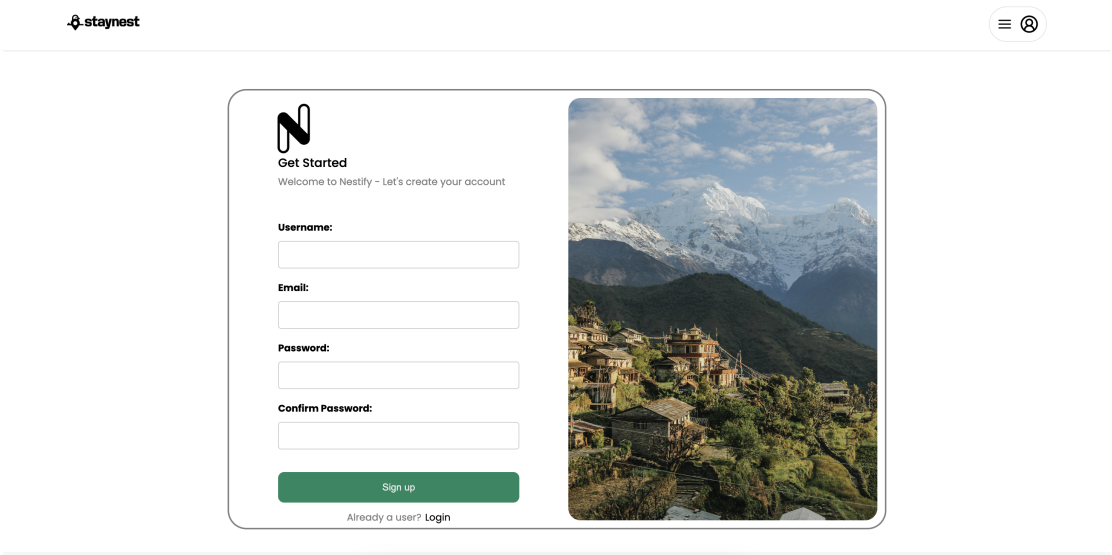
[3] S. O. Mariwa and T. K. Tunduny. “A Web-Based Application for Making Low-Cost Vacation Reservations for Tourists Using the K-Nearest Neighbors Algorithm.” 2022. Retrieved from ISPRS Archives, 2022. URL: https://isprs-archives.copernicus.org/articles/XLVIII-4-W3-2022/67/2022/.

[4] R. W. Wassmer. “The Value of Proximity to a Vacation Home Rental in a Resort Community.” 2019. Retrieved from SAGE Journals, 2019. URL: https://journals.sagepub.com/doi/abs/10.1177/0739456X19844814.

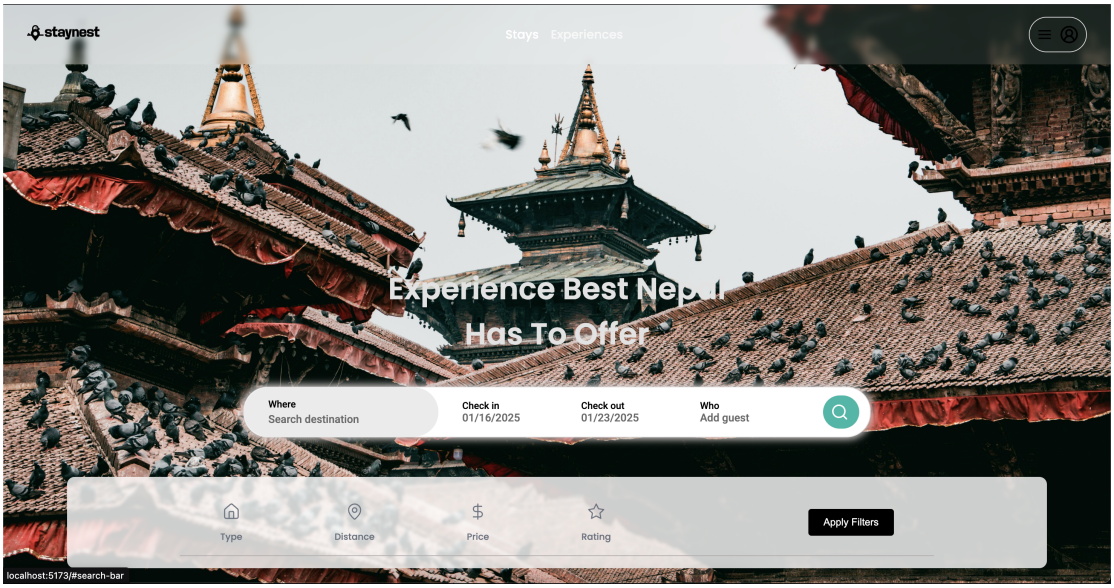
[5] Arrived. “The Best Vacation Rental Sites.” 2022. Retrieved from Arrived, 2022. URL: https://arrived.com/blog/the-10-best-vacation-rental-sites.

APPENDIX

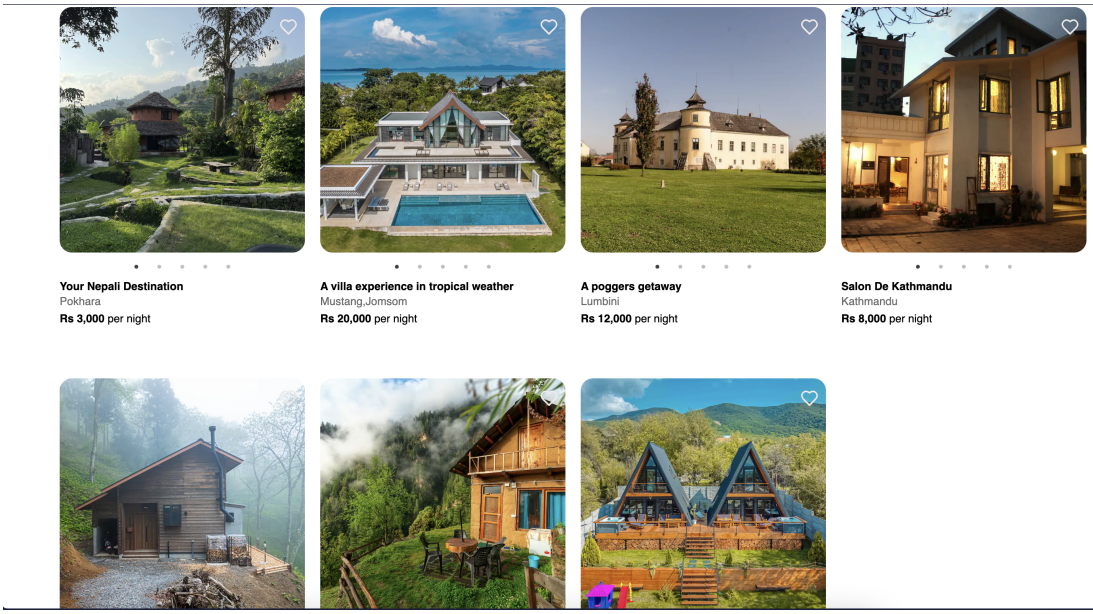
Register / Login Page



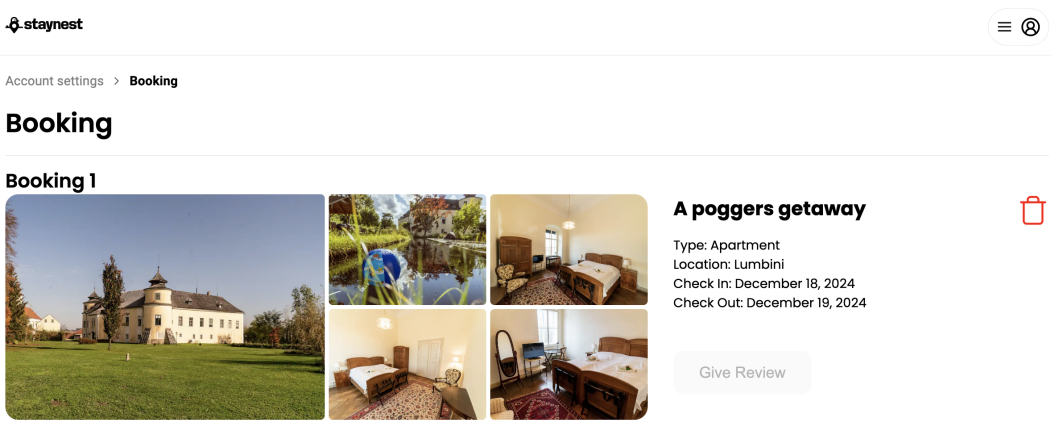
Home Page



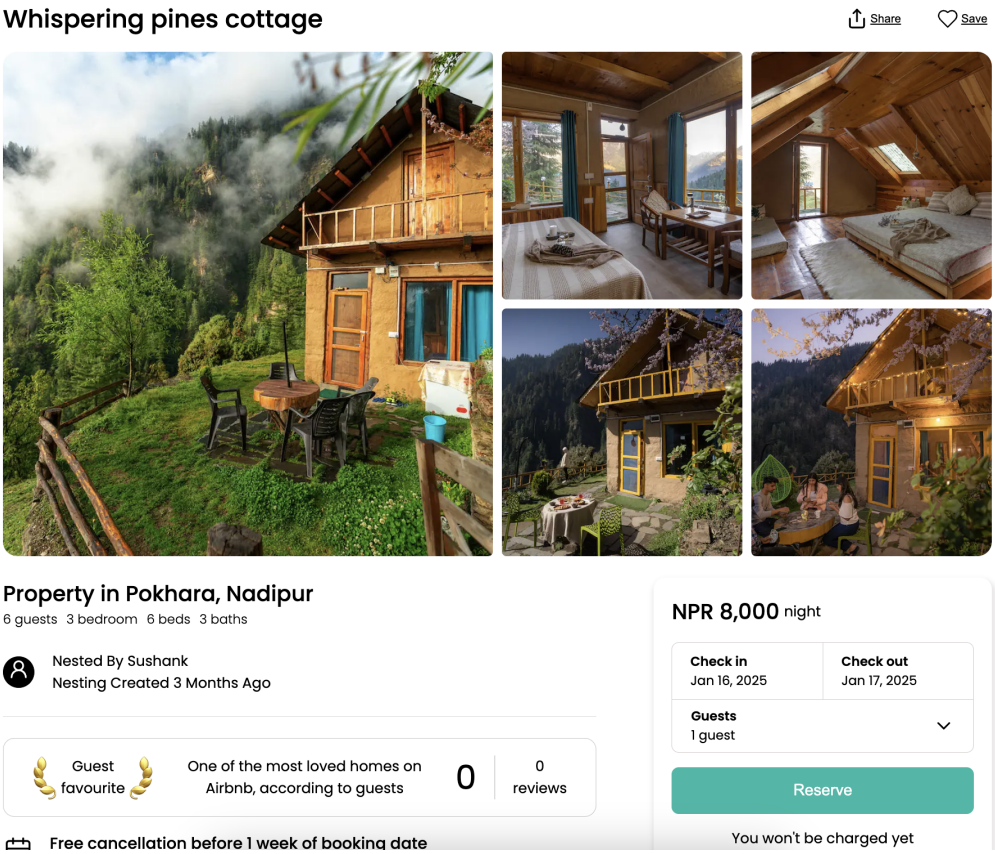
All Properties



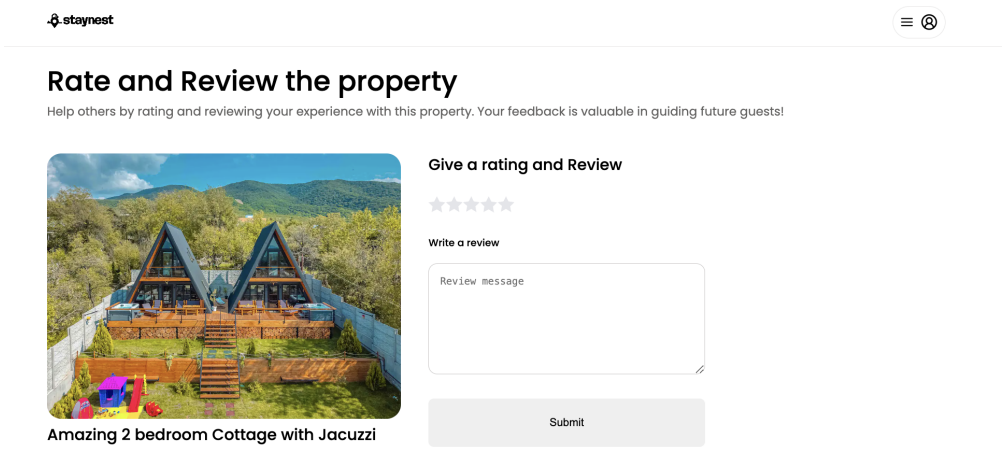
Bookings



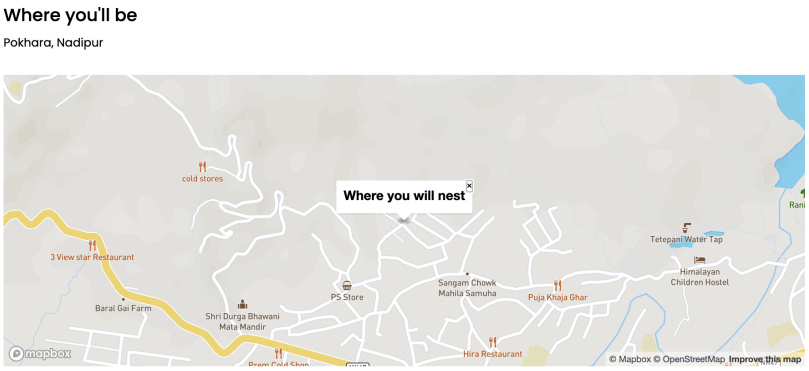
Property Information



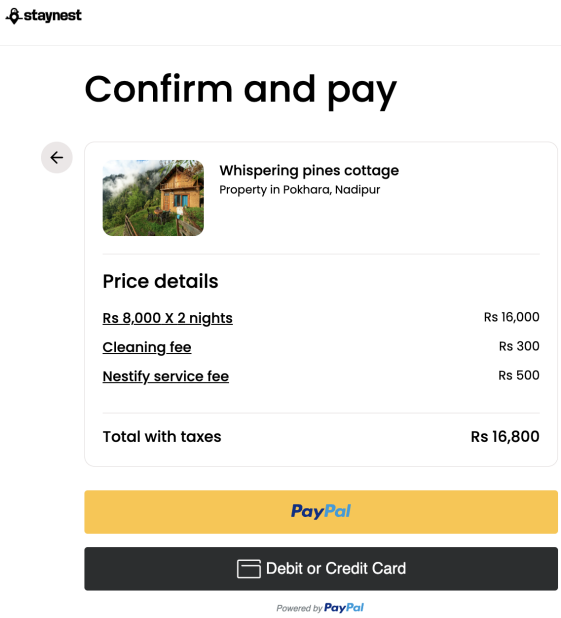
Review



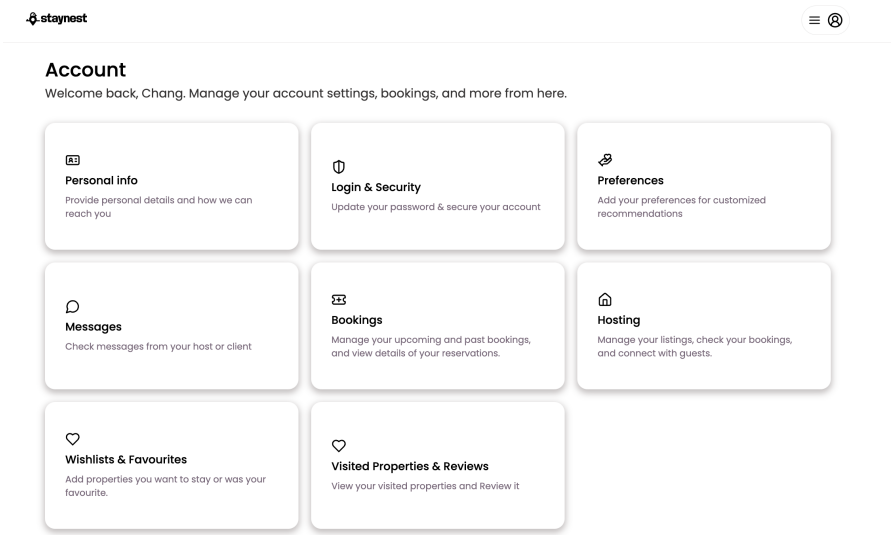
Property Location



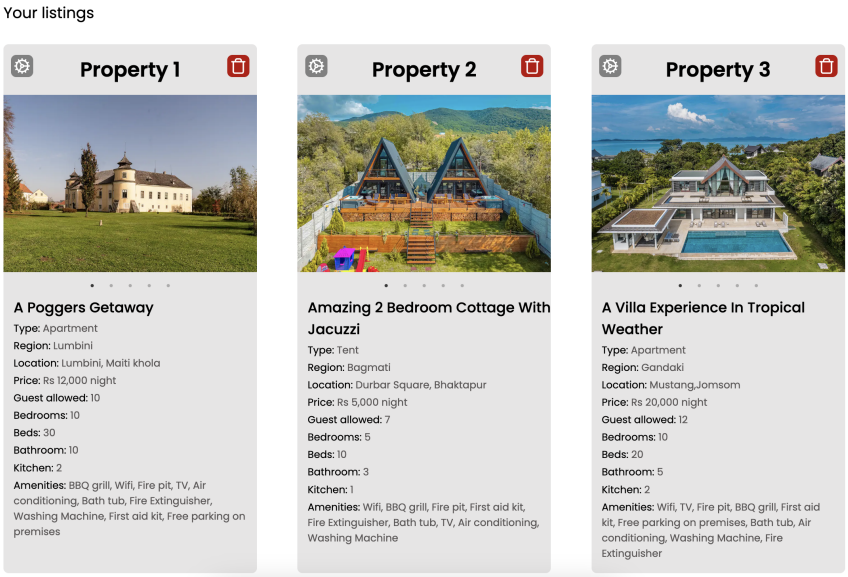
Payment



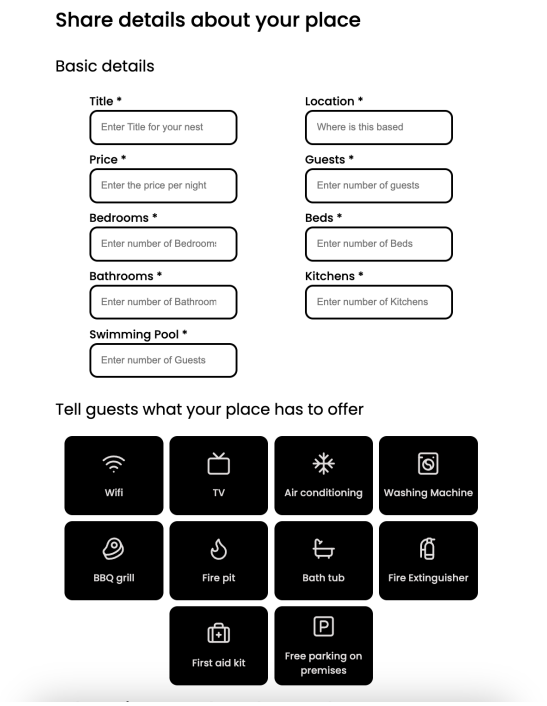
Account Settings

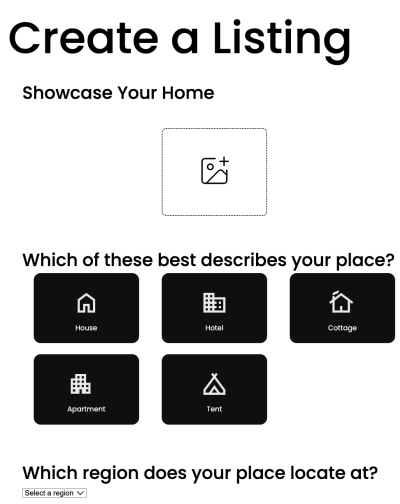


Listed Property

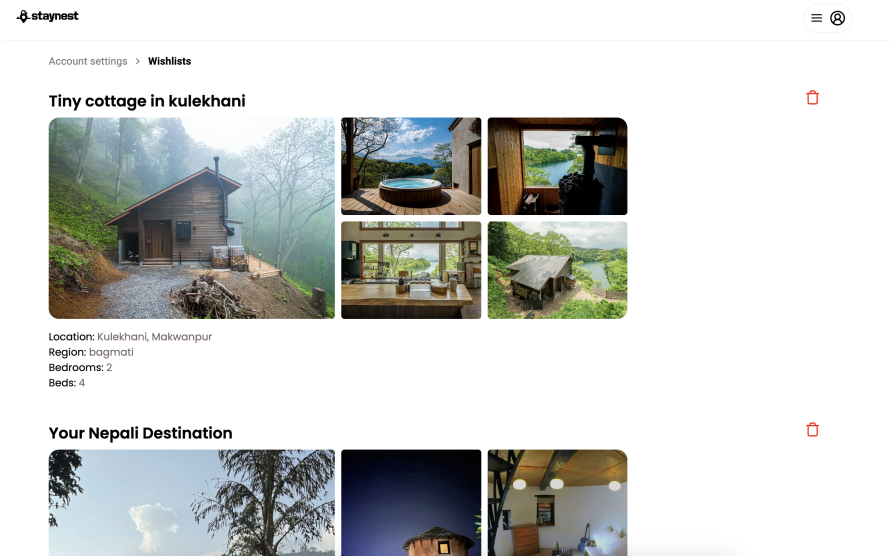


Create Listing

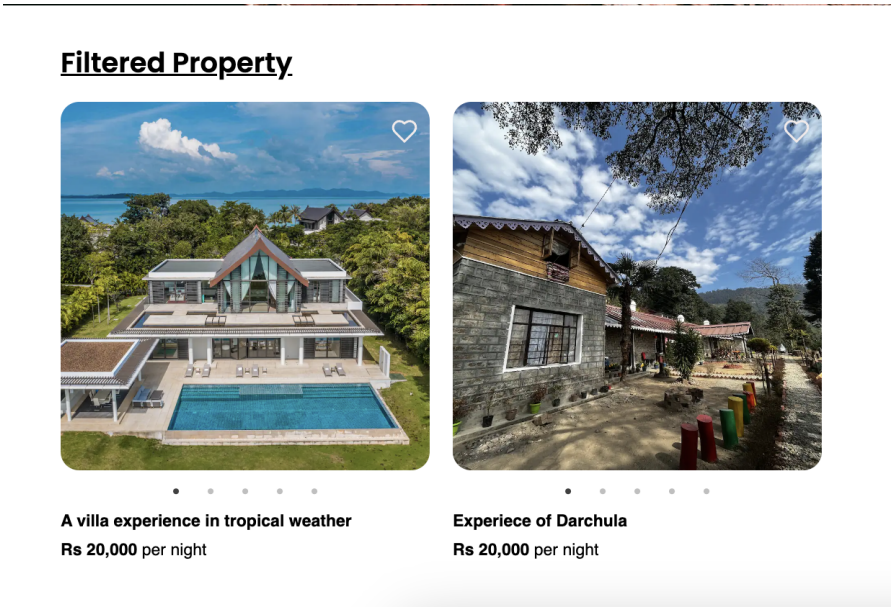




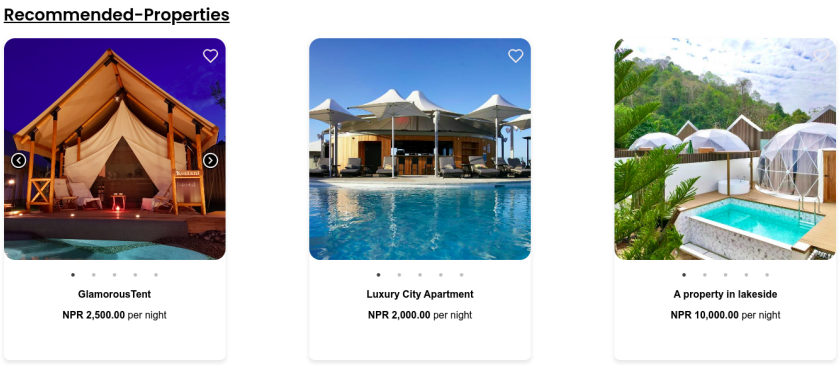
Wishlist



Filtered Properties



Recommended Properties



Admin Dashboard