

EASYSTAY ROOM RESERVATION

A Project Report

Submitted in partial fulfilment of the requirement for
the award of the Degree

BACHELOR OF SCIENCE (COMPUTER SCIENCE)

By

Mr. Surendraprasad Madanlal Yadav

ROLL NO: -102

Under the esteemed guidance of

Mrs. Trupti Rangore

Assistant Professor



DEPARTMENT OF COMPUTER SCIENCE

**SATISH PRADHAN DNYANASADHANA COLLEGE OF ARTS,
SCIENCE AND COMMERCE**

(Affiliated To University Of Mumbai)

THANE, 400604

MAHARASHTRA

2024-2025

**SATISH PRADHAN DNYANASADHANA COLLEGE OF ARTS,
SCIENCE AND COMMERCE**
(Affiliated To University Of Mumbai)
THANE, MAHARASHTRA , 400604

DEPARTMENT OF COMPUTER SCIENCE



CERTIFICATE

This is to certify that the project entitled, **EasyStay Room Reservation** is bonafide work of **Mr. Surendraprasad Madanlal Yadav** bearing **Seat. No: _____** submitted in partial fulfilment of the requirements for the award of degree of **BACHELOR OF SCIENCE in COMPUTER SCIENCE** from University of Mumbai.

Internal Guide

Head Of Department

External Examiner

Date :

College Seal

ABSTRACT

This project introduces a comprehensive EasyStay Room Reservation System, meticulously designed to incorporate a range of essential features aimed at optimizing the management of room bookings. At its core, the system includes robust room reservation management capabilities, allowing for accurate tracking and control of room availability across multiple properties. This ensures that room listings are consistently monitored, minimizing the risk of double bookings and ensuring smooth reservation processes.

A standout feature of the system is its alert mechanism, which is programmed to notify administrators when room bookings are approaching full occupancy or when maintenance is required. This proactive alert system ensures that property owners can address any issues before they impact guests, helping maintain high standards of service.

Additionally, the system supports the numerical arrangement of room listings, which allows for an organized and systematic approach to managing properties. This feature simplifies the process of locating and identifying rooms, which is particularly beneficial when dealing with a large volume of listings.

The booking management functionality further enhances the system's efficiency by providing tools to handle the intake of reservations. This includes real-time updates on room availability and booking confirmations, ensuring that all information is accurate and up-to-date.

Central to the EasyStay Room Reservation System is the admin panel, which offers comprehensive oversight and control. The admin panel provides detailed lists of all available rooms, user interactions, and booking logs. Each booking is timestamped, providing a precise record of reservations and guest information. This level of detail is invaluable for maintaining accountability and providing excellent customer service.

Looking ahead, the project envisions future enhancements, including the integration of QR code technology. This feature will allow users to check into their rooms seamlessly by scanning a code, simplifying the process and reducing the potential for errors. Guests will be able to quickly and accurately access room details, streamlining the reservation experience.

In summary, the EasyStay Room Reservation System is a comprehensive solution that addresses the challenges of room reservation management. With features like real-time availability tracking, alert notifications, numerical arrangement of rooms, and detailed administrative oversight, it promises to enhance the efficiency and accuracy of property management. The planned future integration of QR code technology underscores the system's commitment to continuous improvement and innovation.

ACKNOWLEDGEMENT

I would like to extend our heartiest thanks with a deep sense of gratitude and respect to all those who provided me immense help and guidance during my period. I would like to thank my Project Guide **Asst.Prof. Trupti Rangore** for providing a vision about the system. I have greatly benefited from their regular critical reviews and inspiration throughout my work I am grateful for their guidance, encouragement, understanding and support in the development process.

I would also like to thank my college for giving resources whenever I wanted and for giving me the opportunity to develop the project. I would like to express my sincere thanks to our Head of Department **Dr. Sujata Iyer** for having facilitated us with the essential infrastructure resources without which this project would not have seen the light of the day. I am also thankful to the entire staff of CS/IT for their constant encouragement of my suggestions and moral support throughout the duration of my project. Last but not the least I would like to mention here that I am greatly indebted to each and everybody my friends and who has been associated with my project at any stage but whose name does not find a place in this acknowledgement.

With Sincere Regards, Surendra Yadav

DECLARATION

I hereby declare that the project entitled, “**EasyStay Room Reservation**” done at **Satish Pradhan Dnyanasadhana College, Thane**, has not been in any case duplicated to submit to any other university for the award of any degree. To the best of my knowledge other than me, no one has submitted to any other university.

The project is done in partial fulfilment of the requirements for the award of degree of **BACHELOR OF SCIENCE (COMPUTER SCIENCE)** to be submitted as a fifth semester project as part of our curriculum.

Name and Signature of the Student

EVALUATION CERTIFICATE

This is to certify that the undersigned have assessed and evaluated the project on, " **EasyStay Room Reservation** ".

Submitted by , **Surendra Yadav** student of **B.Sc. (Computer Science)**. This project is original to the best of our knowledge and has been accepted for Assessment.

External Examiner

TABLE OF CONTENTS

Sr. No.	Project Chapters	Page No.
1.	Chapter 1: Introduction 1.1 Background 1.2 Objectives 1.3 Purpose & Scope 1.3.1 Purpose 1.3.2 Scope	6
2.	Chapter 2 : System Analysis 2.1 Existing System 2.2 proposed System 2.3 Requirement Analysis 2.4 Hardware Requirements 2.5 Software Requirements 2.6 Justification of selection of Technology	10
3.	Chapter 3 : System Design 3.1 Modulo Division 3.2 Data Dictionary 3.3 Entity Relationship Diagrams 3.4 Data Flow Diagrams 3.5 Use Case Diagram 3.6 Deployment Diagram 3.7 Class Diagram 3.8 Component Diagram 3.9 Activity Diagram 3.10 Object Diagram 3.11 State Chart Diagram 3.12 Gantt Chart	15
4.	Chapter 4: Implementation and Testing 4.1 Code 4.2 Testing Approach 4.2.1 Unit Testing 4.2.2 Integration System	39
5.	Chapter 5 : Result	77
6.	Chapter 6: Conclusion and Future Work	84
7.	Chapter 7: References	86

LIST OF TABLES

Sr.No	Table	Page No
1	Hardware Requirement	14
2	Software Requirement	14

LIST OF FIGURES

Sr.NO	Figure Name	Page No
1	Data Flow Diagrams	21
2	ER Diagram	19
3	Class Diagram	26
4	Use Case Diagram	24
5	Activity Diagram	29
6	Deployment Diagram	25
7	Object Diagram	31
8	Component Diagram	28
9	Flow Chart Diagram	35
10	State Chart Diagram	33
11	Gantt Chart	37

Introduction

The EasyStay Room Reservation system is a sophisticated solution designed to streamline the management of room rentals, ensuring efficient tracking and utilization of available accommodations. This system addresses the complexities and challenges associated with booking and managing room inventories, particularly for both room owners and customers.

This platform enables room owners to list their properties, update details, and monitor bookings while allowing customers to search for, view, and reserve rooms based on their preferences and availability. A key feature of this system is its real-time availability updates, which significantly reduce the potential for double bookings and ensure that customers have access to accurate information.

Moreover, the EasyStay Room Reservation system includes a notification mechanism that alerts users about booking confirmations, payment statuses, and other essential updates. This proactive communication enhances user experience by keeping all parties informed and engaged throughout the reservation process.

Additionally, the system facilitates efficient bill generation, ensuring transparency in financial transactions. The integrated admin panel provides comprehensive oversight of all activities within the platform, allowing for seamless management of listings, user interactions, and overall system performance. This comprehensive approach guarantees a smooth, reliable, and user-friendly experience for all stakeholders involved in the room reservation process.

1.1 Background

In the hospitality industry, effective management of room reservations is crucial for both room owners and customers. The need for accurate tracking and management of bookings is driven by customer satisfaction, operational efficiency, and revenue optimization. However, many establishments still rely on outdated or manual systems for managing room inventories, which can lead to errors, inefficiencies, and potential customer dissatisfaction. Manual tracking can result in double bookings, overbooking, miscommunication about availability, and even financial losses due to missed opportunities.

A robust EasyStay Room Reservation system addresses these challenges by automating and streamlining the booking process. It allows room owners to list their properties, manage reservations, and monitor occupancy levels in real-time, ensuring that customers have access to accurate and up-to-date information.

By integrating features like user-friendly search filters, real-time availability updates, and secure payment processing, EasyStay enhances the overall booking experience for users. Additionally, automated notifications for booking confirmations, cancellations, and reminders improve communication between room owners and customers, reducing misunderstandings and enhancing satisfaction.

The system promotes efficient management by providing an admin panel that offers comprehensive oversight of all listings, bookings, and user interactions. This centralized control enables room owners to make informed decisions quickly, optimizing their occupancy rates and ensuring a smooth operational flow.

Furthermore, with the increasing demand for seamless online experiences, EasyStay Room Reservation stands out by offering scalability and accessibility through a cloud-based platform. This allows multiple stakeholders, including owners and customers, to access the system from various devices, facilitating smooth interactions and data sharing.

In summary, EasyStay Room Reservation not only improves operational efficiency but also enhances user satisfaction by ensuring accurate bookings, effective communication, and a streamlined reservation process. As the hospitality landscape continues to evolve, implementing such a system is vital for any establishment looking to thrive in a com

1.2 Objectives

To provide a reliable system for managing room reservations: The primary objective is to establish a dependable and efficient system that accurately manages room bookings. This system is designed to replace error-prone manual processes with automated solutions, ensuring that all reservations are tracked in real-time. By providing reliable reservation management, EasyStay helps maintain consistent and accurate records of all bookings, facilitating better resource planning and utilization.

To implement an alert system to notify users of booking confirmations and cancellations: Another key objective is to incorporate a robust alert system that proactively notifies users and room owners about booking confirmations, cancellations, and reminders. This notification system is crucial for preventing misunderstandings and ensuring timely communication. By keeping users informed, the alert system helps maintain seamless operations and enhances overall customer satisfaction.

To maintain an organized and easily accessible room database: The system aims to create and maintain a well-organized database of available rooms that is easily accessible to authorized users. This objective focuses on ensuring that all room information is systematically categorized and stored, making it simple for users to locate and retrieve specific listings as needed. An organized database not only improves operational efficiency but also contributes to a better user experience.

To streamline the process of reservation management and record keeping: The system is designed to simplify and optimize the processes involved in managing reservations and maintaining records. This includes automating the entry of new bookings, updating availability in real-time, and maintaining detailed logs of all transactions. By streamlining these processes, **EasyStay** reduces administrative burden and ensures that all reservation records are accurate and up-to-date.

To provide detailed booking logs for better oversight: A critical objective is to offer detailed booking logs that provide comprehensive oversight of room reservations. These logs include information such as the number of rooms booked, the user responsible for the booking, and timestamps for each transaction. By maintaining detailed reservation records, the system enhances accountability, compliance with operational standards, and enables better monitoring and analysis of booking patterns.

1.3 Purpose, Scope

1.3.1 Purpose

The purpose of this project is to develop an efficient and secure EasyStay Room Reservation System to address the challenges of managing room bookings in various settings, including hotels, guesthouses, and rental properties. Many organizations struggle with manual or outdated methods for tracking reservations, which can lead to booking conflicts, customer dissatisfaction, and operational inefficiencies. The project aims to provide an automated solution that enhances user experience, ensures seamless booking management, and improves overall operational efficiency. The EasyStay system will streamline the reservation process, manage user accounts, and facilitate real-time updates while reducing human errors and maximizing resource utilization.

1.3.2 Scope

The project scope includes the design, development, and deployment of a comprehensive Room Reservation System that can track room availability, bookings, user accounts, and payment transactions. It will feature automated notifications for booking confirmations, cancellations, and reminders, ensuring clear communication with users and room owners. The system will cater to various types of accommodation providers, offering customizable features for reservation management, customer engagement, and reporting. Additionally, the project will focus on integrating user-friendly interfaces and responsive design to simplify operations across devices. Testing and validation will be conducted to ensure the system's reliability, scalability, and adherence to industry standards.

System Analysis

2.1 Existing System

In many cases, room reservation management is handled manually or through basic software tools like spreadsheets or paper logs. This leads to several limitations:

- **Manual Data Entry:** Tracking room availability, bookings, and customer details manually is time-consuming and prone to human error. There is no automated process for updating availability when rooms are booked, checked in, or checked out.
- **Lack of Real-Time Data:** The existing system does not provide real-time updates. Room owners must manually check availability, booking statuses, and customer information, leading to potential mismanagement of reservations.
- **No Alerts or Notifications:** There is no automated system in place to alert the owners about new bookings, cancellations, or upcoming check-in/check-out dates. This increases the risk of double bookings or miscommunication with customers.
- **Inefficient Reporting:** The current system does not offer automated reporting, requiring room owners to manually create and update reports on occupancy rates, revenue, and customer feedback.

2.2 Proposed System

The proposed EasyStay Room Reservation system addresses the limitations of the current manual system by automating room booking, improving user experience, and ensuring efficient management with minimal effort.

- **Automated Booking Management:** The system will automate room booking through a user-friendly interface that allows customers to search for available rooms, make reservations, and manage their bookings in real-time. This minimizes human error and ensures accurate records of bookings.
- **Real-Time Availability Monitoring:** The system will provide real-time updates on room availability, ensuring that owners and customers can quickly check the status of rooms. This feature helps avoid double bookings and allows customers to make informed decisions.
- **Automated Notifications:** The proposed system will include an alert system that notifies room owners of new bookings, cancellations, and upcoming check-in/check-out dates. These notifications can be customized to prevent booking conflicts and enhance communication with customers.
- **Comprehensive Reporting:** The EasyStay system will automatically generate reports on occupancy rates, revenue, and customer feedback. These reports will be easily accessible and exportable for further analysis, providing insights into booking trends and business performance.
- **User-Friendly Interface:** The system will be designed with a simple, intuitive interface, making it easy for both room owners and customers to navigate. This ensures that users can manage reservations, monitor notifications, and access information without requiring extensive technical knowledge.

2.3 Requirement Analysis

Functional Requirements:

- **Authentication:** Room owners (admins) can sign up and log in, while customers have a login feature to manage their reservations.
- **Room Management:** Room owners manage room details, including pricing, availability, and amenities, ensuring that all information is current and accurate.
- **Booking System:** Customers can search for available rooms, make reservations, and receive booking confirmations through the system.
- **Payment Processing:** The system supports secure payment options for customers, enabling them to complete their bookings online.
- **Customer Management:** Room owners can access and manage customer information, including contact details and booking history.
- **Review System:** Customers can leave reviews and ratings for their stay, which helps improve service quality.
- **Dashboard:** Room owners can view statistics related to bookings, revenue, and customer feedback in a comprehensive dashboard.

Non-Functional Requirements:

- **Scalability:** The system should efficiently handle an increasing number of users and room listings as the business grows.
- **Performance & Availability:** The system must ensure fast, real-time updates and 24/7 accessibility for all users.
- **Security:** Protect customer and room owner data with encryption and access control measures to maintain privacy and security.
- **Usability:** A user-friendly interface is essential for both room owners and customers, ensuring ease of navigation and interaction with the system.

2.4 Software Requirements

Front-End Development	<ul style="list-style-type: none">• HTML• CSS• Bootstrap
Back-End Development	<ul style="list-style-type: none">• JavaScript• Node.js
Database	<ul style="list-style-type: none">• MongoDB
Operating system	<ul style="list-style-type: none">• Windows 11

2.5 Hardware Requirements

Processor:	Intel core i3/Ryzen 3
Processor Speed:	1 GHz to 3.8 GHz
RAM:	8 GB to 16 GB
SSD:	128 GB to 512

2.1 Justification of Selection of Technology

1. **Scalability:** Technologies like Node.js can handle multiple requests efficiently, making it suitable for growing user demand.
2. **Development Speed:** The use of EJS for server-side templating allows for rapid development and dynamic content rendering, while MongoDB provides flexibility in managing diverse data structures, streamlining the overall development process.
3. **Performance:** Lightweight server-side technologies enhance response times, while caching solutions like Redis improve overall performance.
4. **Community Support:** Popular frameworks provide extensive resources and community support, facilitating problem-solving.
5. **Integration:** The selected technologies can easily integrate with third-party services (e.g., payment gateways, maps).
6. **User Experience:** Modern front-end tools, such as Bootstrap, contribute to a responsive and intuitive user interface, improving accessibility and engagement for both room owners and customers.
7. **Maintainability:** Well-documented technologies ensure easier long-term maintenance and provide a smoother onboarding process for new developers, allowing for continued growth and updates.
8. **Cost-Effectiveness:** Open-source options help minimize costs related to licensing and hosting.

System Design

3.1 Module Division

1. User Authentication Module

- **Purpose:** To manage user access and ensure secure entry into the system.
- **Features:**
 - **Login:** Allows users (room owners and customers) to securely log in using their credentials (email and password).
 - **Logout:** Enables users to securely exit their sessions.
 - **Registration:** Facilitates the registration of new users (both owners and customers) by capturing necessary details.
 - **Password Management:** Allows users to reset their passwords or retrieve forgotten credentials.

2. Room Management Module

- **Purpose:** To handle all aspects related to room listings, from creation to updates.
- **Features:**
 - **Add Room:** Enables owners to input room details, including location, price, availability, and amenities.
 - **Update Room Details:** Allows owners to update existing room information as needed (e.g., price changes, availability).
 - **Delete Room:** Provides the option for owners to remove rooms from the listing.

3. Booking Management Module

- **Purpose:** To manage customer bookings and reservations.
- **Features:**
 - **Create Booking:** Allows customers to book available rooms by providing necessary details (dates, number of guests).
 - **View Booking Details:** Enables customers to view their booking history and details.
 - **Cancel Booking:** Allows customers to cancel their reservations if needed.

4. Customer Management Module

- **Purpose:** To manage customer-related data and interactions.
- **Features:**
 - **View Customer Profiles:** Allows owners to view detailed information about customers, including contact information and booking history.
 - **Manage Customer Information:** Facilitates updating and maintaining customer records for accuracy and relevance.

5. Review and Rating Module

- **Purpose:** To gather feedback and ratings from customers about their stay.
- **Features:**
 - **Submit Review:** Allows customers to submit reviews and ratings for the rooms they have booked.
 - **View Reviews:** Enables owners to view feedback from customers to improve services.

3.2 Data Dictionary

Relationships

- **Users to Bookings:** One-to-Many (One user can make multiple bookings)
- **Room to Bookings:** One-to-Many (One listing can have multiple bookings).
- **Room to Reviews:** One-to-Many (One listing can have multiple reviews).
- **Users to Reviews:** One-to-Many (One user can leave multiple reviews).

1. Users Table

Field Name	Data Type	Key Type	Description
_id	ObjectId	PRI	Unique identifier for each user
Email	String	UNI	User's email
UserName	String		User's username username (unique)
Password	String		Hashed password for user authentication

2. Reviews Table

Field Name	Data Type	Key Type	Description
_id	ObjectId	PRI	Unique identifier for each review
comment	String		Review comment
rating	Number		Rating given by the user (1-5)
createdAt	Date		Reference to the User who wrote the review
author	ObjectId		Reference to the User who wrote the review

3. Room Table

Field Name	Data Type	Key Type	Description
_id	ObjectId	PRI	Unique identifier for each room
Title	String		Title of the room
Description	String		Description of the room
Image	Array		Array of objects containing URL and filename
Price	Number		Price per
Location	String		Location of the room
Country	String		Country of the room
Owner	String		Reference to the User (owner of the listing)
Review_id	String		Reference to the User who wrote the review

4. Booking Table

Field Name	Data Type	Key Type	Description
_id	ObjectId	PRI	Unique identifier for each booking
Room_id	ObjectId		Reference to the room being booked
Customer_id	ObjectId		Reference to the customer making the booking
dateFrom	Date		Start date of the booking
dateTo	Date		End date of the booking
bookingFacilities	String		Facilities included in the booking

3.3 Entity Relationship Diagram

- **Introduction:** Peter Chen developed the ER diagram in 1976 to represent the structure and logic of databases in a simple and understandable way. The ER model has evolved into variations such as the Enhanced ER Model and the Object Relationship Model.
- **Entity-Relational Model:** This model identifies the entities in the database and their relationships. It provides a graphical representation of the database's structure, showcasing how real-world objects like rooms, customers, and bookings are related.

Entities and Relationships

1. Room:

- Manages → Owner (The owner manages room listings.)
- Has → Review (Rooms can have reviews from customers.)

2. Booking:

- Linked To → Room (A booking is associated with a specific room.)
- Linked To → Customer (A booking is made by a customer.)

3. Customer:

- Requests → Booking (A customer makes a booking request for a room.)
- Leaves → Review (A customer can leave a review for a room.)

4. Owner:

- Manages → Room (An owner can manage multiple rooms.)
- Receives → Booking (The owner receives bookings for their rooms.)

5. Review:

- Given By → Customer (A customer writes a review for a room.)
- For → Room (A review is linked to a specific room.)

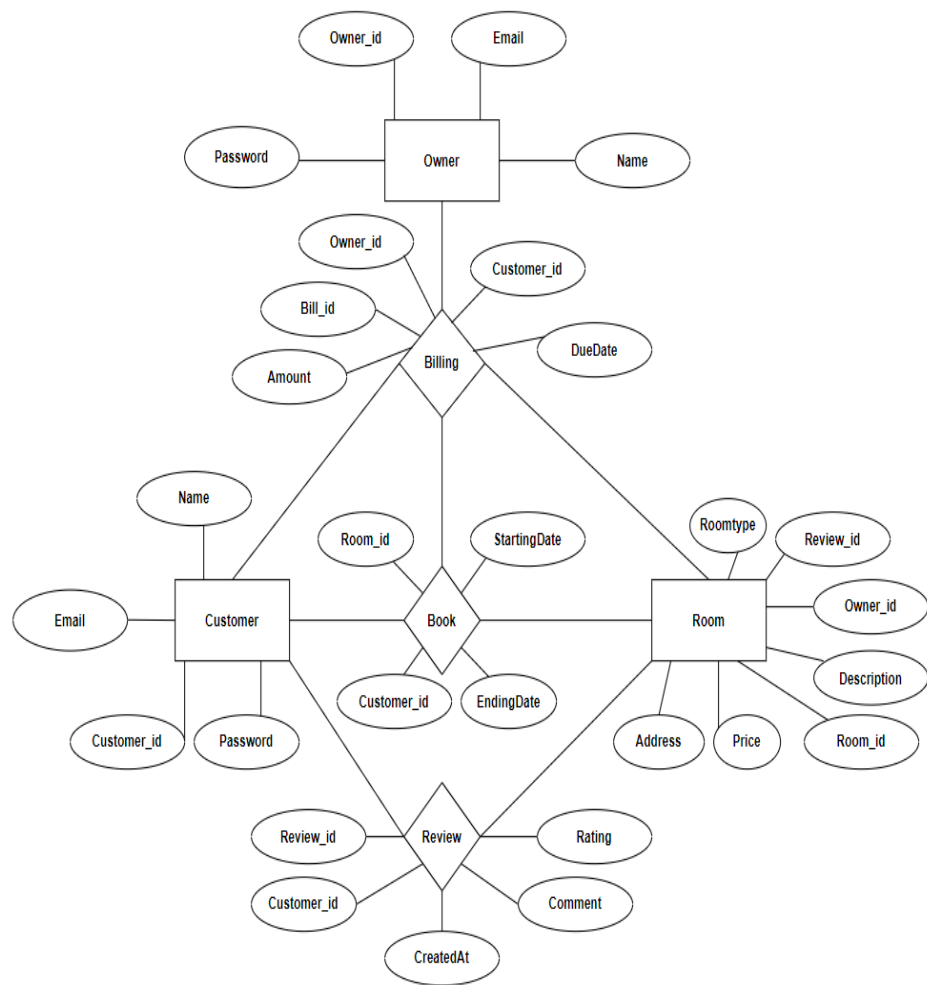


Fig 3.3 Entity Relation diagram

3.4 Data flow Diagram

Definition: A Data Flow Diagram (DFD) illustrates how data moves through a system, showing processes, data stores, and external entities.

Level 0 DFD (Context Diagram)

The Level 0 DFD, also known as the context diagram, offers a high-level overview of the entire system. It represents the system as a single process (a bubble) and illustrates its interactions with external entities, such as users or other systems. This level does not delve into internal processes but focuses on the inputs and outputs, showing how data enters and exits the system through arrows.



Fig 3.4.1 Level 0 DFD

Level 1 DFD

The Level 1 DFD breaks down the single process from the Level 0 DFD into multiple subprocesses. This level provides more detail by illustrating how the main functions of the system are executed and how data flows between these sub-processes. Each sub-process is represented as a separate bubble, and the associated data flows and data stores are also depicted.

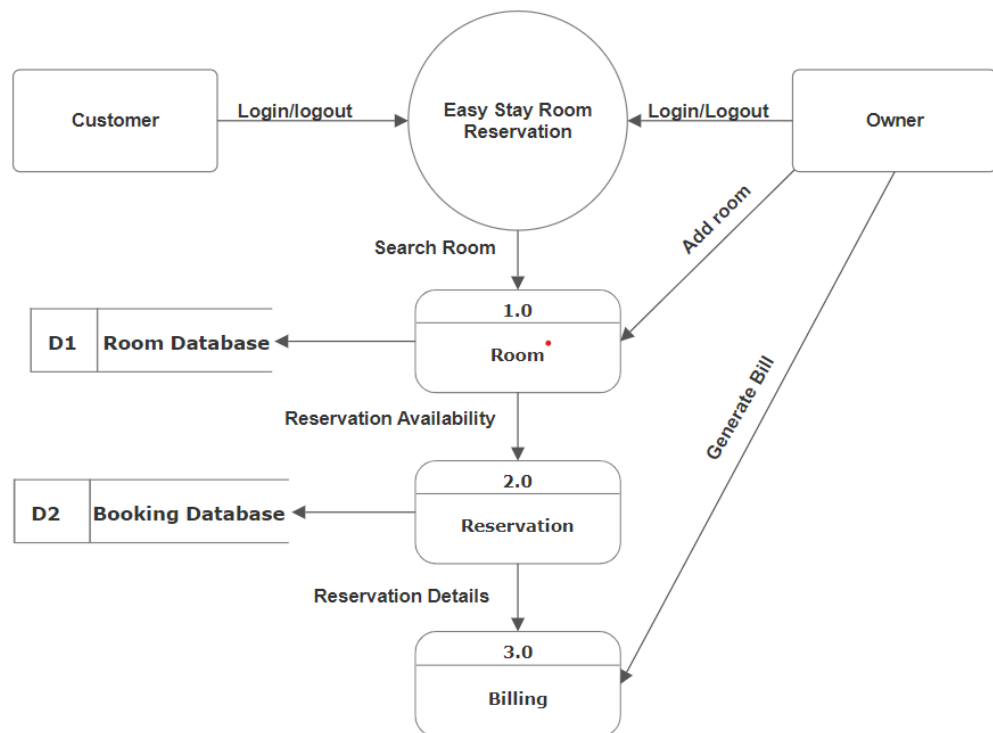


Fig 3.4.2 Level 1 DFD

Level 2 DFD

The Level 2 DFD further decomposes the sub-processes identified in Level 1 into even more detailed sub-processes. This level is used to provide a granular view of the system's operations, detailing how each sub-process interacts with data stores and external entities. The data flows are also elaborated, showing the specifics of data movement within the system.

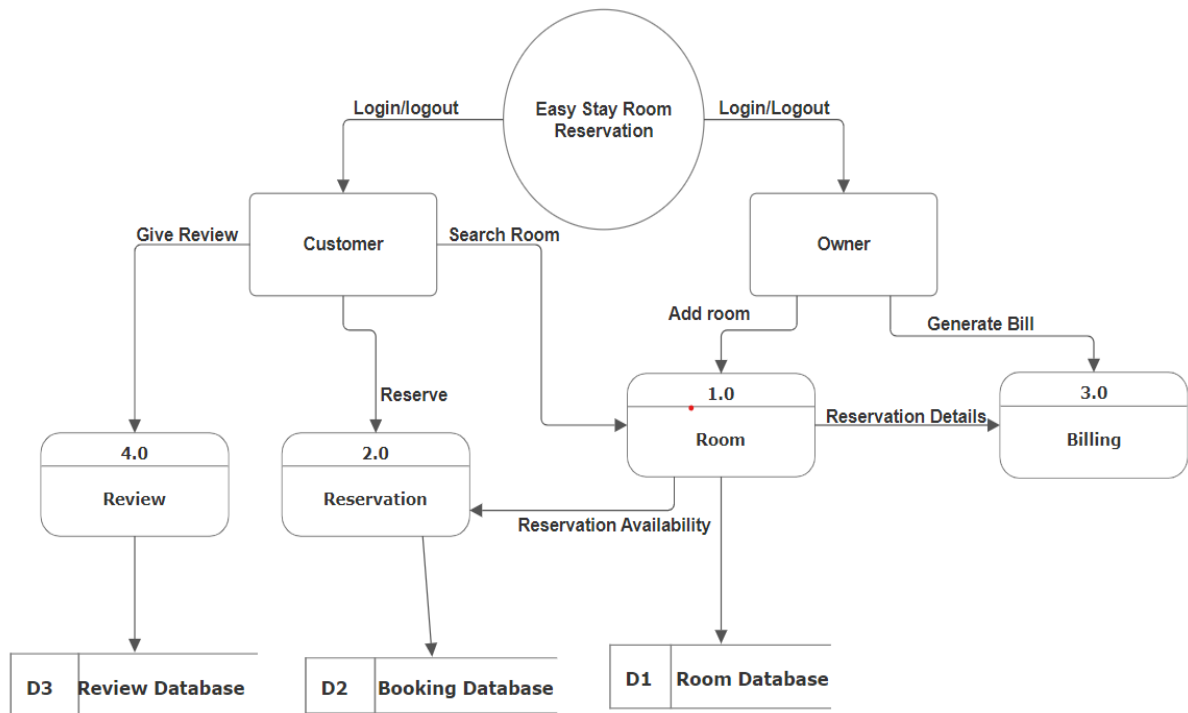


Fig 3.4.3 Level 2 DFD

3.5 Use Case Diagram

Definition: A Use Case Diagram represents the interactions between users (actors) and the system, highlighting the system's functionalities.

Usage in the Project: The Use Case Diagram outlines the primary functionalities of the system, ensuring that all user needs are addressed during development. It serves as a communication tool among stakeholders.

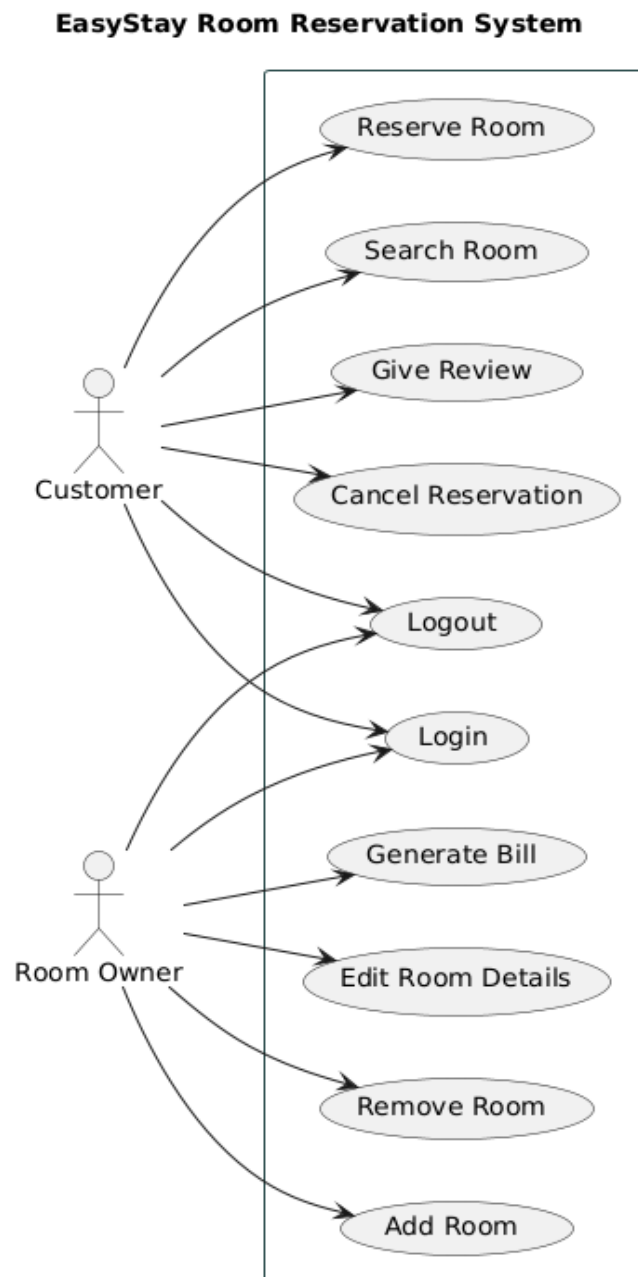


Fig 3.5 Use Case Diagram

3.6 Deployment Diagram

Definition: A Deployment Diagram models the physical deployment of artifacts on nodes in a system, showing how software components are distributed across hardware.

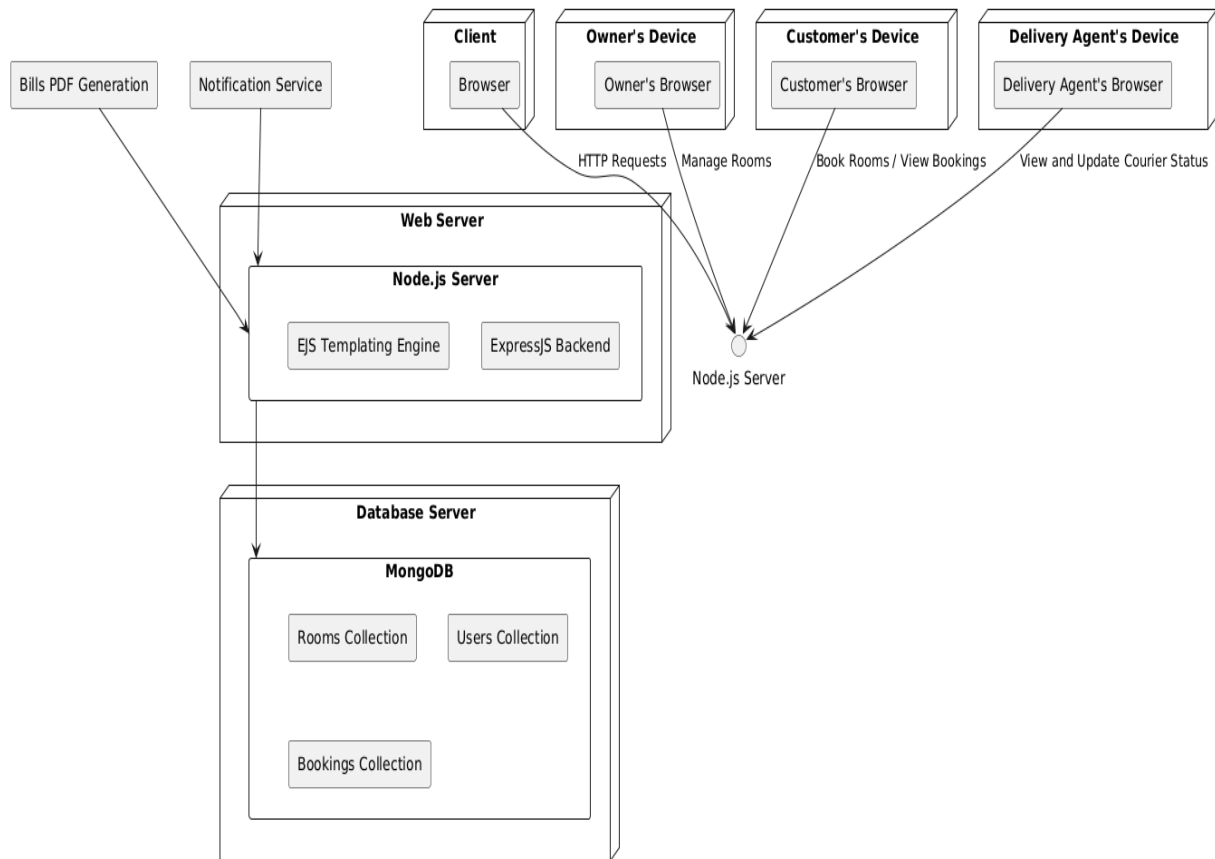


Fig 3.6 Deployment Diagram

3.7 Class Diagram

Class diagrams are a type of UML (Unified Modeling Language) diagram used in software engineering to visually represent the structure and relationships of classes within a system i.e. used to construct and visualize object-oriented systems.

In these diagrams, classes are depicted as boxes, each containing three compartments for the class name, attributes, and methods. Lines connecting classes illustrate associations, showing relationships such as one-to-one or one-to-many.

Class diagrams provide a high-level overview of a system's design, helping to communicate and document the structure of the software. They are a fundamental tool in object-oriented design and play a crucial role in the software development lifecycle.

Class Notation

1. Class Name:

The name of the class is typically written in the top compartment of the class box and is centered and bold.

2. Attributes:

Attributes, also known as properties or fields, represent the data members of the class. They are listed in the second compartment of the class box and often include the visibility (e.g., public, private) and the data type of each attribute.

3. Methods:

Methods, also known as functions or operations, represent the behavior or functionality of the class. They are listed in the third compartment of the class box and include the visibility (e.g., public, private), return type, and parameters of each method.

4. Visibility Notation:

Visibility notations indicate the access level of attributes and methods. Common visibility notations.

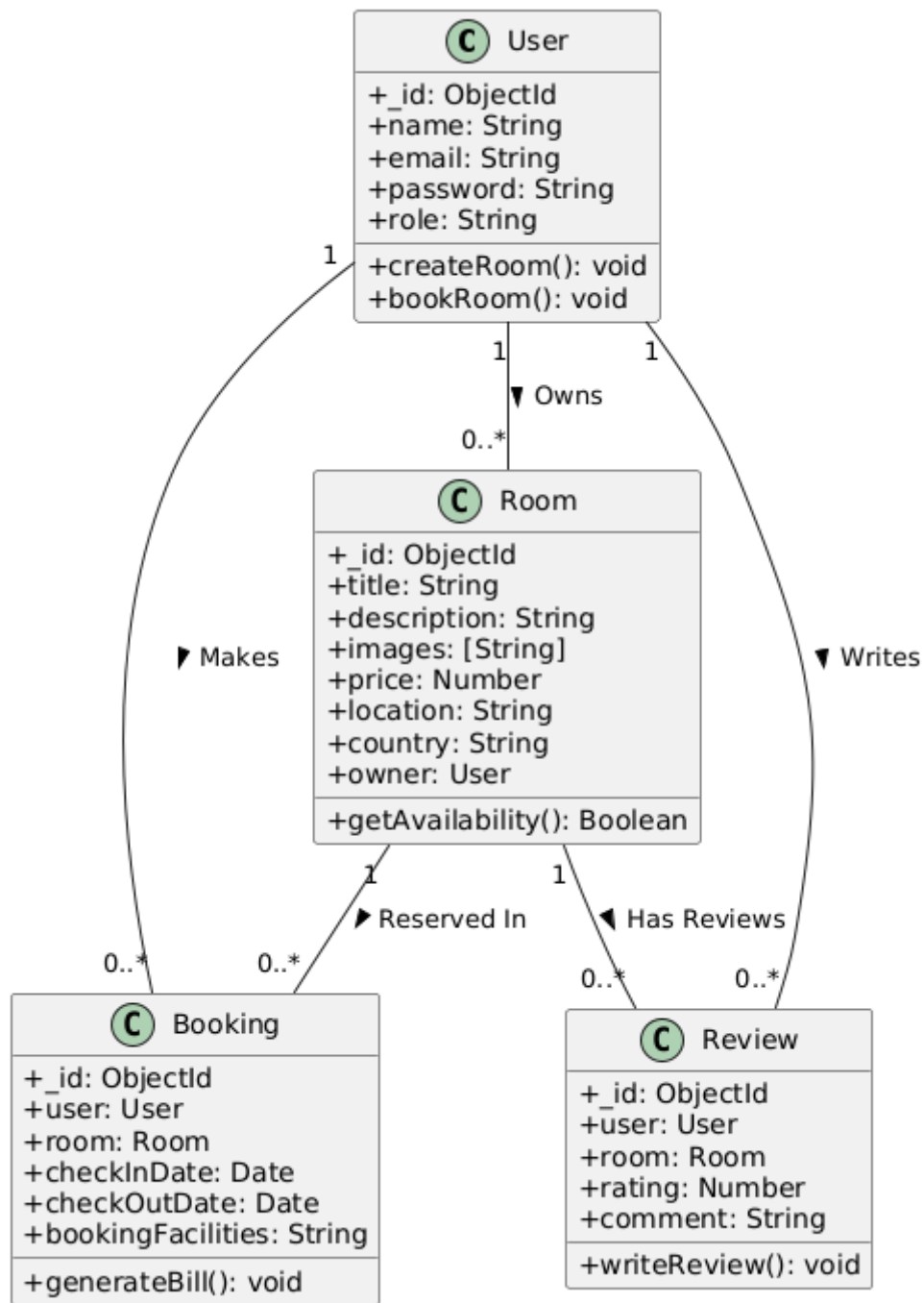


Fig 3.7 Class Diagram

3.8 Component Diagram

Definition: A Component Diagram provides a high-level view of the system's components and their relationship.

Attributes:

Components: Represented as rectangles, these are modular part of system (e.g., User Interface, Business Logic)

Dependencies: Dashed line indicates dependencies between components

Usage in the Project : The Component Diagram illustrates the various software components and their interactions, helping in understanding the system's modularity and facilitating easier maintenance and updates.

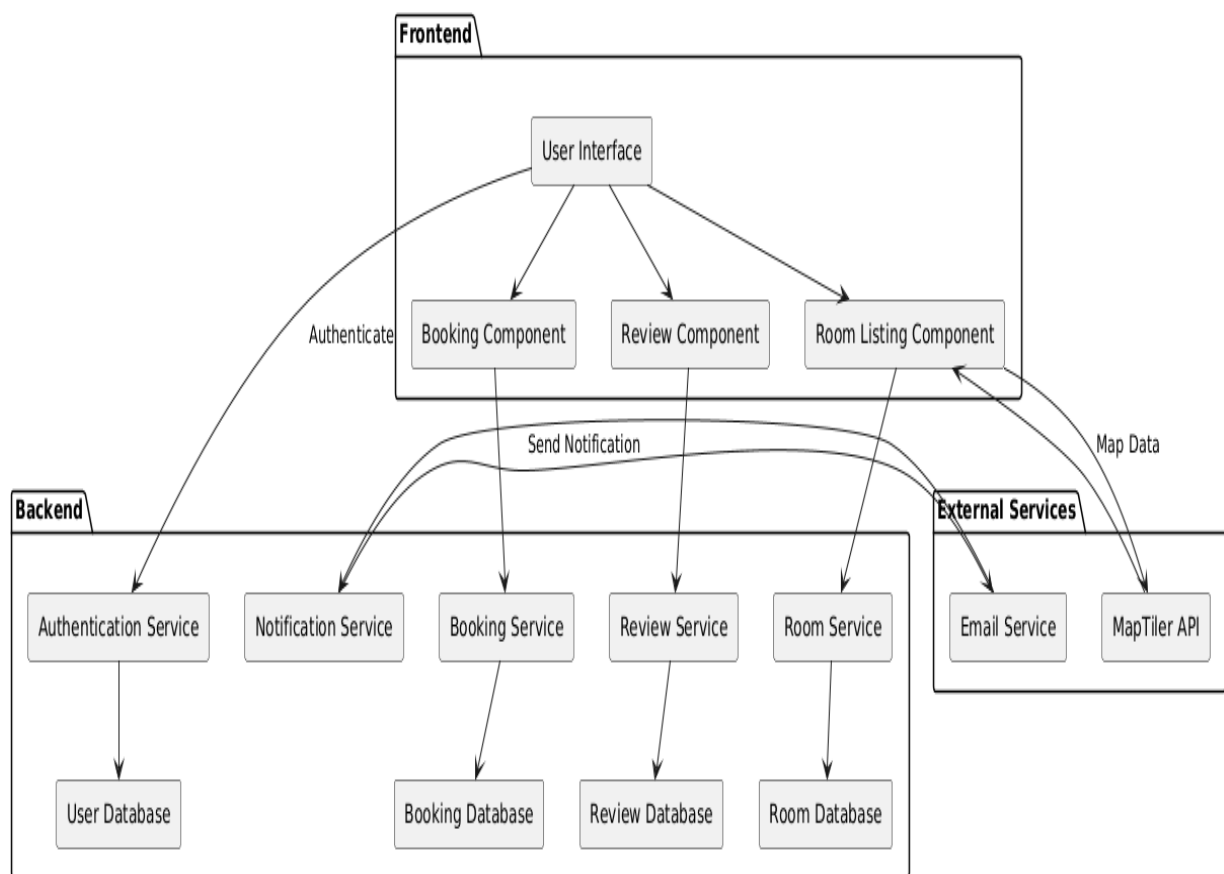


Fig 3.8 Component Diagram

3.9 Activity Diagram

Activity diagrams are behavioral diagrams in UML that model the flow of activities in a system. They are similar to flowcharts but with more advanced features. Key point about

Activity Diagrams:

- Show the flow of control from one activity to another
- Model sequential, branched and concurrent activities
- Use control nodes like forks, joins, decisions and merges to control the flow

Some common components of Activity Diagrams:

- Activities - Represent tasks or actions
- Control Flow - Shows the order of execution
- Object Flow - Shows flow of data/objects between activities
- Initial Node - Start of the activity flow
- Final Node - End of the activity flow
- Decision Node - Represents a conditional branch
- Fork Node - Splits a flow into multiple concurrent flows
- Join Node - Synchronizes multiple flows into one

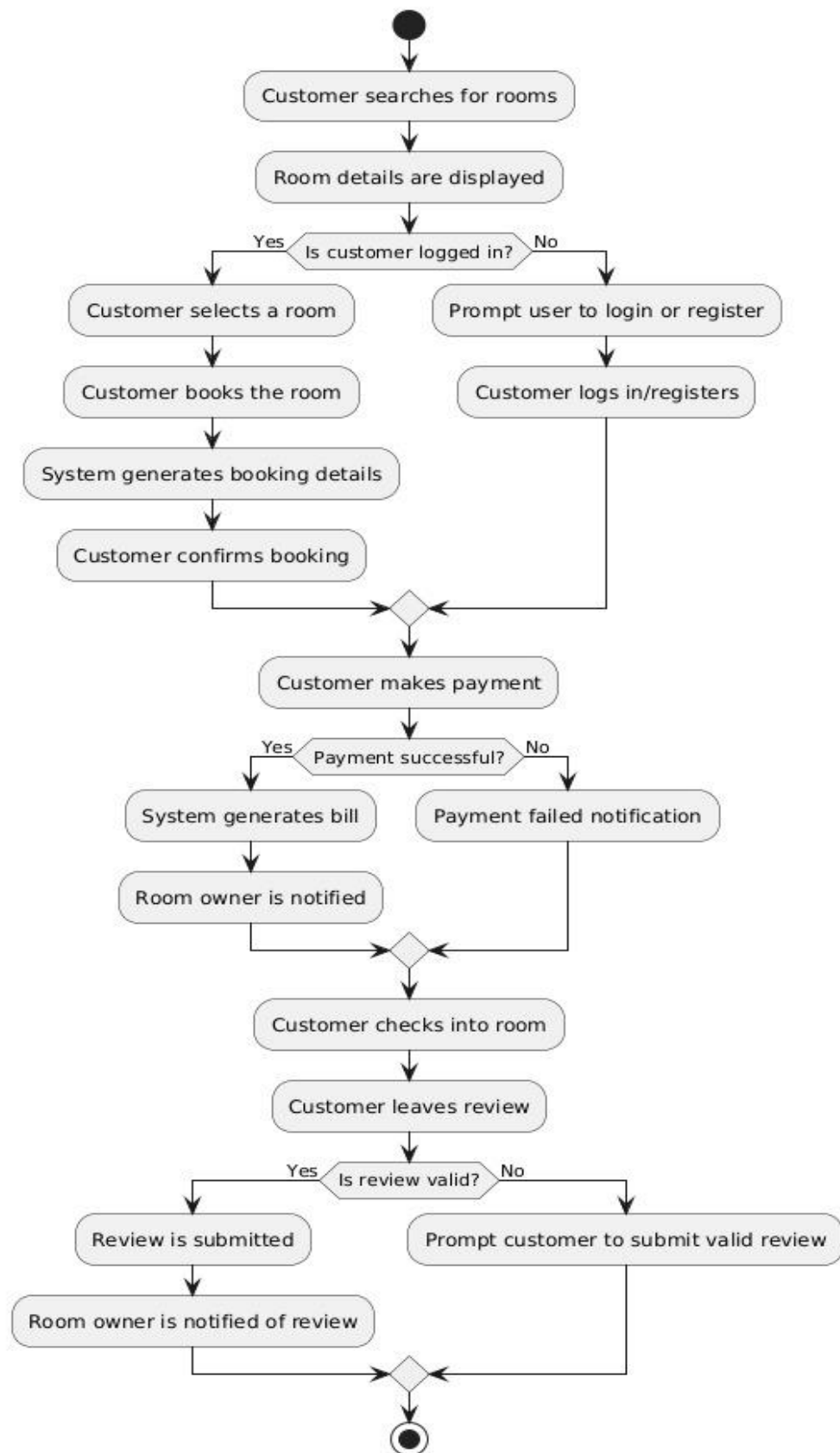


Fig 3.9 Activity Diagram

3.10 Object Diagram

An Object Diagram can be referred to as a screenshot of the instances in a system and the relationship that exists between them.

- An object diagram in UML is useful because it provides a clear and visual representation of specific instances of classes and their relationships at a particular point in time, aiding in understanding and communicating the structure and interactions within a system.
- In other words, “An object diagram in the Unified Modeling Language (UML), is a diagram that shows a complete or partial view of the structure of a modelled system at a specific time.

Object diagrams in UML are depicted using a simple and intuitive notations to show a snapshot of a system at a specific point in time, displaying instances of classes and their relationships.

What is an Object?

An object refers to a specific instance of a class within a system. A class is a blueprint or template that defines the common attributes and behaviors shared by a group of objects. An object, on the other hand, is a concrete and individual occurrence of that class, possessing unique values for its attributes.

What is a Classifier?

In UML a classifier refers to a group of elements that have some common features like methods, attributes and operations. A classifier can be thought of as an abstract metaclass which draws a boundary for a group of instances having common static and dynamic features.

For example:

we refer a class, an object, a component, or a deployment node as classifiers in UML since they define a common set of properties. We are able to design object diagrams by instantiating classifiers.

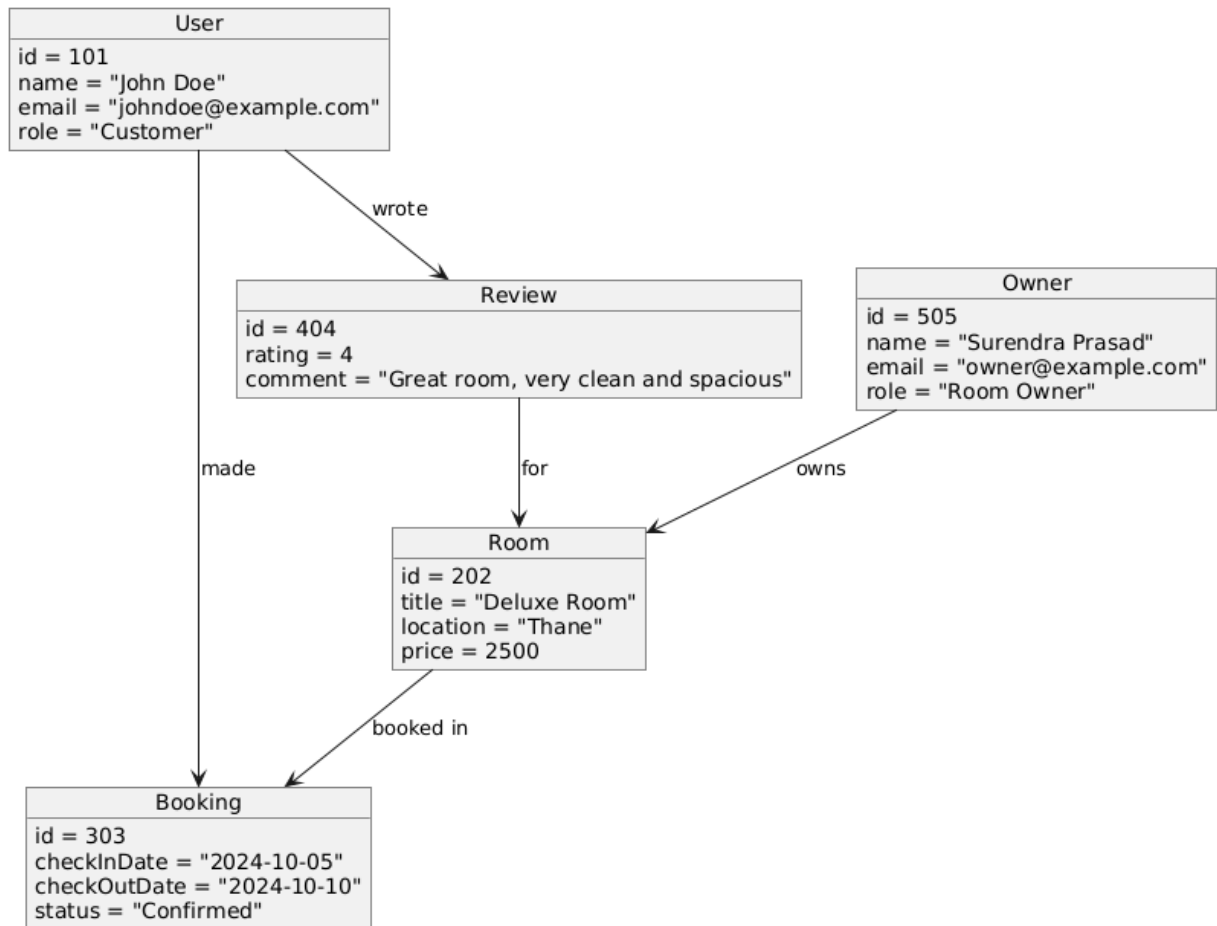


Fig 3.10 Object Diagram

3.11 State Chart Diagram

A state chart diagram (also called a state machine diagram) is a type of behavioral diagram in UML that models the dynamic behavior of a system. It shows the different states an object can be in and the transitions between those states. The key elements of a state chart diagram are:

- **States:** Represent the condition of an object at a particular time. States are depicted as rounded rectangles.
- **Transitions:** Represent the movement from one state to another. Transitions are shown as arrows labeled with the triggering event.
- **Events:** Occurrences that can trigger a state change. Events label the transitions.
- **Initial state:** The default starting state, shown as a solid circle.
- **Final state:** The end state, depicted as a circle surrounding a solid circle.

Purpose of State Chart Diagrams

The main purposes of state chart diagrams are:

- Model the dynamic behavior of an object in response to events
- Specify the lifecycle of an object from creation to termination
- Describe the event-ordered behavior of an object
- Model reactive systems that respond to events
- Provide a visual representation of the states and transitions

State chart diagrams are especially useful for modeling the behavior of an interface, class, or collaboration. They emphasize the event-ordered behavior of an object, which is particularly useful for modeling reactive systems.

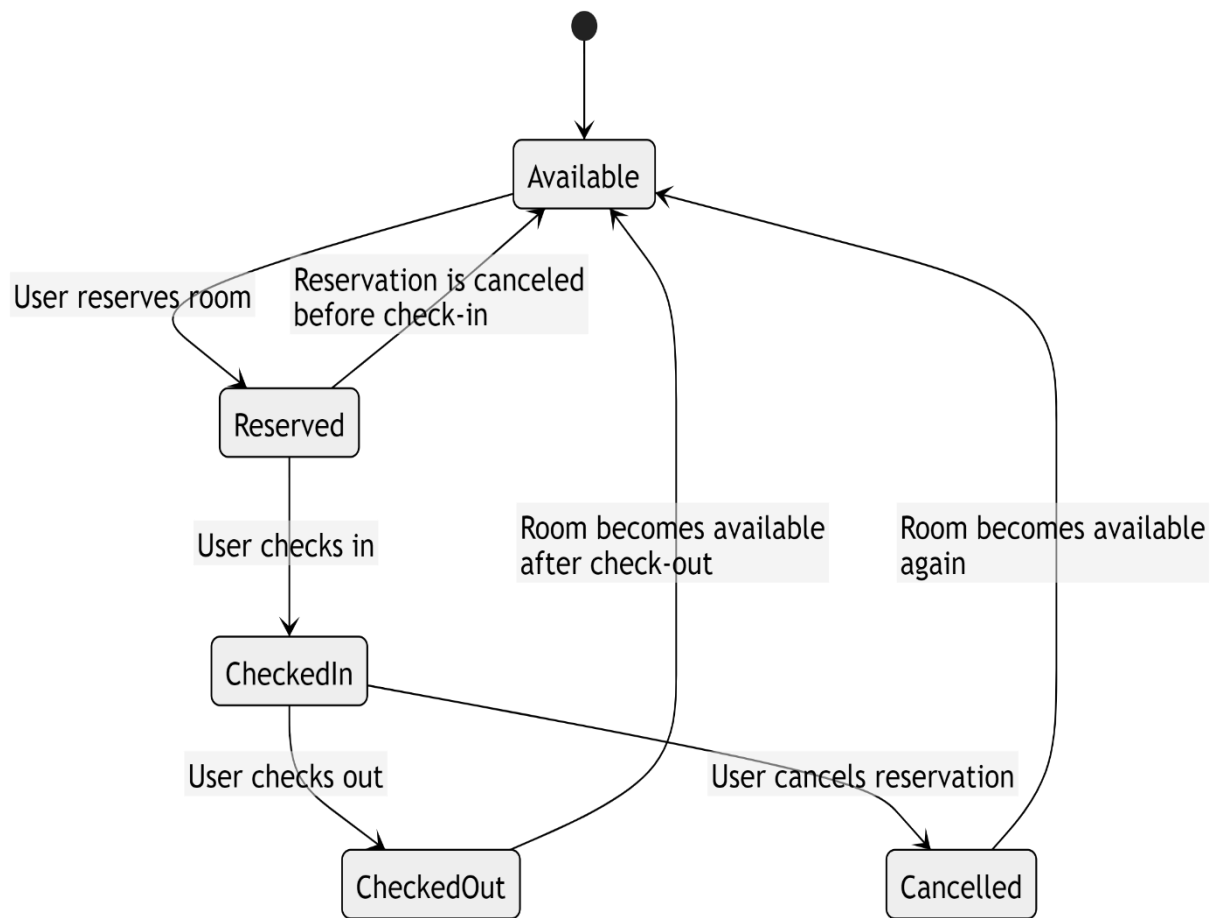


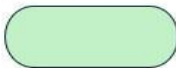
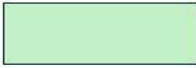

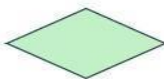

Fig 3.11 State Chart Diagram

3.12 Flow Chart Diagram

A flowchart is a visual representation of a process or workflow that outlines the steps involved in completing a task. It uses standardized symbols to depict different types of actions, decisions, and the flow of information or materials through the process. Flowcharts are widely utilized across various fields, including business, engineering, education, and software development, to facilitate understanding, communication, and analysis of processes.

Flow Direction: Flowcharts typically progress from top to bottom and left to right, guiding the viewer through the steps in a logical sequence.

- **Oval (Terminator):** Represents the start and end points of a process. It typically contains the words "Start" or "End."
- **Rectangle (Process):** Indicates a step in the process where an action or operation occurs (e.g., a task or activity).
- **Diamond (Decision):** Represents a decision point in the process, where a question is posed, leading to different paths based on the answer (e.g., Yes/No).
- **Parallelogram (Input/Output):** Used to show input to or output from a process (e.g., data entry, results display).
- **Arrow (Flow Line):** Indicates the direction of the flow of the process, connecting different steps and showing the sequence of operations.

Symbol	Name	Function
	Oval	Represents the start or end of a process
	Rectangle	Denotes a process or operation step
	Arrow	Indicates the flow between steps
	Diamond	Signifies a point requiring a yes/no
	Parallelogram	Used for input or output operations

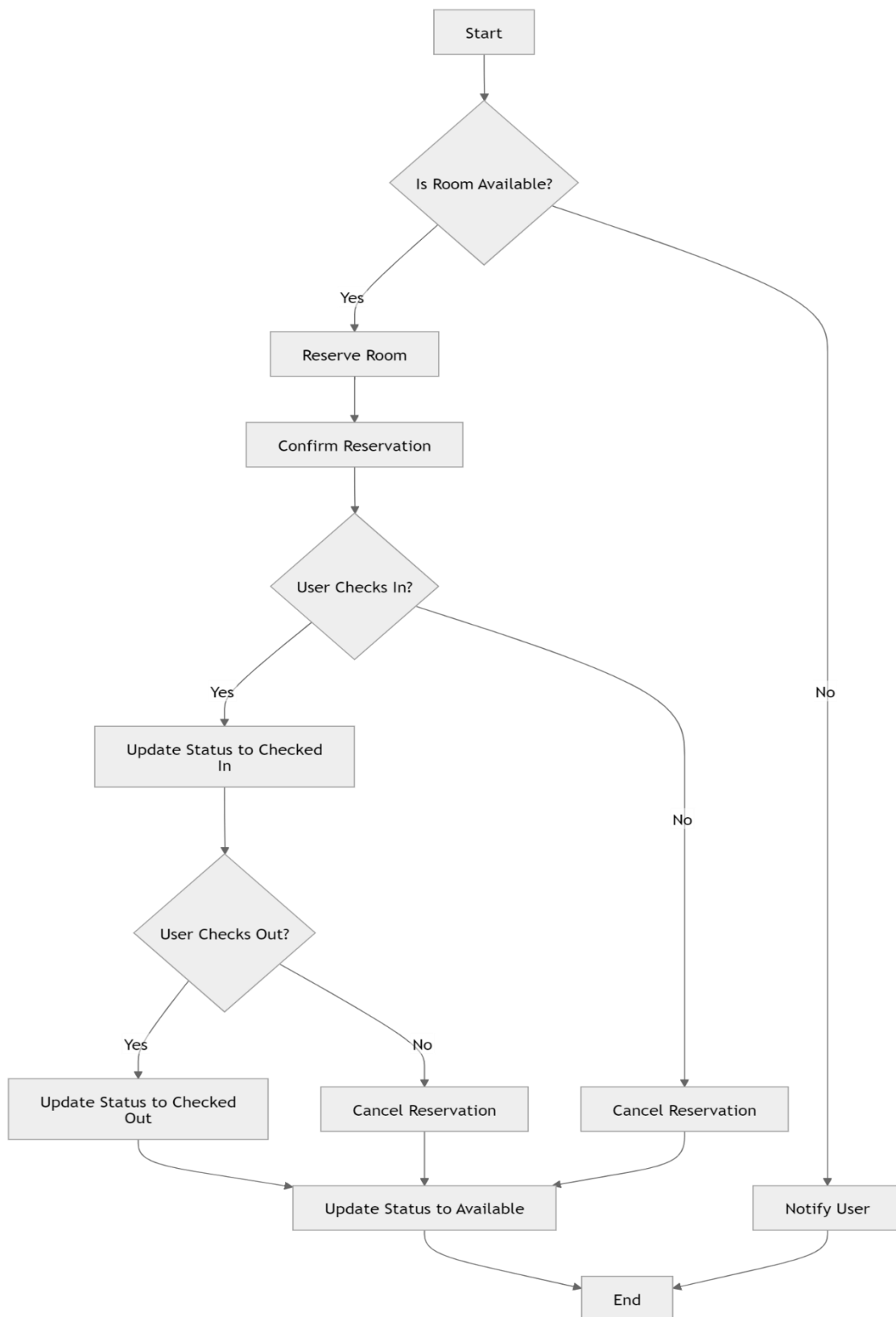


Fig 3.12 Flow Chart Diagram

3.13 Gantt Chart

A **Gantt chart** is a type of bar chart that visually represents a project schedule over time. It shows the start and finish dates of tasks, allowing project managers to plan, track progress, and manage resources. Each task is represented by a horizontal bar, with its length indicating the task's duration.

1. **Requirements Analysis (5 days):** The initial phase where project needs and specifications are analyzed.
2. **Frontend Module Development (10 days):** Development of the user-facing part of the application (e.g., web pages).
3. **Backend Module Integration (10 days):** Development and integration of the server-side components, which handle business logic.
4. **Database Setup (7 days):** Configuration and preparation of the database for storing application data.
5. **API Development (8 days):** Creation of APIs to allow communication between the frontend, backend, and database.
6. **User Interface Refinement (8 days):** Improving and polishing the UI for a better user experience.
7. **System Integration (6 days):** Bringing together all components (frontend, backend, database) for the app to function as a whole.
8. **Testing Phase (5 days):** Testing the application to ensure it works as intended and fixing any bugs.
9. **Final Deployment (5 days):** Launching the app to a live environment after successful testing.

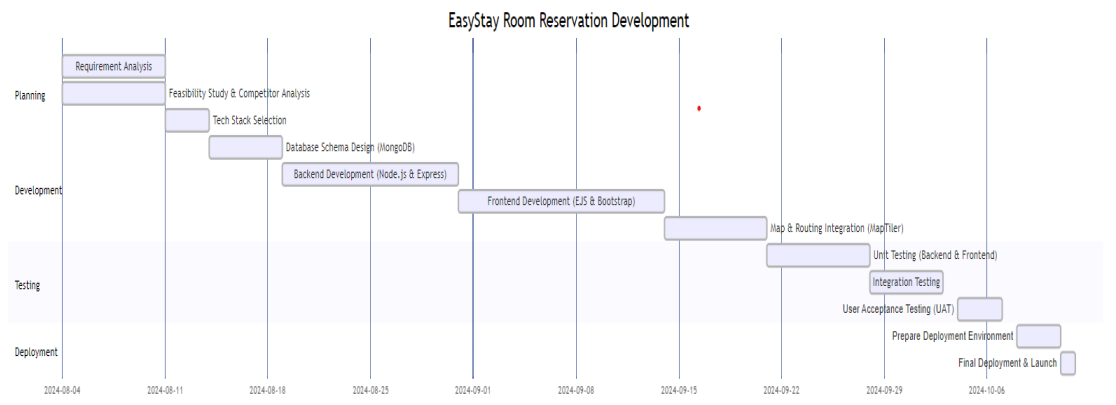


Fig 3.13 Gantt Chart

IMPLEMENTATION AND TESTING

4.1 Code

Frontend Codes

1. boilerPlate.ejs

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>EasyStay</title>

  <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css">

  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" integrity="sha512-SnH5WK+bZxgPHs44uWIX+LLJAJ9/2PkPKZ5QiAj6Ta86w+fsb2TkemfRyvX3pBnMFCv7oQPJkl9QevSCWr3W6A==" crossorigin="anonymous" referrerpolicy="no-referrer" />

  <link rel="stylesheet" href="/css/style.css">

  <link rel="stylesheet" href="/css/rating.css">

  <link rel="stylesheet" href="/css/media.css">

<script src="https://cdn.maptiler.com/maptiler-sdk-js/v2.0.3/maptiler-sdk.umd.js"></script>

  <link href="https://cdn.maptiler.com/maptiler-sdk-js/v2.0.3/maptiler-sdk.css" rel="stylesheet" />

  <link rel="shortcut icon"
href="https://res.cloudinary.com/dkiqoznwu/image/upload/v1722006346/wanderlust_DEV/lkawergj8ejmn1ooxcru.png" type="image/x-icon">

</head>

<body>

  <%- include('../includes/navbar.ejs') %>

  <div class="container" >
```

```

    <%- include('../includes/flash.ejs') %>

    <%- body %>

</div>

<%- include('../includes/footer.ejs') %>

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></script>
    <script src='https://kit.fontawesome.com/a076d05399.js' crossorigin='anonymous'></script>
    <script src="/js/script.js"></script>
    <script src="/js/password.js"></script>
</body>
</html>

```

2. index.ejs

```

<% layout("layouts/boilerPlate") %>

<div id="filters">
    <div class="filter">
        <div> <i class="fa-solid fa-fire"></i></div>
        <p>Trending</p>
    </div>
    <div class="filter">
        <a href="/listings/special/Rooms">
            <div> <i class="fa-solid fa-bed"></i></div>
            <p>Rooms</p></a>
        </div>
    <div class="filter">
        <a href="/listings/special/IconicCities">
            <div> <i class="fa-solid fa-mountain-city"></i></div>
            <p>Iconic Cities</p></a>
        </div>
    <div class="filter">
        <a href="/listings/special/Mountains">
            <div> <i class="fa-solid fa-mountain"></i></div>

```

```

    <p>Mountains</p></a>
</div>
<div class="filter">
    <a href="/listings/special/Castles">
        <div> <i class="fa-brands fa-fort-awesome"></i></div>
        <p>Castles</p></a>
    </div>
<div class="filter">
    <a href="/listings/special/Camping">
        <div> <i class="fa-solid fa-campground"></i></div>
        <p>Camping</p></a>
    </div>
<div class="filter">
    <a href="/listings/special/Farms">
        <div> <i class="fa-solid fa-cow"></i></div>
        <p>Farms</p></a>
    </div>
<div class="filter">
    <a href="/listings/special/Arctic">
        <div><i class="fa-solid fa-snowflake"></i></div>
        <p>Arctic</p></a>
    </div>
<div class="filter">
    <a href="/listings/special/Domes">
        <div><i class="fa-solid fa-igloo"></i></div>
        <p>Domes</p></a>
    </div>
<div class="filter">
    <a href="/listings/special/Boats">
        <div><i class="fa-solid fa-ship"></i></div>
        <p>Boats</p></a>
    </div>
<div class="filter">
    <a href="/listings/special/Pools">
        <div> <i class="fa-solid fa-person-swimming"></i></div>
        <p>Pools</p></a>
    </div>
<div class="tax-toggle">
    <div class="form-check-reverse form-switch">
        <input class="form-check-input" type="checkbox" role="switch"
id="flexSwitchCheckDefault">

```

```

        <label class="form-check-label" for="flexSwitchCheckDefault">Display total
after taxes</label>
    </div>
</div>
<div class="row row-cols-lg-3 row-cols-md-2 row-cols-sm-1 mt-3">
    <% for (let listing of allListings) { %>
        <a href="/listings/<%= listing._id %>" class="listing-link ">
            <div class="card col custom-card mt-3 " style="border-radius: 10%
!important;">
                <div id="carousel<%= listing._id %>" class="carousel slide ">
                    <div class="carousel-inner card-img-top" >
                        <% listing.image.forEach((image, index) => { %>
                            <div class="carousel-item <%= index === 0 ? 'active' : " %>">
                                
                                </div>
                            <% } %>
                        </div>
                    <button class="carousel-control-prev" type="button" data-bs-
target="#carousel<%= listing._id %>" data-bs-slide="prev">
                        <span class="carousel-control-prev-icon" aria-hidden="true"></span>
                        <span class="visually-hidden">Previous</span>
                    </button>
                    <button class="carousel-control-next" type="button" data-bs-
target="#carousel<%= listing._id %>" data-bs-slide="next">
                        <span class="carousel-control-next-icon" aria-hidden="true"></span>
                        <span class="visually-hidden">Next</span>
                    </button>
                </div>

                <div class="card-img-overlay"> </div>

                <div class="card-body">
                    <p class="card-text"><b><%= listing.title %></b><br>
                        &#8377;<%= listing.price.toLocaleString('en-IN') %> /night
                        <i class="tax-info">&nbsp;+18% GST</i>
                    </p>
                </div>
            </div>
        </a>
    <% } %>

```

</div>

3. Show.ejs

<% layout("layouts/boilerPlate") %>

<script>

let mapToken = "<%= process.env.MAP_TOKEN %>"

let mapLocation = "<%= listing.country %> <%= listing.location %> "

let mapName = "<%= listing.title %>"

</script>

<div class="row mt-3">

<div class="col-8 offset-3" >

<h3><%= listing.title %></h3>

</div>

<div class="card custom-card col-6 offset-3 show-card">

<div id="carousel<%= listing._id %>" class="carousel slide" data-bs-ride="carousel">

<div class="carousel-inner card-img-top">

<% listing.image.forEach((image, index) => { %>

<div class="carousel-item <%= index === 0 ? 'active' : " %>">

</div>

<% }) %>

</div>

<button class="carousel-control-prev" type="button" data-bs-target="#carousel<%= listing._id %>" data-bs-slide="prev">

Previous

</button>

```

        <button class="carousel-control-next" type="button" data-bs-target="#carousel<%=
listing._id %>" data-bs-slide="next">

        <span class="carousel-control-next-icon" aria-hidden="true"></span>

        <span class="visually-hidden">Next</span>

    </button>

</div>

<div class="card-body mt-3">

    <% if (currUser && currUser._id.equals(listing.owner._id) && listing.bookings &&
listing.bookings.length) { %>

        <p class="card-text nav-link"><b><i><a href="/booking/customerDetails/<%=
listing._id %>" class="ancor-tag">Bookings Details</a></i></b></p>

        <% } %>

        <p class="card-text"><i>Owned by <%= listing.owner.username %></i></p>

        <p class="card-text"><i>Special there <%= listing.option %></i></p>

        <p class="card-text"><%= listing.description %></p>

        <p class="card-text">&#8377;<%= listing.price.toLocaleString('en-IN') %> /night</p>

        <p class="card-text">Location: <%= listing.country %>, <%= listing.location %></p>

    </div>

</div>

<% if (currUser && listing.owner && currUser._id.equals(listing.owner._id)) { %>

<div class="btns mt-3">

    <a href="/listings/<%= listing._id %>/edit" class="btn    offset-3" style="background-color:
#73AB95;">Edit <i class="fa-solid fa-pen-to-square edit-icon"></i> </a>

    <form action="/listings/<%= listing._id %>?_method=Delete" method="post">

        <button class="btn offset-5" style="background-color: #BDC3C7;"><i class="fa-solid fa-
trash-can "></i></button>

    </form>

```

</div>

```
<% } else if (currUser && listing.bookings && listing.bookings.some(booking =>
booking.user && currUser._id.equals(booking.user._id))) { %>
```

```
<div class="btns mt-3">
```

```
<a href="/booking/<%= listing._id%>/drop" class="btn offset-3" style="background-
color: #73AB95;">Cancel <i class="fa-solid fa-ban" style='font-size:12px'></i> </a>
```

```
<a href="/booking/userdetails/<%= listing._id %>?userId=<%= currUser._id %>"
class="ancor-tag btn ms-4" style="background-color: #F39C12; "> Details <i class='far fa-
question-circle' style='font-size:14px'></i>
```

```
<a>
```

</div>

```
<% } else if (currUser && listing.watchlist && listing.watchlist.includes(currUser._id))
{ %>
```

```
<div class="btns mt-3 ">
```

```
<a href="/booking/<%= listing._id%>/book" class="btn offset-3" style="background-color:
#E67E22;">Reserve </a>
```

```
<form action="/booking/watchlist/<%= listing._id%>/UnWatchlist?_method=Delete"
method="post" >
```

```
<button class="btn offset-5" style="background-color: #BDC3C7;"><i class="fa-solid
fa-bookmark"></i></button>
```

```
</form>
```

</div>

```
<% }else{ %>
```

```
<div class="btns mt-3">
```

```
<a href="/booking/<%= listing._id%>/book" class="btn offset-3" style="background-color:
#E67E22;">Reserve </a>
```

```
<form action="/booking/watchlist/<%= listing._id%>" method="get">
```

```
<button class="btn offset-5" style="background-color: #BDC3C7;"><i class="fa-regular
fa-bookmark"></i></button>
```



```

    </form>

</div>

<% } %>

<!-- comment -->

<div class="col-8 offset-3">

    <hr>

    <div class="col-6 mb-3">

        <h3>where you will be</h3>

        <div id="map"></div>

    </div>

    <hr>

    <% if(currUser){ %>

<h4> Leave A Review</h4>

    <form action="/listings/<%= listing._id%>/reviews" method="post" class="mb-3 needs-
validation" novalidate>

<!-- Rating -->

<div class="mb-3 mt-3">

    <label for="Rating" class="form-label">Rating</label>

    <fieldset class="starability-slot">

        <input type="radio" id="no-rate" class="input-no-rate" name="review[rating]" value="1"
checked aria-label="No rating." /> <!--default 1 -->

        <input type="radio" id="first-rate1" name="review[rating]" value="1" />
        <label for="first-rate1" title="Terrible">1 star</label>

        <input type="radio" id="first-rate2" name="review[rating]" value="2" />
        <label for="first-rate2" title="Not good">2 stars</label>

        <input type="radio" id="first-rate3" name="review[rating]" value="3" />
        <label for="first-rate3" title="Average">3 stars</label>

```

```

<input type="radio" id="first-rate4" name="review[rating]" value="4" />
<label for="first-rate4" title="Very good">4 stars</label>
<input type="radio" id="first-rate5" name="review[rating]" value="5" />
<label for="first-rate5" title="Amazing">5 stars</label>
</fieldset>
</div>

<div class="mb-3 mt-3">
  <label for="Comment" class="form-label">Comment</label>
  <textarea name="review[comment]" id="comment" cols="30" rows="5" class="form-control" required></textarea>
  <div class="invalid-feedback">Please Add some comment for review</div>
</div>

<button class="btn " style="background-color: #2c3e50;color: #fff;">Submit</button>
</form>

<hr>
<% } %>
<% if(listing.reviews.length>0) {%>
  <p><b>All reviews</b></p>
  <div class="row">
    <% for(review of listing.reviews) {%>
      <div class="card col-5 ms-3 mb-3">
        <div class="card-body">
          <h5 class="card-title">@<%= review.author.username %></h5>
          <p class="starability-result card-text" data-rating="<%= review.rating %>"> </p>
          <p class="card-text"><%= review.comment %></p>
          <% if(currUser && currUser._id.equals(review.author._id)) {%>

```

```
<form action="/listings/<%= listing._id %>/reviews/<%=review._id%>?_method=Delete"
method="post" class="mb-3">
```

```
<button class="btn btn-sm " style="background-color: #2c3e50; color:
#fff;">Delete</button>
```

```
</form>
```

```
<% } %>
```

```
</div>
```

```
</div>
```

```
<% } %>
```

```
</div>
```

```
<% } %>
```

```
</div>
```

```
</div>
```

```
<script src="/js/map.js"></script>
```

4. New.ejs

```
<% layout("layouts/boilerPlate") %>
```

```
<div class="row mt-3">
```

```
<div class="col-8 offset-2"> <!--off left se space -->
```

```
<h3>Add a New Property</h3>
```

```
<form action="/listings" method="post" class="needs-validation" novalidate
enctype="multipart/form-data">
```

```
<div class="mb-3">
```

```
<label for="title" class="form-label"> Title</label>
```

```
<input type="text" name="Listing[title]" placeholder="Add a Catchy title" class="form-
control" required>
```

```
<div class="valid-feedback">
```

```
Title Looks good!
```

```
</div>
```

```
</div>
```

```
<div class="mb-3">
```

```
<label for="description" class="form-label"> Description</label>
```

```
<textarea name="Listing[description]" id="" cols="30" rows="5" placeholder="enter  
description" class="form-control" required></textarea>
```

```
<div class="invalid-feedback">
```

```
  please enter a short description
```

```
</div>
```

```
</div>
```

```
<div class="mb-3">
```

```
<label for="image" class="form-label">Upload Image </label>
```

```
<input type="file" name="Listing[image]" class="form-control" multiple required>
```

```
</div>
```

```
<div class="row">
```

```
<div class="mb-3 col-md-4">
```

```
<label for="price" class="form-label"> Price</label>
```

```
<input type="number" name="Listing[price]" placeholder="1200" class="form-control"  
required>
```

```
<div class="invalid-feedback">
```

```
  price Should be valid !
```

```
</div>
```

```
</div>
```

```
<div class="mb-3 col-md-8">
```

```

    <label for="country" class="form-label"> country</label>

    <input type="text" name="Listing[country]" placeholder="India" class="form-control"
required>

    <div class="invalid-feedback">

        country name Should be valid !

    </div>

</div>

<div class="mb-3">

    <label for="location" class="form-label"> location</label>

    <input type="text" name="Listing[location]" placeholder="enter the location"
class="form-control" required >

    <div class="invalid-feedback">

        location Should be valid !

    </div>

</div>

<div class="mb-3 col-md-4">

    <label for="option" class="form-label"> Any special place And Tracking ?</label>

    <select class="form-select" aria-label="Default select example"
name="Listing[option]" required>

        <option value="Mountains" >Mountains</option>

        <option value="Camping">Camping</option>

        <option value="Domes">Domes</option>

        <option value="Boats">Boats</option>

        <option value="Rooms">Rooms</option>

        <option value="Castles">Castles</option>

        <option value="Pools">Pools</option>

```

```

        <option value="Farms">Farms</option>

        <option value="Arctic">Arctic</option>

        <option value="None">None</option>

    </select>

</div>

    <button class="btn col-md-4 offset-4 mb-3 mt-3" style="background-color:
#F39C12;">Add Property</button>

</form>

</div>

</div>

```

5. Login.ejs

```

<% layout("layouts/boilerPlate") %>

<div class="row login-container mt-3">

    <!-- Left side with login form -->

    <div class="col-lg-6 left-side">

        <h1 class="text-center mb-4">Login</h1>

        <form action="/login" method="post" class="needs-validation" novalidate>

            <div class="mb-3">

                <label for="username" class="form-label font-size">UserName</label>

                <input type="text" name="username" placeholder="Enter your username" class="form-
control" required>

            </div>

            <div class="mb-3">

                <label for="password" class="form-label font-size">Password</label>

                <a href="/verifyemail" class="forget-pass-font offset-6 font-size ">Forget
password?</a>

                <div class="input-container">

```

```
<input type="password" id="password" name="password" placeholder="Enter your
password" class="form-control" required>
```

```
<span class="input-group-text icon-button" id="togglePassword">
```

```
<i class="fa fa-eye" aria-hidden="true"></i>
```

```
</span>
```

```
</div>
```

```
</div>
```

```
<button class="btn btnlogin btn-lg col-12" style="background-color:
#F39C12;">Login</button>
```

```
</form>
```

```
</div>
```

```
<!-- Right side with image -->
```

```
<div class="col-lg-6 right-side"></div>
```

```
</div>
```

6. Signup.ejs

```
<% layout("layouts/boilerPlate") %>
```

```
<div class="row login-container mt-3">
```

```
<!-- Left side with sign-up form -->
```

```
<div class="col-lg-6 left-side">
```

```
<div>
```

```
<h1 class="text-center mb-4">SignUp</h1>
```

```
<form action="/signup" method="post" class="needs-validation" novalidate>
```

```
<div class="mb-3">
```

```
<label for="username" class="form-label">UserName</label>
```

```
<input type="text" name="username" placeholder="Enter your username" class="form-
control" required>
```

```
<div class="valid-feedback">Looks good!</div>
```

</div>

<div class="mb-3">

<label for="email" class="form-label">Email</label>

<input type="email" name="email" placeholder="Enter your email" class="form-control" required>

</div>

<div class="mb-3">

<label for="password" class="form-label">Password</label>

<div class="input-container">

<input type="password" id="password" name="password" placeholder="Enter your password" class="form-control" required>

<i class="fa fa-eye" aria-hidden="true"></i>

</div>

</div>

<button class="btn btnlogin col-3 offset-4 mt-3 flash" type="submit" style="background-color: #F39C12;">SignUp</button>

</form>

</div>

</div>

<!-- Right side with image -->

<div class="col-lg-6 right-side" style="background-image: url('https://res.cloudinary.com/dkiqoznwu/image/upload/v1727517257/wanderlust_DEV/swyc5qih3lh8kj9kayuc.svg'); "></div>

</div>

Backend Codes

1. booking.js

```
const Listing = require('../models/listing');
const Bookings = require('../models/booking');
const User = require('../models/user');

module.exports.showBookingPage=async(req,res)=>{
  let {id}=req.params
  const listing = await Listing.findById(id);
  res.render('booking/book.ejs',{listing})
}

module.exports.addBookingData= async (req, res) => {
  const { id } = req.params;
  const listing = await Listing.findById(id);

  let newBookings = new Bookings (req.body.Bookings)
  newBookings.user = req.user._id

  listing.bookings.push(newBookings)

  await newBookings.save();
  await listing.save()

  req.flash('success',' Booking Done !')
  res.redirect(`/listings/${id}`);
};
```

```

module.exports.showWatchlist=async(req,res)=>{

  const listings = await Listing.find({
    $or: [
      { bookings: { $exists: true } },
      { watchlist: { $exists: true } }
    ]
  }).populate({path:'bookings',populate:{path:'user'}})

  const allListings = listings.filter(listing => {
    return (
      (listing.bookings && listing.bookings.some(booking => booking.user
      &&booking.user._id.equals(res.locals.currUser._id))) ||

      (listing.watchlist && listing.watchlist.some(userId => userId
      &&userId.equals(res.locals.currUser._id)))

    );
  });

  // const allListings = listings.filter(listing => {
  //   const hasBooking = listing.bookings && listing.bookings.some(booking =>
  booking.user && booking.user.equals(res.locals.currUser._id));
  //   const isInWatchlist = listing.watchlist && listing.watchlist.some(userId => userId &&
  userId.equals(res.locals.currUser._id));
  //   return hasBooking || isInWatchlist;
  // });

  res.render("listings/index.ejs", { allListings });

```

```
}
```

```
module.exports.addWatchlist=async(req,res)=>{  
  let { id } = req.params;  
  let userId = req.user._id;  
  await Listing.findByIdAndUpdate(id, { $addToSet: { watchlist: userId } });  
  req.flash('success',' Watchlist Added !')  
  res.redirect(`/listings/${id}`);  
}
```

```
module.exports.customerDeleteBooking= async (req, res) => {  
  const { id } = req.params;  
  const listing = await Listing.findById(id)  
  .populate({path:'bookings',populate:{path:'user'}})  
  
  const index = listing.bookings.findIndex(booking => booking.user._id.equals(req.user._id));  
  if (index !== -1) {  
  
    const bookingId = listing.bookings[index]._id;  
    listing.bookings.splice(index, 1);  
  
    await listing.save();  
  
    await Bookings.findByIdAndDelete(bookingId);  
  
  }  
  req.flash('success',' Booking Cancelled !')  
  res.redirect('/booking/watch');
```

```
};
```

```
module.exports.unWatchlist=async(req,res)=>{  
  let {id}= req.params;  
  let userId = req.user._id;  
  await Listing.findByIdAndUpdate(id, { $pull: { watchlist: userId } });  
  req.flash('success',' Watchlist Deleted !')  
  res.redirect(`/listings/${id}`);  
};
```

```
module.exports.cart=async(req,res)=>{  
  let allListings = await Listing.find({ owner: res.locals.currUser._id, bookings: { $exists: true,  
$ne: [] } });  
  res.render("listings/index.ejs",{allListings})  
}
```

```
module.exports.userDetail= async (req, res) => {  
  const { id } = req.params; // Listing ID  
  const userId = req.query.userId; // User ID from query parameter  
  
  if (!userId) {  
    return res.status(400).send('User ID is required');  
  }  
  
  // Find the specific listing by its ID and populate the bookings and their associated users  
  let listing = await Listing.findById(id).populate({  
    path: 'bookings',  
    populate: { path: 'user' }  
  })
```

```

});

if (!listing) {
    return res.status(404).send('Listing not found');
}

// Filter bookings to only those associated with the given user
listing.bookings = listing.bookings.filter(booking => booking.user._id.equals(userId));
const user = await User.findById(userId);
if (!user) {
    return res.status(404).send('User not found');
}

res.render('booking/bookingUser.ejs', { listings: [listing], user });
};

module.exports.customerDetails=async(req,res)=>{
    const { id } = req.params;
    const listing = await Listing.findById(id).populate({
        path: 'bookings',
        populate: { path: 'user' }
    });
    if (!listing) {
        req.flash('error','Listing you requested for does not exists')
        res.redirect('/listings')
    }
    const listings = [listing];
    res.render('booking/customers.ejs',{listings})
}

```

```

}

module.exports.ownerDeleteBooking=async(req,res)=>{
  const { id } = req.params;
  const userId = req.query.userId;

  // Find the specific listing by its ID and populate the bookings and their associated users
  const listing = await Listing.findById(id)
    .populate({
      path: 'bookings',
      populate: { path: 'user' }
    });

  if (!listing.owner.equals(req.user._id)) {
    req.flash('error', 'You are not authorized to perform this action');
    return res.redirect(`/listings/${id}`);
  }

  console.log("user delete",userId)

  const bookingIndex = listing.bookings.findIndex(booking =>
    booking.user._id.equals(userId));

  if (bookingIndex !== -1) {
    const bookingId = listing.bookings[bookingIndex]._id;
    listing.bookings.splice(bookingId, 1);
    await Bookings.findByIdAndDelete(bookingId);
    await listing.save();
  }
}

```

```

    req.flash('success', 'Booking Completed successfully');

    res.redirect(`/booking/customerDetails/${id}`);
  }

module.exports.billing=async(req,res)=>{
  const { id } = req.params;
  const userId = req.query.userId;

  // Find the specific listing by its ID and populate the bookings and their associated users
  const listing = await Listing.findById(id)
    .populate({
      path: 'bookings',
      populate: { path: 'user' }
    }).populate('owner');
  const user = await User.findById(userId)

  // Ensure that the user making the request is the owner of the listing
  if (!listing.owner.equals(req.user._id)) {
    req.flash('error', 'You are not authorized to perform this action');
    return res.redirect(`/listings/${id}`);
  }

  res.render('booking/billing.ejs', {user, listing})
}

module.exports.backPage=(req,res)=>{
  const { id } = req.params;

```

```
    res.redirect(`/listings/${id}`);  
  }  
}
```

2. listings.js

```
const Listing = require('../models/listing');
```

```
module.exports.index = async (req,res)=>{//Index Route  
  const allListings = await Listing.find({});  
  res.render("listings/index.ejs",{allListings})  
};
```

```
module.exports.renderNewFrom=(req,res)=>{  
  res.render('listings/new.ejs')  
};
```

```
module.exports.showListing=async(req,res)=>{  
  let {id} = req.params;  
  const listing = await Listing.findById(id)  
  .populate({path:'reviews',populate:{path:'author'}})  
  .populate('owner')  
  .populate({path:'bookings',populate:{path:'user'}})  
  
  //show in detail  
  if(!listing){  
    req.flash('error','Property you requested for does not exists')  
    res.redirect('/listings')  
  }  
}
```



```

    res.render('listings/show.ejs',{listing})
};

module.exports.createListing=async (req,res,next)=>{

    const newListing = new Listing( req.body.Listing);
    newListing.owner=req.user._id
    if (req.files && req.files.length > 0) {
        newListing.image = req.files.map(file => ({ url: file.path, filename: file.filename }));
    }

    await newListing.save()
    req.flash('success','New Property Created !')
    res.redirect('/listings')

};

module.exports.renderEditForm=async(req,res)=>{
    let {id} = req.params;
    const listing = await Listing.findById(id)
    if(!listing){
        req.flash('error','Property you requested for does not exists')
        res.redirect('/listings')
    }

    let originallamgeUrl = "";
    if (listing.image && listing.image.length > 0) {
        originallamgeUrl = listing.image[0].url.replace('/upload', '/upload/w_250');
    }

```

```

    }

    res.render('listings/edit.ejs', { listing, originalImageUrl });

};

module.exports.updateListing = async (req, res) => {
    let { id } = req.params;
    let listing = await Listing.findById(id);
    if (!listing) {
        req.flash('error', 'Property not found');
        return res.redirect('/listings');
    }
    // Update listing with new data
    Object.assign(listing, req.body.Listing);
    module.exports.destroyListing=async (req,res)=>{
        let {id} = req.params;
        await Listing.findByIdAndDelete(id);
        req.flash('success',' Property Deleted !')
        res.redirect('/listings')
    };
    module.exports.searchListing=async (req, res) => {
        let { country } = req.body;
        // console.log(country)

        // Redirect to the results page with the search query as a query parameter
        res.redirect(`/listings/results?country=${encodeURIComponent(country)}`);
    };
    module.exports.searchResult=async(req,res)=>{

```

```

    let {country}= req.query;
    // console.log(country)
    const allListings = await Listing.find({
      "$or":[
        {"country":{"$regex:country"}},
        {"location":{"$regex:country"}},
      ]
    });
    res.render("listings/index.ejs",{allListings})
  };
  module.exports.specialOptions=async(req,res)=>{
    let {option} =req.params
    const allListings = await Listing.find({option:option})
    res.render("listings/index.ejs",{allListings})
  }
  module.exports.UserOption=async(req,res)=>{ //my Property
    let {user}= req.params
    const allListings = await Listing.find({owner:user});
    res.render("listings/index.ejs",{allListings})
  }
}

```

3. reviews.js

```

const Listing = require('../models/listing');
const Review = require('../models/review');
module.exports.createReview=async(req,res,)=>{
  let listing = await Listing.findById(req.params.id);
  let newReview = new Review(req.body.review);
  newReview.author=req.user._id;

```

```

    listing.reviews.push(newReview)
    await newReview.save();
    await listing.save();
    req.flash('success','New Review Created !')
    res.redirect(`/listings/${listing._id}`)
  };
module.exports.destroyReview=async(req,res)=>{
  let{id,reviewId} = req.params;
  await Listing.findByIdAndUpdate(id,{ $pull: {reviews:reviewId}}) //remove
  await Review.findByIdAndDelete(reviewId);//delete
  req.flash('success','Review Deleted !')
  res.redirect(`/listings/${id}`);
}

```

4. user.js

```

const User = require('../models/user');
const nodemailer = require('nodemailer');
const crypto = require('crypto');

module.exports.renderSignupForm=(req,res)=>{
  res.render('users/signup.ejs')
}

module.exports.signup=async(req,res)=>{
  try{
    let{username,email,password}=req.body;
    const newUser =new User({email,username});
    const registeredUser = await User.register(newUser,password);

```

```

// console.log(registeredUser)
req.logIn(registeredUser,(err)=>{
  if(err){
    return next(err)
  };
  req.flash('success','Welcome to EasyStay')
  res.redirect('/listings');
})
} catch(e){
  req.flash('error',e.message);
  res.redirect('/signup')
}
}
module.exports.renderLoginForm=(req,res)=>{
  res.render('users/login.ejs')
};
module.exports.login=async(req,res)=>{
  req.flash('success','Welcome back to EasyStay! ')
  let redirectUrl = res.locals.redirectUrl || '/listings' //new var
  // console.log(redirectUrl)
  res.redirect(redirectUrl)
}

module.exports.logout=(req,res)=>{
  req.logOut((err)=>{
    if(err){
      next(err)
    }
  })
}

```

```

    req.flash('success','you are logged out !');
    res.redirect('/listings')
  })
}

```

5. App.js

```

if(process.env.NODE_ENV !== "production"){
  require('dotenv').config()
}

const { savRedirectUrl, isLoggedIn } = require('./middleware');
const express = require('express')
const app = express()
const mongoose = require('mongoose')
const Listing = require('./models/listing');
const path = require('path');
const methodOverride = require('method-override');
const ejsMate = require('ejs-mate');
const wrapAsync= require('./utils/wrapAsync')
const ExpressError = require('./utils/ExpressError');
const {listingSchmea,reviewSchema}= require('./schmea')
const listingRouter = require('./routes/listing')
const reviewRouter = require('./routes/review')
const userRouter = require('./routes/user')
const bookingRouter = require('./routes/booking')

const session = require('express-session')
const MongoStore = require('connect-mongo');

```

```

const flash = require('connect-flash');
const passport = require('passport');
const LocalStrategy = require('passport-local');
const User = require('./models/user');
// const { error } = require('console');
app.set('view engine','ejs')
app.set('views',path.join(__dirname,"views"))
app.use(express.urlencoded( {extended:true}))
app.use(methodOverride("_method"));
app.engine('ejs',ejsMate);
app.use(express.static(path.join(__dirname,"/public")));
const dbUrl = 'mongodb://localhost:27017/EasyStay'
const store = MongoStore.create({ //mongo session
  mongoUrl:dbUrl,
  crypto:{
    secret:process.env.SECRET,
  },
  touchAfter:24*3600,
});

store.on('error',()=>{
  console.log('ERROR in MONGO SESSION STORE ',err)
});

const sessionOption ={
  store,
  secret:process.env.SECRET,
  resave:false,

```

```

    saveUninitialized:true,

    cookie:{

        expires>Date.now() + 7 * 24 * 60 * 60 * 1000,

        maxAge: 7 * 24 * 60 * 60 * 1000,

        httpOnly:true,

    }

}

app.use(session(sessionOption))
app.use(flash());


main().then(=>console.log('sucessful connected'))
.catch((err)=>console.log(err));


async function main(){
    await mongoose.connect(dbUrl)
}

app.use(passport.initialize());
app.use(passport.session());
passport.use(new LocalStrategy(User.authenticate())); //user & paswd
passport.serializeUser(User.serializeUser()); //store in session
passport.deserializeUser(User.deserializeUser());
app.use((req,res,next)=>{

    res.locals.success=req.flash('success');

    res.locals.error=req.flash('error');

    res.locals.currUser=req.user; //navbar page (login)

    next();

});

// /show/Boats

```



```
app.use('/listings',listingRouter);
app.use('/listings/:id/reviews',reviewRouter);
app.use('/booking',bookingRouter);
app.use('/',userRouter)
app.get('/',(req,res)=>{
    res.redirect('/listings')
})
app.all('*', (req,res,next)=>{
    next(new ExpressError(404,'Page Not Found '))
});
app.use((err, req, res, next) => {
    let {statusCode=404,message='something went wrong err'}=err;
    // res.status(statusCode).send(message);
    res.status(statusCode).render('error.ejs',{message})
});
app.listen(3000,()=>{
    console.log("listening 3000")
})
```

4.2 Testing Approach

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding. The increasing visibility of software as a system element and attendant costs associated with a software failure are motivating factors for we planned, through testing. Testing is the process of executing a program with the intent of finding an error. The design of tests for software and other engineered products can be as challenging as the initial design of the product itself

There are basically two types of testing approaches:

- One is Black-Box testing – the specified function that a product has been designed to perform, tests can be conducted that demonstrate each function is fully operated.
- The other is White-Box testing – knowing the internal workings of the product, tests can be conducted to ensure that the internal operation of the product performs according to specifications and all internal components have been adequately exercised. White box and Black box testing methods have been used to test this package. The entire loop constructs have been tested for their boundary and intermediate conditions. The test data was designed with a view to check for all the conditions and logical decisions

Testing Strategies:

Testing is a set of activities that can be planned in advance and conducted systematically. A strategy for software testing must accommodate low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high-level tests that validate major system functions against customer requirements.

Software testing is one element of verification and validation. Verification refers to the set of activities that ensure that software correctly implements a specific function

The main objective of software is testing to uncover errors. To fulfill this objective, a series of test steps unit, integration, validation and system tests are planned and executed. Each test step is accomplished through a series of systematic test techniques that assist in the design of test cases

Testing is the only way to assure the quality of software and it is an umbrella activity rather than a separate phase. This is an activity to be performed in parallel with the software effort and one that consists of its own phases of analysis, design, implementation, execution and maintenance.

4.2.1 Unit Testing

Unit testing is a crucial aspect of software development that focuses on testing individual components or units of code in isolation to ensure they perform as expected. In the EasyStay Room Reservation system, unit testing is essential to verify the correctness and reliability of core functionalities.

Key Areas to Focus on for Unit Testing in EasyStay Room Reservation:

1. Room Management:

- **Room Creation and Validation:** Test the creation of valid room listings, including checks for correct details like room type, pricing, availability status, and validation of necessary fields.
- **Room Update and Deletion:** Ensure that updating room details works correctly, including validation of input changes, and that rooms can be deleted without affecting existing reservations.

2. Booking Process:

- **Booking Creation and Validation:** Test the creation of valid bookings, including input validation for dates, guest information, and payment details.
- **Booking Conflict Resolution:** Verify that the system correctly handles booking conflicts by preventing overlapping reservations and notifying users appropriately.

3. User Management:

- **User Registration and Authentication:** Test user registration, including validation of input data (e.g., email format, password strength) and ensure secure authentication processes for user login.
- **User Profile Management:** Verify that users can update their profiles and that changes are correctly reflected in the database.

4. Billing Operations:

- **Bill Generation:** Test the generation of accurate bills based on room bookings, including the correct calculation of total costs, taxes, and discounts.
- **Payment Processing:** Ensure that payment processing functions correctly, validating payment details and confirming successful transactions.

4.2.1.1 Unit Test Cases

EasyStay Room Reservation:

Test Case ID	Description	Expected Result	Pass/Fail
TC1	Test if a block can be created with valid data.	Block creation should be successful.	Pass
TC2	Test if a block with invalid data is rejected.	Block validation should fail with an appropriate error message.	Pass
TC3	Test if the block hash is calculated correctly.	Calculated hash should match the expected hash.	Pass
TC4	Test if transactions with invalid data are rejected.	Transaction validation should fail with an appropriate error message.	Pass
TC5	Test if double-spending attempts are detected.	Double-spending attempts should be rejected.	Pass

Booking Functions Unit Tests

Test Case ID	Description	Expected Result	Actual Result	Pass/Fail
TC6	Test if a booking can be created with valid data.	Booking should be successful.	Booking is successful.	Pass
TC7	Test if a booking with invalid data is rejected.	Booking validation should fail with an appropriate error message.	Booking validation fails with an appropriate error message.	Pass
TC8	Test if booking conflicts are detected.	Conflicting bookings should be rejected.	Conflicting bookings are rejected.	Pass

Room Management Unit Tests

Test Case ID	Description	Expected Result	Actual Result	Pass/Fail
TC9	Test if room details can be updated successfully.	Room details should be updated.	Room details are updated.	Pass
TC10	Test if a room can be deleted.	Room should be deleted successfully.	Room is deleted successfully.	Pass
TC11	Test if invalid room data results in a rejection.	Invalid room data should not be accepted.	Invalid room data is rejected.	Pass

User Authentication Unit Tests

Test Case ID	Description	Expected Result	Actual Result	Pass/Fail
TC12	Test if a user can register with valid details.	User registration should be successful.	User registration is successful.	Pass
TC13	Test if a user cannot register with duplicate email.	Registration should fail with an appropriate error message.	Registration fails with an appropriate error message.	Pass
TC14	Test if a user can log in with valid credentials.	User should be able to log in successfully.	User is able to log in successfully.	Pass
TC15	Test if login fails with invalid credentials.	Login should fail with an appropriate error message.	Login fails with an appropriate error message.	Pass

4.2.2 Integration Testing

Unit testing is a critical aspect of software development that involves testing individual components or units of code in isolation to ensure they function as expected. For a room reservation system like EasyStay, unit testing is particularly important to verify the correctness and reliability of core functionalities.

1. Booking and Room Interactions:

- **Booking creation and validation:** Test the successful creation of valid bookings, including availability checks and user validation.
- **Booking conflict detection:** Ensure that overlapping bookings are accurately identified and rejected.

2. Room Management:

- **Room details management:** Verify that room details can be updated correctly and that the data is stored accurately.
- **Room deletion:** Test the deletion process to confirm that rooms can be deleted successfully, with all associated data handled appropriately.

3. User Authentication:

- **User registration and login:** Validate the user registration process, ensuring that valid users can register while duplicates and invalid entries are rejected.
- **User session management:** Confirm that users can log in and out securely, maintaining session integrity.

4. Notification System:

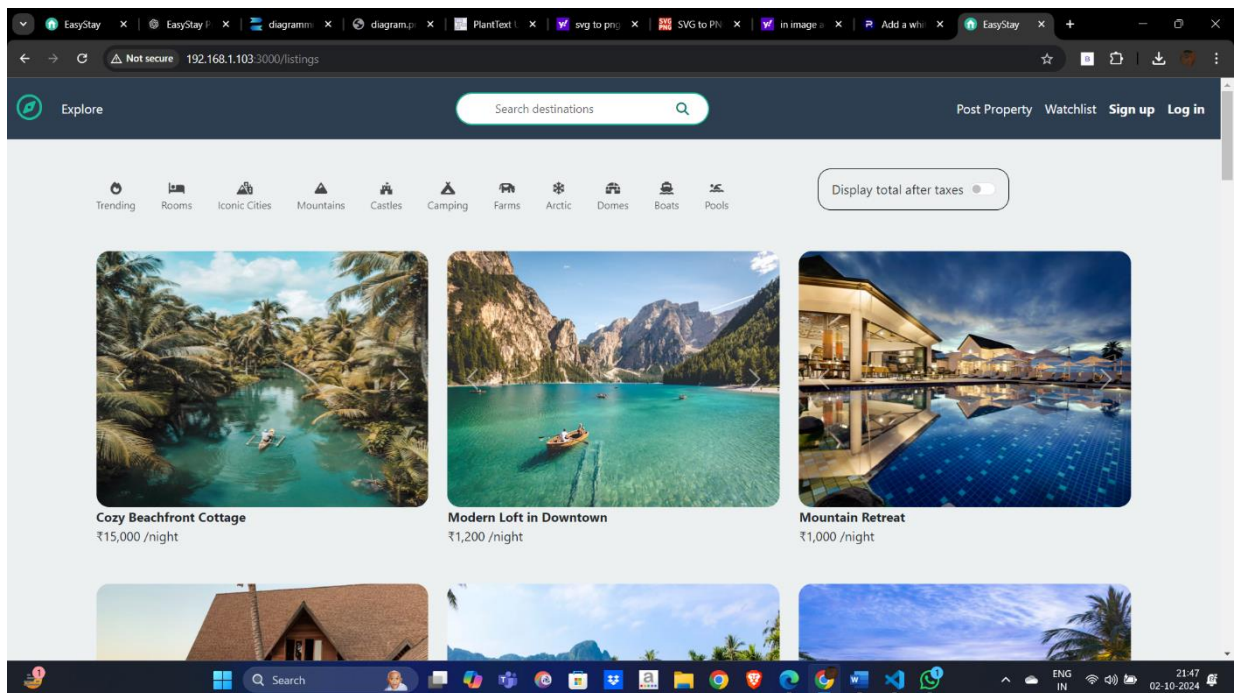
- **Notification functionality:** Test that notifications are sent correctly for booking confirmations, cancellations, and updates.

4.2.2.1 Integration Test Cases

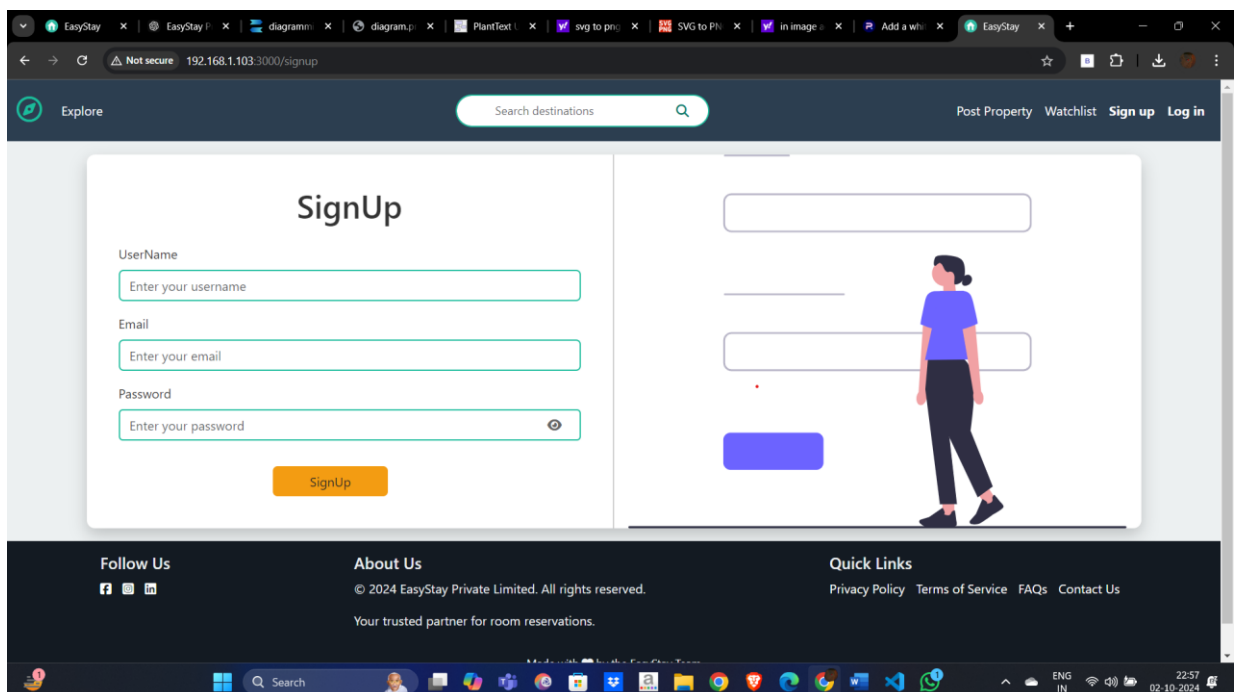
Test Case ID	Description	Expected Result	Actual Result	Pass/Fail
TC1	Test if a booking can be created with valid data.	Booking should be successful.	Booking is successful.	Pass
TC2	Test if a booking with invalid data is rejected.	Booking validation should fail with an appropriate error message.	Booking validation fails with an appropriate error message.	Pass
TC3	Test if booking conflicts are detected.	Conflicting bookings should be rejected.	Conflicting bookings are rejected.	Pass
TC4	Test if room details can be updated successfully.	Room details should be updated.	Room details are updated.	Pass
TC5	Test if a room can be deleted.	Room should be deleted successfully.	Room is deleted successfully.	Pass
TC6	Test if invalid room data results in a rejection.	Invalid room data should not be accepted.	Invalid room data is rejected.	Pass
TC7	Test if a user can register with valid details.	User registration should be successful.	User registration is successful.	Pass
TC8	Test if a user cannot register with duplicate email.	Registration should fail with an appropriate error message.	Registration fails with an appropriate error message.	Pass
TC9	Test if a user can log in with valid credentials.	User should be able to log in successfully.	User is able to log in successfully.	Pass
TC10	Test if login fails with invalid credentials.	Login should fail with an appropriate error message.	Login fails with an appropriate error message.	Pass

RESULT

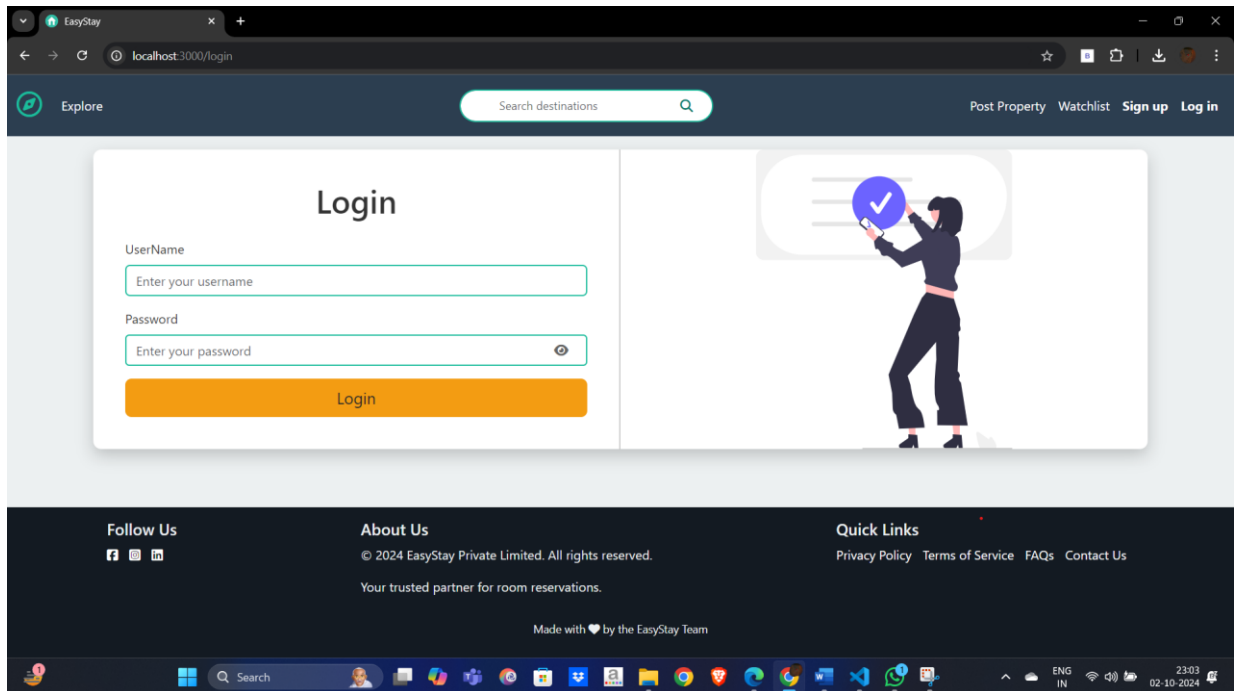
Index.ejs



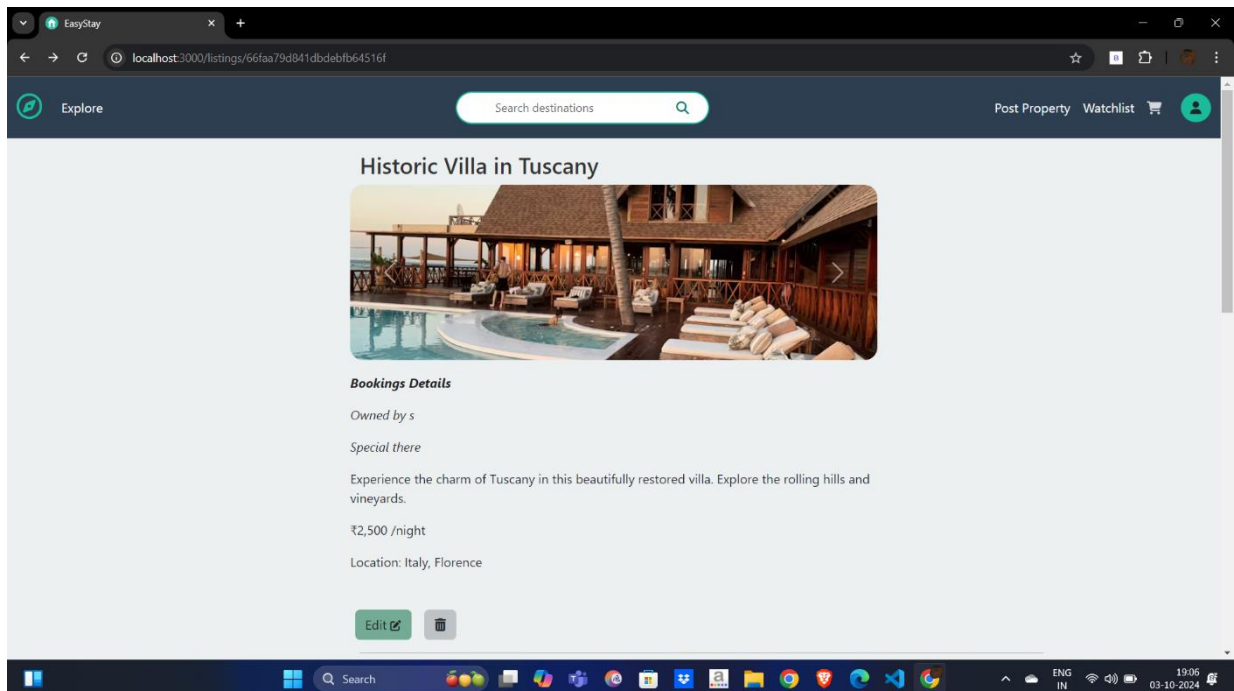
SignUp.ejs



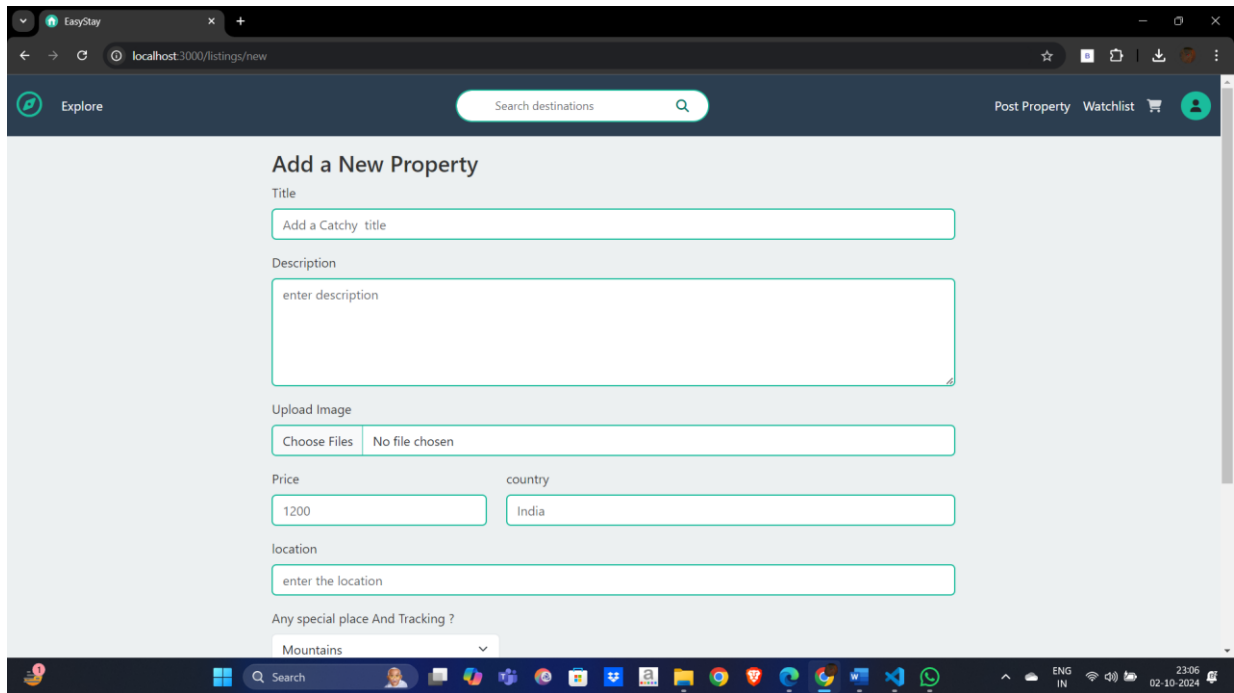
Login.ejs



Show.ejs



New.ejs

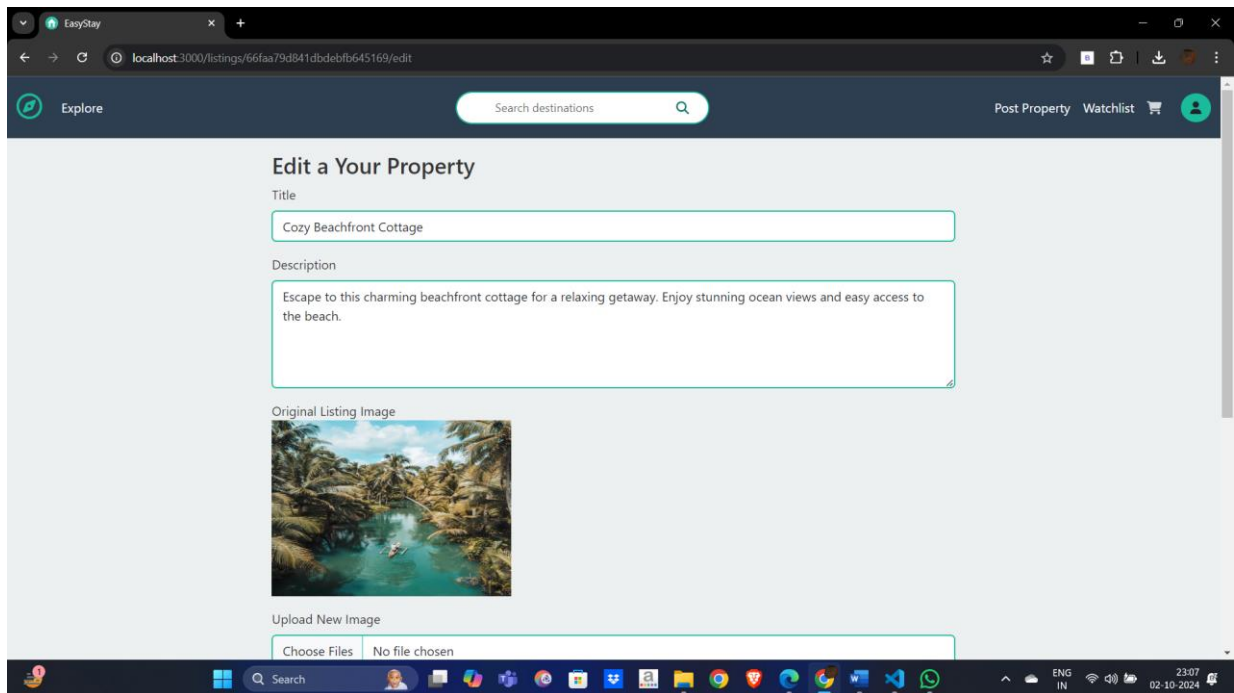


The screenshot shows a web browser window with the URL `localhost:3000/listings/new`. The page has a dark blue header with an 'Explore' button, a search bar labeled 'Search destinations', and links for 'Post Property', 'Watchlist', and a user profile. The main content area is titled 'Add a New Property' and contains the following form fields:

- Title:** A text input field with the placeholder text 'Add a Catchy title'.
- Description:** A large text area with the placeholder text 'enter description'.
- Upload Image:** A file upload button labeled 'Choose Files' and a status indicator 'No file chosen'.
- Price:** A text input field containing the value '1200'.
- country:** A text input field containing the value 'India'.
- location:** A text input field with the placeholder text 'enter the location'.
- Any special place And Tracking ?** A dropdown menu currently showing 'Mountains'.

The Windows taskbar at the bottom shows the time as 23:06 on 02-10-2024.

Edit.esj

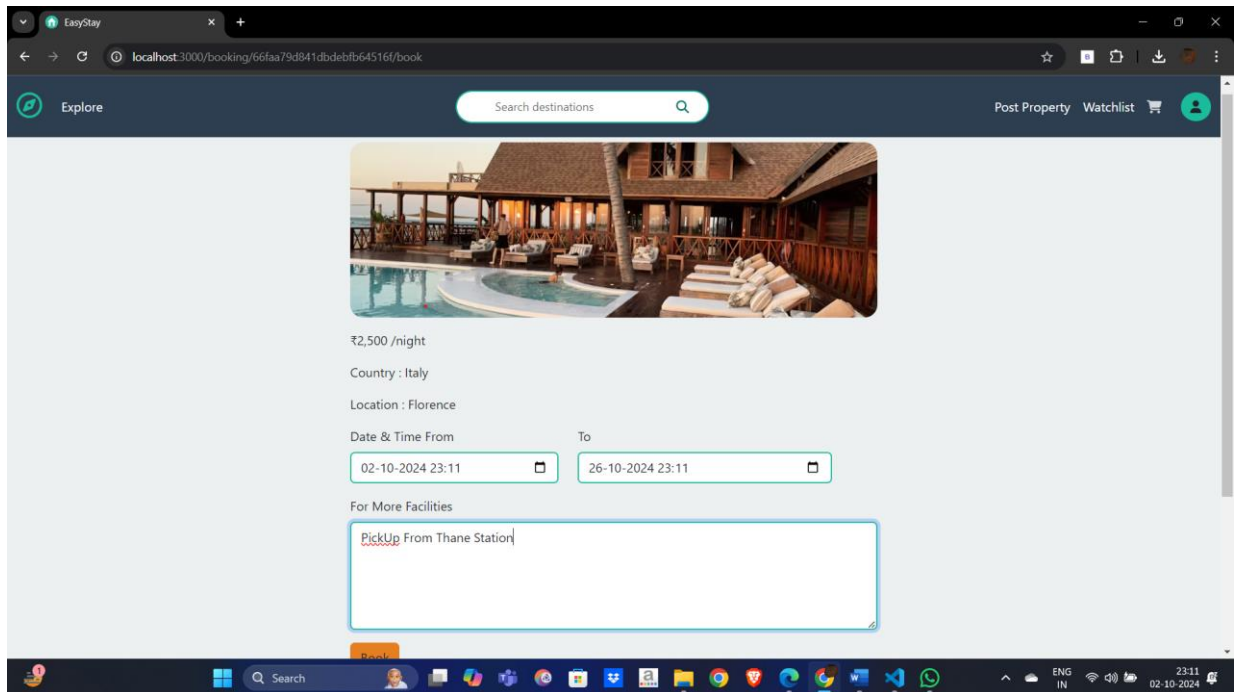


The screenshot shows a web browser window with the URL `localhost:3000/listings/66faa79d841dbdebf645169/edit`. The page layout is identical to the 'New.ejs' page. The form fields are pre-filled with the following data:

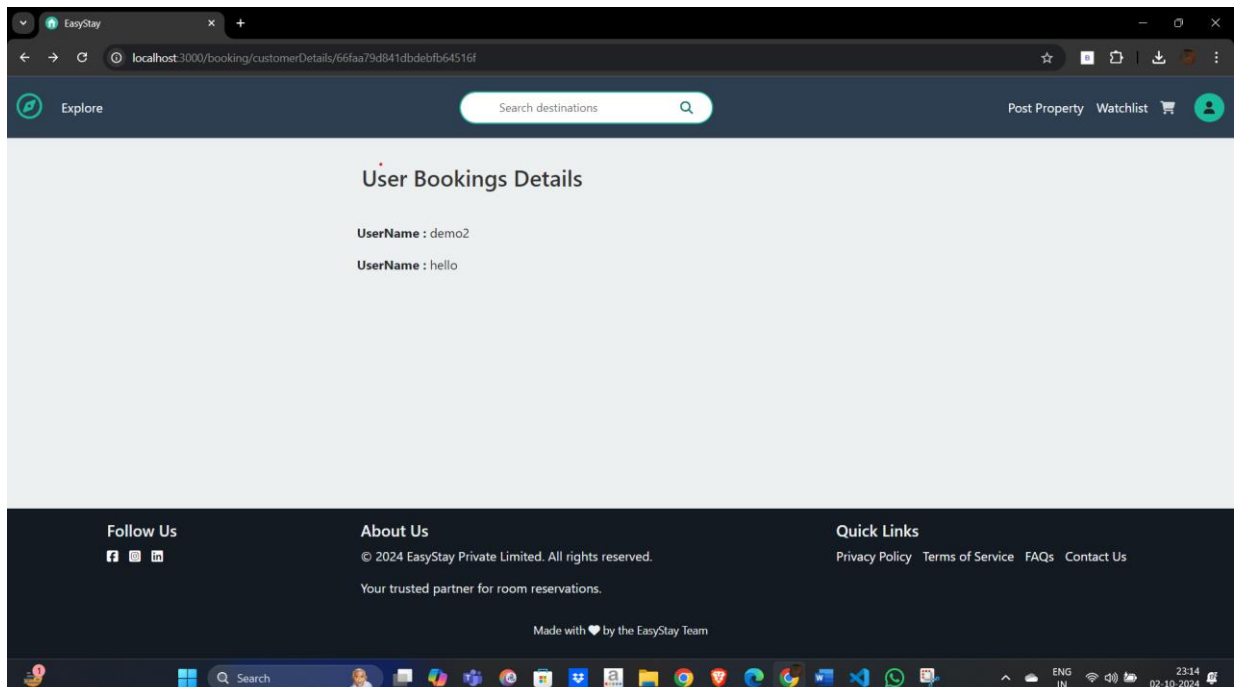
- Title:** 'Cozy Beachfront Cottage'.
- Description:** 'Escape to this charming beachfront cottage for a relaxing getaway. Enjoy stunning ocean views and easy access to the beach.'
- Original Listing Image:** A preview of a tropical beach scene with palm trees and a small boat in the water.
- Upload New Image:** A file upload button labeled 'Choose Files' and a status indicator 'No file chosen'.

The Windows taskbar at the bottom shows the time as 23:07 on 02-10-2024.

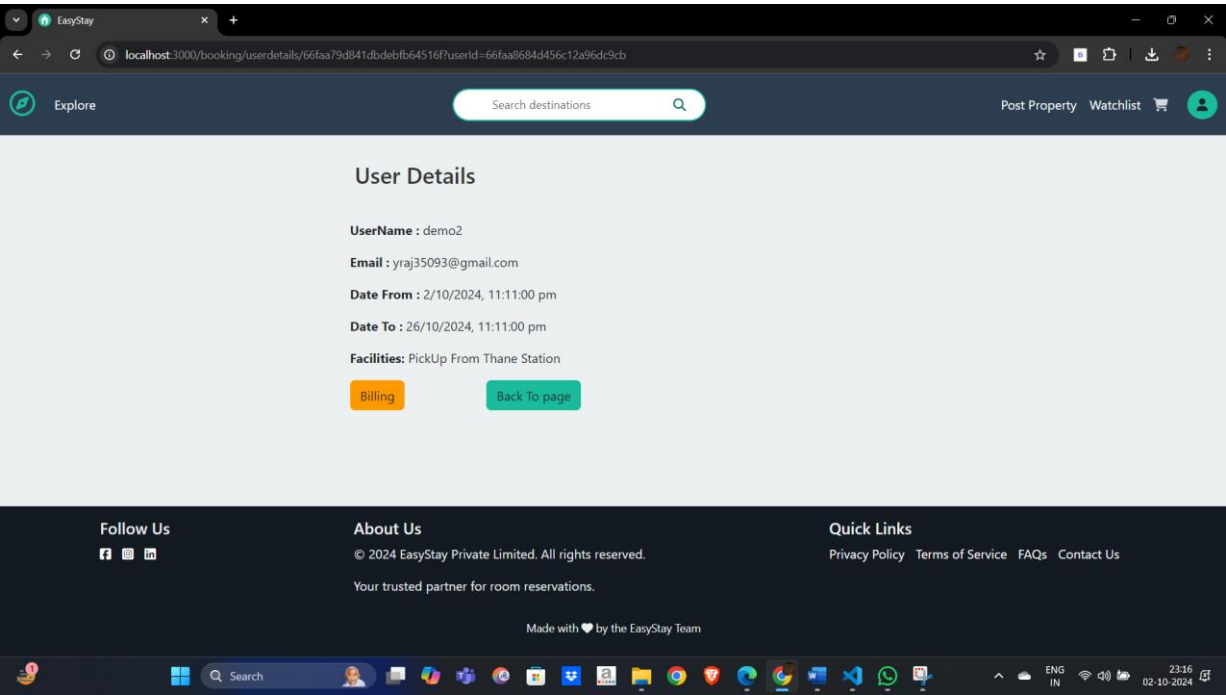
Book.ejs



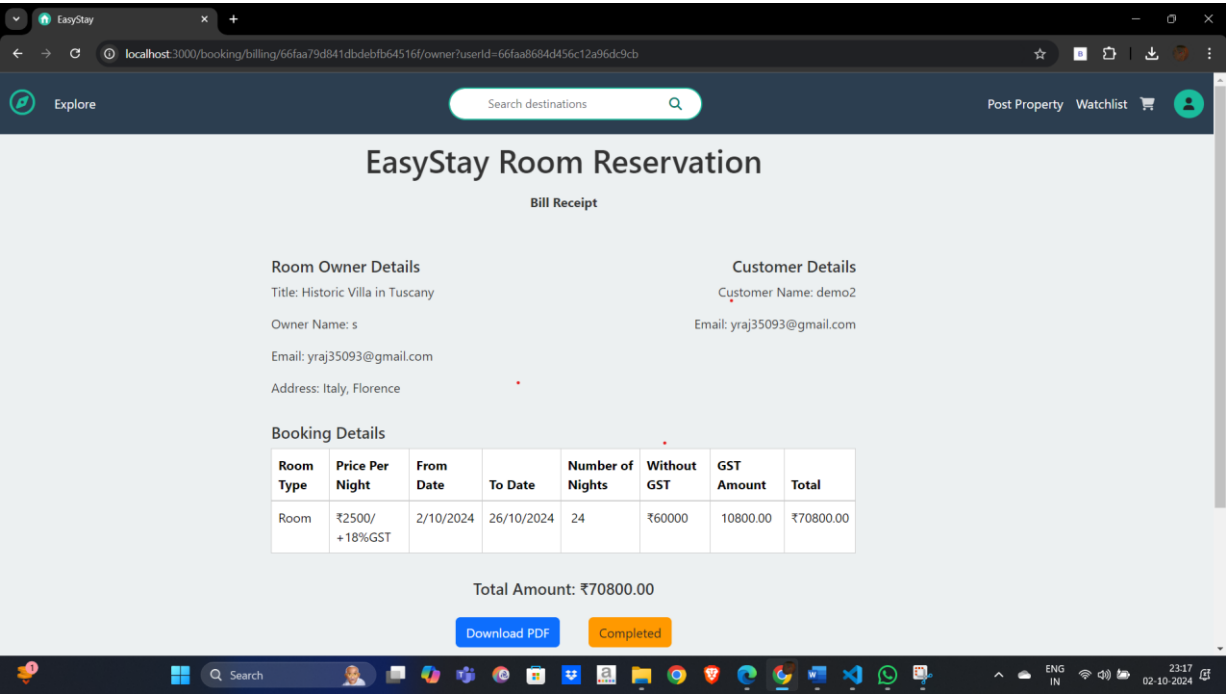
Customer.ejs



BookingUser.ejs



Billing.ejs



MongoDB Database

The screenshot shows the MongoDB Compass interface. The left sidebar displays the 'EasyStay' database with collections: bookings, reviews, rooms, sessions, and users. The 'rooms' collection is selected. The main panel shows two documents from the 'rooms' collection:

```
{ "_id": ObjectId('66faa79d841dbdebfb645169'),  
  "title": "Cozy Beachfront Cottage",  
  "description": "Escape to this charming beachfront cottage for a relaxing getaway. Enj...",  
  "image": Array (1),  
  "0": Object,  
    "price": 15000,  
    "location": "Malibu",  
    "country": "United States",  
  "reviews": Array (empty),  
  "owner": ObjectId('66faa7726057be53a10735aa'),  
  "bookings": Array (empty),  
  "watchlist": Array (empty),  
  "__v": 6,  
  "option": "Farms"  
}
```

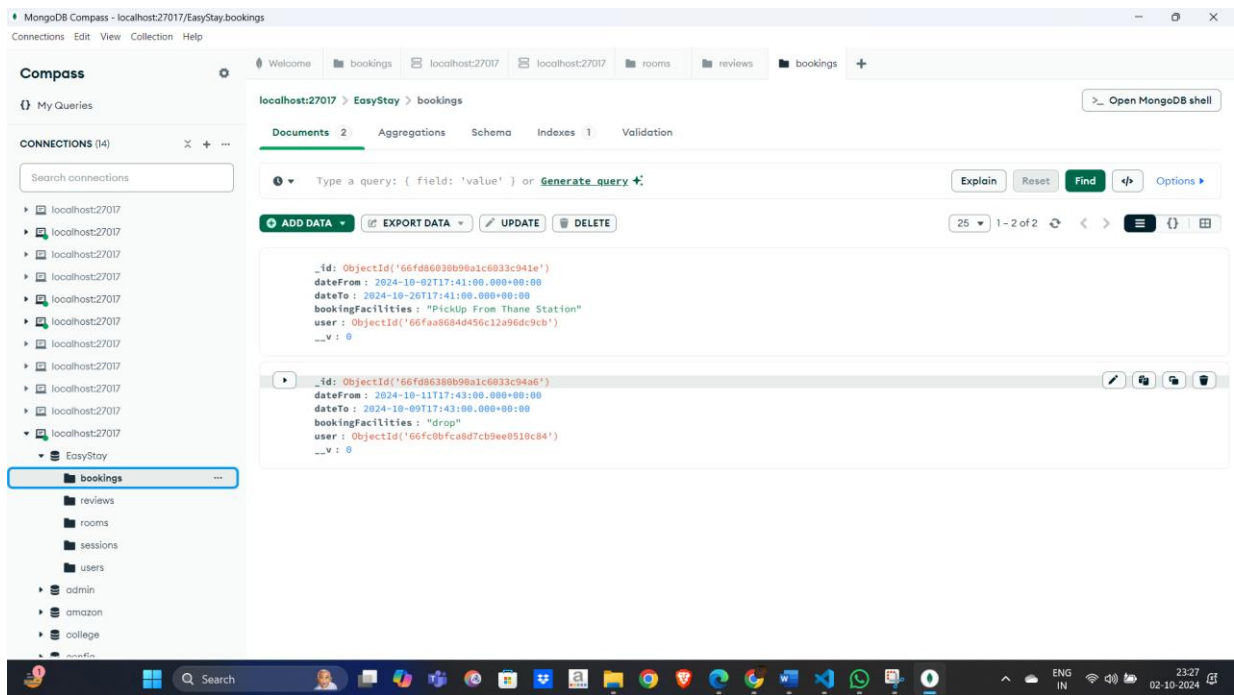
```
{ "_id": ObjectId('66faa79d841dbdebfb64516b'),  
  "title": "Modern Loft in Downtown",  
  "description": "Stay in the heart of the city in this stylish loft apartment. Perfect ...",  
  "image": Array (1),  
  "price": 1200,  
  "location": "New York City",  
  "country": "United States",  
  "reviews": Array (1),  
  "owner": ObjectId('66faa7726057be53a10735aa'),  
  "bookings": Array (empty),  
  "watchlist": Array (empty),  
  "__v": 3  
}
```

The screenshot shows the MongoDB Compass interface with the 'users' collection selected. The main panel displays three documents from the 'users' collection:

```
{ "_id": ObjectId('66faa7726057be53a10735aa'),  
  "email": "yraj35093@gmail.com",  
  "username": "s",  
  "salt": "eab586b14b1116137eb606645499538c21140f32998f7f308e27b42ef2e2a6d4",  
  "hash": "e38d74d51e99f356bf4372caaa857bd46c82362aaff291dd41b9171d266c71f30495d...",  
  "__v": 0  
}
```

```
{ "_id": ObjectId('66faa8684d456c12a96dc9cb'),  
  "email": "yraj35093@gmail.com",  
  "username": "demo2",  
  "salt": "e09a09eaab8df4272cfc87a943018ae41cc461f4f24772ed3ee27fd8c917fc36",  
  "hash": "e42673072d5b107d31bef1484591c6d4860ee396a98b5404f8a02395980f4971859209...",  
  "__v": 0  
}
```

```
{ "_id": ObjectId('66fc0bfca8d7cb9ee0518c84'),  
  "email": "yraj35093@gmail.com",  
  "username": "hello",  
  "salt": "1e4b3df78e1ae2be82a7828d1fd63193e9579a740ace8b29b3e3a61ca228afd",  
  "hash": "0f3fd3d24998696a75d21569f45b81fb7a5cdcb87bba2f2d3cd7ala7df05233b5011f8...",  
  "__v": 0  
}
```



Conclusion

In conclusion, the EasyStay Room Reservation System aims to transform the room booking experience by offering a user-friendly, efficient, and secure platform for both customers and room owners. The system streamlines essential functionalities such as real-time availability, booking management, and billing, ensuring a seamless process for all users involved.

Key benefits of the system include enhanced operational efficiency, improved user experience through intuitive design, and robust security measures that protect sensitive information. Furthermore, the ability to upload multiple images and generate PDF bills adds significant value, allowing room owners to showcase their listings effectively.

By implementing the EasyStay Room Reservation System, customers will enjoy faster booking processes and personalized services, while room owners will gain increased control over their listings and better management capabilities. Ultimately, this project seeks to elevate the room reservation industry, driving customer satisfaction and business growth.

Future Scope

There are several areas in which the EasyStay Room Reservation System could be expanded in the future:

1. **Scalability Improvements:** For larger establishments, the system can be optimized by implementing load balancing and database indexing to enhance performance during high traffic.
2. **Integration with IoT Systems:** Future versions could incorporate IoT devices to monitor room conditions in real-time, improving guest experience and safety compliance.
3. **Advanced Analytics and Reporting:** Adding predictive analytics can help forecast booking patterns and automate reporting, aiding in strategic decision-making.
4. **Mobile Application Development:** Developing mobile apps for iOS and Android would provide on-the-go access for users, enhancing convenience.
5. **Integration with External Systems:** The system could integrate with property management or booking systems to streamline operations and improve user experience.
6. **Support for Multiple Languages:** A multi-language feature could broaden accessibility for international users, enhancing usability across diverse regions.

References

MongoDB Documentation: The official MongoDB documentation was crucial for understanding database design, including schema design, indexing, and querying data in the context of the inventory system.

- URL: <https://www.mongodb.com/docs/>

Node.js Documentation: The official Node.js documentation was used to build the server-side logic for managing API endpoints and database interactions in the CIMS.

- URL: <https://nodejs.org/en/docs/>
- URL: <https://expressjs.com/en/starter/installing.html>

W3Schools JavaScript & MongoDB Tutorials: W3Schools was an important resource for JavaScript basics and MongoDB operations, providing easy-to-understand guides and examples for integrating JavaScript with MongoDB.

- URL: <https://www.w3schools.com/>