

SEMESTER TRAINING REPORT

On

ROOM BOOKING SYSTEM

*Submitted in partial fulfillment of requirements
for the award of the degree*

Bachelor of Technology

In

Computer Science and Engineering

To

IKG Punjab Technical University, Jalandhar

SUBMITTED BY:

Name: Prince Raj

Roll no.: 2003239

Semester: 8th

Batch: 2020-2024

Under the guidance of

Mr. Rajeev Sharma

Assistant Professor



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Chandigarh Engineering College-CGC, Landran

Mohali, Punjab - 140307

CERTIFICATE

This is to certify that **Mr. Prince Raj** has partially completed / completed / not completed the Semester Training during the period from **23th Jan 2024** to **30th Jun 2024** in our Organization / Industry as a Partial Fulfillment of Degree of Bachelor of Technology in Computer Science & Engineering.

(Signature of Project Supervisor)

Date: 27-05-2024

Certification of Appreciation

CONGRATULATIONS TO

Prince Raj

for successfully completing 3 months MERN Stack training program starting from 23rd January 2024 to 10th April 2024. The training program was aimed at providing theoretical knowledge as well as practical skills for optimum learning.

29/04/2024

Date

Alok Ramsisaria

Alok Ramsisaria (CEO)

CANDIDATE DECLARATION

I hereby declare that the Project Report entitled ("**Room Booking System**") is an authentic record of my own work as requirements of 8th semester academic during the period from **23th January 2024** to **30th June 2024** for the award of degree of B.Tech. (Computer Science & Engineering, Chandigarh Engineering College- CGC, Landran, Mohali under the guidance of (Mr. Rajeev Sharma).

(Signature of student) Prince Raj
2003239

Date: 27-05-
2024

Certified that the above statement made by the student is correct to the best of our knowledge and belief.

Signatures Examined by:

1.

2.

3.

4.

Head of Department (Signature and Seal)

ACKNOWLEDGMENT

I take this opportunity to express my sincere gratitude to the Director- Principal **Dr. Rajdeep Singh** Chandigarh Engineering College, Landran for providing this opportunity to carry out the present work.

I am highly grateful to the **Dr. Sukhpreet Kaur** HOD CSE, Chandigarh Engineering College, Landran (Mohali). I would like to express my gratitude to other faculty members of Computer Science & Engineering department of CEC, Landran for providing academic inputs, guidance & Encouragement throughout the training period. The help rendered by **Mr. Rajeev Sharma**, Supervisor for Experimentation is greatly acknowledged. I would also like to acknowledge the help of my Project Partner, **Mayank**, who helped me all along the building of this project. Finally, I express my indebtedness to all who have directly or indirectly contributed to the successful completion of my semester training.

ABOUT COMPANY

As a final semester student pursuing a degree in Computer Science & Engineering, I had the opportunity to gain valuable industry experience through a Six-month Semester Training program with **Grazziti Interactive**.

During my training, I received a certification in MERN Full Stack, which provided me with a strong foundation in programming and software development. However, my major project revolved around the development of an Room Banking System using MERN Full Stack. This project required extensive work on my part, collaborating with my project partner, to meet the requirements for our End-Semester Project. Our focus was on implementing various features such as user authentication, room listings, room booking, and search functionalities to create a robust and user-friendly platform for job seekers and employers. The project demanded independent effort outside of the Institute to ensure its successful completion.

Since 2008, Grazitti has been enabling business transformation with digital solutions leveraging cloud, mobile, and, social media. Grazziti helps businesses grow with Salesforce implementation & consulting, marketing automation, online communities, data analytics, web development, design, and more. As a global consultant, Grazitti collaborates with Adobe, Salesforce, Hubspot, Google, Alteryx, Amazon, Microsoft, Khoros, Optimizely, Acquia, Shopify, and other technology leaders to offer best-in-class solutions to our customers.

We combine the capabilities of these platforms with unique ideas to enable businesses to modify their technological approach and save millions. Grazziti is passionate about building digital solutions which leverage cloud, mobile, and social media technologies that solve not just the current business cases for clients but also meet their changing needs and scalability demand to match future growth strategies.

With offices in the US, India, Australia, Canada, and, Singapore, our growing customer base includes Fortune 100 companies, non-profit organizations, government agencies, and, medium to small-sized enterprises.

ABSTRACT

This project describes the development of a web-based room booking system designed to streamline the process of reserving rooms for various purposes. The system caters to both users seeking rooms and those managing the available spaces.

The system eliminates the need for manual booking processes, allowing users to seamlessly book rooms online. Real-time availability information ensures transparency and facilitates informed decision-making. Automated confirmations streamline communication between users and room managers, saving time and effort for everyone involved. Additionally, the system utilizes a distributed architecture for scalability and performance, with a centralized database guaranteeing data integrity and consistency. Security remains a top priority, with measures implemented to protect user data and prevent unauthorized access.

This room booking system offers a multitude of benefits for both users and administrators. Increased efficiency comes from eliminating manual processes, while improved transparency is achieved through real-time availability information. Enhanced communication is facilitated by automated confirmations, and for authorized users, the system provides a centralized platform for managing space allocation. Overall, this project contributes to a more efficient, user-friendly, and secure room booking experience.

TABLE OF CONTENT

CONTENT	PAGE NO
Certificate.....	i
Candidate Declaration.....	ii
Acknowledgement.....	iii
About Company.....	iv
Abstract.....	v
Table of Contents.....	vi
List of Figures.....	viii
List of Tables.....	ix
Chapter 1: Introduction.....	1
1.1 Brief Overview of Work.....	1
1.2 Objective.....	1
1.3 Scope.....	1
1.4 Project Modules.....	2
1.4.1 Registration.....	2
1.4.2 Rent and Lodging.....	2
1.4.3 Room Post.....	2
1.4.4 Manage Account.....	2
1.5 Project Requirements.....	3
1.5.1 Hardware.....	3
1.5.2 Software.....	3
Chapter 2: System Analysis.....	4
2.1 Literature Review.....	4
2.2 Project Feasibility Study.....	5

2.2.1 Technical Feasibility.....	5
2.2.2 Economical Feasibility.....	5
2.2.3 Operational Feasibility.....	5
2.4 Project Timeline Chart.....	6
2.5 Detailed Module Description with all Functionalities.....	7
2.5.1 Registration.....	7
2.5.2 Rent and Lodging.....	7
2.5.3 Room Post.....	7
2.5.4 Manage Account.....	7
Chapter 3: System Design.....	8
3.1 Use Case Diagrams.....	8
3.2 Data Flow Diagrams.....	10
3.3 Entity Relationship Diagram.....	13
3.4 Activity Diagram.....	14
3.5 Data Dictionary.....	16
Chapter 4: Software Tools.....	19
4.1 Overview of MERN Stack.....	19
4.1.1 Overview.....	19
4.1.2 Features of MERN.....	19
4.1.3 MERN and Internet.....	20
4.1.4 MERN And World Wide Web.....	21
4.1.5 MERN Environment.....	22
4.1.6 Node.js.....	22
4.1.7 Paradigm of MERN.....	23
4.2 About HTML.....	23
4.3 Java Script.....	28
4.4 Introduction to React.....	30

4.5 Introduction to MongoDB.....	46
Chapter 5: Implementation and Testing.....	48
5.1 User Interface and snapshots.....	52
5.2 Test Cases and Result.....	56
Chapter 6: Conclusion & Future work.....	59
References.....	61

List Of Figures

CONTENT	PAGE NO
Figure 2.1- Timeline chart.....	6
Figure 3.1- Use case Diagram.....	8
Figure 3.2.1- DFD 0 level Diagram.....	10
Figure 3.2.2- DFD 1 level Diagram.....	11
Figure 3.2.3- DFD 2 level Diagram.....	12
Figure 3.3- ER Diagram.....	13
Figure 3.4.1- Company Activity Diagram.....	14
Figure 3.4.2- Admin Activity Diagram.....	15
Figure 5.1- Home page I.....	52
Figure 5.2- Home page II.....	52
Figure 5.3- Registration Page.....	53
Figure 5.4- Login Form.....	53
Figure 5.5- Room Posting.....	54
Figure 5.6- Room Booking.....	54
Figure 5.7- User Profile.....	55
Figure 5.8- Room Information.....	55

List Of TABLES

CONTENT	PAGE NO
Table 1- User table.....	16
Table 2 – Room table.....	17
Table 3- Booking table.....	18
Table 4- Additional API table.....	43
Table 5- Session and Cookie table.....	45
Table 6- Test Case.....	56

Chapter 1: Introduction

1.1 Brief Overview of Work

In today's fast-paced world, efficiently managing space and resources is crucial. This project focuses on the development of a web-based room booking system that aims to streamline the process of reserving rooms for various purposes. This system caters to both users seeking rooms and those managing the available spaces. The system offers a user-friendly interface that simplifies the room booking process for users. Gone are the days of cumbersome manual reservations; users can now seamlessly search and book rooms online. This eliminates unnecessary back-and-forth communication and saves valuable time for both users and room managers.

1.2 Objective

The objective of this project is to develop a web-based room booking system that streamlines the process of reserving rooms for meetings, events, or individual use. This system aims to improve efficiency and convenience for both users and room managers. The system will provide users with a user-friendly interface to browse, search, and book available rooms based on their specific needs. This eliminates the need for manual booking processes, saving time and effort for everyone involved. For authorized users, the system may also offer functionalities for managing room details, including adding new rooms and potentially uploading images. This centralized platform allows for efficient space allocation and management.

1.3 Scope

The scope of this room booking system focuses on addressing the core functionalities involved in reserving rooms. This includes providing a user-friendly interface where users can browse available rooms based on specific criteria like date, duration, and desired room type. Advanced search and filtering options are also included to ensure users can find rooms that perfectly match their needs. Additionally, the system offers seamless online booking functionality with real-time availability checks and confirmation processes. For authorized users who manage room allocation, the system may offer functionalities for managing room details.

1.4 Project Modules

1.4.1 Registration and Login

For users seeking to book rooms, the system offers a straightforward registration process. Users will be required to provide valid details to create an account. This information might typically include contact details like phone number and email address, along with basic personal details for identification purposes.

The system may also accommodate authorized room managers who oversee room allocation and management. Their registration process might involve similar details as user registration, potentially with additional verification steps to confirm their authorization for managing rooms within the system. This ensures only authorized personnel can access room management functionalities.

1.4.2 Rent or Lodging

Users seeking long-term solutions can leverage the search function to find rooms ideal for renting. The system might allow users to specify criteria like desired room size, amenities, and budget range. For users requiring short-term accommodation, the system offers search options tailored to finding lodging options. Users could specify their desired dates of stay, potentially including check-in and check-out times. Filters might be available to narrow down choices based on room type (single, double, etc.), desired amenities, and budget constraints.

1.4.3 Room Post

Room managers can post Rooms And include various details regarding address, price, number of rooms available and snapshots of the room.

1.4.4 Manage Account

Users can also manage their accounts and see all their listings they have made and the rooms they have booked in the profile section.

1.5 Project Requirements

1.5.1 Hardware

The system requires the following hardware:

- RAM: 1 GB (further increase that as per requirement.)
- Hard Disk: 80 GB (further increase that as per requirement.)
- Display: 1024 * 768, True Type Color-32 Bit
- Mouse: Any Normal Mouse.
- Keyboard: Any window Supported Keyboard.

1.5.2 Software

- Database Server : MongoDB server
- Web Server : Internet Information Server
- Technologies : MERN: MongoDB, Express, React, Node

Chapter 2: System Analysis

2.1 Literature Review

Before delving into the specifics of this room booking system, it's valuable to acknowledge existing solutions in the realm of room reservation. Traditional methods often involve manual processes, such as contacting room coordinators via phone or email to inquire about availability and make reservations. These methods can be time-consuming and lack transparency regarding real-time availability.

The rise of technology has led to the development of various room booking systems. These web-based or software-based solutions offer a more streamlined approach. Some existing systems cater to specific industries, such as those focused on meeting room booking within office environments. Others might offer broader functionalities for booking various types of rooms across diverse settings.

This project aims to contribute to the landscape of room booking systems by offering a user-friendly and efficient solution. While acknowledging existing options, the focus remains on developing a system that addresses the specific needs and functionalities required for this project's scope.

Importance of Room Booking Systems

In today's fast-paced world, efficient space management is crucial for organizations of all sizes. Traditional methods of room booking, often relying on manual processes and communication, can be cumbersome and time-consuming. This can lead to confusion regarding room availability, double bookings, and wasted time for both room managers and users seeking spaces. Room booking systems offer a compelling solution to these challenges. By leveraging web-based technology, these systems provide a centralized platform for managing and reserving rooms.

2.2 Project Feasibility Study

2.2.1 Technical Feasibility

Technical feasibility assesses whether the project can be built using existing technologies and resources. This involves identifying the necessary hardware, software, and programming languages. The system will likely be a web-based application, requiring a server and a database to manage room information, user details, and booking data. The specific hardware and software requirements will depend on factors like the anticipated user base and the complexity of the functionalities offered. Choosing the right development tools and languages is crucial. Popular options include languages like Python, Java, or JavaScript frameworks like React or Angular. A critical aspect of technical feasibility is ensuring the system delivers outputs within an acceptable time frame. This necessitates optimizing code and server performance to avoid delays in page loading and user interactions.

2.2.2 Economical Feasibility

Economic feasibility evaluates the project's financial viability. This involves considering the costs associated with development, deployment, and ongoing maintenance. Development costs include software licenses, developer time, and potentially server infrastructure setup or cloud hosting fees. The economic benefits of the room booking system can be both tangible and intangible. Tangible benefits could include increased efficiency and reduced administrative costs associated with room booking.

2.2.3 Operational Feasibility

Operational feasibility examines whether the system can be effectively integrated into the existing workflows and infrastructure. The system's user interface and functionalities should be user-friendly and intuitive to encourage adoption by both users seeking rooms and authorized room managers. Seamless integration with existing organizational systems, such as user authentication databases or calendar tools, ensures data consistency and avoids disruptions to existing workflows.

2.3 Project Timeline Chart

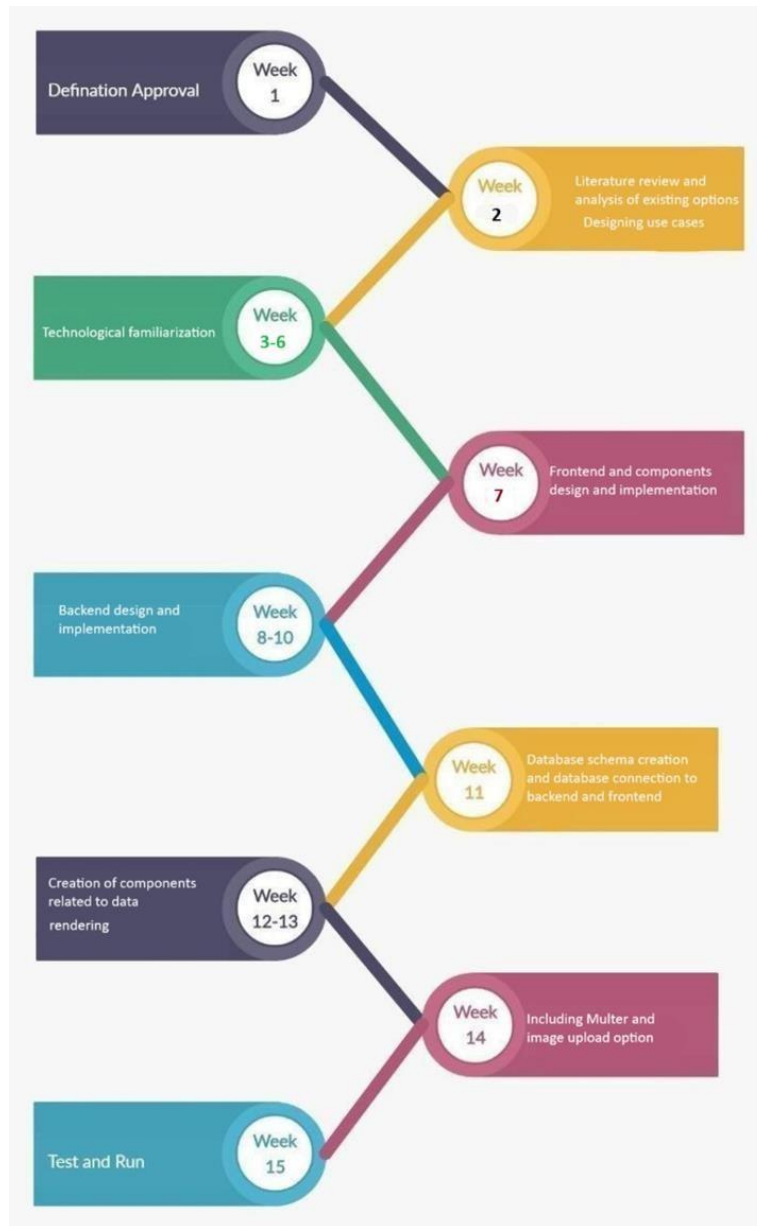


Figure 2.1- Timeline chart

The diagram outlines a software development timeline, for a fifteen-week period. It breaks down the process into phases including initial definition and planning, followed by front-end and back-end development activities. The final stages focus on database integration and testing the completed application.

2.4 Detailed Module Description with all Functionalities

2.4.1 Registration or Login

For users seeking to book rooms, the system offers a straightforward registration process. Users will be required to provide valid details to create an account. This information might typically include contact details like phone number and email address, along with basic personal details for identification purposes.

The system may also accommodate authorized room managers who oversee room allocation and management. Their registration process might involve similar details as user registration, potentially with additional verification steps to confirm their authorization for managing rooms within the system. This ensures only authorized personnel can access room management functionalities.

2.4.2 Rent or Lodging

Users seeking long-term solutions can leverage the search function to find rooms ideal for renting. The system might allow users to specify criteria like desired room size, amenities, and budget range. For users requiring short-term accommodation, the system offers search options tailored to finding lodging options. Users could specify their desired dates of stay, potentially including check-in and check-out times. Filters might be available to narrow down choices based on room type (single, double, etc.), desired amenities, and budget constraints.

2.4.3 Room Post

Authorized room managers can add new rooms to the system, specifying details like address, room type, capacity, and pricing. They can also set availability periods and upload images to showcase the room's features, giving users a clearer picture of the space they're considering.

2.4.4 Manage Account

The room booking system extends its functionality beyond just searching and booking rooms. A dedicated profile section empowers users to manage their account details and booking history. This section might allow users to update their contact information, change passwords, or set notification preferences. More importantly, the profile section acts as a central hub for users to track their booking activity. They can view a list of all rooms they've previously booked, allowing them to easily access booking details or revisit past reservations.

Chapter 3: System Design

3.1 Use Case Diagrams

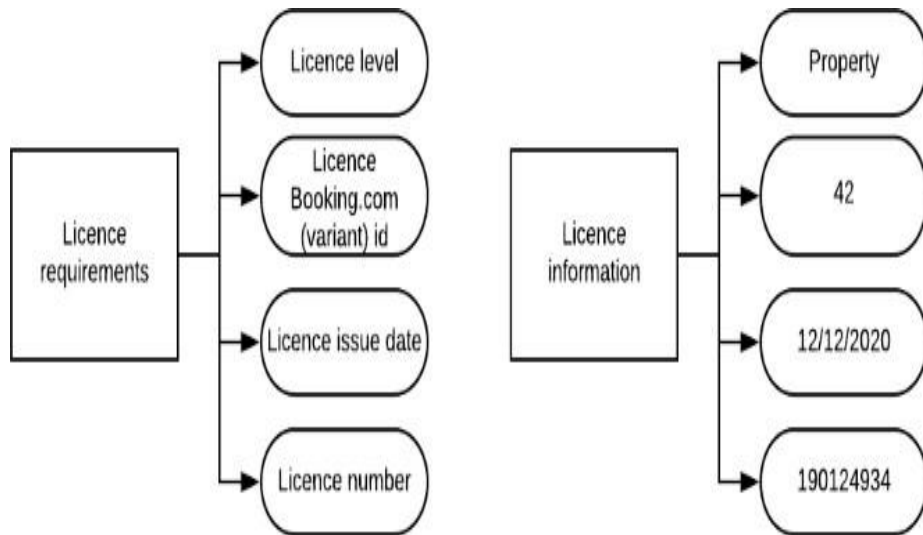


Figure 3.1.1 Licence Management system design

License Management System Design

The system incorporates a well-defined process for managing license requests. The process starts with users selecting the desired license level, which determines the specific requirements they need to fulfill. These requirements are outlined in the system. Once a user submits a request, a unique identifier is generated and linked to the application on the platform (Booking.com variant id). The system then assigns a license issue date upon approval. A unique license number is also generated and stored along with the property information associated with the license. This design ensures clear organization and traceability throughout the licensing process.

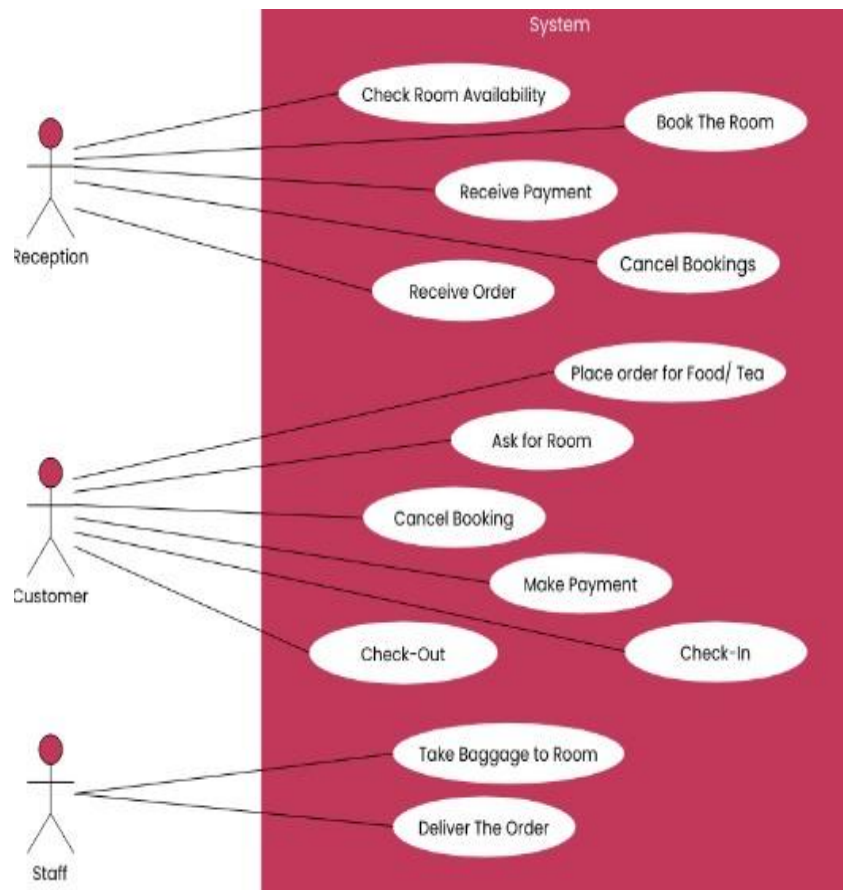


Figure 3.1.2- Use case diagram

Room Booking System Design Overview

The room booking system is designed to facilitate a user-friendly and efficient reservation process. The system leverages a web-based interface for users to search for available rooms based on their preferences, including dates, room type (e.g., single, double, suite), and desired amenities. Once a user finds a suitable room, they can proceed to the booking stage. This might involve creating an account or logging in to an existing one, followed by entering guest details and potentially selecting additional services. The system interacts with a backend server that manages user authentication, verifies room availability and pricing based on predefined rules, and processes booking requests. Upon confirmation, the user receives a booking reference number or confirmation email. The backend server also updates the room inventory to reflect the new reservation and communicates with the database to store all booking details securely.

3.2 Data Flow Diagrams

3.2.1 Context-Level (Level 0) DFD

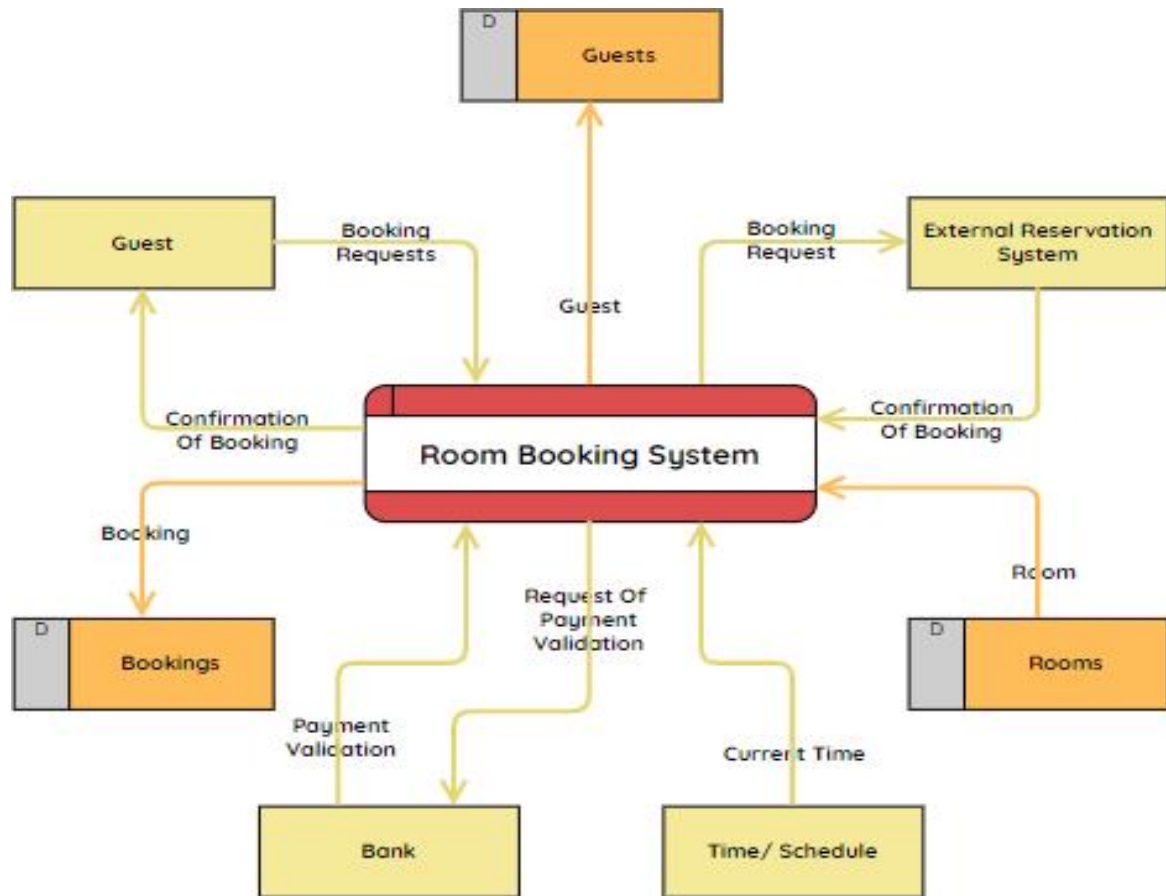


Figure 3.2- DFD 0 level Diagram

The provided Data Flow Diagram (DFD) offers a high-level view of the room booking system's data flow. It shows how external entities like guests and potentially external reservation systems interact with the core booking system. Guests submit reservation requests, the system checks availability and booking details in the database, and interacts with external systems if needed. Finally, the system confirms the reservation and sends a confirmation back to the guest. This DFD provides a foundational understanding of the system's external interactions and data flow.

3.2.2 Level 1 DFD

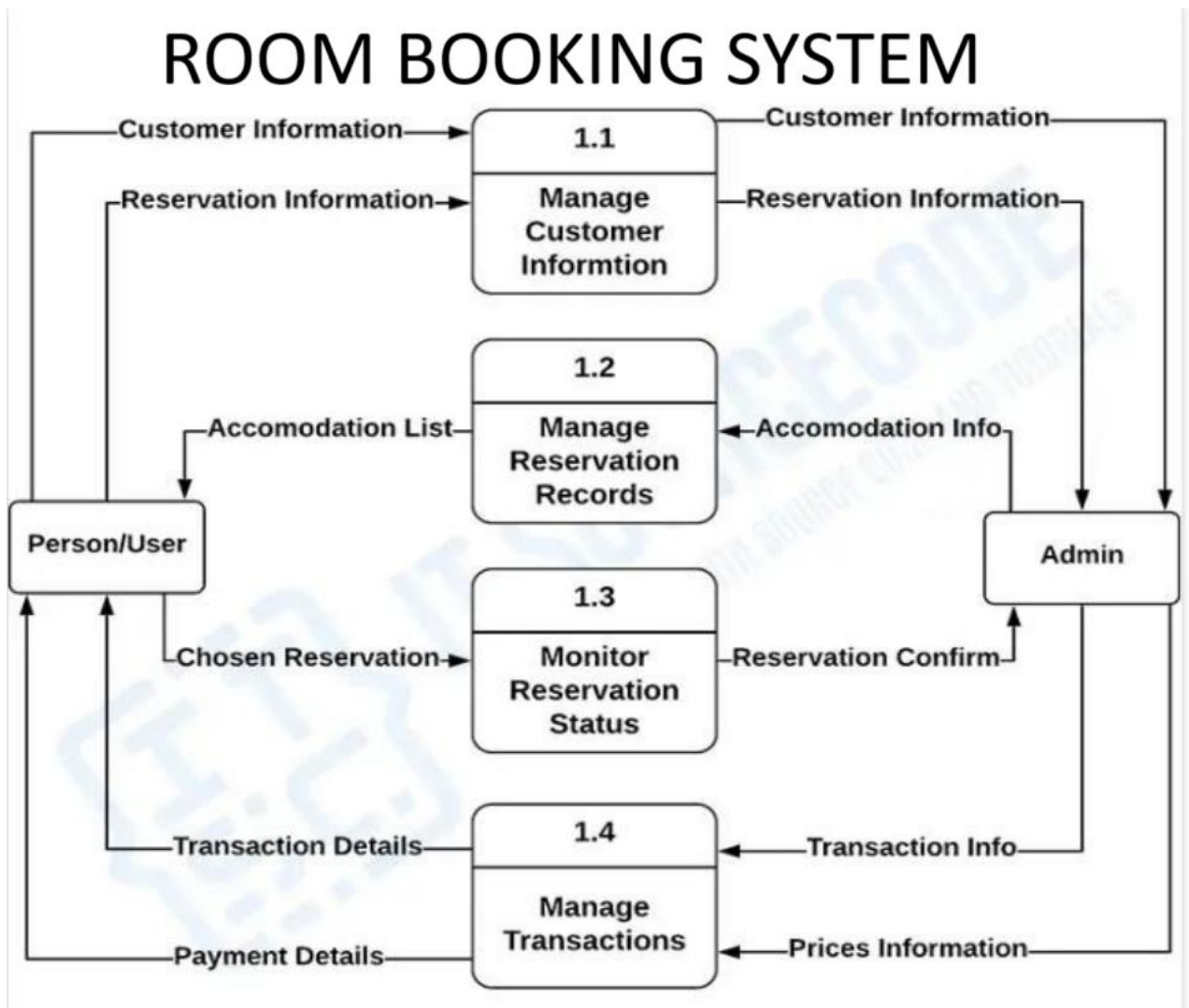


Figure 3.3- DFD 1 level Diagram

The level 1 DFD details the internal workings of the room booking system. Guests submit reservation requests with details like dates and room type. The system checks availability, validates guest information, and manages the reservation process. If a room is available, a reservation is created and a confirmation is sent to the guest. This level 1 DFD offers a clearer picture of the internal data flow and processes involved in handling room booking requests.

```

graph TD
    subgraph Modules
        M11[1.1 Manage Customer Information]
        M12[1.2 Manage Reservation Records]
        M13[1.3 Manage Reservation Records]
        M14[1.4 Manage Transactions]
    end

    subgraph Databases
        CD[Customer Database]
        CatD[Category Database]
        RD[Reservation Database]
        TD[Transaction Database]
    end

    M11 -- "Customer Information" --> CD
    CD -- "Reservation Info" --> M12
    M12 -- "Reservation Info" --> M13
    M13 -- "Reservation Details" --> M14
    M14 -- "Trasaction Details" --> M14
    M14 -- "Updates" --> TD
    TD -- "Confirmation" --> M13
    M13 -- "Reservation Info" --> RD
    RD -- "Category List" --> M11
    RD -- "Category List" --> M12
    M12 -- "Reservation Details" --> RD
    RD -- "Reservation Info" --> M11
    M11 -- "Customer Information" --> C[Customer]
    C -- "Category Chosen" --> M13
    C -- "Category List" --> M11
    Admin[Admin] -- "Transaction Information" --> M14
    Admin -- "Reservation Info" --> M12
    Admin -- "Category List" --> RD
    Admin -- "Reservation Details" --> RD
    
```

The level 2 DFD dives deeper into the room booking process. Guests provide details like dates, room type, and their information. The system checks availability, validates guest information, and manages the reservation. If a room is available, a reservation is created, considering promotions (if applicable), calculating rates, and sending a confirmation email with details like the reservation ID and total cost. Additionally, an optional flow can handle online payments through a transaction manager. This level 2 DFD offers a granular view of the internal data exchange and processes involved in managing room booking requests.

3.3 Entity Relationship Diagram

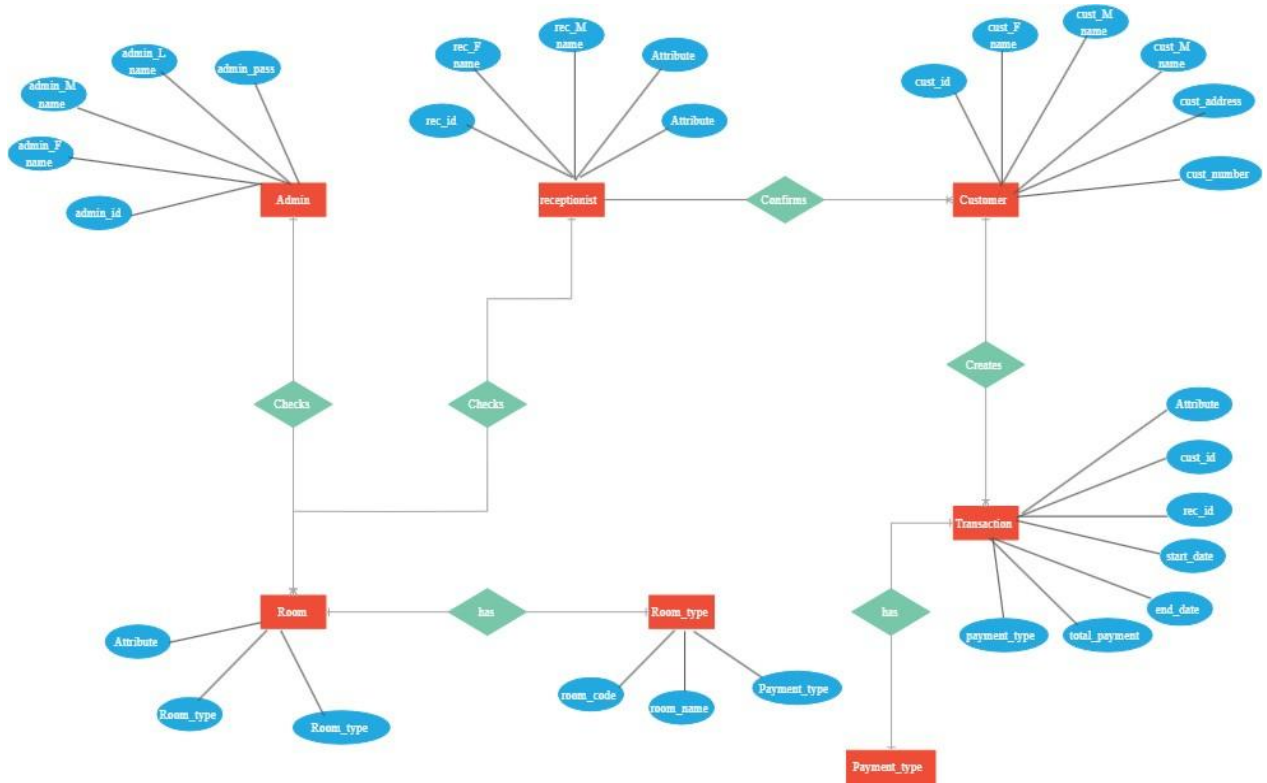


Figure 3.5- ER Diagram

The Entity-Relationship (ER) diagram depicts the entities and their relationships within the room booking system. The core entities likely include: **Guest:** This entity stores information about the guest making the reservation, including contact details and additional information. **Room:** This entity represents the rooms available for booking within the system. It likely captures attributes like room type (e.g.,rent, lodging), capacity etc.. **Reservation:** This entity represents a booking made by a guest. It likely stores details like the guest ID, room ID, reservation dates, total cost etc..

The relationships between these entities can be depicted using cardinality constraints. For example, a single Guest can have many Reservations (one-to-many), and a single Room can be included in many Reservations (one-to-many).

3.4 Activity Diagram

3.4.1 Company

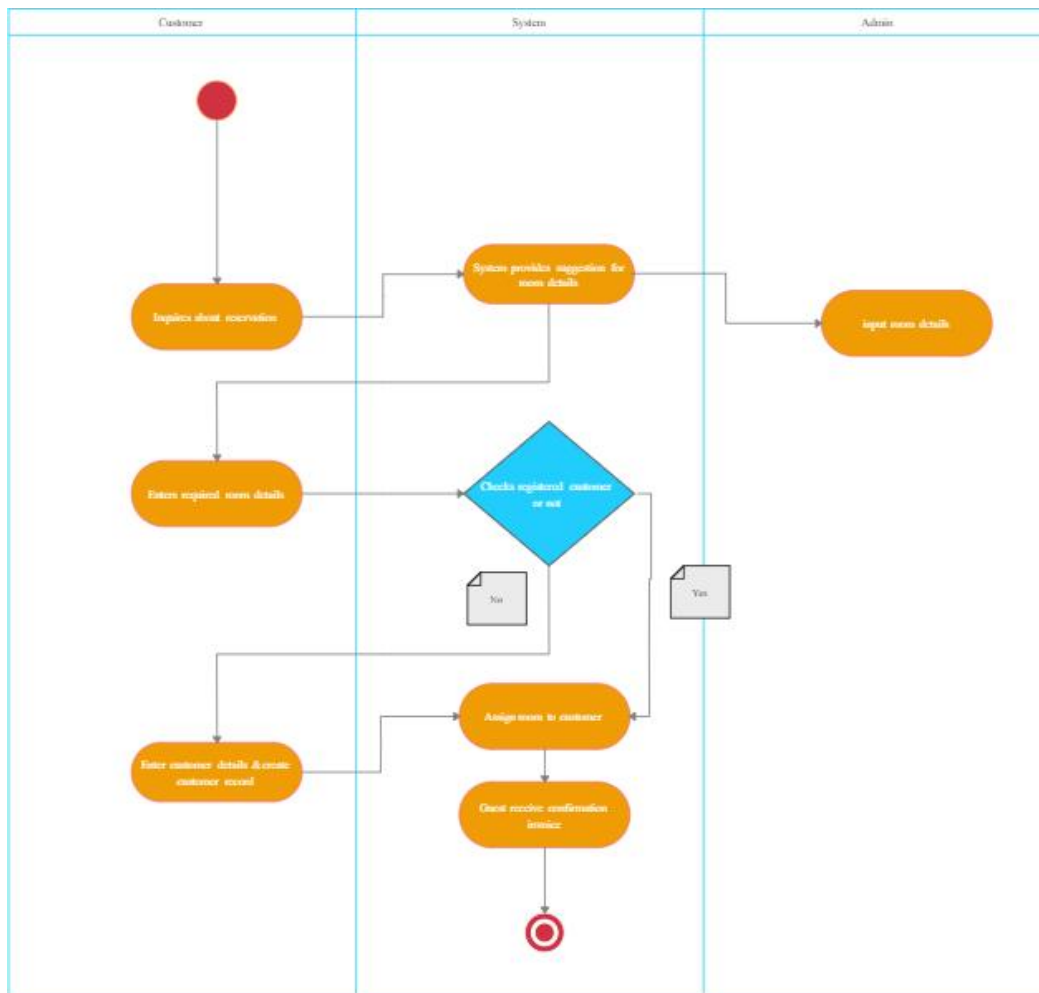


Figure 3.6- Employer Activity Diagram

The activity diagram illustrates the workflow and sequence of activities within the company. It likely depicts various departments or teams involved and how they interact with each other to achieve a specific goal or process. The diagram might show starting and ending points, decision points where different paths can be taken depending on specific conditions, and potentially parallel activities that can occur simultaneously. Arrows between activities indicate the flow of the process, and swimlanes can be used to visually group activities belonging to specific departments or roles.

3.4.2 Admin :-

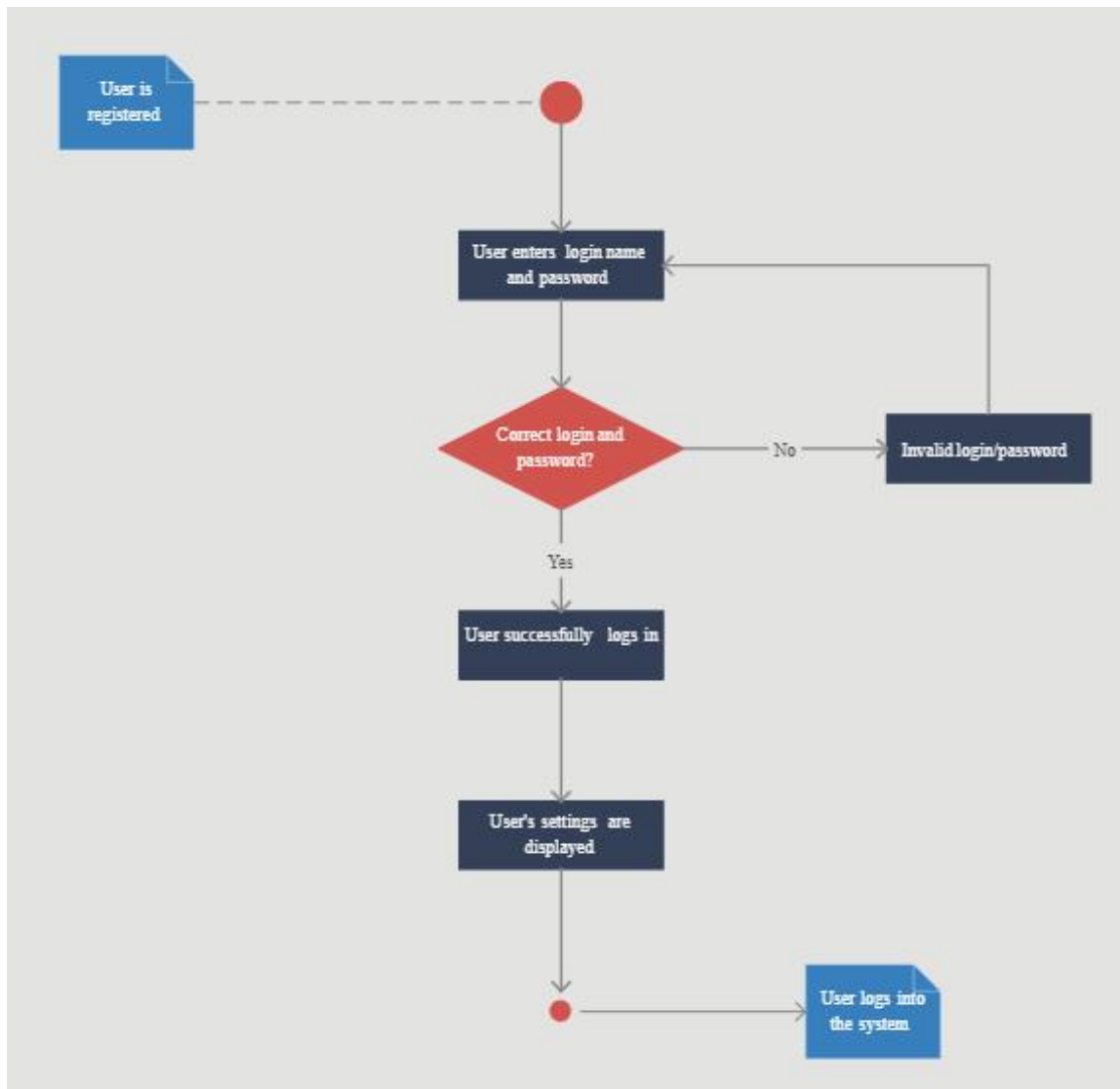


Figure 3.7- Admin Activity Diagram

The admin activity diagram depicts the workflow for administrative tasks within the system. It likely shows actions like user management (creating, editing, or deleting user accounts), content management (adding, editing, or removing system content), system configuration, and potentially data management or reviewing user requests. Analyzing this diagram helps visualize admin tasks, decision points, and how these activities contribute to overall system administration

3.5 Data Dictionary

User Table

This MongoDB schema outlines a blueprint for storing user data within a collection named "user." It encompasses essential fields for managing user accounts, including timestamps for record creation (createdAt), the user's full name (user_full_name), email address (user_email), and a crucial security consideration - the password (user_pwd). However, the schema strictly discourages storing passwords in plain text. Instead, it advocates for robust hashing algorithms like bcrypt or scrypt. These algorithms generate a one-way hash of the password, making it impossible to retrieve the original password even if the database is breached. This approach safeguards user credentials and enhances overall application security.

This schema serves as a solid foundation for user management in your MongoDB database. Remember to tailor it further based on your specific application's needs. You might consider adding additional fields like user roles, profile pictures, or preferences. Additionally, explore advanced features like data validation and indexing to optimize data integrity and query performance. By prioritizing security and flexibility, this schema offers a scalable solution for managing user information within your MongoDB environment.

Table1 – user table

Table 1:				
Name:		user		
Description:		Records information regarding user.		
Fields				
Sr. No.	Field Name	Field Type	Constraints	Description
1	createAt	Date	Null	Holds the user -id.
2	user_full_name	varchar(50)	Null	Holds full name of user.
3	user_email	varchar(50)	Null	Holds email id of user.
4	user_pwd	varchar(250)	Null	Holds password of user.

Room Table

Table 2- room table

Table 2:				
Name:		Room		
Description:		Records information about Room.		
Fields				
Sr. No.	Field Name	Field Type	Constraints	Description
1	room_id	int	primary key	Holds id of room.
2	room_name	Varchar(50)	foreign key	Holds id of room category.
3	room_desc	Text	Null	Holds description of room.
4	email	varchar(50)	Null	Holds city name of room.
5	Room_price	Int	Null	Holds Room Price
6	Room_address	varchar(50)	Null	Holds room Address
7	Room_type	varchar(50)	Null	Holds type of room
8	Room_image	Array	Null	Holds images of room
9	createAt	Date	Null	Holds Date when room is added

This MongoDB schema outlines a collection named "Room" for storing detailed information about rooms in a hospitality or rental application. It incorporates essential fields like room descriptions (room_desc), prices (room_price), addresses (room_address), and types (room_type). While the original schema proposed a foreign key for room categories (room_name), MongoDB doesn't directly translate them. However, relationships can be established through careful data modeling. Additionally, the schema offers flexibility with an array field (room_image) to store multiple images for each room. Notably, it avoids storing email addresses in the "Room" collection, suggesting a separate "User" collection for user information. This schema provides a solid foundation for managing room data, and you can customize it further by adding details like amenities, availability, or maximum occupancy to best suit your application's needs.

Booking Table

This MongoDB schema defines a collection named "Booking" to manage reservation details for your hospitality or rental application. It captures essential booking information like the user's email (email), booked room (room_name), and price (room_price) (potentially redundant if stored in the "Room" collection). The schema also includes user details like address (user_address), mobile number (mobile), and age (age). Notably, it stores potentially sensitive information like Aadhaar numbers (aadhar), requiring adherence to security best practices and regulations. While some fields might be redundant depending on your data modeling, this schema offers a foundation for managing bookings. You can extend it further by including booking duration, payment details, guest count, or special requests.

Table 3- booking table

Table 3:				
Name:		Booking		
Description:		Records information about booking.		
Fields				
Sr. No.	Field Name	Field Type	Constraints	Description
1	Name	varchar(50)	Null	Holds name of room.
2	Email	varchar(50)	Null	Holds email of user.
3	Room_name	varchar(50)	Null	Holds id of job.
4	Room_price	Int	Null	Holds Room Price
5	user_address	varchar(50)	Null	Holds user’s Address
6	mobile	int	Null	Holds mobile number
7	age	int	Null	Holds age of user
8	aadhar	Int	Null	Holds aadhar number
9	bookedAt	Date	Null	Holds Date when room is added

Chapter 4: Software Tools

4.1 Overview Of MERN Stack

4.1.1 Overview

Web development refers to the creating, building, and maintaining of websites. It includes aspects such as web design, web publishing, web programming, and database management. One of the most famous stack that is used for Web Development is MERN stack. This stack provides an end-to-end framework for the developers to work in and each of these technologies play a big part in the development of web applications.

MERN Stack is a JavaScript Stack that is used for easier and faster deployment of full-stack web applications. MERN Stack comprises of 4 technologies namely: MongoDB, Express, React and Node.js. It is designed to make the development process smoother and easier.

4.1.2 Features Of MERN

The MERN stack is a popular technology stack used for building modern web applications. It consists of four main components:

- **MongoDB:** MongoDB is a NoSQL database that stores data in a flexible, JSON-like format. It is a popular choice for web applications due to its scalability, flexibility, and ease of use. MongoDB is used as the database component of the MERN stack.
- **Express.js:** Express.js is a web application framework for Node.js. It provides a robust set of features for building web applications and APIs. Express.js simplifies the process of handling HTTP requests, routing, middleware integration, and more. It serves as the backend framework in the MERN stack.
- **React:** React is a JavaScript library for building user interfaces. It allows developers to create reusable UI components and efficiently update the UI in response to data changes. React follows a component-based architecture, which makes it easy to build complex UIs while maintaining a clear and modular codebase. React is used for building the frontend part of the MERN stack.
- **Node.js:** Node.js is a runtime environment that allows developers to run JavaScript code on the server-side. It uses an event-driven, non-blocking I/O model, which makes it lightweight and efficient for building scalable web applications. Node.js is used as the server-side runtime environment in the MERN stack.

Together, MongoDB, Express.js, React, and Node.js form the MERN stack, providing a full-stack JavaScript solution for building modern web applications. This stack is particularly popular for its flexibility, performance.

4.1.3 MERN And Internet

The MERN stack, consisting of MongoDB, Express.js, React, and Node.js, is a powerful combination for building dynamic and interactive web applications. When it comes to the internet, the MERN stack plays a significant role in shaping the landscape of web development and the overall user experience.

Here's how the MERN stack interacts with the internet:

Client-Server Communication: The MERN stack facilitates communication between the client-side and server-side components of web applications. React, as the frontend library, runs in the user's browser and handles the presentation layer of the application. It communicates with the backend, built with Express.js and Node.js, to fetch and manipulate data. This communication often occurs via HTTP requests and responses, allowing for dynamic content updates without the need for full page reloads.

Data Storage and Retrieval: MongoDB, as the database component of the MERN stack, plays a crucial role in storing and managing the application's data. MongoDB is a NoSQL database, which means it can handle unstructured and semi-structured data efficiently. This flexibility is particularly useful for web applications that deal with large volumes of diverse data types. Through MongoDB, applications can store user information, content, preferences, and more, providing a seamless experience for users as they interact with the application.

Real-Time Interaction: With the MERN stack, developers can implement real-time features in web applications, enhancing user engagement and interactivity. Node.js, with its event-driven architecture and non-blocking I/O capabilities, is well-suited for building real-time applications such as chat apps, collaboration tools, or live data dashboards.

Scalability and Performance: The MERN stack offers scalability and performance benefits, which are crucial for web applications aiming to handle a large number of concurrent users or rapidly growing datasets. Node.js allows for asynchronous and parallel processing, enabling web servers to handle multiple requests simultaneously without blocking the execution of other tasks. Additionally, MongoDB's horizontal scalability features make it well-suited for distributed environments, allowing applications to scale seamlessly as demand grows.

4.1.4 MERN And World Wide Web

The MERN stack, comprised of MongoDB, Express.js, React, and Node.js, is a modern technology stack used for building web applications. When we consider the World Wide Web (WWW), the MERN stack plays a pivotal role in shaping the development and functioning of web-based systems.

Here's how the MERN stack relates to the WWW:

Client-Server Architecture: The WWW operates on a client-server model, where web browsers (clients) communicate with remote servers hosting websites and web applications. The MERN stack facilitates the development of both client-side and server-side components required for this architecture.

Client-Side Interaction: React, one of the key components of the MERN stack, is responsible for handling the client-side interaction within web applications. React allows developers to create dynamic and interactive user interfaces, enabling smooth navigation, data manipulation, and real-time updates—all essential aspects of a modern web experience.

Server-Side Processing: Express.js and Node.js, the backend components of the MERN stack, handle server-side processing. Express.js provides a robust framework for building web servers and APIs, while Node.js serves as the runtime environment for executing JavaScript code on the server. Together, they enable developers to handle incoming requests from clients, process data, interact with databases (such as MongoDB), and generate appropriate responses, all of which are fundamental functionalities for serving web content over the WWW.

Data Management and Persistence: MongoDB, the database component of the MERN stack, plays a crucial role in managing and persisting data for web applications. MongoDB is a NoSQL database that stores data in a flexible, JSON-like format, making it well-suited for

handling the diverse and often unstructured data encountered in web applications. Through MongoDB, developers can store and retrieve information required for various functionalities of their web applications, such as user profiles, content, preferences, and more.

HTTP Protocol: The MERN stack interacts with the WWW through the HTTP (Hypertext Transfer Protocol) protocol. HTTP governs the communication between web clients and servers, allowing for the exchange of various types of data, including HTML documents, CSS stylesheets, JavaScript files, and other resources required for rendering web pages and executing web applications.

4.1.5 MERN Environment

The MERN (MongoDB, Express.js, React, Node.js) stack is a popular environment for developing full-stack web applications. MongoDB serves as the NoSQL database for storing data, Express.js is used for building the backend server and API endpoints, React handles the frontend user interface, and Node.js is the runtime environment for running JavaScript on the server side. Together, these technologies enable developers to create powerful and scalable web applications using JavaScript across the entire stack.

4.1.6 Node.js

Node.js is an open-source, cross-platform JavaScript runtime environment built on Chrome's V8 JavaScript engine. It allows developers to execute JavaScript code server-side, outside of a web browser. Here are some key points about Node.js:

Asynchronous and Event-Driven: Node.js uses an event-driven, non-blocking I/O model, which makes it lightweight and efficient for building scalable network applications. This means that Node.js can handle multiple concurrent connections without getting blocked by I/O operations, making it well-suited for building real-time applications.

JavaScript Everywhere: Node.js enables developers to use JavaScript for both client-side and server-side development, allowing for full-stack JavaScript development. This reduces the need to switch between different programming languages and ecosystems, streamlining the development process.

Vast Ecosystem: Node.js has a rich ecosystem of modules and packages available through npm (Node Package Manager), which is the largest ecosystem of open-source libraries in the world. Developers

an easily leverage existing npm packages or publish their own packages to share reusable code with the community.

Scalability: Node.js is designed to be scalable, allowing applications to handle a large number of concurrent connections efficiently. Its non-blocking I/O model and event-driven architecture make it well-suited for building highly scalable and performant applications, such as web servers, API servers, microservices, and real-time applications.

Community Support: Node.js has a large and active community of developers contributing to its development and ecosystem. This community-driven approach ensures continuous improvement, with regular updates and new features being added to the platform.

4.1.7 Paradigm Of MERN

- MongoDB: NoSQL, document-oriented database for flexible, scalable data storage.
- Express.js: Minimalist web framework for building robust backend APIs.
- React: Declarative, component-based library for building interactive user interfaces.
- Node.js: Event-driven, non-blocking I/O runtime for scalable server-side JavaScript execution.

4.2 About Html

HTML: The Building Blocks of Web Pages

HTML (HyperText Markup Language) is the fundamental language for creating web pages. It's not a programming language in the traditional sense (like JavaScript or Python) but a markup language. This means it uses tags to define the structure and content of a web page, instructing the web browser on how to display the information.

Key Concepts:

- **Tags:** HTML consists of special elements called tags, written as `<tag>` and `</tag>`. These tags enclose content and provide instructions to the browser. For example, `<h1>` and `</h1>` tags define a heading, while `<p>` and `</p>` tags define a paragraph.
- **Structure:** HTML elements provide structure to a web page. Elements can be nested to create a hierarchy, representing different sections of the content. This structure helps both browsers understand the page and users navigate it effectively.
- **Content:** The text, images, videos, and other elements that make up the actual content of the web page are placed within the HTML tags.

Beyond Formatting:

While HTML offers basic formatting capabilities (like bold, italics, headings), it's more than just a formatting language. HTML defines the core structure and content layout, allowing you to build interactive web pages.

Platform Independence:

A significant advantage of HTML is its platform independence. Web pages written in HTML are interpreted by web browsers, regardless of the operating system (Windows, macOS, Linux) the browser runs on. This universality is a cornerstone of the World Wide Web (WWW).

Hypertext and Links:

- **Hypertext:** HTML's name, "HyperText Markup Language," refers to its capability of linking related web pages together using hyperlinks. Hyperlinks are typically embedded within text or images. Clicking on a hyperlink takes the user to the linked web page.
- **Links:** Links are created using the `<a>` tag. The `href` attribute of the `<a>` tag specifies the URL of the linked page.

Examples of Early Web Browsers:

You mentioned Netscape and Internet Explorer as examples of early web browsers. These were indeed dominant players in the early days of the web (1990s and early 2000s). Today, popular web browsers include Chrome, Firefox, Safari, and Edge.

Beyond HTML: A Collaborative Effort

While HTML lays the foundation, it works in conjunction with other technologies like CSS (Cascading Style Sheets) for styling and JavaScript for interactivity to create the rich web experiences we're accustomed to today.

By understanding the core concepts of HTML, you can create the basic structure and content of web pages. From there, you can explore CSS and JavaScript to add visual appeal and dynamic behavior to your web creations.

HTML tags control in part the representation of the WWW page when view with web browser . Examples of browsers used to be web pages include:

- Netscape
- Internet Explorer

HTML: The Building Blocks of Web Pages

HTML (HyperText Markup Language) is the fundamental language for creating web pages. It's not a programming language in the traditional sense (like JavaScript or Python) but a markup language. This means it uses tags to define the structure and content of a web page, instructing the web browser on how to display the information.

Key Concepts:

- **Tags:** HTML consists of special elements called tags, written as `<tag>` and `</tag>`. These tags enclose content and provide instructions to the browser. For example, `<h1>` and `</h1>` tags define a heading, while `<p>` and `</p>` tags define a paragraph.
- **Structure:** HTML elements provide structure to a web page. Elements can be nested to create a hierarchy, representing different sections of the content. This structure helps both browsers understand the page and users navigate it effectively.
- **Content:** The text, images, videos, and other elements that make up the actual content of the web page are placed within the HTML tags.

Beyond Formatting:

While HTML offers basic formatting capabilities (like bold, italics, headings), it's more than just a formatting language. HTML defines the core structure and content layout, allowing you to build interactive web pages.

Platform Independence:

A significant advantage of HTML is its platform independence. Web pages written in HTML are interpreted by web browsers, regardless of the operating system (Windows, macOS, Linux) the browser runs on. This universality is a cornerstone of the World Wide Web (WWW).

Hypertext and Links:

- **Hypertext:** HTML's name, "HyperText Markup Language," refers to its capability of linking related web pages together using hyperlinks. Hyperlinks are typically embedded within text or images. Clicking on a hyperlink takes the user to the linked web page.

- **Links:** Links are created using the `<a>` tag. The `href` attribute of the `<a>` tag specifies the URL of the linked page.

Examples of Early Web Browsers:

You mentioned Netscape and Internet Explorer as examples of early web browsers. These were indeed dominant players in the early days of the web (1990s and early 2000s). Today, popular web browsers include Chrome, Firefox, Safari, and Edge.

Beyond HTML: A Collaborative Effort

While HTML lays the foundation, it works in conjunction with other technologies like CSS (Cascading Style Sheets) for styling and JavaScript for interactivity to create the rich web experiences we're accustomed to today.

By understanding the core concepts of HTML, you can create the basic structure and content of web pages. From there, you can explore CSS and JavaScript to add visual appeal and dynamic behavior to your web creations.

Structure and Content:

`<!DOCTYPE html>`: This declaration at the beginning of an HTML document specifies the document type (HTML5 in this case).

`<html>`: The root element of an HTML document that contains all other elements.

`<head>`: The section containing meta information about the document, such as the title, character encoding, and links to stylesheets.

`<title>`: Defines the title of the web page, displayed in the browser tab.

`<body>`: The section containing the visible content of the web page.

`<h1>` to `<h6>`: Heading tags define headings of different sizes, with `<h1>` being the most prominent.

`<p>`: Defines a paragraph of text.

`
`: Inserts a line break within a paragraph.

``: A generic inline container element used to group inline content (like styling a specific word).

`<div>`: A generic block-level element that defines a division or section within the content. Can contain other elements.

Text Formatting:

`` (deprecated): Makes text bold (use `` for semantic emphasis instead).

`<i>` (deprecated): Makes text italic (use `` for semantic emphasis instead).

``: Defines strong importance (bold emphasis).

``: Defines emphasized text (italic emphasis).

`<u>`: Underlines text (use CSS for styling instead, as underline can have accessibility issues).

`<sup>`: Superscript (text displayed slightly above the baseline).

`<sub>`: Subscript (text displayed slightly below the baseline).

Lists:

``: Defines an unordered list (bullet points).

``: Defines a list item within an unordered list.

``: Defines an ordered list (numbered items).

``: Defines a list item within an ordered list.

Links:

`<a>`: Defines a hyperlink (link) to another web page or resource. `href`: Attribute specifying the URL of the linked resource.

Images:

``: Defines an image element.

`src`: Attribute specifying the URL of the image file.

`alt`: Attribute providing alternative text for the image (important for accessibility).

Tables (less common in modern web design):

`<table>`: Defines a table structure

`<tr>`: Defines a table row.

`<td>`: Defines a table data cell.

`<th>`: Defines a table header cell.

Form Elements (used for user input):

<form>: Defines a form for collecting user input.

<input>: Defines an input field (text, password, email, etc.).

<button>: Defines a clickable button that can submit the form or trigger actions.

4.3 Java Script

Java script is a general purpose prototype based , object oriented scripting language developed jointly by sun and Netscape and is meant for the WWW . it is designed to be embedded in diverse applications and systems , with out consuming much memory . java script borrows most of its syntax from java but also inherits from AWK and PERL , with some indirect influence from self in its object prototype system.

Java scripts dynamically typed that is programs do not declare variable types, and the type of variable is unrestricted and can change at runtime. source can be generated at run time and evaluated against an arbitrary scope. Typical implementations compile by translating source into a specified byte code format, to check syntax and source consistency. Note that the availability to generate and interpret programs at runtime implies the presence of a compiler at runtime.

Java script is a high level scripting language that does not depend on or expose particular machine representations or operating system services. It provides automatic storage management, typically using a garbage collector.

Features:

- **Embedded in HTML:** JavaScript code is seamlessly integrated within HTML documents, allowing for dynamic manipulation of web pages. This means that the browser can interpret and execute JavaScript code directly, enhancing the user's experience beyond static HTML content.
- **Browser Dependence:** While JavaScript is standardized, some minor variations might exist across different web browsers. Developers need to consider these variations to ensure consistent behavior across platforms. Java script is an interpreted language that can be interpreted by the browser at run time.
- **Interpreted Language:** JavaScript is an interpreted language, meaning the browser translates the code line by line during runtime. This eliminates the need for a separate compilation step, making development faster and more iterative.

- **Loosely Typed:** Unlike some stricter languages, JavaScript doesn't require explicit declaration of variable types. This can provide flexibility but also introduces potential runtime errors if not handled carefully.
- **Object-Oriented:** JavaScript embraces object-oriented programming principles, allowing you to structure your code using objects and their properties and methods. This promotes modularity and code reusability.
- **Event-Driven:** JavaScript excels in event-driven programming. Events are specific actions, like clicking a button or hovering over an element. Event handlers are functions that define the actions triggered by these events. This enables interactive web pages that respond to user actions seamlessly.

Advantages

- **Client-Side Applications:** JavaScript empowers you to create client-side applications, which run within the user's web browser. This reduces the workload on the server, resulting in faster performance and a more responsive user experience.
- **Multi-Frame Windows:** JavaScript can manage multiple frames within a web page, allowing for complex layouts and presentations. This can be useful for displaying different sections of content or applications within a single window.
- **Interactive Forms:** JavaScript adds interactivity to forms by validating user input on the client-side. This can include checking login credentials, ensuring correct data formats, and preventing incomplete submissions. This improves data quality and reduces unnecessary server requests.
- **Reduced Network Traffic:** By performing basic data validation and other tasks on the client-side with JavaScript, the amount of data sent back and forth between the browser and server is minimized. This leads to faster loading times and a more efficient user experience.

One of JavaScript's greatest strengths lies in its event-driven nature. Events are user actions like clicking buttons or hovering over elements. JavaScript excels at defining event handlers – functions that dictate the behavior triggered by these events. This enables interactive web pages that respond seamlessly to user interactions, creating a more dynamic and engaging experience.

The advantages of JavaScript are numerous. It empowers developers to build client-side applications that run directly within the user's browser, reducing the workload on the server and

leading to faster performance. JavaScript also facilitates the use of multi-frame windows for complex layouts, and its ability to validate user input on the client-side improves data quality and reduces unnecessary server requests. By performing these tasks locally, JavaScript helps optimize web page performance and network traffic.

In essence, JavaScript's versatility and features make it an indispensable tool for creating rich and interactive web experiences. It simplifies development, enhances user engagement, and optimizes web page performance, making it a cornerstone of modern web development.

4.4 Introduction to React.js :

React is a JavaScript library for building dynamic user interfaces with reusable components. It simplifies UI development through a declarative approach, where developers describe how the UI should look and React handles the rendering efficiently. React's virtual DOM optimizes performance by only updating parts of the UI that have changed, resulting in faster and smoother applications. It's widely used in web development for creating modern, interactive, and scalable user interfaces.

React components are the fundamental building blocks of a React application. They are reusable pieces of code that define how a specific part of the UI (user interface) should look and behave. Here's a detailed explanation of components, props, and state in React:

Components

- **Function Components:** These are the most common type of component. They are simply JavaScript functions that return JSX (a syntax extension for JavaScript that resembles HTML). The JSX code describes the UI elements that make up the component.
- **Class Components** (less common): These are defined using a class syntax that allows for more complex component behavior with lifecycle methods. However, function components are generally preferred in modern React development due to their simpler syntax and better compatibility with hooks (explained later).

Props

- **Data Passing Mechanism:** Props are used to pass data (or information) from a parent component to its child components. They are like arguments passed to a function.
- **Read-Only:** Props are considered read-only within the receiving component. This means you cannot directly modify a prop's value within the child component.

- **Uni-directional Data Flow:** This concept ensures predictability and maintainability in your application by establishing a clear direction for data flow (parent to child).

State:

- **Internal Data Management:** State is used to manage internal data specific to a component. This data can change over time based on user interaction or other events.
- **Managed Within Component:** State is created and managed within the component itself. You cannot directly access or modify the state of another component.
- **Re-rendering Trigger:** When the state of a component changes, React automatically re-renders that component and all its children to reflect the updated data. This ensures the UI stays in sync with the underlying data.

Other Uses for Node.js :-

Node.js, beyond its typical usage as a server-side runtime for web applications, has several other use cases:

Command-Line Tools: Node.js is often used to build command-line tools and utilities. Its JavaScript ecosystem provides libraries and frameworks for building powerful command-line interfaces (CLIs) that automate tasks, manage dependencies, and interact with files and external services.

Desktop Applications: With frameworks like Electron, Node.js can be used to build cross-platform desktop applications. Electron combines Node.js with Chromium to create desktop apps using web technologies (HTML, CSS, JavaScript), allowing developers to leverage their web development skills to build native-like desktop applications.

IoT (Internet of Things): Node.js is well-suited for IoT development due to its lightweight and efficient nature. It can run on embedded devices with limited resources, making it ideal for collecting and processing sensor data, controlling devices, and building IoT applications that connect hardware to the web.

Real-Time Applications: Node.js is commonly used for building real-time applications such as chat apps, collaboration tools, and online gaming platforms. Its event-driven architecture and non-blocking I/O capabilities make it efficient for handling multiple concurrent connections and delivering real-time updates to clients.

API Servers: Node.js is frequently used to build API servers for mobile apps, web applications, and microservices architectures. Its lightweight and scalable nature make it suitable for handling HTTP requests, processing data, and interacting with databases to provide backend service.

Data Streaming: Node.js excels in handling data streaming applications such as multimedia streaming, file uploads/downloads, and real-time data processing. Its non-blocking I/O model allows it to efficiently handle streaming data, making it suitable for applications that deal with large volumes of data in real-time.

Architecture of the Server Package

The `server` package provides interfaces and classes for writing servlets. The architecture of the package is described below.

The Servlet Interface

The central abstraction in the Servlet API is the `Servlet` interface. All servlets implement this interface, either directly or, more commonly, by extending a class that implements it such as `HTTPServlet`.

The `Servlet` interface declares, but does not implement, methods that manage the servlet and its communications with clients. Servlet writers provide some or all of these methods when developing a servlet.

The MERN Request Interface

Client-Side Requests:

In React (the frontend), requests are typically made using methods like `fetch()` or libraries like `Axios` to communicate with the backend.

These requests can be used to fetch data from APIs, send form submissions, or perform other actions requiring server interaction.

Requests are usually asynchronous, meaning they don't block the execution of other code while waiting for a response.

The Servlet Response Interface

Server-Side Response Handling: Upon receiving a request, the server-side code (Node.js with `Express.js`) processes it and generates a response. The response typically includes status codes (e.g.,

200 for success, 404 for not found), headers, and optionally, data in JSON or other formats.

Data Transfer: Responses often include data fetched from a database (MongoDB) or generated dynamically by the server-side code. Data transfer typically occurs in structured formats like JSON, making it easy for the client-side (React) to parse and use the data.

Additional Capabilities of MERN Servlets

MERNServlets could refer to a hypothetical combination of the MERN stack with Java Servlets, a technology typically associated with Java-based web development. While the MERN stack primarily utilizes JavaScript-based technologies for building web applications, incorporating Java Servlets into the stack would introduce additional capabilities:

Enterprise-Grade Capabilities: Java Servlets, being a part of the Java Enterprise Edition (Java EE) ecosystem, bring enterprise-grade capabilities to the stack. This includes features such as transaction management, security, scalability, and integration with other Java EE technologies like Enterprise JavaBeans (EJB) and Java Persistence API (JPA).

Legacy System Integration: Java Servlets can facilitate integration with existing legacy systems written in Java or other languages. This is particularly useful in enterprise environments where there is a need to interface with legacy systems while modernizing the frontend using technologies like React.

Advanced Server-Side Processing: Java Servlets provide advanced server-side processing capabilities, allowing developers to implement complex business logic, data processing, and session management on the server side. This can be advantageous for applications with stringent performance, security, and regulatory requirements.

Concurrency and Multithreading: Java Servlets leverage the concurrency and multithreading capabilities of the Java Virtual Machine (JVM), enabling efficient handling of multiple concurrent requests. This can improve the scalability and responsiveness of web applications, especially in scenarios with high traffic or intensive processing requirements.

Initializing a Servlet

When a server loads a servlet, the server runs the servlet's `init` method. Initialization completes before client requests are handled and before the servlet is destroyed.

Even though most servlets are run in multi-threaded servers, servlets have no concurrency issues during servlet initialization. The server calls the `init` method once, when the server loads the servlet, and will not call the `init` method again unless the server is reloading the servlet.

The server can not reload a servlet until after the server has destroyed the `SERVLET` by calling the

`destroy` method.

The `init` Method

- The `init` method provided by the `HttpServlet` class initializes the servlet and logs the initialization. To do initialization specific to your servlet, override the `init()` method following these rules:
- If an initialization error occurs that renders the servlet incapable of handling client requests, throw an `UnavailableException`.
- An example of this type of error is the inability to establish a required network connection.
- Do not call the `System.exit` method

Initialization Parameters

- The second version of the `init` method calls the `getInitParameter` method. This method takes the parameter name as an argument and returns a `String` representation of the parameter's value.
- The specification of initialization parameters is server-specific. In the Java Web Server, the parameters are specified with a servlet is added then configured in the Administration Tool. For an explanation of the Administration screen where this setup is performed, see the Administration Tool: Adding Servlets online help document. If, for some reason, you need to get the parameter names, use the `getParameterNames` method.

Destroying a Servlet

Servlets run until the server destroys them, for example at the request of a system administrator. When a server destroys a servlet, the server runs the servlet's `destroy` method. The method is run once; the server will not run that servlet again until after the server reloads and reinitializes the servlet. When the `destroy` method runs, another thread might be running a service request. The Handling Service Threads at Servlet Termination section shows you how to provide a clean shutdown when there could be long-running threads still running service requests.

Using the Destroy Method:

The `destroy` method provided by the `HttpServlet` class destroys the servlet and logs the destruction. To destroy any resources specific to your servlet, override the `destroy` method. The `destroy` method should undo any initialization work and synchronize persistent state with the current in-memory state.

The following example shows the `destroy` method that accompanies the [init](#) method shown previously:

```
Public class BOOKDBSERVLET extends GENERICSERVLET
{
    Private BOOKSTORE DB books;

    ...
    ...
    ...
    // the init method

    Public void destroy() {

        // Allow the database to be garbage collected books = null;

    }
}
```

A server calls the `destroy` method after all service calls have been completed, or a server-specific number of seconds have passed, whichever comes first. If your servlet handles any long-running operations, service methods might still be running when the server calls the `destroy` method. You are responsible for making sure those threads complete. The next

section shows you how the `destroy` method shown above expects all client interactions to be completed when the `destroy` method is called, because the servlet has no long-running operations.

Event Listener in Javascript –

In JavaScript, the `addEventListener()` function is used to attach event handlers to HTML elements, allowing developers to respond to various user interactions such as clicks, mouse movements, keyboard presses, and more. Here's a brief overview of how `addEventListener()` works: `element.addEventListener(event, function, useCapture);`

Tracking Service Requests

To track service requests, include a field in your servlet class that counts the number of service methods that are running. The field should have access methods to increment, decrement, and return its value.

The `service` method should increment the service counter each time the method is entered and decrement the counter each time the method returns. This is one of the few times that your `HttpServlet` subclass should override the `service` method. The new method should call `super.service` to preserve all the original `HttpServlet.service` method's functionality.

Providing a Clean Shutdown

To provide a clean shutdown, your `destroy` method should not destroy any shared resources until all the service requests have completed. One part of doing this is to check the service counter. Another part is to notify the long-running methods that it is time to shut down. For this, another field is required along with the usual access methods.

Servlet-client Interaction

For MERN (MongoDB, Express.js, React, Node.js) stack development, the concepts of servlets and HTTP interactions don't directly apply, as it's a JavaScript-based technology stack primarily used for building web applications. However, we can adapt the provided information to fit the context of MERN stack development:

Requests and Responses

Methods in the `nodejs` class that handle client requests take two arguments:

- An `req` object, which encapsulates the data *from* the client
- An `res` object, which encapsulates the response *to* the client

Express.js Request Object:

Purpose: The request object in Express.js encapsulates all the information about an incoming HTTP request, including the URL, query parameters, request body, headers, and more.

Accessing Request Data: Within an Express.js route handler, you can access the request object as the first parameter (`req` by convention).

Common Properties and Methods:

`req.url`: Returns the URL of the request.

`req.params`: Returns route parameters captured in the URL pattern. `req.query`: Returns an object containing the query string parameters.

`req.body`: Returns the parsed body of the request (for requests with a body, like POST or PUT requests).

`req.headers`: Returns an object containing the request headers.

`req.method`: Returns the HTTP method of the request (e.g., GET, POST, PUT, DELETE).

Express.js Response Object:

Purpose: The response object in Express.js represents the HTTP response that the server sends back to the client in response to an incoming request.

Accessing Response Object: Within an Express.js route handler, you can access the response object as the second parameter (`res` by convention).

Common Methods:

`res.send()`: Sends the HTTP response. The method automatically sets the appropriate content type based on the data being sent (e.g., JSON, HTML, plain text).

`res.json()`: Sends a JSON response. It automatically sets the content type to `application/json`.

`res.status()`: Sets the HTTP status code of the response.

`res.setHeader()`: Sets an individual header in the response.

`res.redirect()`: Redirects the client to a different URL. `res.sendFile()`: Sends a file as an HTTP response.

`res.render()`: Renders a view template using a specified template engine.

HTTP Header Data

In the MERN (MongoDB, Express.js, React, Node.js) stack, handling header data typically occurs on the server-side within Express.js and client-side within React. Here's an overview of how header data is managed in the MERN stack:

Express.js (Server-side):

Setting Response Headers: In Express.js, you can set response headers using the `set()` method on the response object (`res`). This method allows you to define custom headers to be sent along with the HTTP response.

React (Client-side):

Setting Request Headers: In React, when making HTTP requests using libraries like Axios or fetch, you can include custom headers by passing an options object with a `headers` property.

Handling GET and POST Requests

The methods to which the `service` method delegates HTTP requests include,

- `DOGET`, for handling GET, conditional GET, and HEAD requests
- `DOPOST`, for handling POST requests
- `DOPUT`, for handling PUT requests
- `DODELETE`, for handling DELETE requests

By default, these methods return a `BAD_REQUEST` (400) error. Your servlet should override the method or methods designed to handle the HTTP interactions that it supports. This section shows you how to implement methods that handle the most common HTTP requests: GET and POST.

The `HTTPServlet`'s `service` method also calls the `doOptions` method when the servlet receives an `OPTIONS` request, and `doTrace` when the servlet receives a `TRACE` request. The default implementation of `doOptions` automatically determines what `HTTPOptions` are supported and returns that information. The default implementation of `doTrace` causes a response with a message containing all of the headers sent in the trace request. These methods are not typically overridden.

About Session Tracking

Session tracking is a flexible, lightweight mechanism that enables `STATEFUL` programming on the web. Its general implementation serves as a basis for more sophisticated state models, such as persistent user profiles or multi-user sessions.

A session is a series of requests from the same user that occur during a time period. This transaction model for sessions has many benefits over the single-hit model. It can maintain state and user identity across multiple page requests. It can also construct a complex overview of user behavior that goes beyond reporting of user hits.

Server-Side Session Objects and Users

Session tracking gives servlets and other server-side applications the ability to keep state information about a user as the user moves through the site. Server-side applications can use this facility to create more `STATEFUL` user experiences and to track who's doing what on the site.

Java Web Server maintains user state by creating a `Session` object for each user on the site. These `Session` objects are stored and maintained on the server. When a user first makes a request to a site, the user is assigned a new `Session` object and a unique session ID. The session ID matches the user with the `Session` object in subsequent requests. The `Session` object is then passed as part of the request to the servlets that handle the request. Servlets can add information to `Session` objects or read information from them.

Session Endurance

After the user has been idle for more than a certain period of time (30 minutes by default), the user's session becomes invalid, and the corresponding Session object is destroyed.

A session is a set of requests originating from the same browser, going to the same server, bounded by a period of time. Loosely speaking, a session corresponds to a single *sitting* of a single anonymous user (anonymous because no explicit login or authentication is required to participate in session tracking).

The first part of the `DOGET` method associates the Session object with the user making the request. The second part of the method gets an integer data value from the Session object and increments it. The third part outputs the page, including the current value of the counter.

When run, this servlet should output the value of the counter that increments every time you reload the page. You must obtain the Session object before you actually write any data to the `SERVLET'S` output stream. This guarantees that the session tracking headers are sent with the response.

The Session object has methods similar to `JAVA.UTIL.DICTIONARY` for adding, retrieving, and removing arbitrary Java objects. In this example, an Integer object is read from the Session object, incremented, then written back to the Session object.

Any name, such as `sessiontest.counter`, may be used to identify values in the Session object. When choosing names, remember that the Session object is shared among any servlets that the user might access. Servlets may access or overwrite each other's values from the Session.

Thus, it is good practice to adopt a convention for organizing the namespace to avoid collisions between servlets, such as: `servletname.name`.

Session Invalidation

Sessions can be invalidated automatically or manually. Session objects that have no page requests for a period of time (30 minutes by default) are automatically invalidated by the Session Tracker `SESSIONINVALIDATIONTIME` parameter. When a session is invalidated, the Session object and its contained data values are removed from the system.

After invalidation, if the user attempts another request, the Session Tracker detects that the user's session was invalidated and creates a new Session object. However, data from the user's previous session will be lost.

Session objects can be invalidated manually by calling `session.invalidate()`. This will cause the session to be invalidated immediately, removing it and its data values from the system.

Handling Non-Cookie Browsers (URL Rewriting)

The Session Tracker uses a session ID to match users with Session objects on the server side. The session ID is a string that is sent as a cookie to the browser when the user first accesses the server. On subsequent requests, the browser sends the session ID back as a cookie, and the server uses this cookie to find the session associated with that request.

There are situations, however, where cookies will not work. Some browsers, for example, do not support cookies. Other browsers allow the user to disable cookie support. In such cases, the Session Tracker must resort to a second method, URL rewriting, to track the user's session.

URL rewriting involves finding all links that will be written back to the browser, and rewriting them to include the session ID. For example, a link that looks like this:

`` might be rewritten to look like this:

``

If the user clicks on the link, the rewritten form of the URL will be sent to the server. The server's Session Tracker will be able to recognize the `;$sessionId$DA32242SSGE2` and extract it as the session ID. This is then used to obtain the proper Session object.

Implementing this requires some reworking by the servlet developer. Instead of writing URLs straight to the output stream, the servlet should run the URLs through a special method before sending them to the output stream.

The `ENCODEURL` method performs two functions:

1. **Determine URL Rewriting:** The `ENCODEURL` method determines if the URL needs to be rewritten. Rules for URL rewriting are somewhat complex, but in general if the server detects that the browser supports cookies, then the URL is not rewritten. The server tracks information indicating whether a particular user's browser supports cookies.

2. **Return URL (modified or the same):** If the `encodeurl` method determined that the URL needs to be rewritten, then the session ID is inserted into the URL and returned. Otherwise, the URL is returned unmodified.

now do this:

```
response.Sendredirect (response.Encode redirecturl ("http://myhost/store/catalog"));
```

The methods `ENCODEURL` and `ENCODEREDIRECTURL` are distinct because they follow different rules for determining if a URL should be rewritten.

Multiple Servlets

URL conversions are required only if the servlet supports session tracking for browsers that do not support cookies or browsers that reject cookies. The consequences of not doing these conversions is that the user's session will be lost if the user's browser does not support cookies and the user clicks on an un-rewritten URL. Note that this can have consequences for other servlets. If one servlet does not follow these conventions, then a user's session could potentially be lost for all servlets.

Using Session Tracking with the Page Compiler

Page compilation is a feature of the Java Web Server that allows HTML pages containing Java code to be compiled and run as servlets. Page compilation also simplifies the task of supporting session tracking. To that end, if URL rewriting is enabled, page compilation automatically adds the `encodeurl` call to links in the HTML page.

Additional APIs

In addition to the Session object, there are a few more classes that may interest the servlet developer.

Additional APIs	
Description	Class
<code>httpsessioncontext</code>	The <code>httpsessioncontext</code> is the object that contains all existing and valid sessions. The <code>http session context</code> can be obtained by calling <code>get Session Context ()</code> on the Session object. The <code>Http Session Context</code> lets you find other Session objects by their IDs and list the IDs of all valid sessions.
<code>HttpSessionBindingListener</code>	<code>HttpSessionBindingListener</code> is an interface that can be implemented by objects placed into a Session. When the Session object is invalidated, its contained values are also removed from the system. Some of these values may be active objects that require cleanup operations when their session is invalidated. If a value in a Session object implements <code>HttpSessionBindingListener</code> , then the value is notified when the Session is invalidated, thereby giving the object a chance to perform any necessary cleanup operations.

Table 4.1 additional APIs

Session Swapping and Persistence

An Internet site must be prepared to support many valid sessions. A large site, for example, might have hundreds, or even thousands, of simultaneously valid sessions.

Because each session can contain arbitrary data objects placed there by the application servlets, the memory requirements for the entire system can grow prohibitively large.

To alleviate some of these problems, the session tracking system places a limit on the number of Session objects that can exist in memory. This limit is set in the `session.maxresidents` property. When the number of simultaneous sessions exceeds

this number, the Session Tracker swaps the least recently-used sessions out to files on disk. Those sessions are not lost: they will be reloaded into memory if further requests come in for those sessions. This system allows for more sessions to remain valid than could exist in memory.

Session invalidation is not affected by session swapping. If a session goes unused for longer than the normal invalidation time, the session is invalidated, whether it is in memory or on disk. Session invalidation is set in the `session.invalidationinterval` property.

Sessions are written to and read from disk using Java serialization. For this reason, only serializable objects put into the Session object will be written to disk. Any objects put into the Session object that are not serializable will remain in memory, even if the rest of the Session object has been written to disk. This does not affect session tracking, but does reduce the memory savings that the Session Tracker gets from swapping a session to disk. For this reason, the servlet developer should try to put only serializable objects into the Session object. Serializable objects are those that implement either `java.io.Serializable` or `java.io.Externalizable`.

The session-swapping mechanism is also used to implement session persistence, if the session persistence feature is enabled. When the server is shut down, sessions still in memory are written to the disk as specified in the `session.swapdirectory` property. When the server starts again, sessions that were written to disk will once again become valid. This allows the server to be restarted without losing existing sessions. Only serializable data elements in the session will survive this shutdown/restart operation.

Customizing Session Tracking

Session-tracking interfaces are in the `node.servlet.http` package.

Properties

You can customize properties in the Session Tracker. The properties are kept in the `server.properties` files

at: `<server_root>/properties/server/javawebserver/server.properties` where `<server_root>` is the directory into which you installed the Java Web Server product.

Note: These property settings are applied to all sessions, and cannot be tuned for individual sessions.

Sessions and Cookies Table:

Session and Cookies in Node.js	
Parameter	Description
req.session	This property on the request object provides access to the current session data. It's an object where you can store key- value pairs of information.
req.session.get(key)	Retrieves the value associated with a specific key from the session object.
req.session.set(key, value)	In Node.js session management libraries like express-session, setting a key-value pair in the session object allows you to store temporary information specific to a user's interaction with your application.
Session Middleware	The express-session library provides middleware that attaches a session object to the request object. This allows you to access and manipulate session data within your application routes.
Setting Cookies	<p>You can use the <code>res.cookie(name, value, options)</code> method within your Express routes to set cookies.</p> <ol style="list-style-type: none">1. name: The name of the cookie (identifier).2. value: The data you want to store in the cookie.3. options (Optional): Configuration options for the cookie, such as <code>maxAge</code> (expiration time), <code>httpOnly</code> (disallow client-side JavaScript access), <code>secure</code> (requires HTTPS), and <code>domain</code> (restrict cookie access to specific domains).
Accessing Cookies	You can access cookies sent by the client using the <code>req.cookies</code> object within your route handlers. This object is a key-value map containing all cookies sent in the request header.

<code>res.cookie(name, value, options)</code>	Sets a cookie with the specified name, value, and options.
<code>req.cookies</code>	An object containing all cookies sent by the client in the request header. Retrieve a specific cookie using <code>req.cookies.name</code> .
Session Expiration	Sessions have a defined lifespan (configurable in <code>express-session</code>). If the user remains inactive for that duration, the session expires, and the session object is discarded from storage. The corresponding session cookie automatically becomes invalid.

Table 4.2 Sessions and Cookies in Node.js

Introduction to MongoDB :

MongoDB is a popular open-source NoSQL database management system designed for the modern era of application development. Unlike traditional relational databases, MongoDB stores data in flexible, JSON-like documents with dynamic schemas. This schema flexibility allows developers to easily iterate and evolve their data models as application requirements change.

MongoDB is well-suited for handling unstructured or semi-structured data and is particularly advantageous for use cases such as real-time analytics, content management, mobile applications, and IoT (Internet of Things) platforms.

Distinct Features Of MongoDB:-

- **Document-Oriented:** MongoDB stores data in JSON-like documents called BSON (Binary JSON), which are schema-less and can vary in structure from one document to another within the same collection. This flexibility simplifies the development process and allows for easy representation of complex hierarchical relationships.
- **High Scalability:** MongoDB is designed for horizontal scalability, allowing it to scale out across multiple servers or clusters to handle large volumes of data and high throughput. It supports sharding, which partitions data across multiple machines, and replica sets for high availability and fault tolerance.

- **Performance:** MongoDB offers high-performance read and write operations by using memory-mapped files for storage and employing indexes, query optimization, and aggregation pipelines to efficiently process queries and data manipulations.
- **Ad Hoc Queries :** MongoDB supports rich query capabilities, including ad hoc queries, range queries, text searches, and geospatial queries. Developers can use MongoDB's powerful query language, which is similar to JSON, to perform complex data retrieval operations.
- **Aggregation Framework:** MongoDB provides an expressive aggregation framework for performing data aggregation operations such as grouping, sorting, and transforming documents. This allows for complex data analysis and reporting directly within the database.
- **Flexible Schema Design:** MongoDB's schema flexibility enables agile development practices, allowing developers to iterate quickly and adapt to changing requirements without needing to modify existing schema definitions.
- **Automatic Sharding:** MongoDB's built-in automatic sharding feature dynamically distributes data across multiple shards in a cluster, ensuring horizontal scalability and transparent data distribution.
- **Document Validation:** MongoDB supports document validation, allowing developers to enforce data integrity and consistency by defining rules for document validation at the collection level.

Chapter 5: Implementation and Testing

Thorough testing is essential to ensure the reliability and quality of the room booking system. Following the development phase, rigorous testing procedures will be implemented to identify and rectify any potential errors. Testing involves simulating various user scenarios and functionalities to verify the system's behavior against defined requirements. This process helps ensure the system functions as intended and meets the needs of both users and room managers. Test data creation plays a vital role in this process. By carefully crafting test data sets, we can simulate real-world usage patterns and identify potential shortcomings. Identified errors will be documented and addressed during the testing phase, ensuring a robust and reliable system before deployment. This focus on testing safeguards the successful implementation of the room booking system.

Unit Testing

The first step in the testing is the unit testing. Unit test is normally considered as an adjunct to the coding step. After the coding has been developed, received and verified for correct syntax, unit testing begins. The standalone modules were tested individually for their correct functionality, with the corresponding data. This ensures the reliability of the modules when integrated. Each and every module is tested independently with sample data and it was found that all modules are properly functioning. Using the unit test plans, prepared in the design phase of the system as a guide, important control paths are tested to uncover errors within the boundary of the modules. Boundary conditions were checked, all independent paths were exercised to ensure that all statements in the module are checked at least once and all error handling paths were tested. Each unit was thoroughly tested to check if it might fall in any possible situation. This testing was carried out during the programming itself. At the end of this testing phase, each unit was found to be working satisfactory, as regard to the expected output from the module.

Integration Testing

The second step in the testing process is the Integration testing. Integration testing is the systematic technique for constructing the program structure while conducting tests to uncover errors associated with interfacing. All the modules when unit testing will work properly but after interfacing the data can be lost across an interface, one module can have an inadvertent, adverse effect on other, sub functions when combined may not produce the desired major function, global data structures can cause problems, etc.

Integration testing was performed by integrating all the individual modules and the activities of the user such as loading layers, retrieving information from any functions applying themes based on the records present in the database etc. and is found that it works good to the examination of the end users. Hence, the objective of integration testing is to take unit tested modules and build a final program structure.

All the modules developed are independent. Even the whole process of approval for all. Each module is integrated well with other modules. And all the interfaces are tested successfully.

Functional Testing

This test involves testing the system under typical operating conditions with sample input values. Functional testing was performed on the system by giving existing industry id or plot number and a null or string as the input for any field in which case the user should be redirected to the same state with the appropriate message, rather than proceeding and crashing in the system.

Functional testing was performed on the system by raising the demand with an eye to check all the validations. The total processing of the system is satisfactory with the following results.

- All the validations are clearly notified to the user regarding jobseekers reg, new client reg, job order, job providers, and job search preparation etc.
- Almost all the functional errors, data storage errors and all types of logical errors are tested successfully.

Acceptance Testing

User acceptance test of a system is the factor for the success of the system. The system under consideration was listed for user acceptance by keeping constant touch with the perspective user of the system at the time of design, development and making changes whenever required for unit testing.

The requirements of the customer are gathered at regular intervals at the developing site itself. The problems that are to be visualized through this tool are been gathered by the customer and are reported.

The user at the user's site carried this test. Live data entered and the system's output was compared with what was manually prepared. Here the system has met the user's requirement in the following fields:

1. Data Entry
2. Error Handling
3. Reporting and corrections
4. Data Access Protections
5. System Output

Implementation

Implementation includes all those activities that take place to convert the old system to the new system. The new system will replace the existing system. The aspects of implementation are as follows. Conversion, Post Implementation Review.

Conversion

Conversion means changing from one system to another. The objective is to put the tested system into operation. It involves proper installation of the software package developed and training the operating staff.

The software has been installed and found to be functioning properly. The users should to be trained to handle the system effectively. Sample data to be provided to the operating staff and

are asked to operate on the system. The operating staffs now have a clear out look of the software and are ready for practical implementation of the package.

Post Implementation Review

A post implantation review is an evaluation of system in terms of the extent to which the system accomplishes the stated objectives. This starts after the system is implemented and conversation is complete.

5.1 User Interface and snapshots

Home Page

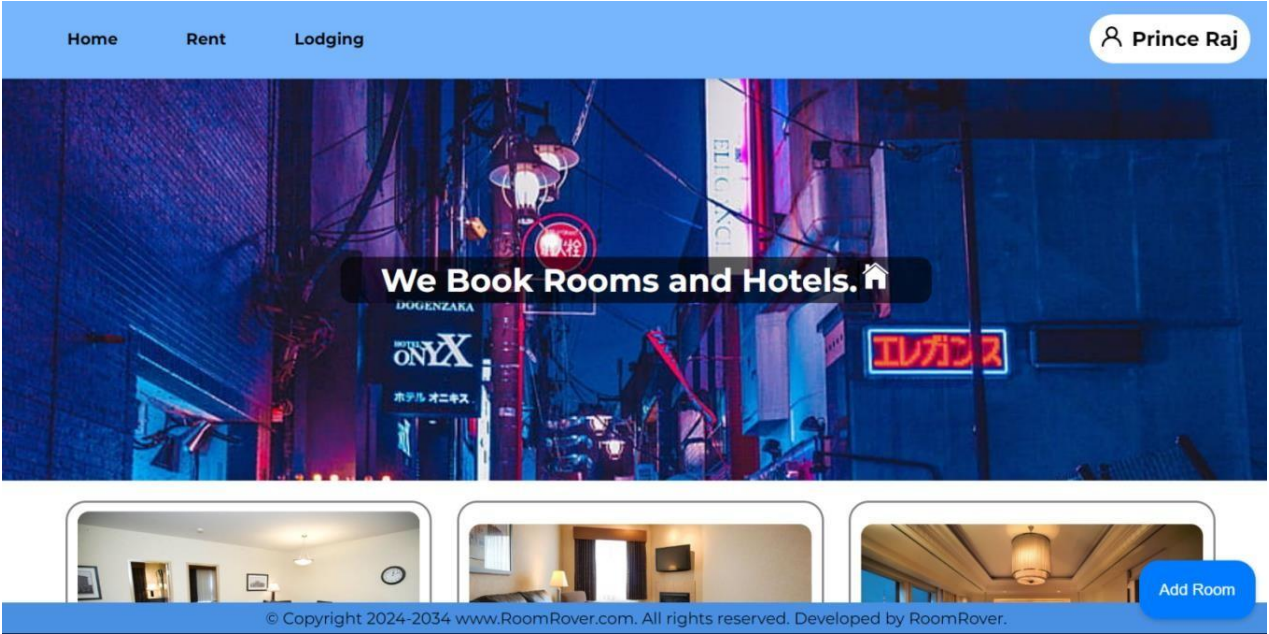


Figure 5.1- Home page I

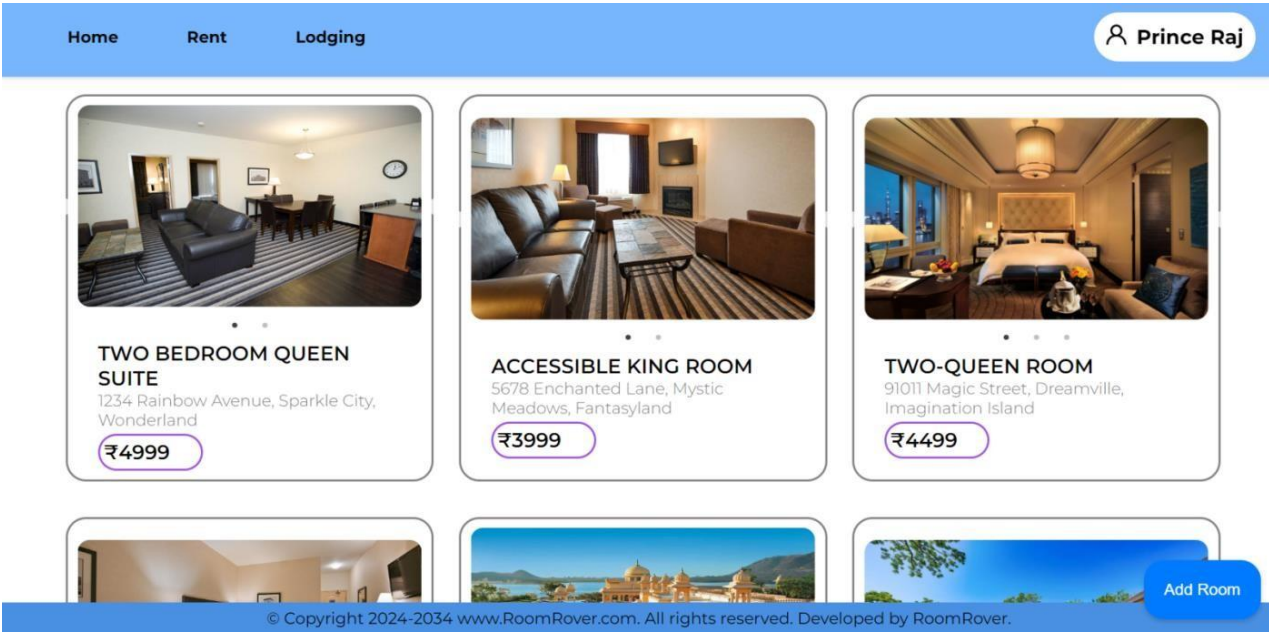
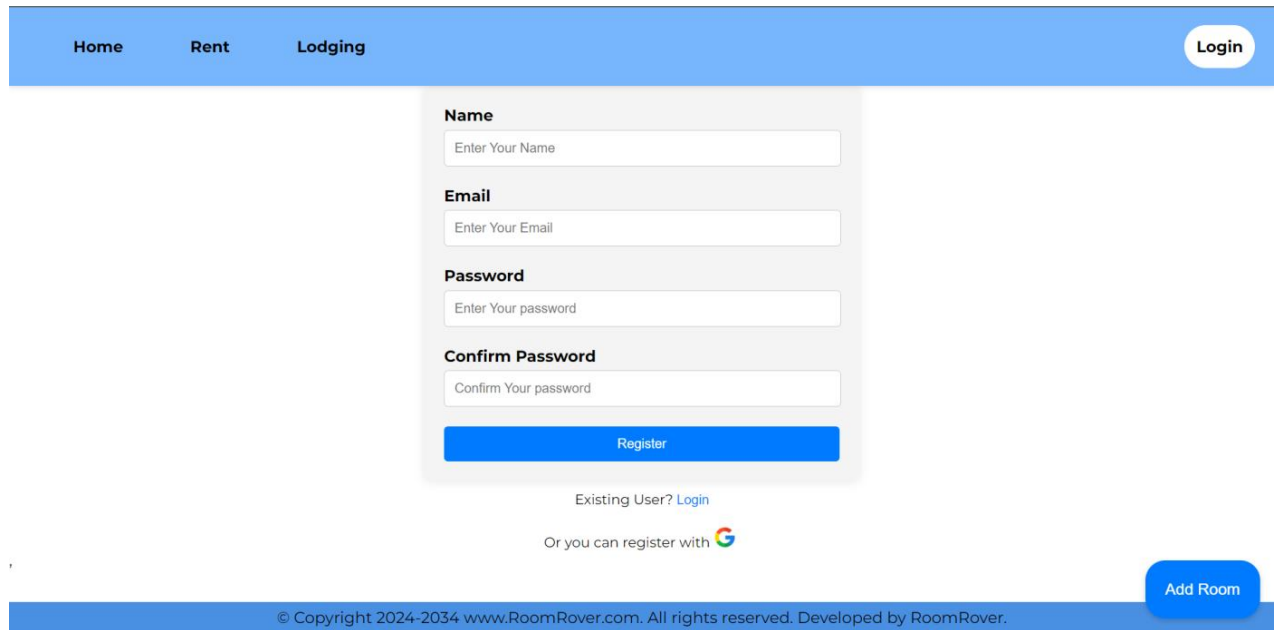


Figure 5.2- Home page II

Registration Page



The Registration Page features a blue header with navigation links: Home, Rent, Lodging, and a Login button. The main content area contains a registration form with fields for Name, Email, Password, and Confirm Password, each with a placeholder text. A Register button is at the bottom of the form. Below the form, there is a link for Existing User? Login and a Google login option. A footer bar contains the copyright notice and an Add Room button.

Home Rent Lodging Login

Name
Enter Your Name


Email
Enter Your Email

Password
Enter Your password

Confirm Password
Confirm Your password

Register

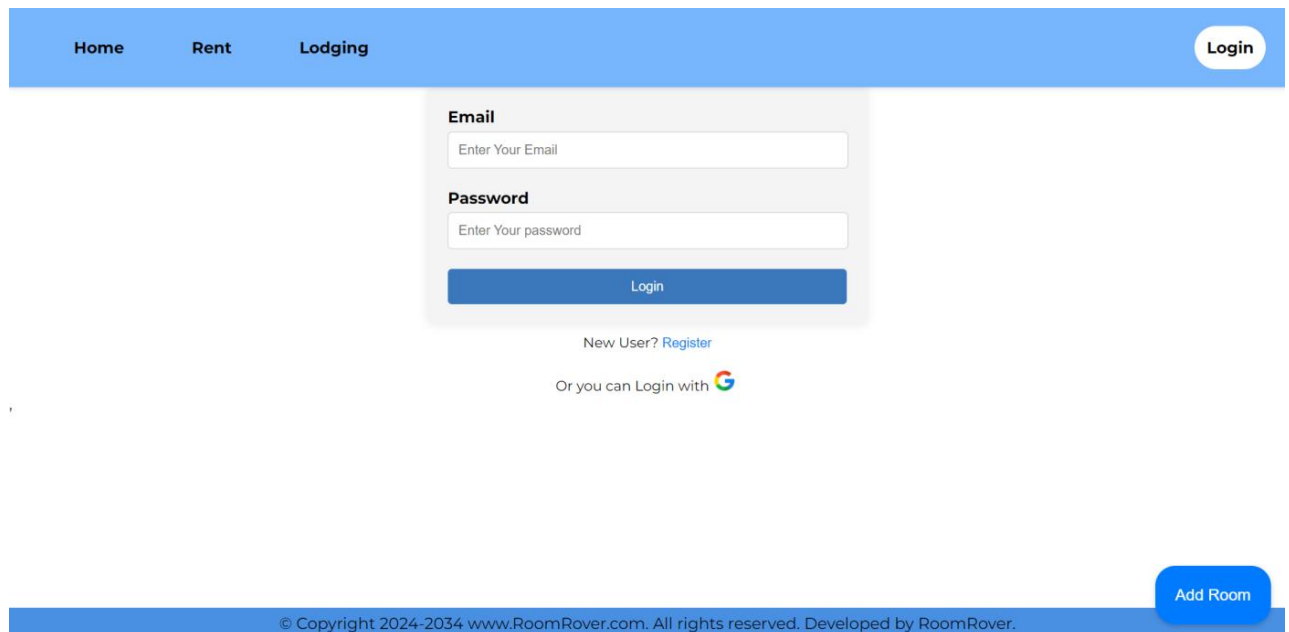
Existing User? [Login](#)

Or you can register with 

© Copyright 2024-2034 www.RoomRover.com. All rights reserved. Developed by RoomRover. Add Room

Figure 5.3- Registration Page

Login Form



The Login Form features a blue header with navigation links: Home, Rent, Lodging, and a Login button. The main content area contains a login form with fields for Email and Password, each with a placeholder text. A Login button is at the bottom of the form. Below the form, there is a link for New User? Register and a Google login option. A footer bar contains the copyright notice and an Add Room button.


Home Rent Lodging Login

Email
Enter Your Email

Password
Enter Your password

Login

New User? [Register](#)

Or you can Login with 

© Copyright 2024-2034 www.RoomRover.com. All rights reserved. Developed by RoomRover. Add Room

Figure 5.4- Login Form

Room Posting

ADD YOUR ROOM

User Name :

Room Name :

Address :

Price :

Description :

Room Type

☐ Lodging

☐ Rent

Upload Images :

Choose Files

No file chosen

submit

Figure 5.5- Room Posting

Room Booking

Washington Resort Hotel & Spa
1415 Secret Way, Twilight Town, Mystique Mountain

Name

Age

aadhar Number

Mobile Number

Address

Submit

Figure 5.6- Room Booking

User Profile

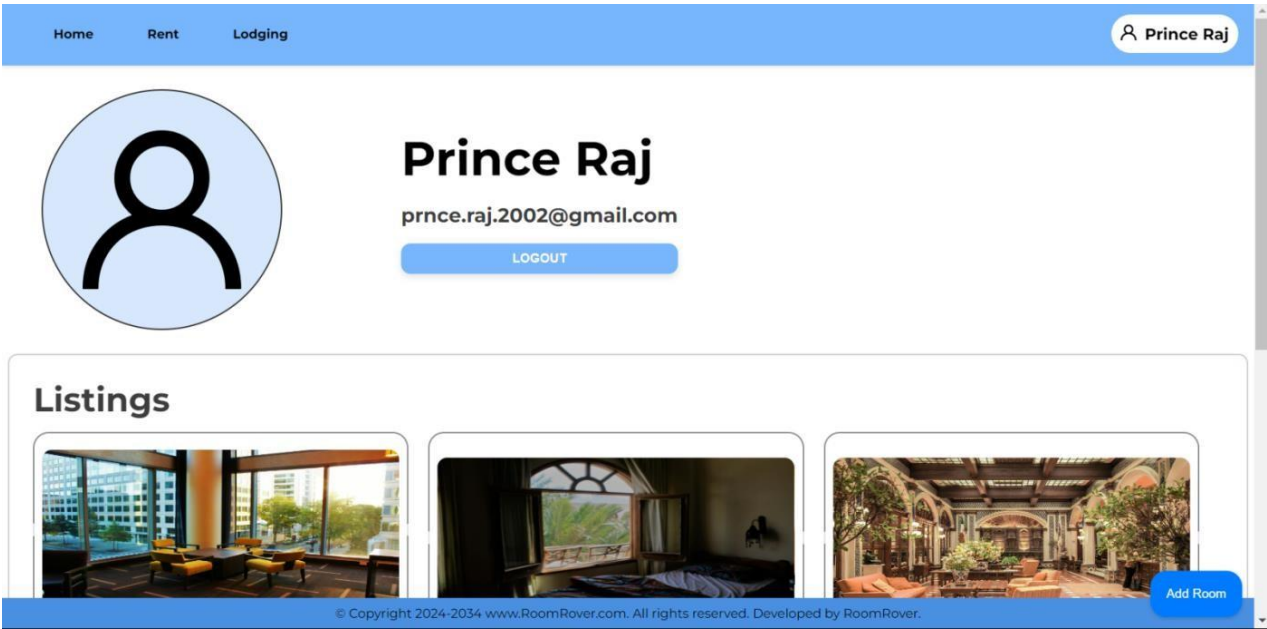


Figure 5.7- User Profile

Room information

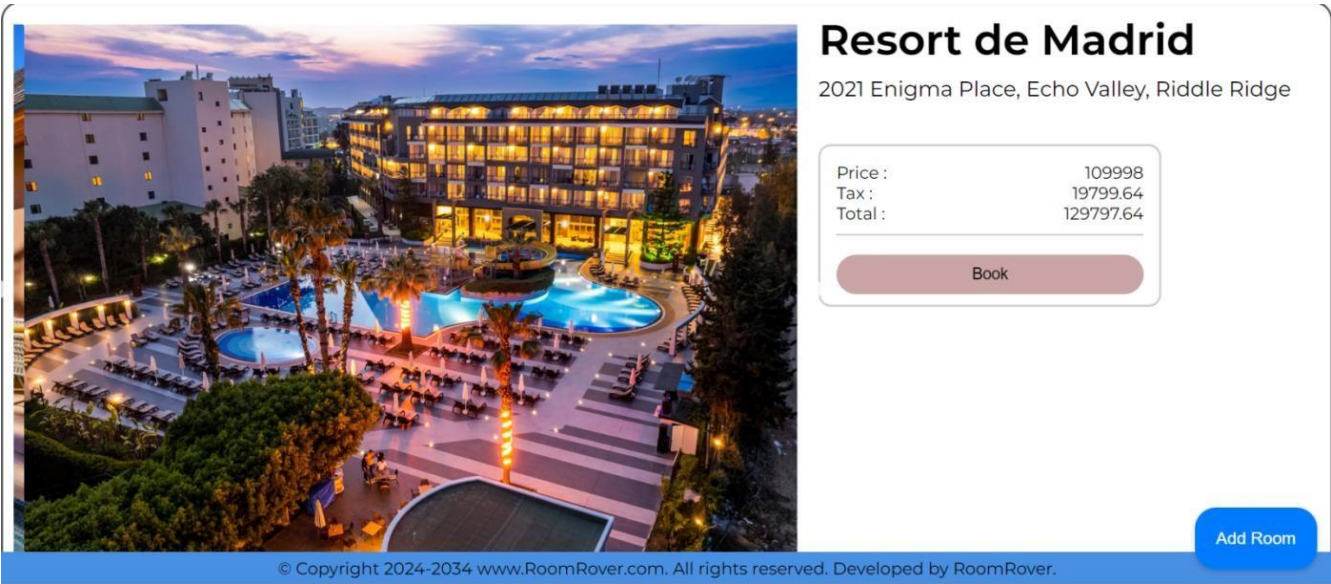


Figure 5.8- Room Information

5.2 Test Cases and Result

Table 5.1 Test Case

TEST CASE					
Test Case Id	Test scenario	Test steps	Test data	Expected result	Actual result
T01	Login through google authentication	Open website and clicked on google icon in login/registration	Data is retrieved from google auth	User is able to login to the website	As expected
T02	Register through normal register form	1. Open website and clicked on Registration and submitted the registration form	Information about the user to be added in room registration form	Data is successfully added in roomUser table	As expected
T03	Login to the website	1. open website and clicked on login 2. Logged into using the data	User enter valid data	Data is fetch from roomUser table	As expected
T04	Opening home page	Automatically opens the home page after login/ register	Fetch data from database that is shown on the home page	Data is fetched from RoomBookings table	As expected
T05	ADD ROOM	Click on the add room option at the bottom of the website	A form appears that need to filled	Data is added successfully in RoomBookingRooms and on submit the data is again fetched.	As expected
T06	Rent page	All the listings that are listed as rooms for rent ie the rooms that are to for rent.	User sees all the rooms available for rent.	Data from the RoomBookingRooms table is fetched and	As expected

		<p>be rented for a longer duration of time are listed in this section.</p> <p>Click on the rent option on the navbar to navigate to the page.</p>		then filter on that data is applied	
T07	Lodging page	<p>All the listings that are listed as rooms for Lodging ie the rooms that are to be lodged for a short duration of time are listed in this section.</p> <p>Click on the Lodging option on the navbar to navigate to the page.</p>	User sees all the rooms available for Lodging.	Data from the RoomBookingRooms table is fetched and then filter on that data is applied	As expected
T08	Profile page	User can see their profile page by clicking on their name being displayed on the navbar.	User can see their profile page that allows them to logout and also allows them to see all their listings they have made till now.	Data is fetched from the users table and rooms table and the necessary filters are applied.	As expected
T09	Seeing room info	On clicking the room cards one can see the description for the rooms	The description of the rooms can be viewed on clicking the room card which lets one book the room and navigate to the profile of the person who posted the room.	Data is fetched from the room table.	As expected

T10	Profile of host	On clicking on the room info we can see option of host's profile. Clicking on it we see the rooms listed by them and other profile details provided.	Users see the details of the host and all the provided rooms by the host on navigating to the profile of the host.	Data fetched from users table and the rooms table	As expected
T11	Room book option	On clicking to the room card we can go to room info and there we have option to book a room. On clicking that we can book a room.	On navigating to book room option we get a form of further details and filling those details we can then book the room	We are able to book the room using form which adds data in the database table named booking	As expected
T12	Logout	click on user profile option click on logout	User can logout of the account using this option	Successfully logout and go to login page	As expected

Chapter 6: Conclusion and Future Scope

Conclusion

It's been incredibly rewarding to develop this room booking system. This project has not only provided valuable practical experience in building a web-based application, but it has also deepened my understanding of technologies like [list relevant technologies used, e.g., ASP.NET, VB.NET, SQL Server]. Throughout the development process, I've gained valuable insights into project management and problem-solving.

Benefits:

The project is identified by the merits of the system offered to the user. The merits of this project are as follows: -

- **Web-Based Access:** Convenient web-based access allows users to search for and book rooms from any device with an internet connection, eliminating the need for phone calls or in-person visits
- **Centralized Management:** Authorized room managers gain a centralized platform to manage room details, availability, and potentially user access permissions. This simplifies room management and ensures all relevant information is readily accessible.
- **Intuitive Interface:** A user-friendly interface with clear navigation and simple forms streamlines the room booking process, making it easy for users to find the space they need.
- User is provided the option of monitoring the records he entered earlier. He can see the desired records with the variety of options provided by him.
- **Data-Driven Decision Making:** The system can provide valuable reports and insights on room usage patterns, which can inform strategic decisions regarding space allocation and resource management.

- **Clear Availability:** Real-time availability information fosters transparency and eliminates confusion regarding room occupancy.

Future Work

- Given the web-based nature of this project and its development by the Cyber Security Division, security is paramount. Before deploying the system, rigorous testing procedures will be implemented to identify and eliminate any potential security vulnerabilities. This comprehensive testing process ensures the system safeguards sensitive user data and maintains a high level of security.
- To further enhance security, the system may integrate with a data center console. This allows authorized personnel to monitor user activity and identify any suspicious behavior that might indicate unauthorized access attempts. Additionally, the system will implement comprehensive logging practices
- This room booking system is designed to be a foundational platform with the potential for future expansion. The core functionalities can be leveraged for various auditing operations beyond the initial scope of room booking. By designing the system's architecture with scalability in mind, we ensure the ability to integrate additional features or functionalities as future needs arise
- By prioritizing these future considerations and security testing, we ensure this room booking system becomes a robust, secure, and adaptable platform that effectively meets the organization's needs

References

Books

- React.js Essentials (2015), Author: Artemij Fedosejev
- Eloquent JavaScript, 3rd Edition: A Modern Introduction to Programming (2019), Author: Marijn Haverbeke
- Sams Teach Yourself UML in 24 Hours, Third Edition(2004), Author: Schmuller, J.

Articles/Journals

- Hafner, A. W., Seaton, R. L., & Faust, B. D. (2010). Open Room: Making Room Reservation Easy for Students and Faculty. Code4Lib Journal, (10). Retrieved from <https://doaj.org/article/f03e86ee6ca54827a209798ccb3ba7e4>
- Telepen. (2020). Juno Secure Room Booking. Retrieved October 23, 2020, from <https://telepen.co.uk/library-room-booking-system/>
- Teng, P. (2014). Online Room-Booking System Based on Database. Applied Mechanics and Materials, 513–517(Applied Science, Materials Science and Information Technologies in Industry), 1748,1751.

Websites

- https://www.olx.in/en-in/for-rent-houses-apartments_c1723
- <https://www.agoda.com/>
- <https://www.makemytrip.com/>
- <https://www.oyorooms.com/>