

HOTEL BOOK KARO

**A PROJECT REPORT
for
Major Project (KCA-451)
Session (2023-24)**

Submitted by

**Shivanshu Panwar
2200290140148**

**Submitted in partial fulfilment of the
Requirements for the Degree of**

MASTER OF COMPUTER APPLICATION

**Under the Supervision of
Ms. Komal Salgotra
Assistant Professor**



Submitted to

**DEPARTMENT OF COMPUTER APPLICATIONS
KIET Group of Institutions, Ghaziabad
Uttar Pradesh-201206
JUNE 2024**

CERTIFICATE

Certified that **Shivanshu Panwar (2200290140148)** has carried out the project work having “**Hotel Book Karo**” (**Major Project KCA-451**) for **Master of Computer Application** from Dr. A.P.J. Abdul Kalam Technical University (AKTU) (formerly UPTU), Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

Date:

Shivanshu Panwar (2200290140148)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date:

Ms. Komal Salgotra
Assistant Professor
Department of Computer Applications
KIET Group of Institutions, Ghaziabad

Dr. Arun Kumar Tripathi
Professor & Head
Department of Computer Applications
KIET Group of Institutions, Ghaziabad

ABSTRACT

The Hotel Book Karo Project aims to develop a comprehensive online platform for users to search, browse, and book accommodations seamlessly. Leveraging the MERN Stack technology – MongoDB, Express.js, React.js, and Node.js – the project focuses on providing a user-friendly interface, efficient booking process, and robust backend functionality.

The project begins with user registration, where users can create accounts securely, enabling them to access personalized features such as saving favourite hotels and managing bookings. Hotel listings are meticulously curated, allowing property owners to submit detailed information about their establishments, which is then reviewed and approved by administrators before being displayed on the website. Users can explore a wide range of hotels, filtering results based on location, amenities, pricing, and more.

The booking process is streamlined, with users able to select desired accommodations, input booking details, and complete transactions securely through integrated payment gateways. Administrators have access to a centralized dashboard for managing bookings, facilitating modifications, cancellations, and monitoring payment statuses. The project prioritizes user experience and security, implementing stringent validation checks and encryption protocols to safeguard user data and payment transactions.

In summary, the Hotel Book karo Project endeavours to revolutionize the way users interact with hotel reservations online, offering a dynamic, intuitive, and secure platform that caters to the diverse needs of modern travellers while setting new standards in the hospitality industry.

ACKNOWLEDGEMENTS

Success in life is never attained single-handedly. My deepest gratitude goes to my project supervisor, **Ms. Komal Salgotra** for her guidance, help, and encouragement throughout my project work. Their enlightening ideas, comments, and suggestions.

Words are not enough to express my gratitude to **Dr. Arun Kumar Tripathi**, Professor and Head, Department of Computer Applications, for his insightful comments and administrative help on various occasions.

Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me with moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

Shivanshu Panwar

TABLE OF CONTENTS

	Page Number
Certificate	ii
Abstract	iii
Acknowledgement	iv
Table of Contents	v
List of Tables	vi
List of Figures	vii
Chapter 1: Introduction	1 - 3
1.1 Project description	1
1.2 Project Scope	2
1.3 Hardware / Software used in Project	3
Chapter 2: Feasibility Study	4 - 5
2.1 Technical Feasibility	4
2.2 Operational Feasibility	4
2.3 Economical Feasibility	5
Chapter 3: Database Design	6 - 21
3.1 E-R Diagram	6
3.2 Data Flow Diagram	9
3.3 Use Case Diagram	18
Chapter 4: Design	22 - 30
Chapter 5: Discussion	31 – 33
5.1 Project Planning and Design	31
5.2 Implementation Challenges and Solution	31
5.3 User Feedback and Iterative Implementation	32
5.4 Future Enhancements	32
Chapter 6: Testing	34 - 37
6.1 Unit Testing	34
6.2 Integrating Testing	35
6.3 System Testing	36 - 37
Chapter 7: Coding	38 – 59
Chapter 8: Conclusion	60 – 62
8.1 Project Goal and Achievement	60
8.2 Technical Aspects	60
8.3 Challenges and Solution	61
8.4 Future Enhancement	61
8.5 Recommendation	62
Chapter 9: Bibliography	63

LIST OF TABLES

Table No.	Name of Table	Page No.
1.1	Hardware Requirements	3
1.2	Software Requirements	3

LIST OF FIGURES

Figure No.	Name of Figure	Page No.
3.1	ER Diagram	9
3.2	0 level DFD	10
3.3	1 st level DFD	14
3.4	2 nd level DFD (a)	17
3.5	2 nd level DFD (b)	18
3.6	Use Case Diagram	21
4.1	Sign In	22
4.2	Sign In	23
4.3	Home Page	24
4.4	Search Hotels	25
4.5	Hotel Details	26
4.6	Booking	27
4.7	Booking Confirmation	28
4.8	Add Hotel Form	29
4.9	Database	30

CHAPTER 1

INTRODUCTION

1.1 Project Description

The Hotel Book karo Website Project aims to develop a robust online platform that simplifies the process of booking accommodations for travellers while providing hotel owners with an efficient means of showcasing their properties. Leveraging the MERN Stack technology – MongoDB, Express.js, React.js, and Node.js – the project focuses on delivering a seamless user experience, comprehensive hotel listings, and secure transaction processing.

The key features of the Hotel book karo website are:

- User Registration and Authentication enable users to create accounts securely and access personalized features.
- Hotel Listings allow hotel owners to submit detailed property information, which is reviewed and approved by administrators before being displayed.
- Advanced Search Functionality provide users with intuitive search filters to find accommodations based on location, amenities, pricing, and more.
- Booking Management facilitate seamless booking transactions, including check-in/out dates, room selection, and payment processing.
- Responsive Design ensure accessibility across various devices, enhancing user experience.
- Secure Payment Processing integrate trusted payment gateways for secure and reliable transactions.

In summary, the Hotel Book Karo Project strives to revolutionize the online booking experience, providing a user-friendly platform that caters to the needs of both travellers and hotel owners in the hospitality industry.

In this project Technology Used

i. Front End Technology:

- React JS

ii. Back-End Technology:

1. Node JS
2. Express JS

iii. Database:

3. MongoDB

1.2 Project Scope

The Hotel Book karo Website project aims to develop a comprehensive online platform that facilitates seamless booking experiences for travelers and efficient property management for hotel owners. The scope encompasses the creation of a user-friendly interface, robust backend functionality, and secure payment processing system.

- The scope of the project will be to build a fully functional hotel booking website with features multiple features.
- The main focus will be on the frontend - building an easy-to-use and responsive customer-facing website.
- The following features will be in scope:
 - User Registration and Authentication
 - Hotel Listings
 - Search and Filter Functionality
 - Booking Management System
 - Secure Payment Processing
 - Communication Channels
 - Responsive Design
 - Reviews and ratings
- The following features will be out of scope for the initial version:
 - Recommendation engine
 - Loyalty programs
 - Advanced analytics and business intelligence
 - Mobile app
 - Drop shipping or order fulfillment
 - Multi-vendor support
- Security, performance and scalability will be considered but not optimized to the level of Amazon. The initial focus will be on building a minimum viable product.
- The website will be developed using React JS, Node JS, and Mongo-DB.

In summary, the scope of the project is to build the basic yet essential features required for a hotel booking website, with an emphasis on the customer-facing website while keeping security, performance and scalability in mind to a reasonable extent.

1.3 Hardware/Software Used

Here are the hardware and software that can be used for hotel book karo website project:

1.3.1 Hardware Requirements:

- Web server: A high-performance web server will be required to handle the traffic and load of an e-commerce website. Options include AWS EC2 instances, Google Cloud Compute Engine, or a dedicated server.
- Database server: A database server will be needed to store product data, order details, user information, and more. Options include AWS RDS, Google Cloud SQL, or a self-hosted SQL database server.
- Load balancer (optional): A load balancer can be used to distribute traffic across multiple web and database servers for high availability and scalability.

Tabel 1.1: Hardware Requirements

Title	Required
Processor	i3 processor or more
RAM	4GB or more
Hard Disk	40GB OR MORE SSD
Window	Window 8 or higher

1.3.2 Software Requirements:

Tabel 1.2: Software Requirements

Title	Uses
Database	MongoDB
Server	Node Js
IDE	Vs Code

In summary, to develop a hotel book karo, you'll need a combination of hardware resources like web and database servers, and software like programming languages, frameworks, libraries and tools. The specific stack depends on your preferences and requirements.

The important thing is to choose technologies that you are comfortable with while keeping performance, scalability and security in mind.

CHAPTER 2

FEASIBILITY STUDY

Building a hotel book karo requires careful consideration of technical, financial and operational feasibility.

The purpose of this feasibility study is to assess the viability and potential success of the proposed Hotel Book Karo project. The project aims to replicate the functionality and features of the existing Hotel Booking websites. This study evaluates the technical, operational, economic, and scheduling aspects to provide stakeholders with comprehensive insights into the feasibility of the project.

The feasibility study aims to evaluate the practicality and potential success of developing a hotel book karo as an academic project. The study encompasses technical, economic, legal, operational, and scheduling aspects to ensure comprehensive analysis and informed decision-making.

2.1 Technical Feasibility

a. **Technology Stack:** The project will utilize a robust and scalable technology stack comprising front-end, back-end, and database technologies. The front-end will be developed using Tailwind CSS, TypeScript, and React.js to ensure a responsive and user-friendly interface. The back-end will leverage Node.js and Express.js to handle server-side operations efficiently. A database MongoDB will be employed to manage user data and booking information.

b. **Infrastructure:** The website will be hosted on a reliable cloud platform such as Amazon Web Services (AWS) or Google Cloud Platform (GCP). These services offer scalability, security, and uptime guarantees, essential for handling fluctuating traffic and ensuring data safety.

c. **Security:** Implementing robust security measures is critical. SSL encryption, two-factor authentication, and secure payment gateways will be integrated to protect user data and transactions. Regular security audits and updates will be conducted to address potential vulnerabilities.

2.2 Operational Feasibility

a. **Development Team:** A skilled team comprising front-end and back-end developers, UI/UX designers, and database administrators will be assembled. The team will follow agile development methodologies to ensure iterative progress and timely delivery.

b. **User Support:** A dedicated support team will be established to handle user inquiries and issues. This will include FAQs, live chat, and email support to enhance user satisfaction and retention.

c. **Maintenance and Support:** Evaluate the feasibility of maintaining and supporting the website post-launch, including updates, bug fixes, and customer support.

2.3 Economic Feasibility

a. **Initial Costs:** Initial costs include development tools, hosting services, domain registration, and potential third-party integrations. Estimations suggest a budget of \$5,000 to cover these expenses.

b. **Operational Costs:** Ongoing costs will consist of hosting fees, maintenance, marketing, and customer support. An estimated \$500 monthly budget will cover these operational expenses.

c. **Revenue Model:** The website can generate revenue through multiple streams:

- **Commission-based Model:** Charging hotels a commission on each booking.
- **Subscription Model:** Offering premium listings or services to hotels for a monthly fee.
- **Advertising:** Displaying targeted ads to users.

The feasibility study demonstrates that developing a hotel booking platform is a viable academic project. The technical infrastructure is sound, economic prospects are promising, legal requirements are manageable, operational processes are clear, and the project timeline is feasible. With careful planning and execution, the project is poised to deliver a valuable, user-centric platform that meets the needs of modern travelers and offers significant learning opportunities for the development team.

CHAPTER 3

DATABASE DESIGN

3.1 E-R Diagram

Designing an efficient database is crucial for the success of a hotel book karo. The database should be well-structured to handle various entities such as users, hotels, rooms, bookings, and payments. This design aims to ensure data integrity, minimize redundancy, and facilitate easy data retrieval. The main entities and relationships for a Hotel Book Karo project are:

Entity Definitions and Attributes:

Users:

The Users entity stores information about the users of the website. Attributes include:

- **U_id (Primary Key):** A unique identifier for each user.
- **FirstName:** The user's first name.
- **LastName:** The user's last name.
- **Email:** The user's email address, unique to each user.
- **Password:** The hashed password for user authentication.
- **Phone:** The user's contact number.
- **DateCreated:** The date the user account was created.

The Users entity represents individual user register on the website for booking hotels.

Hotels:

The Hotels entity contains details about the hotels listed on the platform. Attributes include:

- **H_id (Primary Key):** A unique identifier for each hotel.
- **H_name:** The name of the hotel.
- **Address:** The hotel's address.
- **City:** The city where the hotel is located.
- **State:** The state where the hotel is located.
- **Country:** The country where the hotel is located.
- **Phone:** The contact number of the hotel.
- **Description:** A description of the hotel.
- **Rating:** The average user rating for the hotel.
- **Facilities:** shows the facilities available at hotel.

The **Hotels** entity classifies hotels into distinct city, state, and country. Attributes may include **h_id** and **name**. Each Hotel is associated with a specific location, aiding with facility available and searching for hotels available in city, state and country.

Rooms:

The **Rooms** entity details the rooms available in each hotel. Attributes include:

- **Room_id** (Primary Key): A unique identifier for each room.
- **H_id** (Foreign Key): A reference to the **H_id** in the **Hotels** entity, establishing a relationship between rooms and their respective hotels.
- **RoomNumber**: The room number within the hotel.
- **RoomType**: The type of the room (e.g., single, double, suite).
- **PricePerNight**: The cost per night for the room.
- **Availability**: The availability status of the room.
- **MaxOccupancy**: The maximum number of occupants allowed in the room.
- **Amenities**: A list of amenities available in the room.

The **rooms** entity represents the different categories of room that hotel provides, like Ac, Non-Ac, etc.

Booking:

The **Booking** entity records information about each booking made by users. Attributes include:

- **Book_id** (Primary Key): A unique identifier for each booking.
- **U_id** (Foreign Key): A reference to the UserID in the Users entity, linking the booking to a specific user.
- **Room_id** (Foreign Key): A reference to the RoomID in the Rooms entity, linking the booking to a specific room.
- **CheckInDate**: The date the user checks into the hotel.
- **CheckOutDate**: The date the user checks out of the hotel.
- **NumberOfGuests**: The number of guests for the booking.
- **TotalPrice**: The total price of the booking.
- **BookingDate**: The date the booking was made.
- **Status**: The current status of the booking (e.g., confirmed, cancelled).

The **Booking** entity represents records information about each booking made by users and admin.

Payment:

The **Payment** entity handles payment details for bookings. Attributes include:

- **Pay_id** (Primary Key): A unique identifier for each payment.

- **Book_id (Foreign Key):** A reference to the BookingID in the Booking entity, linking the payment to a specific booking.
- **PaymentDate:** The date the payment was made.
- **Amount:** The amount paid.
- **PaymentMethod:** The method of payment (e.g., credit card, PayPal).
- **Trans_id:** A unique identifier for the transaction provided by the payment gateway.

The **Payment** entity represents records of handles payment details for bookings

Relationships Between Entities

Users and Booking:

There is a one-to-many relationship between Users and Booking. A user can make multiple bookings, but each booking is associated with only one user. This is represented by the foreign key U_id in the Booking entity, which references the U_id the Users entity.

Hotels and Rooms:

There is a one-to-many relationship between Hotels and Rooms. A hotel can have multiple rooms, but each room belongs to only one hotel. This is represented by the foreign key H_id in the Rooms entity, which references the H_id in the Hotels entity.

Rooms and Booking:

There is a one-to-many relationship between Rooms and Booking. A room can be booked multiple times, but each booking is for only one room. This is represented by the foreign key Room_id in the Booking entity, which references the Room_id in the Rooms entity.

Booking and Payment:

There is a one-to-one relationship between Booking and Payment. Each booking has one corresponding payment, and each payment is associated with one booking. This is represented by the foreign key Book_id in the Payment entity, which references the Book_id in the Booking entity.

Entity-Relationship Diagram (ERD): An ERD visualizes these entities and their relationships. It includes:

- **Users'** entity with a one-to-many relationship to **Booking**.
- **Hotels** entity with a one-to-many relationship to **Rooms**.
- **Rooms** entity with a one-to-many relationship to **Booking**.
- **Booking** entity with a one-to-one relationship to **Payment**.

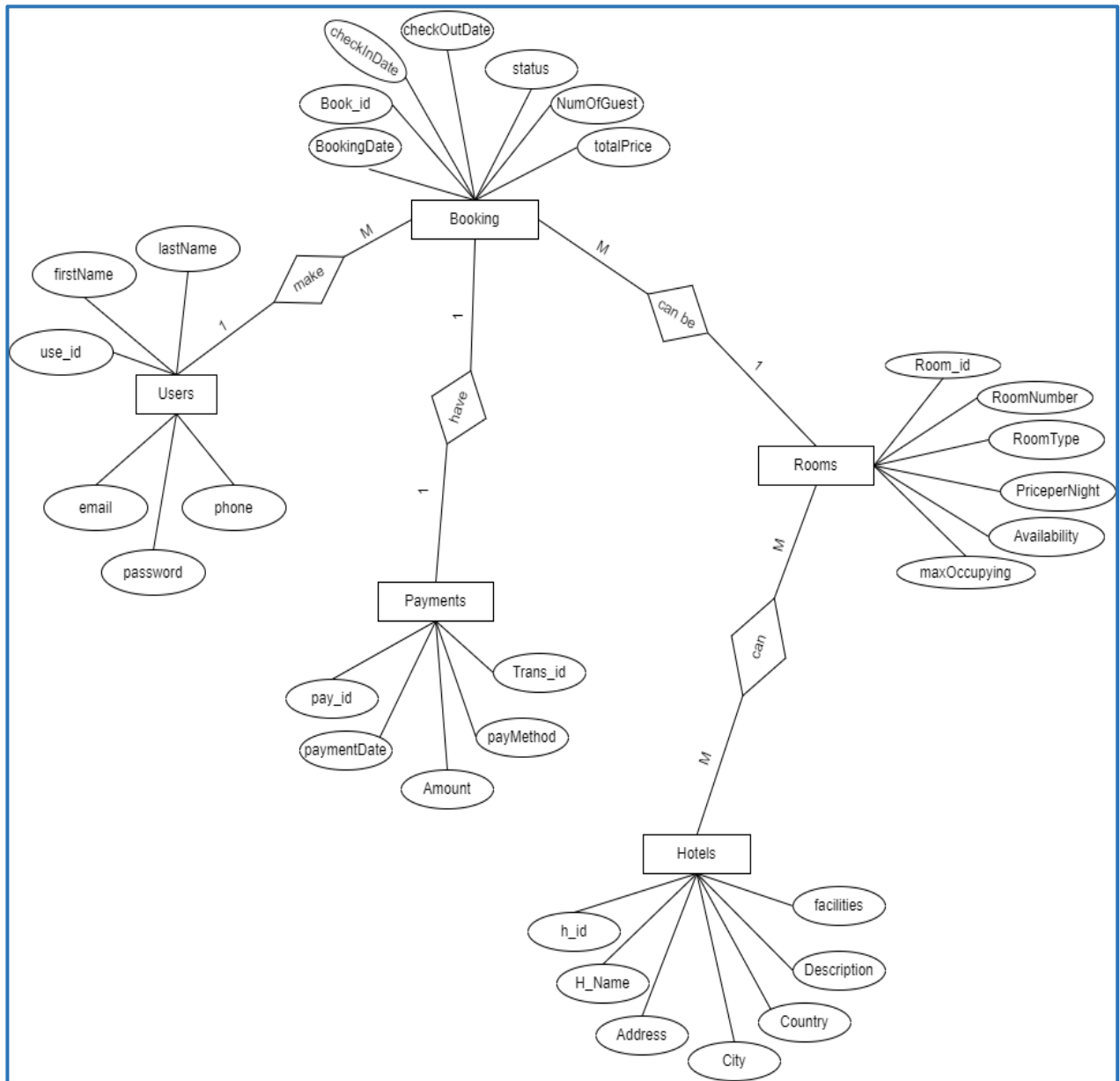


Figure 3.1: ER Diagram

The proposed database design ensures efficient data management for the hotel book karo. By clearly defining entities, their attributes, and relationships, the design supports scalable and secure operations. This structured approach facilitates seamless booking processes, enhances user experience, and ensures data integrity across the platform.

3.2 Data Flaw Diagram

A Data Flow Diagram (DFD) for a hotel book karo illustrates the flow of information between processes, data stores, and external entities. Key processes include user registration, hotel search, booking, and payment processing. Data flows between users (guests and admins), the

booking system, hotel database, room database, and payment gateway. Users input search criteria and booking details, which are processed and stored in the respective databases. The system retrieves hotel and room information for display and processes payments securely, updating booking statuses accordingly. The DFD helps visualize the data movement and interactions within the system.

Context level DFD – 0 level:

A Context Level DFD, also known as a Level 0 DFD, provides an overview of the system and how it interacts with external entities. For the hotel book karo website project, the main external entities are Guests, Admins, and Payment Gateways. The DFD outlines the flow of data between these entities and the system.

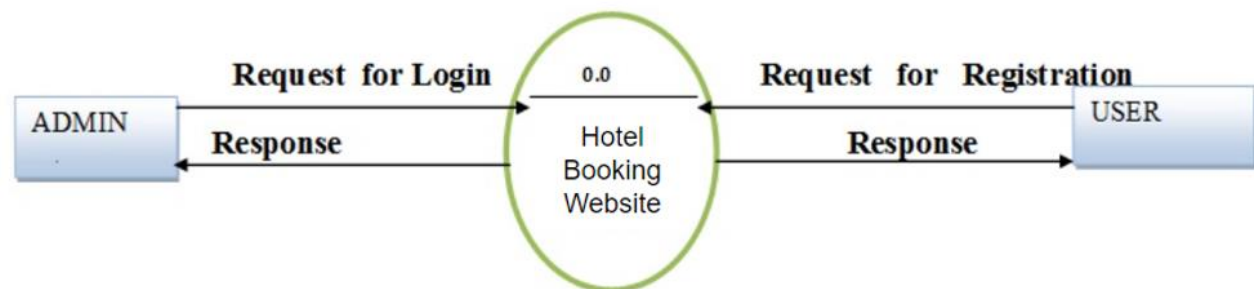


Figure 3.2: 0 level DFD

External Entities:

- Guests
- Admins
- Payment Gateways

Processes:

- User Registration and Authentication
- Hotel and Room Search
- Booking Management
- Payment Processing
- Admin Management

Data Stores:

- User Database (DS1)
- Hotel Database (DS2)
- Booking Database (DS3)

- Payment Database (DS4)

Data Flow:

Guests to System:

- **Registration Data:** Guests provide personal details for registration.
- **Login Credentials:** Guests provide credentials to log in.
- **Search Criteria:** Guests input search criteria for hotels and rooms.
- **Booking Details:** Guests provide booking details (dates, number of guests, etc.).
- **Payment Information:** Guests provide payment details during booking.

System to Guests:

- **Confirmation Data:** System sends registration confirmation.
- **Search Results:** System provides hotel and room search results.
- **Booking Confirmation:** System sends booking confirmation details.
- **Payment Confirmation:** System sends payment confirmation.

Admins to System:

- **Admin Commands:** Admins manage users, hotels, rooms, and bookings.

System to Admins:

- **Management Reports:** System provides reports and data for management tasks.

System to Payment Gateways:

- **Payment Requests:** System sends payment processing requests.

Payment Gateways to System:

- **Payment Responses:** Payment gateways send back payment confirmation or rejection.

Level 0 DFD Summary:

- User Registration and Authentication (Process 1):** Handles registration and login data flows between Guests and the User Database.
- Hotel and Room Search (Process 2):** Manages search queries and results between Guests and the Hotel Database.
- Booking Management (Process 3):** Processes booking details, stores booking information in the Booking Database, and updates Guests.
- Payment Processing (Process 4):** Facilitates payment transactions, interacting with Payment Gateways and updating the Payment Database.

- v. **Admin Management (Process 5):** Enables Admins to manage data across User, Hotel, Booking, and Payment Databases.

This high-level overview encapsulates the main interactions and data flows within the hotel booking website system, ensuring a clear understanding of how data moves through the system and interacts with external entities.

1st level – User side Data flow Diagram:

The 1st level DFD for the user side of a hotel book karo website illustrates how users interact with the system to perform various tasks.

Processes:

1. User Registration/Login:

- Input: User details (username, password)
- Output: User account creation, authentication token
- Data Store: User Database

2. Search Hotels:

- Input: Search criteria (location, dates, number of guests)
- Output: List of available hotels
- Data Store: Hotel Database

3. View Hotel Details:

- Input: Hotel selection
- Output: Detailed hotel information (room types, amenities, pricing)
- Data Store: Hotel Database

4. Book Room:

- Input: Booking details (hotel, room type, dates)
- Output: Booking confirmation
- Data Store: Booking Database

5. Make Payment:

- Input: Payment information (credit card details)
- Output: Payment confirmation, receipt
- Data Store: Payment Database

6. View Booking History:

- Input: User request
- Output: List of past and upcoming bookings
- Data Store: Booking Database

Data Stores:

- User Database: Stores user details and credentials
- Hotel Database: Stores hotel information

- **Booking Database:** Stores booking details
- **Payment Database:** Stores payment transactions

Level DFD - Admin Side Diagram:

The 1st level DFD for the admin side of a hotel book karo illustrates how administrators interact with the system to manage users, hotels, and bookings.

Processes:

1. Manage Users:

- **Input:** Admin user actions (add, update, delete user)
- **Output:** Updated user information
- **Data Store:** User Database

2. Manage Hotels:

- **Input:** Admin actions (add, update, delete hotel information)
- **Output:** Updated hotel details
- **Data Store:** Hotel Database

3. Manage Bookings:

- **Input:** Admin actions (view, modify, cancel bookings)
- **Output:** Updated booking information
- **Data Store:** Booking Database

4. Generate Reports:

- **Input:** Admin request (specify report type and parameters)
- **Output:** Generated reports (booking statistics, financial reports)
- **Data Store:** Booking Database, Payment Database

Data Stores:

- **User Database:** Stores user details and credentials
- **Hotel Database:** Stores hotel information
- **Booking Database:** Stores booking details
- **Payment Database:** Stores payment transactions

The development of the hotel book karo project involved creating detailed Data Flow Diagrams (DFDs) at the first level for both user and admin sides, which effectively mapped out the flow of information and interactions within the system. On the user side, the DFD encapsulated key processes such as user registration, hotel search, room booking, and payment transactions. This ensured a seamless user experience by illustrating how data is handled from the point of user input to the final booking confirmation. On the admin side, the DFD highlighted critical administrative functions including user management, hotel and room maintenance, booking oversight, and report generation. This enabled efficient management and monitoring of the platform, ensuring data integrity and operational stability.

Overall, these DFDs provided a clear and structured visualization of the system's workflows, facilitating better understanding and communication among the development team and stakeholders. They laid a solid foundation for developing a robust, user-friendly, and efficient hotel booking website that meets both user expectations and administrative needs.

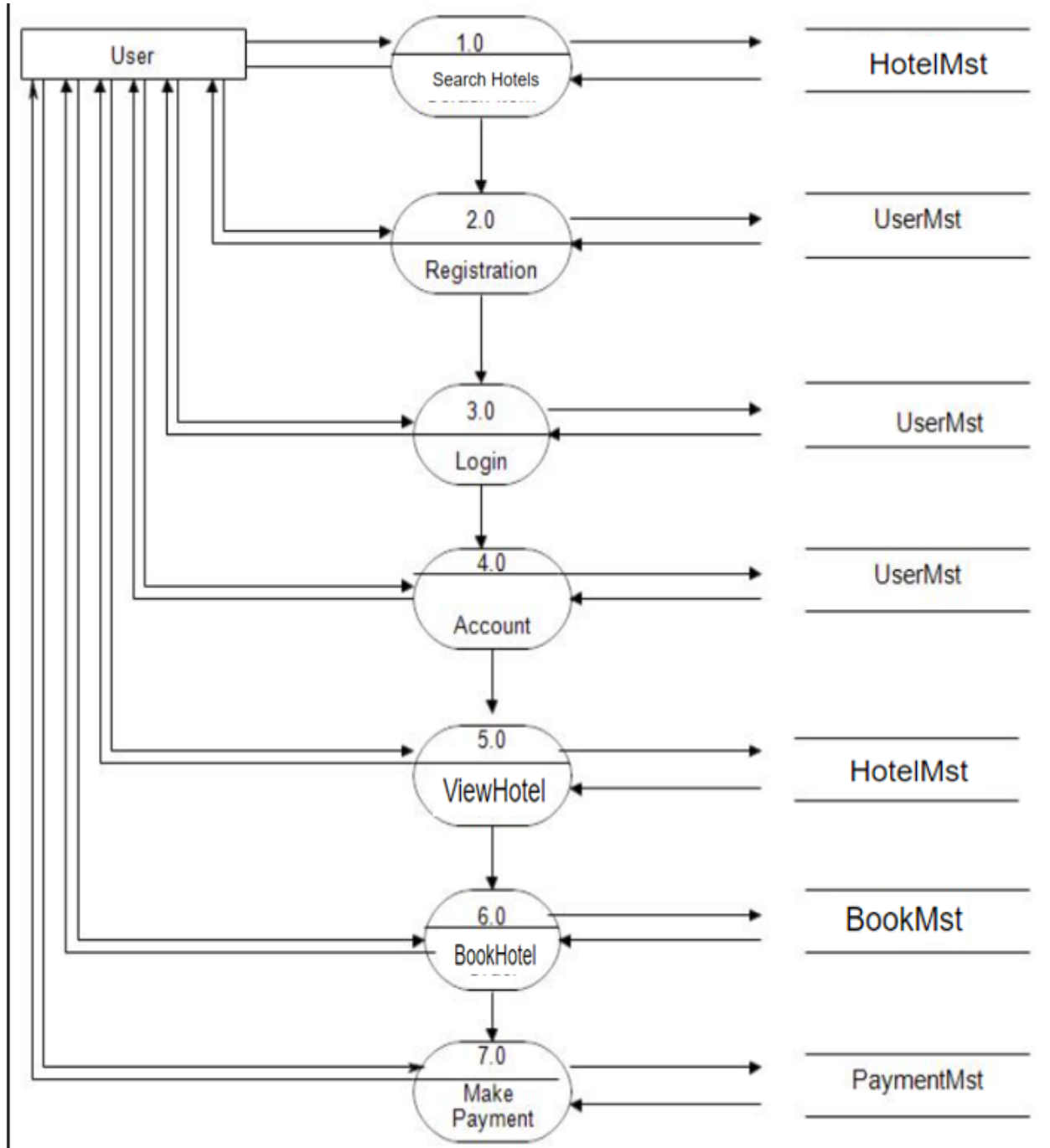


Figure 3.3: 1st level DFD

2nd level – User side DFD:

A 2nd level Data Flow Diagram (DFD) provides a detailed view of the processes involved in the user interactions with the hotel book karo. It breaks down the high-level processes into more granular tasks.

User-Side DFD Explanation

1. User Registration/Login:

- a. **Process 1.1: User Enters Details:** The user inputs their registration or login information.
- b. **Process 1.2: Validate User:** The system validates the credentials against the User Database.
- c. **Data Stores:** User Database for storing and verifying user credentials.
- d. **External Entities:** Users interact with this process to create accounts or log in.

2. Search and View Hotels:

- a. **Process 2.1: Enter Search Criteria:** Users input search criteria (location, dates, number of guests).
- b. **Process 2.2: Retrieve Hotel Data:** The system retrieves matching hotels from the Hotel Database.
- c. **Process 2.3: Display Results:** The system displays search results to the user.
- d. **Data Stores:** Hotel Database storing hotel information.
- e. **External Entities:** Users initiate searches and view results.

3. Booking Process

- a. **Process 3.1: Select Room:** The user selects a room from the available options.
- b. **Process 3.2: Enter Booking Details:** The user provides booking details (check-in/check-out dates, guests).
- c. **Process 3.3: Confirm Booking:** The system confirms the booking and updates the Booking Database.
- d. **Data Stores:** Booking Database for storing booking information.
- e. **External Entities:** Users finalize their booking details.

4. Payment Process:

- a. **Process 4.1: Enter Payment Details:** Users enter payment information.
- b. **Process 4.2: Process Payment:** The system processes the payment via an external Payment Gateway.
- c. **Process 4.3: Update Booking Status:** The system updates the booking status in the Booking Database.
- d. **Data Stores:** Payment Database for storing payment records.
- e. **External Entities:** Users complete the payment process.

5. View Booking History

- a. **Process 5.1: Request Booking History:** Users request their booking history.
- b. **Process 5.2: Retrieve Booking Data:** The system fetches booking records from the Booking Database.
- c. **Process 5.3: Display Booking History:** The system displays past and upcoming bookings to the user.
- d. **Data Stores:** Booking Database containing booking records.
- e. **External Entities:** Users view their booking history.

The user-side 2nd level DFD provides a detailed breakdown of how users interact with the hotel booking system, from registration and searching for hotels to booking rooms and making payments.

2nd Level Admin-Side Data Flow Diagram (DFD)

The admin-side 2nd level Data Flow Diagram (DFD) illustrates the detailed processes involved in the administrative functions of the hotel book karo. It covers managing users, hotels, and bookings.

Admin-Side DFD Explanation

1. Manage Users:

- a. **Process 1.1: View Users:** Admin views the list of registered users.
- b. **Process 1.2: Add/Edit/Delete Users:** Admin performs actions to add, edit, or delete user information.
- c. **Data Stores:** User Database storing user details.
- d. **External Entities:** Admins interacting with user management functionalities.

2. Manage Hotels:

- a. **Process 2.1: View Hotels:** Admin views all hotel listings.
- b. **Process 2.2: Add/Edit/Delete Hotels:** Admin adds, updates, or removes hotel information.
- c. **Data Stores:** Hotel Database storing hotel details.
- d. **External Entities:** Admins managing hotel listings.

3. Manage Rooms:

- a. **Process 3.1: View Rooms:** Admin views rooms associated with hotels.
- b. **Process 3.2: Add/Edit/Delete Rooms:** Admin manages room details including availability and pricing.
- c. **Data Stores:** Room Database containing room information.
- d. **External Entities:** Admins interacting with room management.

4. Manage Bookings:

- a. **Process 4.1: View Bookings:** Admin views all current and past bookings.

- b. **Process 4.2: Confirm/Cancel Bookings:** Admin updates the status of bookings (confirming or canceling).
- c. **Data Stores:** Booking Database storing booking records.
- d. **External Entities:** Admins managing booking details.

5. Generate Reports:

- a. **Process 5.1: Select Report Type:** Admin selects the type of report to generate.
- b. **Process 5.2: Generate Report:** The system generates the report based on selected parameters.
- c. **Process 5.3: Display Report:** The system displays the generated report.
- d. **Data Stores:** Report Database containing report data.
- e. **External Entities:** Admins generating and viewing reports.

The admin-side 2nd level DFD provides a comprehensive view of how administrators manage various aspects of the hotel book karo. It ensures efficient handling of user accounts, hotel and room listings, bookings, and reporting functionalities, supporting the smooth operation of the platform.

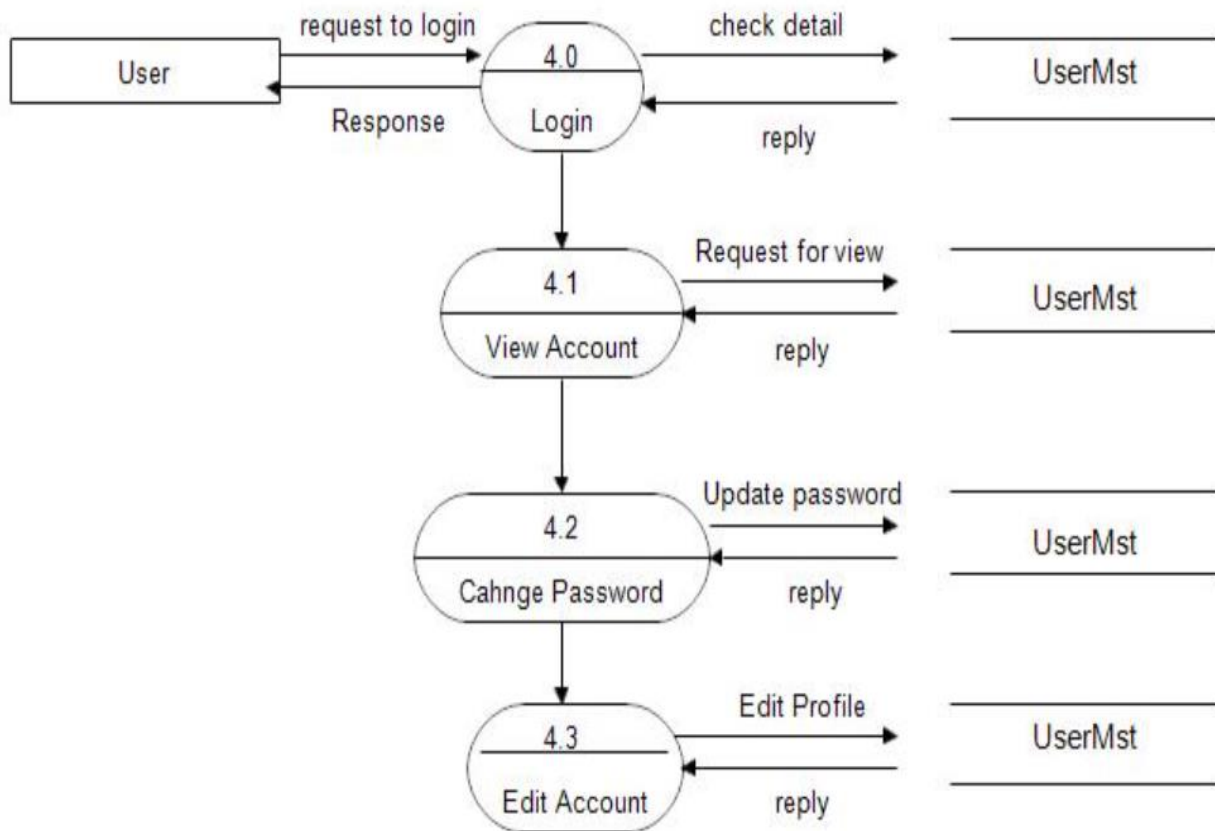


Figure 3.4: 2nd level DFD (a)

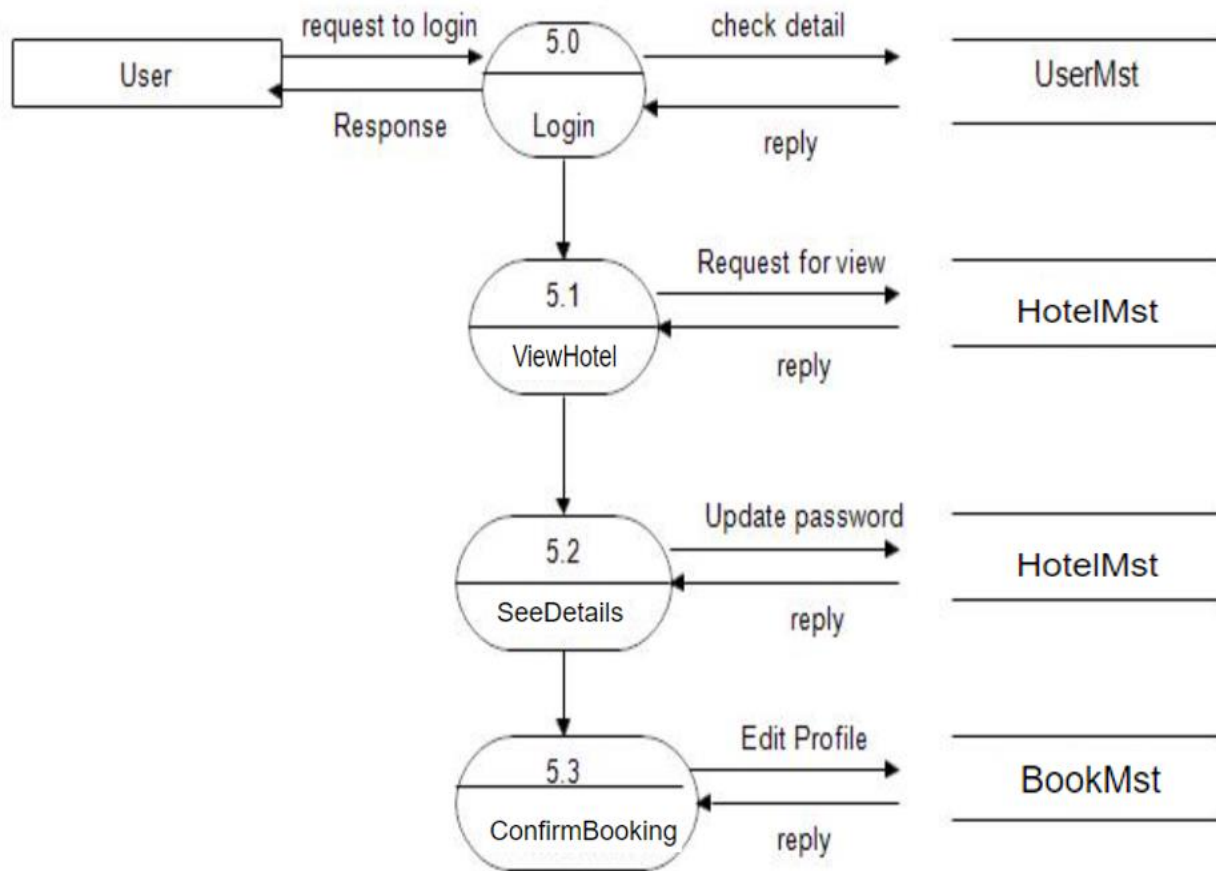


Figure 3.5: 2nd level DFD (b)

3.3 Use Case Diagram

A use case diagram is a visual representation that describes the interactions between users (actors) and the system (use cases). It captures the functional requirements of a system and shows the relationship between actors and use cases. For a hotel book karo, the use case diagram provides a clear depiction of the various functionalities offered by the system and how different users interact with these functionalities.

3.3.1 Actors:

In the context of a hotel book karo, the primary actors include:

- a. **Guest:** Guests are the primary users of the hotel book karo. They perform various activities such as searching for hotels, viewing hotel details, booking rooms, and making payments. They may also manage their accounts and view booking history.

- b. **Admin:** Admins are responsible for managing the website, including user accounts, hotel listings, and bookings. They have access to administrative functions that are not available to regular users.

3.3.2.1 Use Cases for Guests:

- a. **Register:** Allows new users to create an account. Followings are the steps in registering users:
 - i. Guest navigates to the registration page.
 - ii. Guest fills out the registration form with personal details (name, email, password, etc.).
 - iii. System validates the information and creates a new user account.
 - iv. Guest receives a confirmation email.
- b. **Login:** Authenticates users and grants access to their account. Following steps required to login user:
 - i. Guest navigates to the login page.
 - ii. Guest enters email and password.
 - iii. System verifies credentials.
 - iv. Guest is granted access to their account.
- c. **Search Hotels:** Allows users to search for hotels based on various criteria. Following are steps in searching hotels:
 - i. Guest navigates to the search page.
 - ii. Guest enters search criteria (location, dates, number of guests, etc.).
 - iii. System displays a list of available hotels matching the criteria.
- d. **View Hotel Details:** Displays detailed information about a selected hotel. Following are the steps in view hotel details:
 - i. Guest selects a hotel from the search results.
 - ii. System displays hotel details (rooms, amenities, reviews, pricing, etc.).
- e. **Book Room:** Allows users to book a room in a selected hotel. Followings are the steps in booking room in hotels:
 - i. Guest selects a room from the hotel details page.
 - ii. Guest enters booking details (check-in, check-out dates, number of guests, etc.).
 - iii. System calculates the total price.
 - iv. Guest confirms the booking and proceeds to payment.
- f. **Make Payment:** Processes the payment for a room booking. Following steps required to make payments:
 - i. Guest selects a payment method.

- ii. Guest enters payment details.
 - iii. System processes the payment and confirms the booking.
 - iv. Guest receives a booking confirmation email.
- g. **View Booking History:** Allows users to view their past and upcoming bookings.
 - i. Guest navigates to the booking history page.
 - ii. System displays a list of the guest's bookings.

3.3.2.2 Use Cases for Admins:

- a. **Manage Hotels:** Allows admins to add, edit, or remove hotel listings. Followings are the steps to manage hotels:
 - iii. Admin navigates to the hotel management page.
 - iv. Admin performs actions such as adding, editing, or deleting hotel listings.
 - v. System updates the hotel database accordingly.
- b. **Manage Bookings:** Allows admins to view and manage all bookings. Followings are the steps to manage bookings of the hotels:
 - i. Admin navigates to the booking management page.
 - ii. Admin views booking details and performs necessary actions (confirm, cancel, etc.).
 - iii. System updates the booking records.

3.3.3 Relationships:

- a. **Generalization:** There is a generalization relationship between the Guest and Registered User actors, indicating that a registered user is a specialized type of user compared to a guest.
- b. **Association:** Both the Registered User and Guest actors are associated with all the primary use cases, such as browsing products, searching, and managing accounts. This reflects that these functionalities are accessible to both user types.
- c. **Dependency:** The Administrator actor has a dependency relationship with all administrative use cases, indicating that these functionalities depend on the administrator's role.

3.3.4 System Boundary:

The system boundary defines the scope of the Hotel Book karo project, delineating the interactions between actors and the system itself. It helps in visualizing the boundaries of the platform and clarifies the external entities that interact with it.

The use case diagram for the hotel book karo project highlights the essential functionalities and interactions between users and the system. It provides a clear understanding of the requirements and serves as a foundation for further development and design. This structured approach ensures

that both guest and admin needs are met, leading to a comprehensive and user-friendly booking platform.

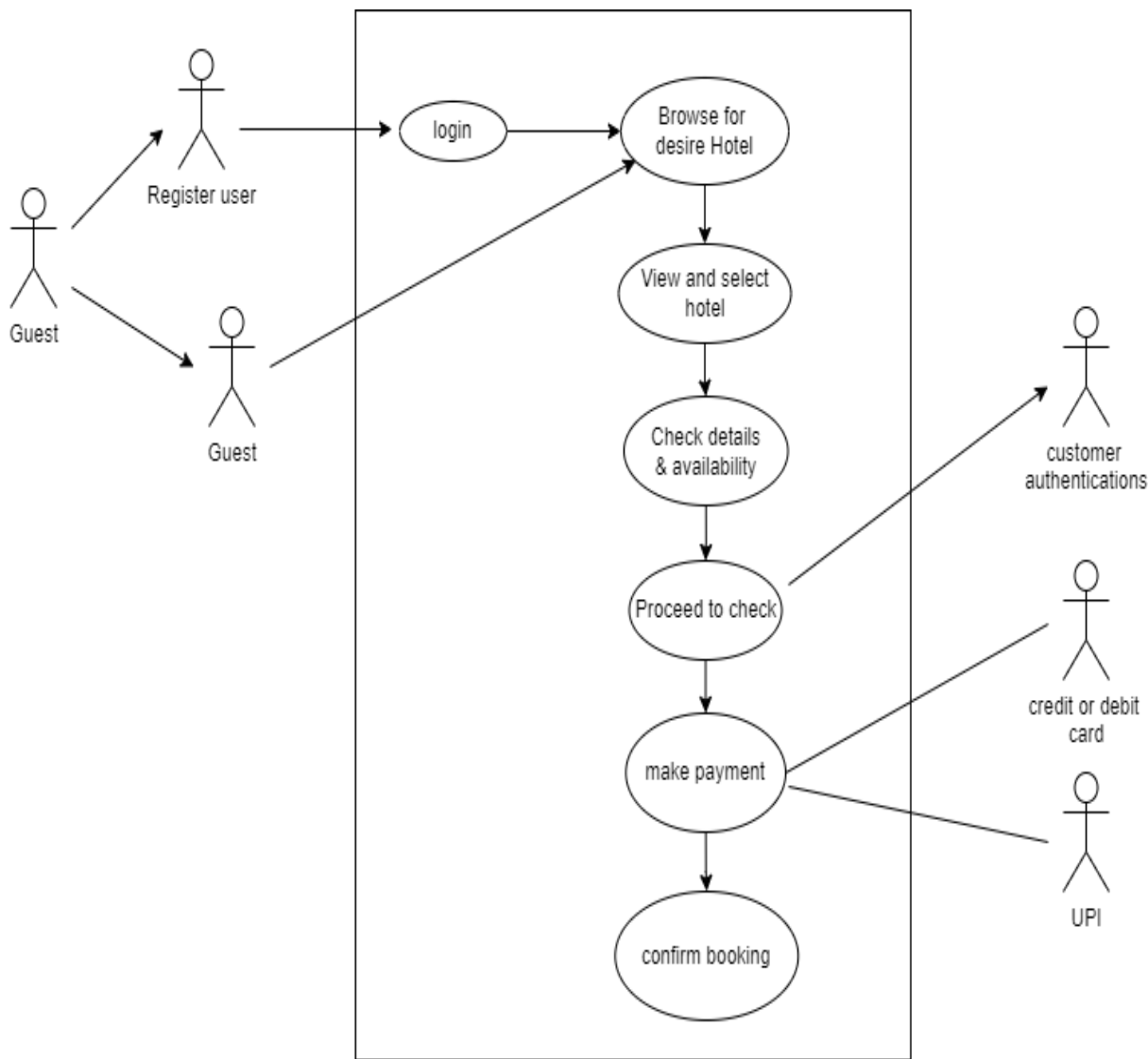


Figure 3.6: Use Case Diagram

CHAPTER 4

DESIGN

4.1 Module Wise Image

Sign-In Module:

The sign-in module of the hotel book karo website enables users to securely access their accounts by entering their email and password. Utilizing encrypted authentication protocols, it ensures data protection and user privacy, allowing guests to manage bookings and personal information efficiently.

Create an account here'. At the bottom right, there is a blue 'Login' button." data-bbox="104 389 899 703"/>

Find your next stay
Search low prices on hotels for your dream vacation...

Where are you going? Adults: 1 Children: 1 05/22/2024 05/22/2024 Search Clear

Sign In

Email

Password

Not Registered? [Create an account here](#)

Login

Figure 4.1 Sign In

Sign-Up Module:

The sign-up module of the hotel book karo website allows new users to create an account by providing their personal details, such as name, email, and password. It includes form validation and secure storage of user information, ensuring a smooth and safe registration process for users.

Create an Account

First Name

Last Name

Email

Password

Confirm Password

Create Account

Figure 4.2 Sign Up

Home Page :

The home page module of the hotel book karo website serves as the primary interface for users. It features an intuitive design with a search bar for finding hotels by location, check-in/check-out dates, and number of guests. Additionally, it showcases featured hotels, special offers, and user reviews. The module also includes navigation links to user registration/login, account management, and customer support for a seamless user experience.

Hotel Book Karo



Sign In

Find your next stay

Search low prices on hotels for your dream vacation...

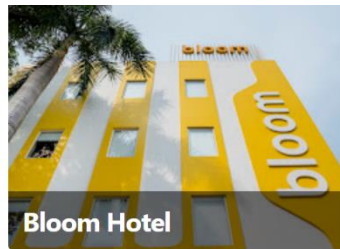
Latest Destinations

Most recent destinations added by our hosts

Latest Destinations

Most recent desinations added by our hosts



Hotel Book Karo

© 2024 Shivanshu Panwar

[Privacy Policy](#) [Terms of Service](#)


Figure 4.3 Home Page

Searching Module:

The search module of the hotel book karo website allows users to find hotels based on location, availability on specific dates, and room capacity. Users enter a destination, check-in/check-out dates, and the number of guests. The system then queries the database for hotels that match these criteria, displaying only those with available rooms that meet the specified capacity and date requirements, ensuring a tailored and efficient search experience.

Find your next stay

Search low prices on hotels for your dream vacation...

 Where are you going?

Adults: 2 Children: 1

05/23/2024

05/24/2024

Search

Clear

Filter by:

Property Rating

☐ 5 Stars

☐ 4 Stars

☐ 3 Stars

☐ 2 Stars

☐ 1 Stars

Hotel Type

☐ Budget

☐ Boutique

☐ Luxury

☐ Ski Resort

☐ Business

☐ Family


☐ Romantic

☐ Hiking Resort

☐ Cabin

7 Hotels found

Sort By ▾



★★★★★ Luxury

Sandal Suites by Lemon Tree Hotels

SANDAL SUITES, Assotech Business Cresterra, 22, Noida-Greater Noida Expy, INFOSPACE, Sector 135, Noida, Uttar Pradesh 201301 Set in a tech business park off the Noida-Greater Noida Expressway, this contemporary all-suite hotel is 15 km from India Expo Mart and 34 k...

₹2000 per night


View More

Free WiFi

Parking

Family Rooms

+3 more



★★★★★ Business

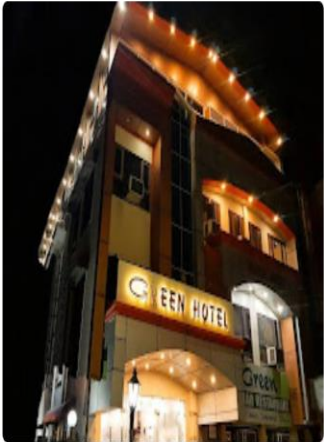
Hotel White Castle

Figure 4.4 Search Hotel

25

Hotel Details Module :

The Hotel Details module of the hotel book karo website provides comprehensive information about each hotel, including room availability, pricing, amenities, user reviews, and high-quality images. This module allows users to view detailed descriptions of the hotel’s facilities, check real-time room availability, and make informed decisions. The intuitive design ensures users can easily navigate through different sections to find all relevant details before making a booking.



Free WiFi

Parking

Non-Smoking Rooms

Fitness Center

Address & contact information

Near Ram Jhula, Behind Parmarth Nikethan, Swarg Ashram, Rishikesh, Uttarakhand 249304

060099 60097

A 2-minute walk from the Ganges River, this unpretentious modern hotel is 1.8 km from the Parmarth Niketan ashram and 23 km from the Neelkanth Mahadev Temple.

The functional rooms come with en suite bathrooms, free Wi-Fi and satellite TV. Room service is available.

Amenities include a restaurant offering Indian, Italian and Chinese cuisine, as well as a traditional Ayurvedic spa. Activities such as yoga classes are also available, and parking is free.

Check-in time: 14:00

Check-out time: 11:00

₹3965

05/23/2024

05/24/2024

Adults: 2 Children: 1

Book Now

Figure 4.5 Hotel Details

Booking Module :

The hotel booking module of the hotel book karo website academic project enables users to select and reserve rooms. It includes features for searching hotels, viewing room details, selecting check-in/check-out dates, and confirming bookings. Integrated with secure payment gateways, it ensures seamless transactions. This module also updates room availability in real-time, ensuring accurate booking status and enhancing user experience.

₹1448

05/19/2024

05/21/2024

Adults: **2** Children: **2** ▲▼

Book Now

Your Booking Details

Location:

Ghaziabad (U.P.), Ghaziabad, India

Check-in Check-out

Sun May 19 2024 Tue May 21 2024

Total length of stay:

2 nights

Guests

2 adults & 2 children

Confirm Your Details

First Name Last Name

Email

Your Price Summary

Total Cost: ₹2896.00
Includes taxes and charges

Payment Details

Card number MM / YY CVC

Confirm Booking

Figure 4.6 Booking

Booking Confirmation with Payment Module :

Hotel Booking Confirmation with Payment Module allows users to finalize their room reservations. After selecting a room and entering booking details, users proceed to the payment page where they input their payment information. The system processes the payment via a secure gateway, updates the booking status, and sends a confirmation email to the user. This ensures a seamless and secure transaction, providing users with instant confirmation of their bookings.

The screenshot displays a web form for booking confirmation. On the left, a sidebar titled "Your Booking Details" contains fields for "Location:", "Check-in", "Check-out", "Total length of stay:", and "Guests". The main content area is titled "Confirm Your Details" and includes input fields for "First Name", "Last Name", and "Email". Below these is a section titled "Your Price Summary" with a blue bar indicating the price "Includes taxes and charges". At the bottom, there is a "Payment Details" section with two input fields and a blue "Confirm Booking" button. A green banner in the top right corner reads "Booking Saved!".

Figure 4.7 Booking Confirm

Add Hotel Form Module :

The Add Hotel Form module in the hotel book karo website allows administrators to input new hotel information into the system. This form includes fields for the hotel name, address, city, state, country, postal code, phone number, email, description, and rating.

Once submitted, the data is validated and stored in the Hotel Database, making the hotel available for user searches and bookings.

Add Hotel

Name

City Country

Description

Price Per Night

Star Rating

Price Per Night

Star Rating

Type

Budget	Boutique	Luxury	Ski Resort	Business
Family	Romantic	Hiking Resort	Cabin	Beach Resort
Golf Resort	Motel	All Inclusive	Pet Friendly	Self Catering

Facilities

<input type="checkbox"/> Free WiFi	<input type="checkbox"/> Parking	<input type="checkbox"/> Airport Shuttle	<input type="checkbox"/> Family Rooms	<input type="checkbox"/> Non-Smoking Rooms
<input type="checkbox"/> Outdoor Pool	<input type="checkbox"/> Spa	<input type="checkbox"/> Fitness Center		

Guests

Adults	Children
<input type="text"/>	<input type="text"/>

Images

Choose Files No file chosen

Save

Figure 4.8 Add Hotel Form

Filter Module: The filter module of the hotel book karo website allows users to refine their search results by rating, hotel type, and facilities. Additionally, users can sort hotels by price and rating, enabling them to view options from highest to lowest or vice versa. This functionality enhances user experience by providing tailored search results, ensuring users find accommodations that meet their preferences and budget.

Property Rating

- ☐ 5 Stars
- ☐ 4 Stars
- ☐ 3 Stars
- ☐ 2 Stars
- ☐ 1 Stars

Facilities

- ☐ Free WiFi
- ☐ Parking
- ☐ Airport Shuttle
- ☐ Family Rooms
- ☐ Non-Smoking Rooms
- ☐ Outdoor Pool
- ☐ Spa
- ☐ Fitness Center

Sort By ▼

Hotel Type

- ☐ Budget
- ☐ Boutique
- ☐ Luxury
- ☐ Ski Resort
- ☐ Business
- ☐ Family
- ☐ Romantic
- ☐ Hiking Resort
- ☐ Cabin
- ☐ Beach Resort
- ☐ Golf Resort
- ☐ Motel
- ☐ All Inclusive
- ☐ Pet Friendly
- ☐ Self Catering

Max Price

Select Max Price ▼

Select Max Price

500

Figure 4.9 filtering

CHAPTER 5

DISCUSSIONS

The hotel book karo website academic project represents a comprehensive effort to address the dynamic needs of the modern travel industry through technology. This discussion delves into the various aspects of the project, from initial planning and design to implementation and potential future improvements. It reflects on the challenges encountered, the solutions implemented, and the overall impact of the project on the user experience and operational efficiency.

5.1 Project Planning and Design: The planning phase involved extensive market research and user needs analysis. Understanding the competitive landscape and the common pain points of users provided a solid foundation for the project. Key findings from this research highlighted the need for an intuitive user interface, robust search functionality, seamless booking processes, and strong security measures.

- 1. User-Centric Design:** The user-centric approach was pivotal in the design phase. By prioritizing user experience (UX) and user interface (UI) design principles, the project aimed to create a platform that was both aesthetically pleasing and functionally robust. Wireframes and prototypes were developed and tested with potential users to gather feedback and make iterative improvements. This process ensured that the final design met user expectations and facilitated easy navigation and usability.
- 2. Technical Architecture:** The technical architecture was designed to be scalable, secure, and efficient. The choice of a technology stack involving React.js for the front end, Node.js and Express.js for the back end, and MongoDB for the database was driven by the need for flexibility, performance, and ease of development. The cloud hosting solution, selected for its reliability and scalability, ensured that the platform could handle varying levels of traffic and data loads efficiently.

5.2 Implementation Challenges and Solutions: The implementation phase encountered several challenges, primarily related to integrating various functionalities and ensuring seamless performance across different modules.

- 1. Integration of Payment Systems:** Integrating secure payment gateways was a complex task due to the need to comply with financial regulations and ensure user data protection. The project utilized industry-standard encryption protocols and secured API integrations to facilitate safe and reliable transactions. Rigorous testing was conducted to identify and rectify any potential security vulnerabilities.

2. **Real – Time Room Availability:** Ensuring real-time room availability posed another challenge, especially with simultaneous bookings. Implementing an efficient database query system and utilizing caching mechanisms helped manage data synchronization issues. Transactions were handled with ACID (Atomicity, Consistency, Isolation, Durability) properties to maintain data integrity.
3. **User Authentication and Authorization:** Developing a robust user authentication and authorization system was critical to protect user data and provide personalized experiences. The use of JWT (JSON Web Tokens) for secure authentication, along with role-based access control, ensured that users and admins had appropriate access levels.

5.3 **User Feedback and Iterative Improvements:** User feedback played a crucial role in refining the platform. Beta testing with a group of potential users provided insights into areas that required improvement. Common feedback included the need for more advanced search filters, faster page loading times, and more detailed hotel information. These insights guided subsequent development iterations, focusing on enhancing performance and user satisfaction.

1. **Advance Search Filters:** Incorporating advanced search filters allowed users to refine their searches based on criteria such as price range, amenities, location, and user ratings. This enhancement significantly improved the user experience by enabling more precise and relevant search results.
2. **Performance Optimization:** Optimizing performance involved improving server response times and reducing page load times. Techniques such as lazy loading for images and content, efficient database indexing, and minimizing API response times were implemented to achieve these goals.

5.4 **Future Enhancements:** While the project successfully delivered a functional and user-friendly hotel booking platform, there are several areas for future enhancement.

1. **Machine Learning for Personalization:** Integrating machine learning algorithms could enhance personalization by providing tailored recommendations based on user preferences and booking history. This would further improve user engagement and satisfaction.
2. **Integration with Additional Services:** Expanding the platform to include additional services such as car rentals, tour packages, and local experiences could provide a more comprehensive travel solution, attracting a broader user base.
3. **Mobile Application Development:** Developing a dedicated mobile application could expand the platform's reach and usability. While the website is mobile-optimized, a native mobile app could offer a more seamless and responsive user experience.

In conclusion, the hotel book karo website academic project exemplifies how technology can be harnessed to meet the evolving needs of the travel industry. Through careful planning, user-centric design, and iterative development, the project has successfully created a platform that enhances the booking experience for users while ensuring operational efficiency and security. Continued feedback and future enhancements will further solidify its position as a valuable tool for modern travellers.

CHAPTER 6

TESTING

6.1 INTRODUCTION

In Testing is a critical phase in the software development lifecycle, ensuring that the system meets the specified requirements and functions correctly. For the hotel book karo website academic project, testing encompasses unit testing, integration testing, and system testing. Each type of testing targets different aspects of the software to identify and resolve issues effectively. In this context, three essential types of testing:

1. Unit Testing,
2. Integration Testing and
3. System Testing

form the cornerstone of quality assurance for the Hotel Book Karo website project.

6.2 TYPES OF TESTING

6.2.1 Unit Testing

Unit testing involves testing individual components or functions of the software in isolation to ensure they perform as expected. It focuses on the smallest testable parts of an application, such as functions, methods, or classes.

Scope and Objectives:

- Verify the correctness of individual functions and methods.
- Ensure that each component behaves as expected under various conditions.
- Identify and fix bugs at an early stage of development.

Test Cases:

User Registration:

- **Test Case 1:** Verify that a new user can register successfully with valid data.
- **Test Case 2:** Check that the system returns an error when trying to register with an already registered email.
- **Test Case 3:** Ensure password validation rules are enforced (e.g., minimum length, special characters).

Hotel Search:

- **Test Case 1:** Verify that searching for hotels by city returns relevant results.
- **Test Case 2:** Check that the search results are filtered correctly by date availability.
- **Test Case 3:** Ensure that invalid search criteria (e.g., invalid dates) return appropriate error messages.

Room Booking:

- **Test Case 1:** Verify that a room can be booked successfully with valid data.
- **Test Case 2:** Ensure the system prevents double booking of the same room for the same dates.
- **Test Case 3:** Check that the total price calculation is accurate based on the selected room and dates.

6.2.2. Integration Testing

Integration testing involves testing the interaction between different components of the system to ensure they work together correctly. It focuses on the interfaces and data flow between modules.

Scope and Objectives:

- Verify that different modules integrate seamlessly.
- Ensure data is correctly passed between modules.
- Identify issues related to module interactions.

Test Cases:

User and Booking Integration:

- **Test Case 1:** Verify that a user can successfully log in and book a room, with booking details correctly linked to the user account.
- **Test Case 2:** Check that booking history displays accurate information for the logged-in user.

Hotel and Room Integration:

- **Test Case 1:** Ensure that adding a new hotel automatically makes its rooms available for booking.
- **Test Case 2:** Verify that updates to hotel information (e.g., address, contact details) reflect correctly in room details.

Payment and Booking Integration:

- **Test Case 1:** Verify that successful payments update the booking status to confirmed.
- **Test Case 2:** Ensure that payment failures do not confirm bookings and prompt appropriate error messages to the user.

6.2.3 System Testing

System testing involves testing the entire system as a whole to ensure it meets the specified requirements. It validates the complete and integrated software to ensure it functions as intended.

Scope and Objectives:

- Verify the overall functionality of the system.
- Ensure the system meets all specified requirements.
- Identify defects that may arise from system-level interactions.

Test Cases:

End-to-End Booking Process:

- **Test Case 1:** Verify that a user can search for a hotel, view details, book a room, and make a payment, completing the entire process without issues.
- **Test Case 2:** Ensure that users receive confirmation emails after successful bookings.

User Account Management:

- **Test Case 1:** Verify that users can update their profile information (e.g., name, address) and that changes are saved correctly.
- **Test Case 2:** Ensure that users can view and manage their booking history accurately.

Admin Functionality:

- **Test Case 1:** Verify that admins can add, edit, and delete hotel listings, with changes reflecting correctly in the user-facing interface.
- **Test Case 2:** Ensure that admins can generate reports on bookings, payments, and user activities.

Performance Testing:

- **Test Case 1:** Verify that the system can handle a high volume of simultaneous users performing searches and bookings without performance degradation.

- **Test Case 2:** Ensure that page load times remain within acceptable limits under peak usage conditions.

Security Testing:

- **Test Case 1:** Verify that user passwords are securely hashed and stored.
- **Test Case 2:** Ensure that sensitive data, such as payment information, is transmitted securely using SSL encryption.

Effective testing of the hotel book karo website involves a layered approach encompassing unit testing, integration testing, and system testing. Each type of testing targets different levels of the software to ensure comprehensive coverage and early identification of issues. Unit testing verifies individual components, integration testing ensures proper module interactions, and system testing validates the overall functionality and performance of the system. This thorough testing strategy helps deliver a reliable, secure, and user-friendly hotel booking website.

CHAPTER 7

CODING

7.1 Frontend Code

package.json

```
{
  "name": "frontend",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite --port 5174",
    "build": "tsc && vite build",
    "lint": "eslint . --ext ts,tsx --report-unused-disable-directives --max-warnings 0",
    "preview": "vite preview"
  },
  "dependencies": {
    "@stripe/react-stripe-js": "^2.4.0",
    "@stripe/stripe-js": "^2.2.1",
    "react": "^18.2.0",
    "react-datepicker": "^4.24.0",
    "react-dom": "^18.2.0",
    "react-hook-form": "^7.48.2",
    "react-icons": "^4.12.0",
    "react-query": "^3.39.3",
    "react-router-dom": "^6.18.0"
  },
  "devDependencies": {
    "@types/react": "^18.2.15",
    "@types/react-datepicker": "^4.19.3",
    "@types/react-dom": "^18.2.7",
    "@typescript-eslint/eslint-plugin": "^6.0.0",
    "@typescript-eslint/parser": "^6.0.0",
    "@vitejs/plugin-react-swc": "^3.3.2",
    "autoprefixer": "^10.4.16",
    "eslint": "^8.45.0",
```

```

    "eslint-plugin-react-hooks": "^4.6.0",
    "eslint-plugin-react-refresh": "^0.4.3",
    "postcss": "^8.4.31",
    "tailwindcss": "^3.3.5",
    "typescript": "^5.0.2",
    "vite": "^4.4.5"
  }
}

```

index.html

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Hotel Book Karo</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.tsx"></script>
  </body>
</html>

```

App.tsx

```

import {
  BrowserRouter as Router,
  Route,
  Routes,
  Navigate,
} from "react-router-dom";
import Layout from "../layouts/Layout";
import Register from "../pages/Register";
import SignIn from "../pages/SignIn";
import AddHotel from "../pages/AddHotel";
import { useAppContext } from "../contexts/AppContext";
import MyHotels from "../pages/MyHotels";

```

```

import EditHotel from "../pages/EditHotel";
import Search from "../pages/Search";
import Detail from "../pages/Detail";
import Booking from "../pages/Booking";
import MyBookings from "../pages/MyBookings";
import Home from "../pages/Home";

```

```

const App = () => {
  const { isLoggedIn } = useAppContext();
  return (
    <Router>
      <Routes>
        <Route
          path="/"
          element={
            <Layout>
              <Home />
            </Layout>
          }
        />
        <Route
          path="/search"
          element={
            <Layout>
              <Search />
            </Layout>
          }
        />
        <Route
          path="/detail/:hotelId"
          element={
            <Layout>
              <Detail />
            </Layout>
          }
        />
        <Route
          path="/register"
          element={
            <Layout>
              <Register />

```

```

        </Layout>
    }
/>
<Route
  path="/sign-in"
  element={
    <Layout>
      <SignIn />
    </Layout>
  }
/>

{isLoggedIn && (
  <
    <Route
      path="/hotel/:hotelId/booking"
      element={
        <Layout>
          <Booking />
        </Layout>
      }
    </>
  )
}

<Route
  path="/add-hotel"
  element={
    <Layout>
      <AddHotel />
    </Layout>
  }
/>
<Route
  path="/edit-hotel/:hotelId"
  element={
    <Layout>
      <EditHotel />
    </Layout>
  }
/>
<Route
  path="/my-hotels"

```



```

        element={
          <Layout>
            <MyHotels />
          </Layout>
        }
      />
    <Route
      path="/my-bookings"
      element={
        <Layout>
          <MyBookings />
        </Layout>
      }
    />
  </>
)}
  <Route path="*" element={<Navigate to="/" />} />
</Routes>
</Router>
);
};

export default App;

```

AddHotel.tsx

```

import { useMutation } from "react-query";
import ManageHotelForm from "../forms/ManageHotelForm/ManageHotelForm";
import { useAppContext } from "../contexts/AppContext";
import * as apiClient from "../api-client";

const AddHotel = () => {
  const { showToast } = useAppContext();

  const { mutate, isLoading } = useMutation(apiClient.addMyHotel, {
    onSuccess: () => {
      showToast({ message: "Hotel Saved!", type: "SUCCESS" });
    },
    onError: () => {
      showToast({ message: "Error Saving Hotel", type: "ERROR" });
    }
  });

```

```

    },
  });

  const handleSave = (hotelFormData: FormData) => {
    mutate(hotelFormData);
  };

  return <ManageHotelForm onSave={handleSave} isLoading={isLoading} />;
};

export default AddHotel;

```

Booking.tsx

```

import { useQuery } from "react-query";
import * as apiClient from "../api-client";
import BookingForm from "../forms/BookingForm/BookingForm";
import { useSearchContext } from "../contexts/SearchContext";
import { useParams } from "react-router-dom";
import { useEffect, useState } from "react";
import BookingDetailsSummary from "../components/BookingDetailsSummary";
import { Elements } from "@stripe/react-stripe-js";
import { useAppContext } from "../contexts/AppContext";

const Booking = () => {
  const { stripePromise } = useAppContext();
  const search = useSearchContext();
  const { hotelId } = useParams();

  const [numberOfNights, setNumberOfNights] = useState<number>(0);

  useEffect(() => {
    if (search.checkIn && search.checkOut) {
      const nights =
        Math.abs(search.checkOut.getTime() - search.checkIn.getTime()) /
        (1000 * 60 * 60 * 24);

      setNumberOfNights(Math.ceil(nights));
    }
  },
  [search.checkIn, search.checkOut]);

```

```

const { data: paymentIntentData } = useQuery(
  "createPaymentIntent",
  () =>
    apiClient.createPaymentIntent(
      hotelId as string,
      numberOfNights.toString()
    ),
  {
    enabled: !!hotelId && numberOfNights > 0,
  }
);

const { data: hotel } = useQuery(
  "fetchHotelById",
  () => apiClient.fetchHotelById(hotelId as string),
  {
    enabled: !!hotelId,
  }
);

const { data: currentUser } = useQuery(
  "fetchCurrentUser",
  apiClient.fetchCurrentUser
);

if (!hotel) {
  return <></>;
}

return (
  <div className="grid md:grid-cols-[1fr_2fr]">
    <BookingDetailsSummary
      checkIn={search.checkIn}
      checkOut={search.checkOut}
      adultCount={search.adultCount}
      childCount={search.childCount}
      numberOfNights={numberOfNights}
      hotel={hotel}
    />
    {currentUser && paymentIntentData && (

```

```

    <Elements
      stripe={stripePromise}
      options={{
        clientSecret: paymentIntentData.clientSecret,
      }}
    >
      <BookingForm
        currentUser={currentUser}
        paymentIntent={paymentIntentData}
      />
    </Elements>
  )}
</div>

);
};

```

```
export default Booking;
```

Detail.tsx

```

import { useQuery } from "react-query";
import { useParams } from "react-router-dom";
import * as apiClient from "../api-client";
import { AiFillStar } from "react-icons/ai";
import GuestInfoForm from "../forms/GuestInfoForm/GuestInfoForm";

const Detail = () =>
{
  const { hotelId } = useParams();

  const { data: hotel } = useQuery(
    "fetchHotelById",
    () => apiClient.fetchHotelById(hotelId || ""),
    {
      enabled: !!hotelId,
    }
  );

  if (!hotel) {
    return <></>;
  }
}

```

```

return (
  <div className="space-y-6">
    <div>
      <span className="flex">
        { Array.from({ length: hotel.starRating }).map(() => (
          <AiFillStar className="fill-yellow-400" />
        ))}
      </span>
      <h1 className="text-3xl font-bold">{hotel.name}</h1>
    </div>

    <div className="grid grid-cols-1 lg:grid-cols-3 gap-4">
      {hotel.imageUrls.map((image) => (
        <div className="h-[300px]">
          <img
            src={image}
            alt={hotel.name}
            className="rounded-md w-full h-full object-cover object-center"
          />
        </div>
      ))}
    </div>

    <div className="grid grid-cols-1 lg:grid-cols-4 gap-2">
      {hotel.facilities.map((facility) => (
        <div className="border border-slate-300 rounded-sm p-3">
          {facility}
        </div>
      ))}
    </div>

    <div className="grid grid-cols-1 lg:grid-cols-[2fr_1fr]">
      <div className="whitespace-pre-line">{hotel.description}</div>
      <div className="h-fit">
        <GuestInfoForm
          pricePerNight={hotel.pricePerNight}
          hotelId={hotel._id}
        />
      </div>
    </div>
  </div>

```

```

    </div>
  );
};

export default Detail;

```

7.2 Backend Code

package.json

```

{
  "name": "backend",
  "version": "1.0.0",
  "description": "",
  "main": "./src/index.ts",
  "scripts": {
    "dev": "nodemon",
    "build": "npm install && npx tsc",
    "start": "node ./dist/index.js",
    "e2e": "cross-env DOTENV_CONFIG_PATH=.env.e2e nodemon"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "cloudinary": "^1.41.0",
    "cookie-parser": "^1.4.6",
    "cors": "^2.8.5",
    "cross-env": "^7.0.3",
    "dotenv": "^16.3.1",
    "express": "^4.18.2",
    "express-validator": "^7.0.1",
    "jsonwebtoken": "^9.0.2",
    "mongodb": "^6.2.0",
    "mongoose": "^8.0.0",
    "multer": "^1.4.5-lts.1",
    "stripe": "^14.8.0"
  },

```

```

    "devDependencies": {
      "@types/bcryptjs": "^2.4.6",
      "@types/cookie-parser": "^1.4.6",
      "@types/cors": "^2.8.16",
      "@types/express": "^4.17.21",
      "@types/jsonwebtoken": "^9.0.5",
      "@types/multer": "^1.4.11",
      "@types/node": "^20.9.0",
      "nodemon": "^3.0.1",
      "ts-node": "^10.9.1",
      "typescript": "^5.2.2"
    }
  }
}

```

index.ts

```

import express, { Request, Response } from "express";
import cors from "cors";
import "dotenv/config";
import mongoose from "mongoose";
import userRoutes from "./routes/users";
import authRoutes from "./routes/auth";
import cookieParser from "cookie-parser";
import path from "path";
import { v2 as cloudinary } from "cloudinary";
import myHotelRoutes from "./routes/my-hotels";
import hotelRoutes from "./routes/hotels";
import bookingRoutes from "./routes/my-bookings";

cloudinary.config({
  cloud_name: process.env.CLOUDINARY_CLOUD_NAME,
  api_key: process.env.CLOUDINARY_API_KEY,
  api_secret: process.env.CLOUDINARY_API_SECRET,
});

mongoose.connect(process.env.MONGODB_CONNECTION_STRING as string);

const app = express();
app.use(cookieParser());
app.use(express.json());

```

```

app.use(express.urlencoded({ extended: true }));
app.use(
  cors({
    origin: process.env.FRONTEND_URL,
    credentials: true,
  })
);

app.use("/api/auth", authRoutes);
app.use("/api/users", userRoutes);
app.use("/api/my-hotels", myHotelRoutes);
app.use("/api/hotels", hotelRoutes);
app.use("/api/my-bookings", bookingRoutes);

app.listen(7000, () => {
  console.log("server running on localhost:7000");
});

```

Models :

hotel.ts

```

import mongoose from "mongoose";
import { BookingType, HotelType } from "../shared/types";

const bookingSchema = new mongoose.Schema<BookingType>(
  {
    firstName: { type: String, required: true },
    lastName: { type: String, required: true },
    email: { type: String, required: true },
    adultCount: { type: Number, required: true },
    childCount: { type: Number, required: true },
    checkIn: { type: Date, required: true },
    checkOut: { type: Date, required: true },
    userId: { type: String, required: true },
    totalCost: { type: Number, required: true },
  }
);

const hotelSchema = new mongoose.Schema<HotelType>(
  {

```



```

    userId: { type: String, required: true },
    name: { type: String, required: true },
    city: { type: String, required: true },
    country: { type: String, required: true },
    description: { type: String, required: true },
    type: { type: String, required: true },
    adultCount: { type: Number, required: true },
    childCount: { type: Number, required: true },
    facilities: [{ type: String, required: true }],
    pricePerNight: { type: Number, required: true },
    starRating: { type: Number, required: true, min: 1, max: 5 },
    imageUrls: [{ type: String, required: true }],
    lastUpdated: { type: Date, required: true },
    bookings: [bookingSchema],
  });

const Hotel = mongoose.model<HotelType>("Hotel", hotelSchema);
export default Hotel;

```

user.ts

```

import mongoose from "mongoose";
import bcrypt from "bcryptjs";
import { UserType } from "../shared/types";

const userSchema = new mongoose.Schema(
  {
    email: { type: String, required: true, unique: true },
    password: { type: String, required: true },
    firstName: { type: String, required: true },
    lastName: { type: String, required: true },
  }
);

userSchema.pre("save", async function (next) {
  if (this.isModified("password")) {
    this.password = await bcrypt.hash(this.password, 8);
  }
  next();
});

const User = mongoose.model<UserType>("User", userSchema);

```

```
export default User;
```

types.ts

```
export type UserType =  
{  
  _id: string;  
  email: string;  
  password: string;  
  firstName: string;  
  lastName: string;  
};
```

```
export type HotelType =  
{  
  _id: string;  
  userId: string;  
  name: string;  
  city: string;  
  country: string;  
  description: string;  
  type: string;  
  adultCount: number;  
  childCount: number;  
  facilities: string[];  
  pricePerNight: number;  
  starRating: number;  
  imageUrls: string[];  
  lastUpdated: Date;  
  bookings: BookingType[];  
};
```

```
export type BookingType =  
{  
  _id: string;  
  userId: string;  
  firstName: string;  
  lastName: string;  
  email: string;  
  adultCount: number;
```

```

    childCount: number;
    checkIn: Date;
    checkOut: Date;
    totalCost: number;
};

export type HotelSearchResponse =
{
    data: HotelType[];
    pagination: {
        total: number;
        page: number;
        pages: number;
    };
};

export type PaymentIntentResponse =
{
    paymentIntentId: string;
    clientSecret: string;
    totalCost: number;
};

```

auth.ts

```

import { NextFunction, Request, Response } from "express";
import jwt, { JwtPayload } from "jsonwebtoken";

declare global
{
    namespace Express {
        interface Request {
            userId: string;
        }
    }
}

const verifyToken = (req: Request, res: Response, next: NextFunction) =>
{
    const token = req.cookies["auth_token"];
    if (!token) {

```

```

    return res.status(401).json({ message: "unauthorized" });
  }

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET_KEY as string);
    req.userId = (decoded as JwtPayload).userId;
    next();
  } catch (error) {
    return res.status(401).json({ message: "unauthorized" });
  }
};

export default verifyToken;

```

hotels.ts

```

import express, { Request, Response } from "express";
import Hotel from "../models/hotel";
import { BookingType, HotelSearchResponse } from "../shared/types";
import { param, validationResult } from "express-validator";
import Stripe from "stripe";
import verifyToken from "../middleware/auth";

const stripe = new Stripe(process.env.STRIPE_API_KEY as string);

const router = express.Router();

router.get("/search", async (req: Request, res: Response) =>
{
  try {
    const query = constructSearchQuery(req.query);

    let sortOptions = { };
    switch (req.query.sortOption) {
      case "starRating":
        sortOptions = { starRating: -1 };
        break;
      case "pricePerNightAsc":
        sortOptions = { pricePerNight: 1 };
        break;
      case "pricePerNightDesc":

```

```

        sortOptions = { pricePerNight: -1 };
        break;
    }

    const pageSize = 5;
    const pageNumber = parseInt(
        req.query.page ? req.query.page.toString() : "1"
    );
    const skip = (pageNumber - 1) * pageSize;

    const hotels = await Hotel.find(query)
        .sort(sortOptions)
        .skip(skip)
        .limit(pageSize);

    const total = await Hotel.countDocuments(query);

    const response: HotelSearchResponse =
    {
        data: hotels,
        pagination: {
            total,
            page: pageNumber,
            pages: Math.ceil(total / pageSize),
        },
    };

    res.json(response);
} catch (error) {
    console.log("error", error);
    res.status(500).json({ message: "Something went wrong" });
}
});

router.get("/", async (req: Request, res: Response) =>
{
    try {
        const hotels = await Hotel.find().sort("-lastUpdated");
        res.json(hotels);
    } catch (error) {
        console.log("error", error);
    }
}

```

```

    res.status(500).json({ message: "Error fetching hotels" });
  }
});

router.get(
  "/:id",
  [param("id").notEmpty().withMessage("Hotel ID is required")],
  async (req: Request, res: Response) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }

    const id = req.params.id.toString();

    try {
      const hotel = await Hotel.findById(id);
      res.json(hotel);
    } catch (error) {
      console.log(error);
      res.status(500).json({ message: "Error fetching hotel" });
    }
  }
);

router.post(
  "/:hotelId/bookings/payment-intent",
  verifyToken,
  async (req: Request, res: Response) =>
  {
    const { numberOfNights } = req.body;
    const hotelId = req.params.hotelId;

    const hotel = await Hotel.findById(hotelId);
    if (!hotel) {
      return res.status(400).json({ message: "Hotel not found" });
    }

    const totalCost = hotel.pricePerNight * numberOfNights;

    const paymentIntent = await stripe.paymentIntents.create({

```

```

    amount: totalCost * 100,
    currency: "gbp",
    metadata: {
      hotelId,
      userId: req.userId,
    },
  });

  if (!paymentIntent.client_secret) {
    return res.status(500).json({ message: "Error creating payment intent" });
  }

  const response = {
    paymentIntentId: paymentIntent.id,
    clientSecret: paymentIntent.client_secret.toString(),
    totalCost,
  };

  res.send(response);
}

);

router.post(
  "/:hotelId/bookings",
  verifyToken,
  async (req: Request, res: Response) =>
  {
    try {
      const paymentIntentId = req.body.paymentIntentId;

      const paymentIntent = await stripe.paymentIntents.retrieve(
        paymentIntentId as string
      );

      if (!paymentIntent) {
        return res.status(400).json({ message: "payment intent not found" });
      }

      if (
        paymentIntent.metadata.hotelId !== req.params.hotelId ||
        paymentIntent.metadata.userId !== req.userId

```

```

    ) {
      return res.status(400).json({ message: "payment intent mismatch" });
    }

    if (paymentIntent.status !== "succeeded")
  {
    return res.status(400).json({
      message: `payment intent not succeeded. Status: ${paymentIntent.status}`,
    });
  }

  const newBooking: BookingType =
  {
    ...req.body,
    userId: req.userId,
  };

  const hotel = await Hotel.findOneAndUpdate(
    { _id: req.params.hotelId },
    {
      $push: { bookings: newBooking },
    }
  );

  if (!hotel) {
    return res.status(400).json({ message: "hotel not found" });
  }

  await hotel.save();
  res.status(200).send();
} catch (error) {
  console.log(error);
  res.status(500).json({ message: "something went wrong" });
}
);

const constructSearchQuery = (queryParams: any) =>
{
  let constructedQuery: any = {};

```



```

if (queryParams.destination) {
  constructedQuery.$or = [
    { city: new RegExp(queryParams.destination, "i") },
    { country: new RegExp(queryParams.destination, "i") },
  ];
}

if (queryParams.adultCount) {
  constructedQuery.adultCount = {
    $gte: parseInt(queryParams.adultCount),
  };
}

if (queryParams.childCount) {
  constructedQuery.childCount = {
    $gte: parseInt(queryParams.childCount),
  };
}

if (queryParams.facilities) {
  constructedQuery.facilities = {
    $all: Array.isArray(queryParams.facilities)
      ? queryParams.facilities
      : [queryParams.facilities],
  };
}

if (queryParams.types) {
  constructedQuery.type = {
    $in: Array.isArray(queryParams.types)
      ? queryParams.types
      : [queryParams.types],
  };
}

if (queryParams.stars) {
  const starRatings = Array.isArray(queryParams.stars)
    ? queryParams.stars.map((star: string) => parseInt(star))
    : parseInt(queryParams.stars);

  constructedQuery.starRating = { $in: starRatings };
}

```

```
}

if (queryParams.maxPrice) {
  constructedQuery.pricePerNight = {
    $lte: parseInt(queryParams.maxPrice).toString(),
  };
}

return constructedQuery;
};

export default router;
```

CHAPTER 8

CONCLUSION

In the development of the hotel book karo website as an academic project has been a comprehensive and insightful endeavour, encompassing the full spectrum of project planning, research, design, implementation, and evaluation. This project aimed not only to create a functional and user-friendly hotel booking platform but also to provide a deep understanding of the practical applications of software development principles and project management strategies.

8.1 Projects Goal and Achievements:

The primary objective of the project was to design and develop a robust hotel book karo website that could cater to the needs of modern traveller's by providing a seamless and intuitive booking experience. The project set out to achieve several specific goals, which included:

- **User-Friendly Interface:** Ensuring that the platform is easy to navigate for users of all technical proficiencies.
- **Responsive Design:** Creating a website that performs well on various devices, including desktops, tablets, and smartphones.
- **Comprehensive Search and Filter Options:** Implementing advanced search functionalities to allow users to find hotels based on location, price range, amenities, and user reviews.
- **Secure Payment Processing:** Integrating secure payment gateways to protect users' financial information.
- **Scalable Architecture:** Building a scalable backend infrastructure capable of handling increasing user traffic and data loads.

Through diligent planning, development, and testing phases, the project successfully met these goals. The final product is a sophisticated booking platform that offers an enhanced user experience, robust performance, and high security standards.

8.2 Technical Insights:

From a technical standpoint, this project provided an invaluable learning experience in several key areas:

- **Front-End Development:** Utilizing Tailwind CSS, TypeScript, and React.js allowed for the creation of a responsive and dynamic user interface. This aspect of the project reinforced the importance of user-centric design and the need for continuous usability testing.

- **Back-End Development:** Implementing Node.js and Express.js for server-side logic and database management using MongoDB highlighted the significance of efficient data handling and server management. This included learning about RESTful API development and CRUD operations.
- **Security Measures:** Integrating SSL encryption, secure authentication methods, and PCI-compliant payment processing provided practical experience in safeguarding user data and building trust through robust security protocols.
- **Database Design:** Designing a relational database schema to support various entities like users, hotels, rooms, bookings, and payments was a critical part of the project. This involved understanding entity relationships, ensuring data normalization, and optimizing query performance. success.

8.3 Challenges and Solutions:

Throughout the project, several challenges were encountered, each presenting unique learning opportunities:

- **Technical Challenges:** Issues such as optimizing database queries and ensuring cross-browser compatibility required thorough research and problem-solving skills. The team leveraged online resources, community forums, and expert consultations to overcome these hurdles.
- **Time Management:** Balancing academic commitments with project deadlines necessitated effective time management strategies. Breaking down tasks into manageable sprints and setting realistic milestones helped in maintaining steady progress.
- **User Feedback Integration:** Incorporating feedback from user testing into the development cycle was essential for refining features and enhancing user experience. This iterative feedback loop was vital for ensuring that the platform met user expectations.

8.4 Future Enhancement:

While the project has successfully achieved its initial objectives, there are several areas for future enhancement:

- **AI Integration:** Incorporating AI-driven recommendations and chatbots could further personalize the user experience and provide real-time assistance.
- **Advanced Analytics:** Implementing advanced analytics tools to track user behaviour and booking trends can offer valuable insights for continuous improvement.

- **Global Expansion:** Expanding the platform to support multiple languages and currencies would make it accessible to a broader audience.
- **Mobile Application:** Developing a dedicated mobile application could provide a more seamless experience for smartphone users, leveraging native mobile capabilities. resilience.

8.5 Recommendation:

Considering the comprehensive examination of all the above aspects, it is recommended to proceed with the Hotel Book karo project. The potential for market success, coupled with the robust technical foundation and operational considerations, positions the project favourably. Continuous monitoring and adaptation to evolving industry trends will be essential for sustained success.

The hotel book karo academic project stands as a testament to the practical application of software development skills, project management techniques, and collaborative teamwork. It underscores the importance of a user-centric approach, robust technical foundations, and iterative development processes. This project not only resulted in a functional and efficient hotel booking platform but also provided a comprehensive learning experience that will inform future endeavours in the field of software development.

CHAPTER 9

BIBLIOGRAPHY

The following are the websites that we had analysed for our Hotel Book Karo website projects:

- W3School
- React.js Official website link: <https://react.dev/>
- Mongo DB official website link: <https://www.mongodb.com/>
- TypeScript official website link: <https://www.typescriptlang.org/>
- Tailwind official website link: <https://tailwindcss.com/>
- Stripe payment gateway link: <https://stripe.com/in/payments>