**A PROJECT ON ROOM FINDER WEB APP**

SUBMITTED BY

JEETEN KUMAR MANDAL

STUDENT ID: LC00016000981

FINAL YEAR PROJECT REPORT SUBMITTED IN

PARTIAL FULFILLMENT FOR THE DEGREE OF

BACHELOR OF INFORMATION TECHNOLOGY

FACULTY OF COMPUTER SCIENCE IN MULTIMEDIA

LINCOLN UNIVERSITY COLLEGE

MALAYSIA

JULY 2024

# Declaration

We hereby declare that this report is based on our own independent work, except for Quotation and summaries which have been dully acknowledged. I also declare that no part of this work has been submitted for any degree to this or any other university.

|  |  |
| --- | --- |
| Date: 12 June 2024 | Signature: |
|  | Name: |

Supervisor:

Faculty of Computer Science in Multimedia,

Nepal Business College,

Lincoln University

Supervisor:

# Acknowledgement

First of all, I thank Almighty for inspiring me with strength and energy to complete this project on time. I am deeply grateful to all those who have encouraged and helped me, discussed ideas and insights, which have contributed in various ways to complete this project.

My deepest gratitude and appreciation go to my supervisor Er. Abhay Das for his patience and very helpful comments, her bright ideas, guidance and for being generous with her knowledge and experience in supervising

# Abstract

The Room Finder Web App is designed to simplify the process of finding and booking rooms for rent. This application allows users to post available rooms and search for rooms based on various criteria such as location, price, and amenities. By providing a centralized platform for room listings and bookings, the Room Finder Web App aims to bridge the gap between room seekers and room providers, making the rental process more efficient and user-friendly. This project leverages modern web technologies, including Next.js, to deliver a responsive and interactive user experience.

Table of Contents

[Declaration ii](#_Toc173908940)

[Acknowledgement iii](#_Toc173908941)

[Abstract iv](#_Toc173908942)

[List of tables viii](#_Toc173908943)

[List of figures ix](#_Toc173908944)

[List of abbreviations x](#_Toc173908945)

[Chapter one Introduction 1](#_Toc173908946)

[1.1 Background of the Study 1](#_Toc173908947)

[1.2 Problem Statement 1](#_Toc173908948)

[1.3 Project Objectives 2](#_Toc173908949)

[1.4 Significance of the Project 2](#_Toc173908950)

[1.5 Limitations of the Project 3](#_Toc173908951)

[1.6 Organisation of the Project 3](#_Toc173908952)

[Chapter Two Literature Review 5](#_Toc173908953)

[CHAPTER THREE THORETICAL BACKGROUND OF THE PROJECT 7](#_Toc173908954)

[CHAPTER FOUR PROPOSED METHODOLOGY 9](#_Toc173908955)

[Chapter Five Result and Discussion 13](#_Toc173908956)

[5.1 System Analysis 13](#_Toc173908957)

**[5.1.1 Functional Requirements](#_Toc173908958)** [13](#_Toc173908958)

**[5.1.2 Non Functional Requirements](#_Toc173908959)** [14](#_Toc173908959)

**[5.1.3 Use Case Diagram](#_Toc173908960)** [16](#_Toc173908960)

**[5.1.4 Context Diagram](#_Toc173908961)** [17](#_Toc173908961)

**[5.1.5 Level 1 Data Flow Diagram](#_Toc173908962)** [18](#_Toc173908962)

**[5.1.6 ER Diagram](#_Toc173908963)** [19](#_Toc173908963)

**[5.1.7 Activity Diagram](#_Toc173908964)** [20](#_Toc173908964)

[5.2 System Design 21](#_Toc173908965)

**[5.2.1 System Architecture](#_Toc173908966)** [21](#_Toc173908966)

**[5.2.2](#_Toc173908967)****[User Interface (UI) Design](#_Toc173908967)** [22](#_Toc173908967)

**[5.2.3 Deployment Design](#_Toc173908968)** [23](#_Toc173908968)

**[5.3.4 Component Diagram](#_Toc173908969)** [24](#_Toc173908969)

[5.3 Implementation 25](#_Toc173908970)

**[5.3.1 Technology Stack](#_Toc173908971)** [25](#_Toc173908971)

[5.3.2 Frontend Implementation 28](#_Toc173908972)

**[5.3.3 Backend Implementation](#_Toc173908973)** [29](#_Toc173908973)

**[5.3.4 Challenges in Implementation](#_Toc173908974)** [31](#_Toc173908974)

[5.4 Testing 32](#_Toc173908975)

**[5.4.1 Overview of Testing Strategy](#_Toc173908976)** [32](#_Toc173908976)

**[5.4.2 Unit Testing](#_Toc173908977)** [33](#_Toc173908977)

**[5.4.3 Integration Testing](#_Toc173908978)** [33](#_Toc173908978)

**[5.4.4 Performance Testing](#_Toc173908979)** [35](#_Toc173908979)

**[5.4.5 User Acceptance Testing](#_Toc173908980)** [36](#_Toc173908980)

**[5.4.6 Bug Tracking and Regression Testing](#_Toc173908981)** [37](#_Toc173908981)

[5.5 Deployment 38](#_Toc173908982)

**[5.5.1 Deployment Strategy](#_Toc173908983)** [38](#_Toc173908983)

**[5.5.2 Deployment Process](#_Toc173908984)** [39](#_Toc173908984)

[Chapter SIX Conclusion and Recommendation 41](#_Toc173908985)

[6.1 Conclusion 41](#_Toc173908986)

[6.2 Limitations 42](#_Toc173908987)

[6.3 Recommendations 43](#_Toc173908988)

[Chapter Seven References 45](#_Toc173908989)

# List of tables

|  |  |  |
| --- | --- | --- |
| **Table** | **Description** | **Page** |

# List of figures

|  |  |  |
| --- | --- | --- |
| **Figures** | **Description** | **Page** |
|  |  |  |
| Figure 2.1: | **cxc** | 107 |

# List of abbreviations

|  |  |
| --- | --- |
| **Abbreviations** | **Meaning** |
|  |  |
| SCM | Supply Chain Management |

# Chapter one Introduction

## Background of the Study

The rental market is often fragmented, with room seekers relying on multiple sources such as classified ads, social media, and word-of-mouth to find suitable accommodations. This can be time-consuming and often leads to frustration due to misinformation or outdated listings. Similarly, room providers face challenges in reaching a broad audience and managing inquiries efficiently. The Room Finder Web App addresses these issues by creating a centralized, easy-to-use platform where users can list and search for rooms, streamlining the rental process for both parties.

## Problem Statement

Finding a suitable room for rent is often a complex and tedious task, involving multiple visits, extensive negotiations, and the risk of encountering outdated or incorrect information. Traditional methods, such as classified ads and informal networks, are inefficient and unreliable. Room providers also struggle to manage listings and reach potential tenants effectively. There is a need for a comprehensive, user-friendly platform that simplifies the process of finding and booking rooms while providing accurate and up-to-date information (Shrestha, 2021). The Room Finder Web App aims to address these challenges by offering a unified solution that enhances the rental experience for all users.

## Project Objectives

* To develop a web application that enables users to easily find and book rooms for rent.
* To provide a platform for room providers to list their rooms with detailed descriptions and images.
* To implement advanced search and filter functionalities to help users find rooms that meet their specific requirements.
* To ensure a secure and efficient user experience through robust authentication and data management systems.
* To create a responsive design that is accessible on various devices, including mobile phones and tablets.

## Significance of the Project

The Room Finder Web App significantly improves the room rental process by reducing the time and effort required to find suitable accommodations. It benefits room seekers by providing a comprehensive platform with accurate and up-to-date listings, advanced search options, and a user-friendly interface. Room providers benefit from an efficient system to manage listings and communicate with potential tenants. By fostering transparency and trust through features like reviews and ratings, the app enhances the overall rental experience. The project thus addresses a critical need in the rental market and has the potential to become a valuable tool for users.

## Limitations of the Project

Despite its many advantages, the Room Finder Web App has several limitations. The initial development phase may face technical challenges, such as ensuring data security and maintaining server performance under high traffic. The accuracy of room listings depends heavily on user-generated content, which can lead to issues with misinformation or outdated information. Additionally, attracting a sufficient user base of both room seekers and providers requires significant marketing efforts. The app's functionality may also be limited in areas with low internet penetration or technological adoption. Addressing these limitations is crucial for the app's success and widespread adoption.

## Organisation of the Project

The report is organized as follows:

* **Chapter 1:** Introduction, which includes the abstract, background of the study, problem statement, project objectives, significance, limitations, and organization of the report.
* **Chapter 2:** Literature Review, providing an overview of existing solutions and related research in the field of room rental applications.
* **Chapter 3:** Theoretical background, provides the theory of techniques used in the project
* **Chapter 4:** Methodology, detailing the design and development process of the Room Finder Web App, including technology stack, system architecture, and implementation steps.
* **Chapter 5:** Conclusion and Discussion, presenting the outcomes of the project, testing results, and user feedback.
* **Chapter 6:** Conclusion and Future Work, summarizing the project’s achievements, limitations, and potential areas for future enhancement.
* **References:** A list of all sources cited in the report.
* **Appendices:** Additional material supporting the report, such as code snippets, screenshots, and user manuals.

# Chapter Two Literature Review

The choice to develop a Room Finder Web App for my final year project emerged from a recognition of the unique challenges in the rental housing market in Nepal, particularly in rapidly urbanizing areas like Kathmandu and Pokhara. Traditional methods of finding rooms, such as through local classifieds or word-of-mouth, are often inefficient and unreliable, leading to frustration for both room seekers and providers. Studies in Nepal have highlighted the growing demand for more organized and user-friendly platforms that can streamline the process of finding rental accommodations.

Research conducted by local scholars, such as the study by Shrestha (2021), emphasizes the difficulties faced by students and young professionals in locating affordable housing in Kathmandu. The study points out the lack of a centralized system for rental listings, which often results in outdated information and missed opportunities. Moreover, the rapid urbanization in cities like Kathmandu has exacerbated these challenges, making it increasingly difficult for newcomers to navigate the rental market efficiently. (Shrestha, 2021)

Technological advancements, coupled with the increasing use of the internet and smartphones in Nepal, presented a clear opportunity to address these issues. The availability of web development frameworks like Next.js and the growing accessibility of cloud-based solutions in Nepal made the development of a Room Finder Web App not only feasible but also timely. The application aims to centralize room listings, provide accurate and up-to-date information, and improve the overall rental experience in Nepal. (Nepali, 2022)

Given the practical implications and the potential to make a meaningful impact on the local rental market, this topic was chosen as an ideal focus for my final year project.

# CHAPTER THREE THORETICAL BACKGROUND OF THE PROJECT

The development of the Room Finder Web App is grounded in several key theoretical concepts from computer science, web development, and user experience (UX) design. These theories form the foundation of the project, guiding the design and implementation of the app to ensure it meets the needs of users effectively.

**Web Development Frameworks**

The Room Finder Web App is built using Next.js, a React-based framework that enables server-side rendering (SSR) and static site generation (SSG). These features are crucial for enhancing the performance and SEO of the web application. SSR allows the app to pre-render pages on the server before sending them to the client, which significantly reduces load times and improves the user experience. The use of SSG enables the app to generate static HTML pages at build time, further improving performance, especially in a high-traffic environment. The choice of Next.js aligns with modern web development practices that prioritize speed, scalability, and ease of maintenance.

**Database Management Systems**

A critical component of the Room Finder Web App is its ability to store, retrieve, and manage data efficiently. The app likely uses a NoSQL database, such as MongoDB, due to its flexibility and scalability. MongoDB is particularly well-suited for applications that handle large volumes of unstructured data, such as user profiles, room listings, and booking transactions. Its document-based data model allows for the storage of complex data structures, which can be easily queried and updated as needed. This approach is essential for managing dynamic and varied data that the app needs to handle.

**User Experience (UX) Design**

UX design principles are integral to the success of the Room Finder Web App. The goal is to create an intuitive, user-friendly interface that meets the needs of both room seekers and providers. This involves applying key UX principles, such as simplifying navigation, providing clear visual feedback, and ensuring consistency across different parts of the app. A well-designed user interface (UI) that follows these principles can significantly enhance user satisfaction and engagement, making the app more likely to be adopted and used regularly.

# CHAPTER FOUR PROPOSED METHODOLOGY

The development of the Room Finder Web App will follow a systematic and iterative approach, leveraging agile methodology to ensure flexibility and responsiveness to changes throughout the project lifecycle. The methodology is divided into several key phases:

**1. Requirement Analysis**

* **Stakeholder Consultation:** The first step involves gathering detailed requirements from key stakeholders, including potential users, landlords, and real estate professionals. This phase will include interviews, surveys, and focus groups to identify the core features and functionalities required for the app.
* **Requirement Specification:** Based on the collected data, a comprehensive requirements specification document will be created, outlining the functional and non-functional requirements, user stories, and acceptance criteria.

**2. System Design**

* **Architectural Design:** The system architecture will be designed using a three-tier architecture model, comprising the presentation layer (front-end), application layer (back-end), and data layer (database). This will ensure a clear separation of concerns, making the system easier to develop, maintain, and scale.
* **Database Design:** A NoSQL database, such as MongoDB, will be used to store user profiles, room listings, booking details, and other related data. The database schema will be designed to optimize query performance and accommodate the dynamic nature of the data.
* **User Interface Design:** Wireframes and prototypes of the user interface (UI) will be created using tools like Figma. The design will focus on usability, accessibility, and responsiveness to ensure a seamless user experience across different devices.

**3. Technology Stack Selection**

* **Front-End Development:** The front-end will be developed using Next.js, a React-based framework that supports server-side rendering (SSR) and static site generation (SSG). These features will enhance the app’s performance and SEO, providing a fast and responsive user experience.
* **Back-End Development:** The back-end will be built using Node.js and Express.js to handle server-side logic, manage user authentication, and process requests. RESTful APIs will be developed to facilitate communication between the front-end and back-end.
* **Database Management:** MongoDB will be used as the primary database due to its scalability and flexibility in handling unstructured data. The database will be hosted on a cloud service like MongoDB Atlas to ensure high availability and security.
* **Authentication and Security:** User authentication will be managed using OTPs to ensure secure access control. HTTPS will be implemented to encrypt data transmitted between clients and servers, safeguarding user privacy.

**4**. **Development Phase**

* **Agile Sprints:** Development will be carried out in iterative sprints. Each sprint will focus on implementing specific features, followed by testing and review. This approach allows for continuous integration and testing, ensuring that any issues are identified and resolved early in the development process.
* **Front-End Implementation:** The UI components will be developed using React.js, with Next.js handling the SSR and routing. The front-end will be integrated with the back-end through RESTful APIs, allowing for dynamic data rendering and real-time updates.
* **Back-End Implementation:** The server-side logic will be implemented using Node.js. The back-end will handle user authentication, data processing, and interaction with the MongoDB database. API endpoints will be created to support CRUD (Create, Read, Update, Delete) operations on room listings and user data.

**5. Testing and Quality Assurance**

* **Unit Testing:** Individual components and modules will undergo unit testing to ensure they function correctly in isolation.
* **Integration Testing:** Integration testing will be conducted to ensure that different parts of the application work together as expected. This includes testing the interaction between the front-end, back-end, and database.
* **User Acceptance Testing (UAT):** A group of users will be invited to test the application in a real-world scenario. Their feedback will be used to refine and improve the user experience and address any remaining issues.

**6. Deployment**

* **Cloud Deployment:** The application will be deployed on a cloud platform like Vercel, which supports serverless deployment for Next.js applications. MongoDB will be hosted on MongoDB Atlas, ensuring scalability and reliability.
* **Continuous Deployment:** A CI/CD (Continuous Integration/Continuous Deployment) pipeline will be set up to automate the deployment process. This will enable automatic deployment of updates and bug fixes, ensuring the app remains up-to-date.

**7. Maintenance and Future Enhancements**

* **Post-Launch Support:** After deployment, the application will undergo regular monitoring and maintenance to ensure it operates smoothly. Any bugs or issues reported by users will be promptly addressed.
* **Feature Enhancements:** Based on user feedback and market trends, additional features such as payment integration, advanced filtering options, and AI-driven recommendations will be considered for future updates.

# Chapter Five Result and Discussion

## 5.1 System Analysis

System analysis involves understanding and defining the functional and non-functional requirements of the Room Finder Web App, identifying system components, and creating models to represent the system’s architecture and data flow. This section includes the functional and non-functional requirements, system architecture, data flow diagrams, use case diagrams, and entity-relationship diagrams.

### **5.1.1 Functional Requirements**

Functional requirements define the specific behavior or functions of the Room Finder Web App. These include:

* **User Registration and Authentication:**
  + Users can register with email and password.
  + Users can log in using secure authentication methods.
  + Forgot password functionality with email verification.
* **Room Listings Management:**
  + Landlords can post new room listings with details (e.g., price, location, amenities, photos).
  + Users can search for rooms using filters such as location, price, and amenities.
  + Users can view detailed information about a room.
  + Landlords can edit or delete their room listings.
* **Booking System:**
  + Users can book rooms directly through the platform.
  + Booking confirmation and notification emails are sent to both users and landlords.
  + Users can view their booking history.
* **User Profile Management:**
  + Users can update their profile information, including contact details and preferences.
  + Landlords can manage their listings and view booking requests.
* **Search and Filtering:**
  + Users can search for rooms based on various criteria such as location, price range, and room type.
  + Search results are sorted and filtered to match user preferences.

### **5.1.2 Non Functional Requirements**

Non-functional requirements address the quality attributes of the system:

* **Performance:**
  + The system should handle concurrent users without performance degradation.
  + Average page load time should be under 3 seconds.
* **Security:**
  + Authentication system for user’s security.
  + SSL encryption.
* **Usability:**
  + The user interface should be intuitive and easy to navigate.
  + The application should be accessible on both desktop and mobile devices.
* **Scalability:**
  + The system should be scalable to handle an increasing number of users and data.
* **Reliability:**
  + The system should have an uptime of 99.9%, with automated backups and disaster recovery protocols.
* **Maintainability:**
  + The code-base should be modular and well-documented to facilitate maintenance and updates.

### **5.1.3 Use Case Diagram**

The use case diagram outlines the different functionalities of the Room Finder Web App from the user's perspective. It includes key actions such as posting a room, searching for a room, booking a room, and leaving a review. Actors in the diagram include room seekers, landlords, and the system administrator. This diagram helps in understanding the system's features and the roles of different users in interacting with these features.

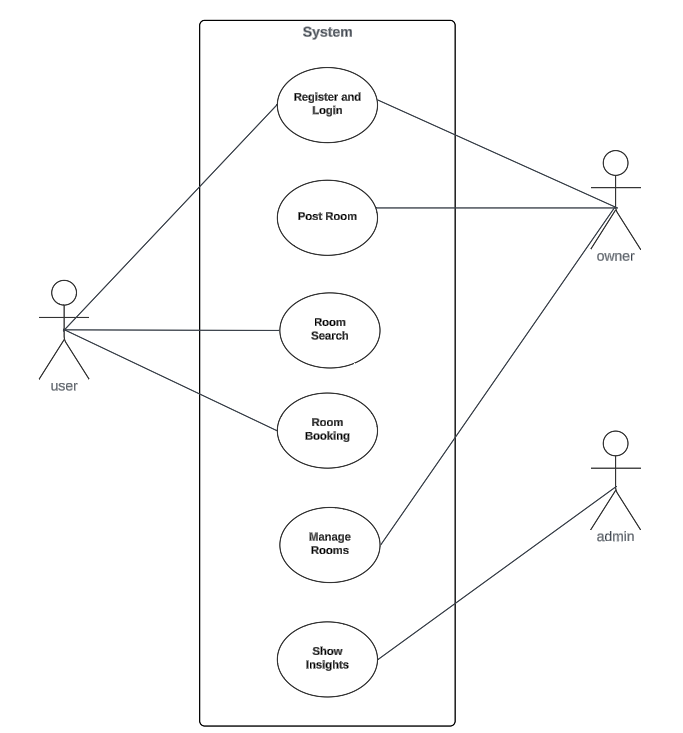


Fig 5.1 Use case diagram

### **5.1.4 Context Diagram**

Context diagram (level 0 DFD) depicts the interaction between users and the system and shows main processes like user registration, room posting, and booking.

The context diagram provides a high-level view of the Room Finder Web App, illustrating its interactions with external entities. It shows how users (both room seekers and landlords) interact with the system to post or search for rooms, while the system processes their requests and manages the data. The diagram highlights the flow of information between users, the system, and external databases or services involved in authentication and data storage.

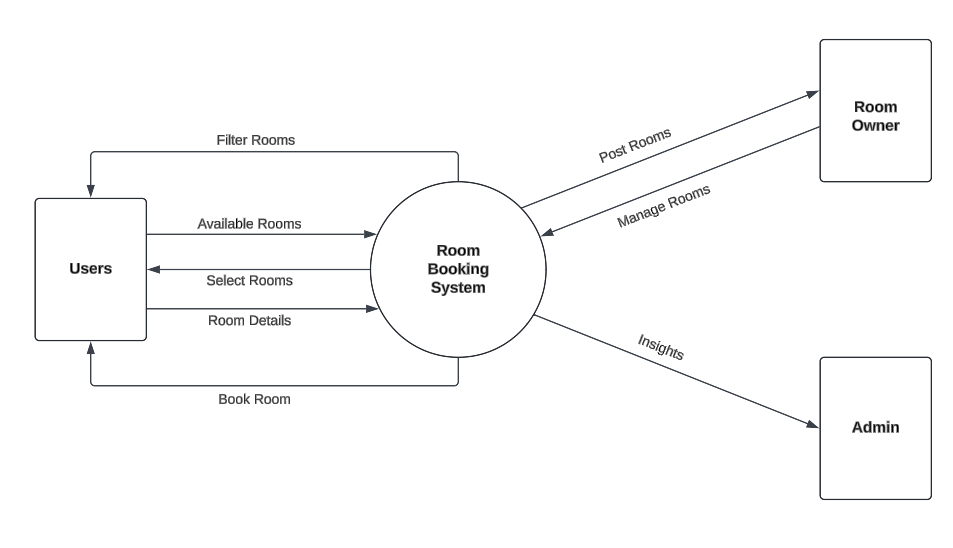


Fig 5.2 Context Diagram

### **5.1.5 Level 1 Data Flow Diagram**

Expands on Level 0 DFD to show detailed processes such as authentication, data retrieval from the database, and room filtering. This Level 1 DFD shows how data is processed within the system, from receiving user inputs (like room postings and booking requests) to updating the database and generating responses. This diagram shows how information moves between processes such as user authentication, room management, booking management providing a clear picture of the system’s internal workings.

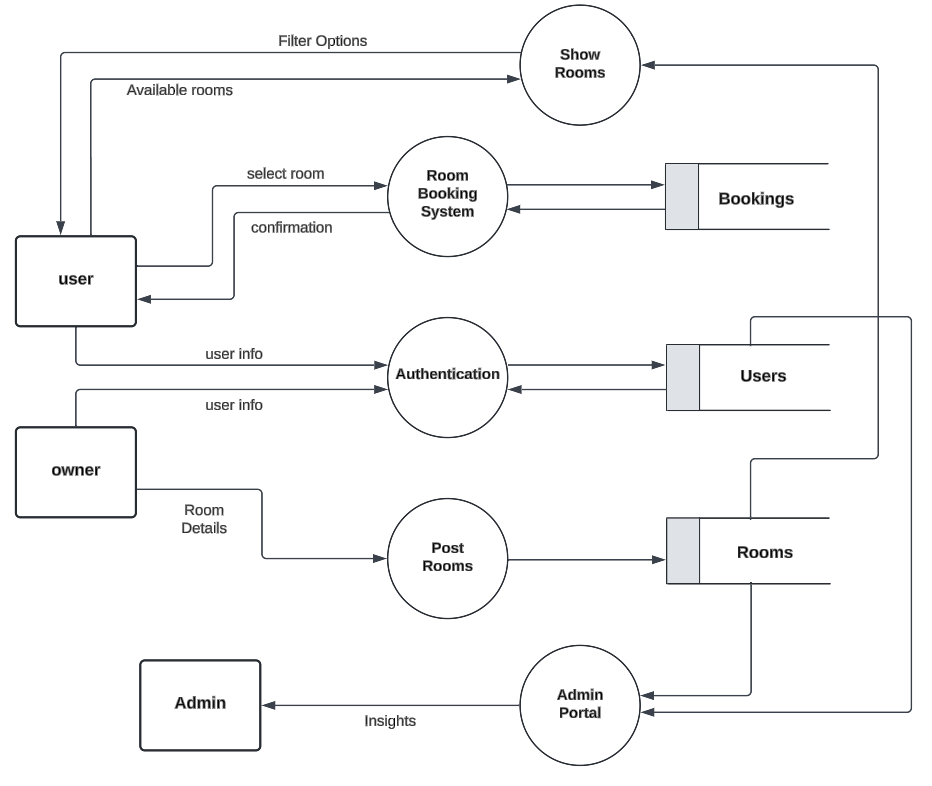


Fig 5.3 Data Flow Diagram Level 1

### **5.1.6 ER Diagram**

The Entity-Relationship Diagram depicts the data model of the Room Finder Web App, showing how entities such as users, rooms, bookings, and reviews are related.

* **User:** Represents users of the platform, including attributes like user ID, name, email, and password.
* **Room:** Represents room listings, including attributes like room ID, location, price, and amenities.
* **Booking:** Represents room bookings, including attributes like booking ID, user ID, room ID, and booking date.

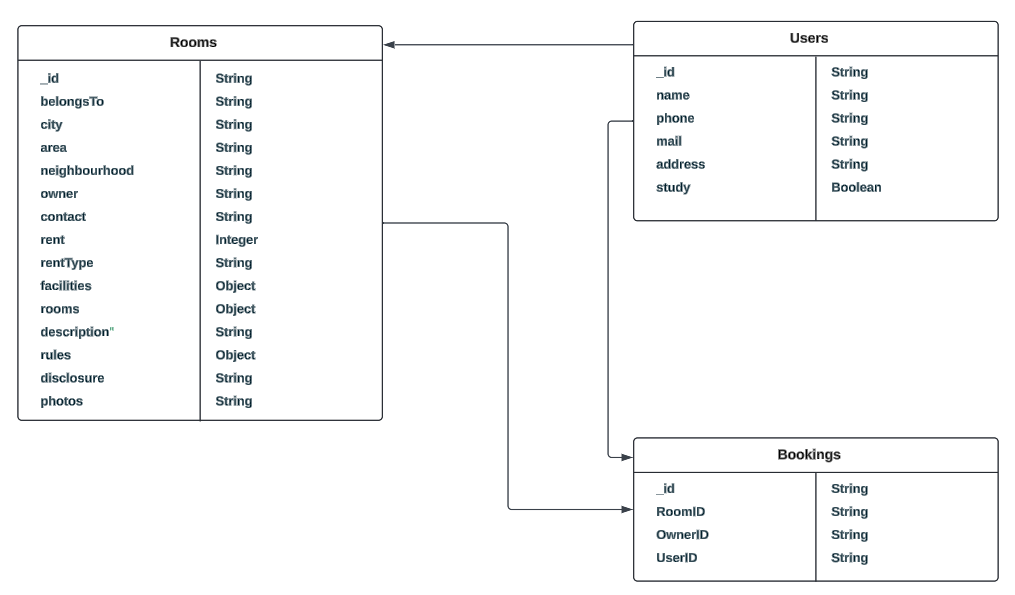


Fig 5.4: ER Diagram

### **5.1.7 Activity Diagram**

The activity diagram maps out the workflow of specific processes within the Room Finder Web App, such as booking a room or posting a listing. It shows the sequence of actions and decision points from the start of an activity to its completion. This diagram helps in understanding the flow of operations.

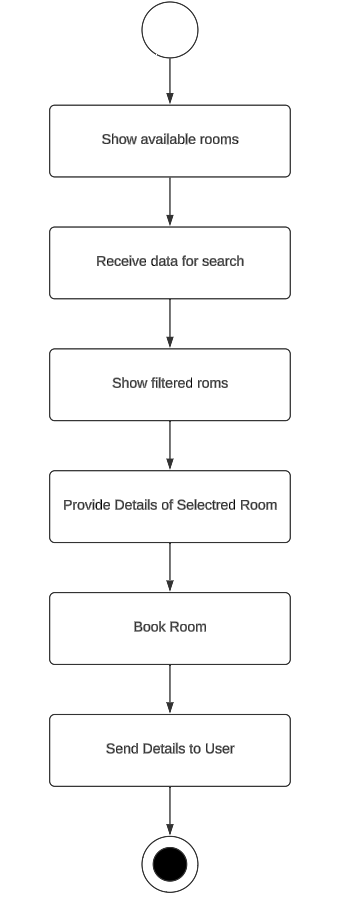


Fig 5.5: Activity Diagram

## 5.2 System Design

System design involves creating a detailed plan for implementing the Room Finder Web App, focusing on both the software architecture and the user experience. The design phase translates the requirements and analysis into a blueprint that guides the development process. Below are the key components of the system design:

### **5.2.1 System Architecture**

The Room Finder Web App is designed using a three-tier architecture:

* **Presentation Layer (Front-End):**
  + Developed using Next.js (React framework) for building the user interface.
  + Handles the user interface, including forms, navigation, and content display.
  + Communicates with the application layer through RESTful APIs.
* **Application Layer (Back-End):**
  + Built using Node.js and Express.js to manage server-side logic.
  + Processes user requests, handles business logic, and interacts with the database.
  + Implements authentication, authorization, and data validation.
* **Data Layer (Database):**
  + Uses MongoDB as the primary database to store user profiles, room listings, bookings, and reviews.
  + Implements database indexing for optimized search queries.

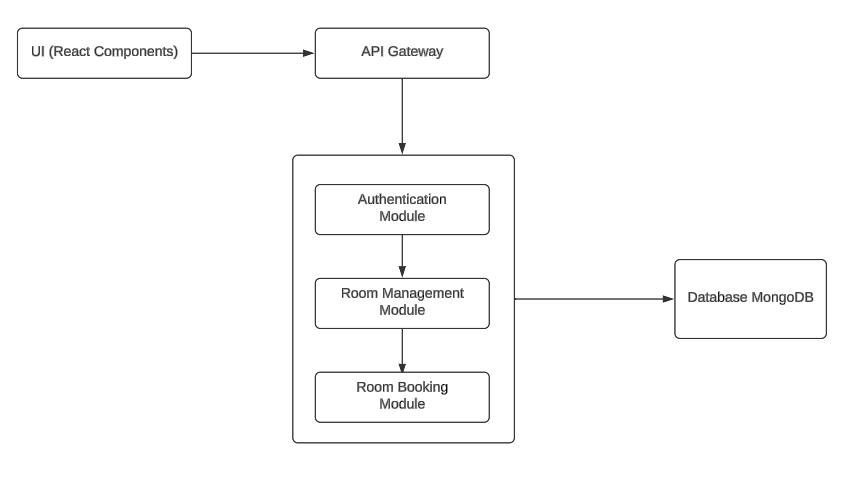


Fig 5.6: System Architecture Design

### **5.2.2** **User Interface (UI) Design**

UI design is centered on creating an intuitive, user-friendly interface that ensures a positive user experience across all devices.

* **Consistency:** Maintain a uniform design language throughout the app, with consistent use of colors, fonts, and button styles.
* **Components:** The contents in the web app are broken down into components to ensure code reuse and efficiency.
* **Responsive Design:** The UI will adapt to various screen sizes and orientations, ensuring a seamless experience on devices like desktops, tablets, and mobile devices.

### **5.2.3 Deployment Design**

Deployment design ensures that the Web App is properly hosted and accessible to users.

* **Cloud Deployment:**
  + The app will be deployed on a cloud platform like Vercel, providing scalability and reliability.
  + MongoDB Atlas will be used for the database, offering automated backups and easy scaling options.
* **Continuous Integration/Continuous Deployment (CI/CD):**
  + A CI/CD pipeline will be set up using tools like GitHub to automate the deployment process, ensuring that new updates are deployed smoothly.
* **Monitoring and Logging:**
  + Vercel Analytics will be used to monitor the app’s performance and log errors, helping to quickly identify and resolve issues.

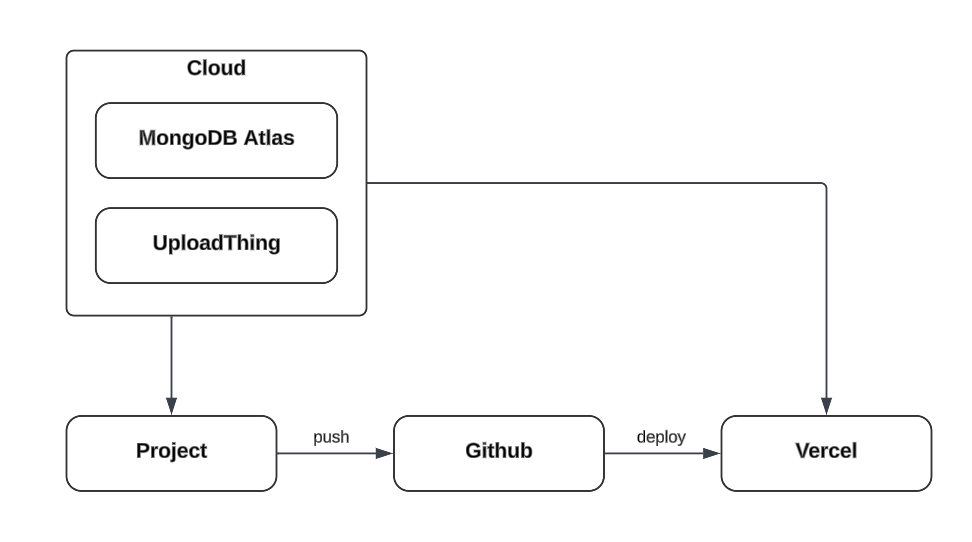


Fig 5.7: Deployment Design

### **5.3.4 Component Diagram**

The component diagram provides significant benefits for the Room Finder Web App by offering a clear, modular view of the system's architecture. It helps in understanding how different parts of the front-end interact with each other and with the back-end, ensuring that each component, such as the user interface, API interactions, and various management modules, is properly isolated and can be developed, tested, and maintained independently. This modularity enhances the scalability and flexibility of the web app, making it easier to add new features or update existing ones without disrupting the entire system. Additionally, it aids in efficient collaboration among development teams by clearly defining component boundaries and responsibilities.

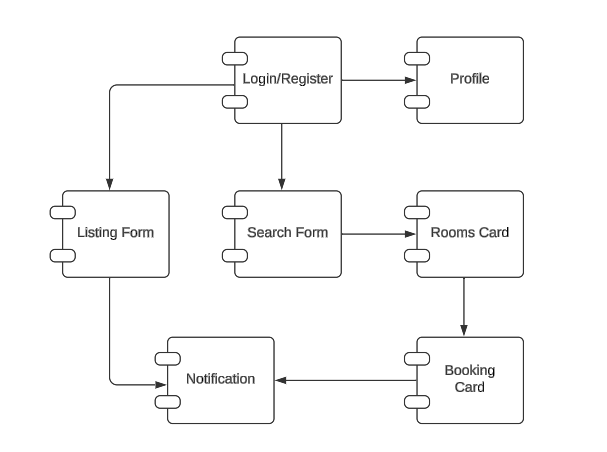


Fig 5.8: Component Diagram

## 5.3 Implementation

The implementation phase is the critical stage where the design specifications are translated into functional code. In this chapter, we will explore the detailed process of implementing the Room Finder Web App, focusing on the technologies, tools, and methodologies used. The implementation is aligned with the structure and components outlined in the design phase and leverages modern web development practices to ensure efficiency, scalability, and maintainability.

### **5.3.1 Technology Stack**

The choice of technology stack plays a pivotal role in the overall performance and user experience of the web app. For "HamroDera," the following technologies were selected:

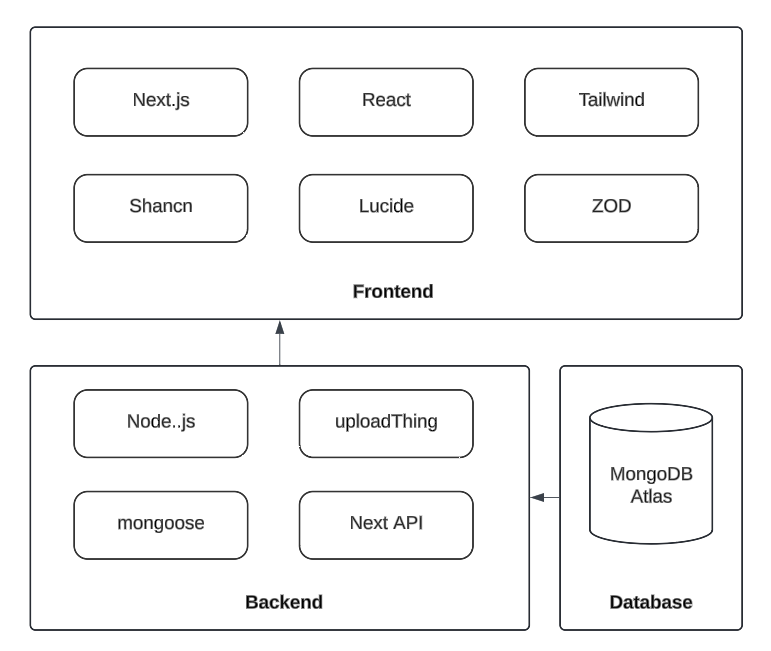


Fig 5.9: Technology Stack

* **Front-End** **Technology Stack**
  + **Next.js (React Framework)**: Provides a server-side rendering capability, ensuring faster load times and better SEO performance. It also offers a flexible file-based routing system, making it easier to manage different pages within the app. (Docs | Next.js, 2024)
  + **React.js**: Used for building reusable UI components, enabling a dynamic and responsive user interface. React’s component-based architecture allows for efficient development and easier maintenance.
  + **Tailwind CSS**: A utility-first CSS framework that simplifies styling by providing a set of predefined classes, allowing for rapid UI design and ensuring a consistent look and feel across the application.
  + **Lucide React**: Lucide React was used to incorporate a wide range of customizable icons into the React components, significantly improving the app's visual appeal and user experience. (Lucide React, 2024)
  + **Shadcn**: It is a component library that provides react components that can be easily integrated to Next.js application. Forms, cards, toast, dialog, accordion, buttons, lists, alert, etc. are implemented in this project with the help of shadcn components. (shadcn, 2024)
  + **ZOD**: By defining schemas with Zod, the project can enforce strict validation rules on user inputs and API responses, preventing invalid or malicious data from being processed. This is particularly useful in form handling and API requests, where Zod automatically checks that incoming data meets the required structure and types. (Zod | Documentation, 2024)
* **Back-End Technology Stack**
  + **Node.js**: A JavaScript runtime that allows for the development of server-side applications. It’s known for its non-blocking, event-driven architecture, which is ideal for building scalable web applications.
  + **Mongoose**: Mongoose is used for making connection with the database and integrating the backend operations with the database. It provides a straight-forward, schema-based solution to model your application data. It includes built-in type casting, query building, validation, business logic hooks and more, out of the box.
  + **UploadThing**: This service provided a seamless and secure way for users to upload room photos directly from the front-end, with minimal configuration and without the need for complex back-end file management. (What is UploadThing?, 2024)
* **Database**:
  + **MongoDB (MongoDB Atlas)**: A NoSQL database that stores data in a flexible, JSON-like format. MongoDB’s schema-less design was used in this project, allowing for easy updates to the data model as the project evolved. MongoDB Atlas, the cloud version, was used to ensure high availability, security, and scalability. (What is MongoDB Atlas?, 2024)
* **Authentication**:
  + **OTP based:** Whenever the user tries to login, an OTP is sent to the user to log in the web app. A unique hashed key is stored that helps to identify the user during the api calls and processes.
* **Hosting and Deployment**:
  + **Vercel**: The web app is deployed on Vercel, which is particularly optimized for Next.js applications. It provides continuous deployment and automatic scaling, ensuring that the app can handle traffic efficiently.

## 5.3.2 Frontend Implementation

The front-end of the Room Finder Web App was implemented using Next.js and React.js. The development process began by setting up the Next.js project, configuring the necessary files, and integrating Tailwind CSS for styling.

* **Page Structure**: The web app's pages are structured according to Next.js's file-based routing system. Key pages include the home page, search results, room details, user profile, and booking confirmation. Each page corresponds to a component in the React architecture, ensuring that the UI is built in a modular and reusable manner.
* **Component Development**: React components were developed for each section of the UI, including the header, footer, room listing, and search bar. Components were designed to be reusable, meaning that the same component could be used in different parts of the app with varying data inputs. For instance, the RoomCard component is used both on the search results page and the user's profile page, but with different data sources.
* **State Management**: React's useState and useEffect hooks were extensively used to manage state within components, ensuring that the UI reacts to changes in data in real-time. For more complex state management across different components, the Context API was used, allowing for a global state that could be accessed by any component in the tree.
* **Styling**: Tailwind CSS was integrated to handle the styling of the web app. The utility-first approach of Tailwind allowed for rapid development, with classes being applied directly to elements to control layout, spacing, typography, and more. This also made it easier to maintain a consistent design across the app.
* **Icons:** Lucide React was used to incorporate a wide range of customizable icons into the React components, significantly improving the app's visual appeal and user experience. By using Lucide React, the project could easily include scalable vector icons that are consistent in style, lightweight, and responsive across different devices. This helped in creating intuitive and aesthetically pleasing UI elements, such as buttons, navigation menus, and status indicators, which are crucial for user interaction.

### **5.3.3 Backend Implementation**

The back-end of the Room Finder Web App was built using Node.js, mongoose, with MongoDB as the database.

* **API Development**: The back-end serves as a RESTful API, handling requests from the front-end and interacting with the MongoDB database. Next.js API was used to set up routes for different endpoints, including user authentication, room management, booking management, and review management. The API follows REST principles, ensuring that it is stateless and scalable.
* **Database Design**: The MongoDB database was designed to store various collections such as Users, Rooms, and Bookings. Each collection is structured to store data efficiently, with relationships between collections managed through references (e.g., a booking references a room and a user). This flexible schema design allows the app to adapt to changing requirements without the need for significant database restructuring.
* **OTP based Authentication**: Whenever the user tries to login, an OTP is sent to the user to log in the web app. A unique hashed key is stored that helps to identify the user during the api calls and processes.
* **Image Handling**: UploadThing facilitated the handling of file uploads, particularly images of rooms, which are a central feature of the web app. By integrating UploadThing, the project could efficiently manage the process of uploading, storing, and retrieving images from the cloud. This service provided a seamless and secure way for users to upload room photos directly, with minimal configuration and without the need for complex back-end file management. This greatly simplified the development process and ensured that users could easily attach images to their room listings, enhancing the overall functionality and user experience of the app. (What is UploadThing?, 2024)
* **Error Handling and Validation**: The back-end was equipped with robust error handling and input validation. Try-catch was used to catch errors and send appropriate responses to the client. Additionally, input data from the client was validated using libraries like ZOD, ensuring that only valid data is processed and stored in the database.

### **5.3.4 Challenges in Implementation**

During the implementation phase, several challenges were encountered:

* **Handling Asynchronous Operations**: Managing asynchronous operations, especially in the context of API calls and database interactions, posed a challenge. This was addressed by using async/await syntax in JavaScript, which allowed for writing cleaner and more manageable asynchronous code.
* **State Management**: Managing state across multiple components and ensuring that updates were reflected in real-time required careful planning. The Context API and custom hooks in React were leveraged to maintain global state effectively.
* **Database Schema Evolution**: As the project evolved, the database schema needed to be updated to accommodate new features. This was managed using MongoDB’s flexible schema design, which allows for changes without disrupting existing data.

The implementation phase successfully converted the designs and plans into a functional Room Finder Web App. The app was developed using best practices in web development, with a focus on modularity, scalability, and security. The combination of modern web technologies and best practices resulted in a robust and scalable application, ready to meet the needs of users in searching for and booking rooms online. The challenges faced during implementation were successfully overcome through innovative solutions, setting a solid foundation for future enhancements and scalability.

## 5.4 Testing

Testing is a critical phase in the software development lifecycle that ensures the functionality, reliability, and performance of the Room Finder Web App. In this sub chapter, we will delve into the various testing strategies employed in the project, aligned with the best practices to deliver a robust and error-free application. The testing process was meticulous and iterative, covering all aspects of the web app, from individual components to full system integration, ensuring that the final product meets the specified requirements and provides a seamless user experience. (Software Testing, 2024)

### **5.4.1 Overview of Testing Strategy**

The testing strategy for the Room Finder Web App was designed to encompass both automated and manual testing approaches. Automated testing was heavily utilized to ensure the codebase's stability through continuous integration, while manual testing focused on user experience and edge cases that might not be captured by automated tests. The following key types of testing were implemented:

* **Unit Testing**: To validate the smallest parts of the application, such as individual functions and components, ensuring that they work as expected in isolation.
* **Integration Testing**: To verify that different modules and components work together seamlessly, particularly the interaction between the front-end and back-end systems.
* **Performance Testing**: To assess the application's responsiveness, stability, and speed under various conditions, ensuring it can handle the expected load.
* **User Acceptance Testing (UAT)**: To validate the application against user requirements, ensuring that it meets the business needs and provides a satisfactory user experience.
* **Regression Testing:** After fixing bugs, regression tests were conducted to ensure that the fixes did not introduce new issues or break existing functionality. Automated test suites were particularly valuable in this phase, allowing for quick re-testing of the entire application.

### **5.4.2 Unit Testing**

In this project, unit testing was a fundamental approach to ensuring the reliability and correctness of individual components and functions within the Room Finder Web App. We manually created test cases for each function and component, focusing on validating their expected behavior in isolation. This involved designing specific inputs and predicting their corresponding outputs, then verifying that the actual output matched the expected result. By breaking down the application into its smallest parts, we were able to thoroughly examine each piece of logic and user interface element. This method allowed us to identify and address errors early in the development process, ensuring that each component functioned correctly before it was integrated into larger parts of the system. Through this meticulous process, we maintained high code quality and minimized the risk of bugs in the final product.

### **5.4.3 Integration Testing**

Integration testing focused on ensuring that different modules and services within the application interacted correctly. Given the modular nature of the Room Finder Web App, it was crucial to verify that the front-end components properly communicated with the back-end API and that data was correctly handled across the system.

* **Testing API Endpoints**: The integration between the front-end and the Next API back-end was tested by sending HTTP requests using tools like **ThunderClient**. These tests ensured that API endpoints returned the correct data and status codes when invoked by the front-end. For example, the room search functionality was tested to ensure that it correctly retrieved and displayed rooms based on the user's search criteria.
* **Database Interaction**: Tests were conducted to ensure that the application could correctly interact with the MongoDB database. This included verifying that data was correctly inserted, updated, retrieved, and deleted in the various collections (Users, Rooms, Bookings). MongoDB's flexible schema was a particular focus, ensuring that any changes in the data structure did not break existing functionality.
* **Component Interactions**: React component integration was tested by simulating user actions that would typically trigger interactions between multiple components. For example, testing the booking flow involved ensuring that selecting a room triggered the appropriate API call, updated the booking state, and displayed a confirmation message.

### **5.4.4 Performance Testing**

Performance testing was conducted to assess how well the Room Finder Web App could handle different levels of user load and data volume. The goal was to ensure that the application remained responsive and stable under normal and peak conditions.

* **Scalability Testing**: The application was tested for its ability to scale, particularly in the context of increasing user traffic. *Vercel's* auto-scaling features were leveraged to ensure that the app could handle sudden spikes in traffic without downtime or degraded performance. The back-end's ability to scale was also tested by simulating high numbers of concurrent database operations, ensuring that *MongoDB Atlas* could handle the load efficiently.
* **Page Load Speed**: The performance of the front-end was assessed using tools like *Lighthouse*. The focus was on optimizing the page load times, particularly on the home page and search results page, where multiple images and data points were loaded simultaneously. Optimizations such as lazy loading of images and reducing the size of JavaScript bundles were implemented to improve performance.

The web app was tested with the help of Lighthouse extension to study and test the Performance, Accessibility, Best Practices, and SEO. The following results were calculated with the help of metrics like first contentful paint, total blocking time, speed index, and largest contentful paint. (Overview | Lighthouse, 2016)

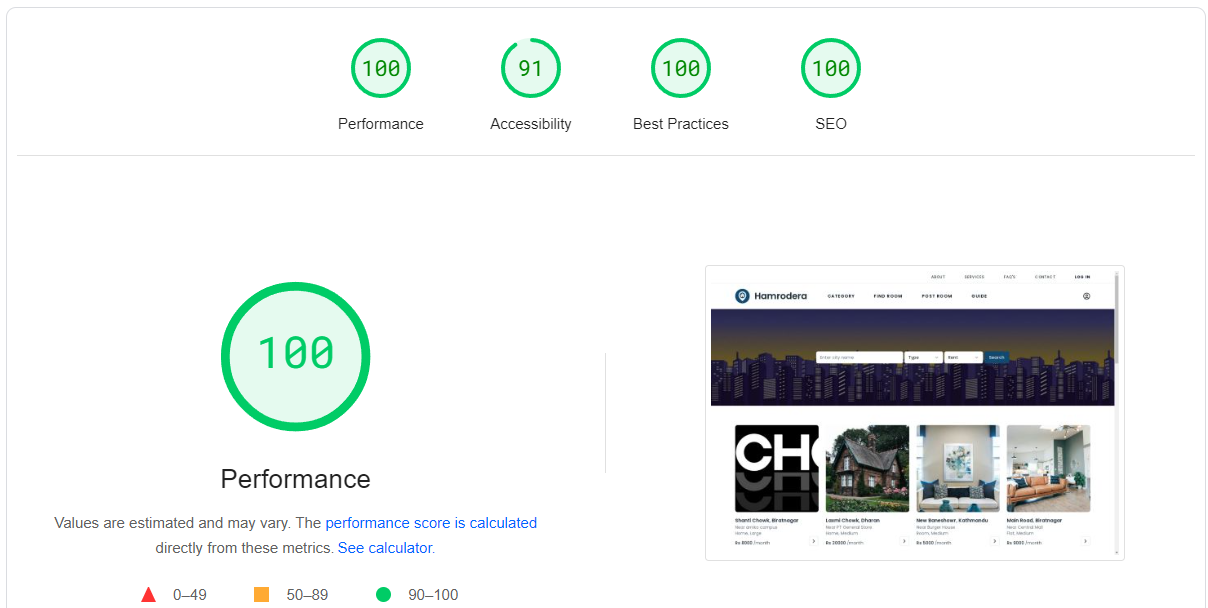


Fig 5.10: Lighthouse Performance Report

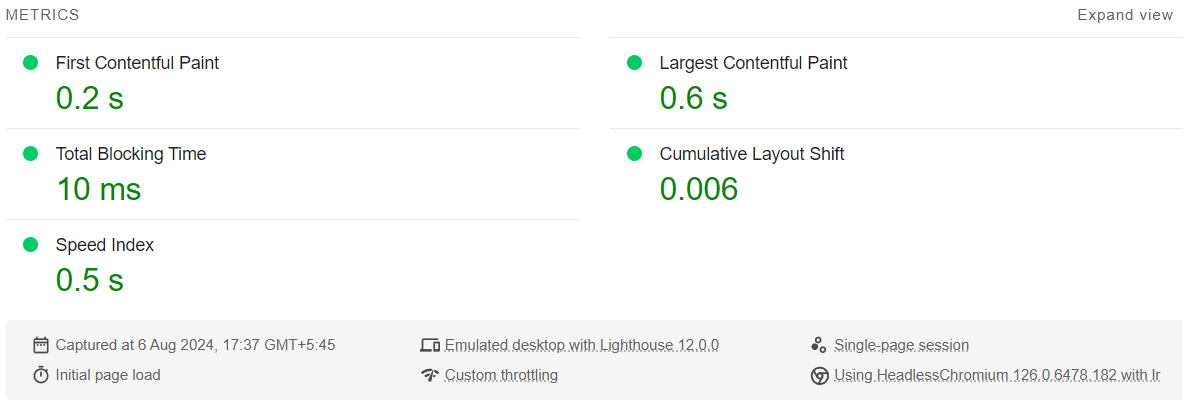


Fig 5.11: LightHoue Performance Metrices

### **5.4.5 User Acceptance Testing**

User Acceptance Testing (UAT) was the final stage of testing, focusing on ensuring that the Room Finder Web App met the business requirements and provided a satisfactory user experience.

* **Test Scenarios**: Real-world scenarios were developed and tested by end-users, who interacted with the application to ensure it met their needs. These scenarios included typical tasks such as posting a room, searching for a room by location and price, booking a room, and leaving a review. Feedback from users during UAT was instrumental in identifying areas of improvement, particularly in terms of usability and feature completeness.
* **Feedback Loop**: UAT involved continuous feedback from users, which was used to make iterative improvements to the application. For example, feedback on the ease of use of the booking process led to minor adjustments in the UI, making the steps clearer and more intuitive.

### **5.4.6 Bug Tracking and Regression Testing**

Throughout the testing phase, a systematic approach to bug tracking and resolution was maintained and every bug identified during testing was logged with details such as the severity, steps to reproduce, and the environment in which it occurred.

* **Bug Triage**: Bugs were triaged based on their impact on the application. Critical bugs, such as those affecting core functionality like login or booking, were prioritized and fixed immediately. Lower priority bugs, such as minor UI inconsistencies, were addressed after the critical issues were resolved.
* **Regression Testing**: After fixing bugs, regression tests were conducted to ensure that the fixes did not introduce new issues or break existing functionality. Automated test suites were particularly valuable in this phase, allowing for quick re-testing of the entire application.

## 5.5 Deployment

The deployment strategy for the Room Finder Web App was centered around ensuring continuous integration and delivery (CI/CD). The goal was to automate as much of the deployment process as possible, enabling quick and reliable updates to the live application without disrupting the user experience.

### **5.5.1 Deployment Strategy**

The strategy involved deploying the front-end and back-end components separately, leveraging cloud services that provide robust infrastructure and seamless scalability.

* **Project Deployment**: The project, built using Next.js, was deployed on *Vercel*, a platform optimized for modern front-end frameworks. Vercel was chosen for its ease of use, automatic scaling, and integration with GitHub, which allowed for continuous deployment. Every time code was pushed to the main branch on GitHub, Vercel automatically built and deployed the latest version of the app, ensuring that the live site was always up-to-date. (Getting started with Vercel, 2024)
* **Database Deployment**: The database, was hosted on *MongoDB**Atlas*, a cloud-based database service that offers high availability, security, and scalability. MongoDB Atlas was chosen for its ability to automatically manage database clusters, handle backups, and provide real-time monitoring. The database is geographically distributed to minimize latency and ensure that users have a fast and reliable experience, regardless of their location.
* **Image Deployment**: *UploadThing* facilitated the handling of file uploads, particularly images of rooms, which are a central feature of the web app. By integrating UploadThing, the project could efficiently manage the process of uploading, storing, and retrieving images from the cloud. This service provided a seamless and secure way for users to upload room photos directly from the front-end, with minimal configuration and without the need for complex back-end file management. This greatly simplified the development process and ensured that users could easily attach images to their room listings, enhancing the overall functionality and user experience of the app.

### **5.5.2 Deployment Process**

The deployment process was divided into several stages, each designed to ensure that the application was properly configured, tested, and monitored before going live. The following steps outline the deployment process:

1. **Preparation and Configuration**:
   1. The first step involved configuring environment variables for both the front-end and back-end. This included setting API keys, database connection strings, and other sensitive information that should not be hard-coded in the source code.
   2. The Next.js front-end and API was configured for production builds, optimizing the app for performance. This involved minifying JavaScript, optimizing images, and enabling server-side rendering for faster load times.
2. **Building and Testing**:
   1. The front-end and back-end were built for production using their respective build tools. The Next.js app was built using the next build command, while the Node.js back-end was prepared with the appropriate build scripts.
   2. Before deployment, both the front-end and back-end were tested in a staging environment. This staging environment mirrored the production environment as closely as possible, allowing the team to catch any issues that might arise from the build or deployment process.
3. **Deploying to Production**:
   1. The front-end was deployed to Vercel, where it automatically handled the DNS configuration and SSL certificates, ensuring the site was accessible via HTTPS.
   2. MongoDB Atlas handled the database deployment, ensuring that the database was secure, with proper indexing and backups in place.
4. **Post-Deployment Monitoring**:
   1. After deployment, the application was closely monitored using tools provided by Vercel, Railway, and MongoDB Atlas. These tools provided real-time analytics on performance, user interactions, and potential errors.
   2. Alerts were set up to notify the development team of any critical issues, such as downtime or high error rates, ensuring that any problems could be addressed immediately.

# Chapter SIX Conclusion and Recommendation

## 6.1 Conclusion

The Room Finder Web App, HamroDera, was developed to address the growing need for a streamlined and efficient platform that connects landlords and tenants in Nepal. The project aimed to simplify the process of finding and booking rental properties, providing a user-friendly interface and powerful backend to handle the complexities of rental transactions. Throughout the development process, we leveraged modern web technologies, including Next.js for the front-end, Node.js and Express.js for the back-end, and MongoDB Atlas for database management. The deployment strategy utilized Vercel and Railway, ensuring scalability and reliability in the production environment.

This project has successfully achieved its objectives, providing a fully functional web application that allows users to post, search, and book rooms with ease. The implementation of features like real-time room availability, secure user authentication, and responsive design ensures a smooth user experience across different devices. The project's success is also attributed to the rigorous testing and deployment processes, which ensured that the app was both robust and scalable.

In conclusion, HamroDera stands as a significant contribution to the housing rental market in Nepal, offering a digital solution that caters to the needs of both landlords and tenants. The project has demonstrated the potential of modern web technologies in solving real-world problems and has set a solid foundation for future enhancements and expansions.

## 6.2 Limitations

Despite the successful implementation of the web app, several limitations were encountered during development, which could affect the system's overall performance and usability:

* **Limited Geographical Scope**: The current version of this web app is primarily focused on specific urban regions within Nepal. This geographical limitation restricts its usability to users outside these regions. Expanding the platform to cover more areas would require additional resources, including data collection, localization, and support infrastructure.
* **Scalability Concerns**: While the deployment strategy utilizes cloud services like Vercel, the scalability of the system may be challenged if there is a sudden surge in user traffic. Although these platforms offer auto-scaling features, the system's performance might be affected during peak usage times, leading to potential slowdowns or downtimes.
* **Limited Payment Integration**: The app currently lacks integrated payment gateway support, meaning users must handle payments outside the platform. This limitation could lead to inconvenience and missed opportunities for streamlining the rental transaction process. Adding secure payment options in future iterations would address this gap.
* **Maintenance and Support**: The platform requires ongoing maintenance and support to ensure it remains secure, up-to-date, and free from bugs. However, limited resources and the need for continuous updates can challenge the development team's ability to keep the system running smoothly over time.
* **Data Privacy and Security**: Although efforts have been made to secure the platform, managing sensitive user data (such as personal details, and location and rooms information) poses inherent risks. Ensuring compliance with data protection regulations and maintaining user trust remains a continuous challenge.

## 6.3 Recommendations

While the Room Booking web app has met its initial goals, there are several areas where further improvements and expansions could enhance its functionality and user experience:

1. **Mobile Application Development**: To reach a wider audience and provide a more accessible platform, it is recommended to develop a mobile application version of web app. This would cater to users who prefer mobile apps over web-based platforms and could leverage native device features for an enhanced user experience.
2. **Advanced Filtering and Search Capabilities**: Introducing more advanced filtering options, such as price range, room size, and additional amenities, would allow users to find rooms that more precisely match their preferences. Implementing AI-based recommendations could further improve the search experience by suggesting rooms based on user behavior and preferences.
3. **Integration with Payment Gateways**: Adding secure payment gateway integration like *esewa* and *khalti* would streamline the booking process by allowing users to make payments directly through the app. This feature could also include automated rent payment reminders and invoicing, making the app more useful for both landlords and tenants.
4. **Expansion to Other Regions**: While the initial deployment of HamroDera focuses on specific urban regions in Nepal, expanding the platform to cover other areas or even neighboring countries could significantly increase its user base. This expansion would require localization features, including language support and region-specific content.
5. **User Feedback and Continuous Improvement**: Regularly collecting user feedback and analyzing usage data will be crucial for the continuous improvement of the app. This feedback loop can guide future updates, ensuring that the platform evolves to meet the changing needs of its users.

By following these recommendations, this room booking web appcan continue to grow and adapt, becoming an even more valuable tool for those seeking rental properties in Nepal.

# Chapter Seven References

Bhattarai. (2020). Rental Market Dynamics in Nepal. The need for Centralized Platforms. *Journal of Nepal Housing and Development*, 34-50.

*Docs | Next.js*. (2024, August 06). Retrieved from nextjs: https://nextjs.org/docs

*Getting started with Vercel*. (2024, August 08). Retrieved from Vercel: https://vercel.com/docs/getting-started-with-vercel

*Lucide React*. (2024, August 06). Retrieved from lucide: https://lucide.dev/guide/packages/lucide-react

Nepali, S. (2022). The Role of Technology in Modernizing the Rental Market in Nepal. . *Information Technology in Nepal*, 21-37.

*Overview | Lighthouse*. (2016, October 27). Retrieved from developer.chrome: https://developer.chrome.com/docs/lighthouse/overview

*shadcn*. (2024, August 06). Retrieved from Introduction | shadcn/ui : https://ui.shadcn.com/docs

Shrestha. (2021). Challenges in Finding Affordable Housing in Kathmandu: A Study on the Impact of Urbanization. . *Nepal Journal of Urban Studies*, 45-59.

*Software Testing*. (2024, August 08). Retrieved from https://www.geeksforgeeks.org/software-testing-basics/: https://www.geeksforgeeks.org/software-testing-basics/

*What is MongoDB Atlas?* (2024, August 07). Retrieved from mongodb: https://www.mongodb.com/docs/atlas/

*What is UploadThing?* (2024, August 07). Retrieved from UploadThing: https://docs.uploadthing.com/

*Zod | Documentation*. (2024, August 06). Retrieved from Zod: https://zod.dev/