MINOR PROJECT REPORT

Intelligent Ranking and Chat System for Property Rental

(ES-451)

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

Bachelor of Technology
Computer Science & Engineering

Project Guide:
Ms. Apurva Jain
Assistant Professor

Submitted by:
Akshat Jain (00115607223)
Khushal (00715607223)
Pankaj Singh (04315602722)
Raghav Dwivedi (01015602722)

Department of Computer Science & Engineering

Dr. Akhilesh Das Gupta Institute of Professional Studies

(Formerly ADGITM)

FC-26, SHASTRI PARK, NEW DELHI

Affiliated to

GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY

Sector - 16C Dwarka, Delhi - 110075, India

2022-26

DECLARATION

I/We, the undersigned student(s) of B.Tech. (CSE), hereby declare that the minor/major project titled "**Intelligent Ranking and Chat System for Property Rental**" submitted to **Apurva Jain (Associate Professor)**, the Department of Computer Science & Engineering, at **Dr. Akhilesh Das Gupta Institute of professional Studies**, Delhi, affiliated with Guru Gobind Singh Indraprastha University, Dwarka New Delhi, in partial fulfillment of the requirements for the degree of Bachelor of Technology in Computer Science and Engineering, has not previously been used as the basis for the award of any degree, diploma, or similar title or recognition. The list of members involved in the project is as follows:

S. No.	Student Name	Enrollment Number	Student Signature
1	Akshat Jain	00115607223	
2	Khushal	00715607223	
3	Pankaj Jain	04315602722	
4	Raghav Dwivedi	01015602722	

Place: Delhi

Date:

ACKNOWLEGEMENT

The note starts with thanks to Almighty who actually created this piece of work and helped us when things were not easy for us.

I am very grateful and indebted to Apurva Jain (Associate Professor) who immensely helped and rendered her/his valuable advice, precious time, knowledge and relevant information regarding the collection of material. She has been a major source of inspiration throughout the project as he not only guided me throughout the Minor Project Report but also encouraged me to solve problems that arose during this report.

Her guidance and suggestions about this Minor Project report have really enlightened me. It has been a great help to support to have her/him around.

And finally, I would like to mention appreciation to our parents and friends who have been instrumental throughout this period by providing unrelenting encouragement.

S. No.	Student Name	Enrollment Number	Student Signature
1	Akshat Jain	00115607223	
2	Khushal	00715607223	
3	Pankaj Jain	04315602722	
4	Raghav Dwivedi	01015602722	

Place: Delhi

Date:

CERTIFICATE

This is to certify that this project entitled "**Intelligent Ranking and Chat System for Property Rental**" submitted in partial fulfillment of the degree of B.Tech. (CSE) as a part of the curriculum bearing Course Code ES-451 submitted to the Department of Computer Science & Engineering, at Dr. Akhilesh Das Gupta Institute of professional Studies, Delhi, affiliated to Guru Gobind Singh Indraprastha University,
New Delhi-110078 to me by

S. No.	Student Name	Enrollment Number	Student Signature
1	Akshat Jain	00115607223	
2	Khushal	00715607223	
3	Pankaj Jain	04315602722	
4	Raghav Dwivedi	01015602722	

is an is an authentic work carried out by them under my guidance. The matter embodied in this project work has not been submitted earlier for award of any degree to the best of my knowledge and belief.

Signature of the Supervisor

Apurva Jain

Countersigned by:

Director/Principal/HoD/Project In charge

Place: Delhi

Date:

Abstract

The rapid growth of rental demand in metropolitan cities has highlighted the need for smarter, more reliable, and user-centric digital platforms. Traditional property rental systems rely heavily on static filters such as location, rent, or amenities, which often fail to capture deeper user preferences, behaviour patterns, and real-time requirements. To overcome these limitations, this project presents an **Intelligent Ranking and Chat System for Property Rental**, designed to deliver personalized recommendations, verified listings, and seamless communication in a unified platform.

The system integrates a modern, scalable architecture comprising a **React.js frontend**, a **Spring Boot backend**, and a **MySQL–Redis storage layer**. A dedicated **AI module built using Python and Flask** enhances the platform by applying ranking algorithms and sentiment analysis through the VADER model. This enables dynamic reordering of property listings based on user interaction history, feedback, and contextual cues. Real-time chat functionality provides instant communication between renters and property owners, while verification mechanisms help reduce misleading or outdated listings, improving trust and transparency.

Comprehensive testing—including unit, integration, functional, and performance evaluation—ensured system reliability across all modules. The platform successfully delivers faster search results, improved relevance, and a more engaging user experience compared to conventional rental systems. Although current limitations include dataset size and potential performance drops under heavy loads, the system establishes a robust foundation for future upgrades such as advanced machine learning models, mobile applications, automated verification, and integrated booking features.

Overall, the project demonstrates an effective blend of AI-driven intelligence and scalable software engineering to create a smarter, more personalized rental search experience.

Table of Contents

Introduction

Finding rental accommodation in major metropolitan cities often becomes a stressful and unorganized task for most users. Even though there are several real-estate and rental platforms available, they typically depend on basic filters such as rent range, location, amenities, or room type. These rigid filters fail to understand deeper preferences—like a user's lifestyle, priorities, interaction history, or evolving requirements. As a result, users regularly come across listings that are irrelevant, outdated, or mismatched with their expectations. This not only wastes time but also reduces trust in digital rental platforms.

To address these issues, this project introduces an advanced web application titled "Intelligent Ranking and Chat System for Property Rental." The main goal of the system is to make the house-hunting process more seamless, user-friendly, and personalized. Instead of relying solely on static search parameters, the platform analyzes several dynamic signals including a user's browsing patterns, past selections, real-time conversational queries, feedback behavior, and detailed listing characteristics. By processing these factors collectively, the system generates a more meaningful and adaptive ranking of available rooms.

The architecture of the application relies on a modern and scalable technology stack. The frontend is developed using React.js to ensure an interactive, responsive, and smooth user experience. The backend is powered by Spring Boot (Java), which manages APIs, authentication, business logic, and communication with the database. For persistent storage, MySQL is used to organize user data, property details, and chat histories. To improve speed and reduce load on the database, Redis functions as a caching layer for frequently accessed information.

A specialized AI module, built with Python (Flask), plays a key role in enhancing recommendation quality. It uses the VADER sentiment analysis model to interpret user feedback, gauge satisfaction levels, and refine ranking outputs. Additional components—such as a real-time chat interface for owners and tenants, listing verification mechanisms to reduce fake entries, and a continuous feedback loop—help create a more transparent and reliable ecosystem.

Currently, the system runs as a web-based solution and operates with certain practical limitations. The ranking model is trained on a relatively small dataset, and heavy traffic conditions may introduce performance delays or increased latency. However, despite these constraints, the platform establishes a strong groundwork for building a smart, trustworthy, and efficient rental search tool that can evolve with user needs and the fast-changing urban housing environment.

System Analysis

2.1 General

The demand for rental rooms in metropolitan cities has grown rapidly, bringing with it the need for faster and more reliable search systems. Traditional online rental platforms often present large volumes of unorganized data, leaving users to shift through many irrelevant listings. Most systems fail to understand user intent or adjust results based on preferences, feedback, or conversation history.

To overcome these drawbacks, a deeper analysis of the existing environment, user requirements, and operational challenges is required. This chapter outlines the investigation carried out to understand the current problems, evaluate possible solutions, and determine the feasibility of the proposed Intelligent Ranking and Chat System for Property Rental.

2.2 Preliminary Investigation

The preliminary investigation focused on identifying the core issues faced by users and property owners while interacting with rental platforms. Through informal discussions, observation, and review of existing systems, the following major problems were identified:

• **Search results are often generic and lack personalization:** Most platforms rely on basic filters like price or location, which do not account for individual user preferences or past behavior. This results in generic search results that frequently fail to match what users are actually looking for.

• **Many listings are outdated, misleading, or unverified, leading to trust issues:** Users often encounter properties with old photos, incorrect details, or incomplete information. The lack of verification mechanisms allows misleading listings to stay online, reducing user confidence and wasting significant time.

• **Communication between owners and seekers is slow, fragmented, and not tracked properly:** Interactions often take place through external apps or irregular phone calls, resulting in missed messages, delays, and no central record of conversations. This disrupts the inquiry process and increases confusion.

• **Users spend considerable time applying filters repeatedly and comparing options manually:** With limited personalization, users must constantly refine filters and individually compare listings. This makes the search process lengthy, repetitive, and mentally exhausting.

• **Most platforms do not incorporate AI-based ranking or conversational support, making the experience mechanical:** The absence of intelligent recommendation systems means platforms cannot prioritize listings based on user behaviour or sentiment. Additionally, lack of chat-based assistance makes the process less interactive and more difficult for first-time users.

This initial study confirmed the need for an intelligent, user-friendly system capable of delivering personalized results, faster communication, and higher data reliability.

2.3 Feasibility Study

A feasibility study was conducted to evaluate whether the proposed system could be developed with the available resources, skills, and constraints. The study covers technical, economic, and operational feasibility.

2.3.1 Technical Feasibility

From a technical perspective, the proposed solution is practical and achievable with the selected tools and technologies. The system uses:

• **Frontend:** React.js for dynamic UI and smooth user interaction. React.js provides a component-based architecture that makes the interface fast, interactive, and easy to maintain. Its virtual DOM ensures efficient rendering, and its integration with APIs enables seamless communication with the backend. The technology is mature and widely supported, making it highly feasible for the project's requirements.

• **Backend:** Spring Boot (Java) to handle API logic, security, and core operations. Spring Boot simplifies backend development through built-in configurations, dependency management, and REST API support. It also offers strong security features and excellent performance, which makes it capable of managing user authentication, listing operations, chat services, and data processing without stability issues.

• **Database:** MySQL for structured data storage and Redis for caching to improve speed. MySQL is suitable for storing relational data like user profiles, property details, chat records, and feedback. It ensures reliability and consistency through ACID properties. Redis acts as an in-memory cache to speed up frequent queries, reducing load on the database and improving response time. Both technologies are mature and easy to integrate with Spring Boot.

• **AI Layer:** Python and Flask for handling ranking logic and sentiment analysis using VADER. Python provides flexibility for building AI-driven features. Flask is lightweight and efficient for exposing AI functionalities as API endpoints. The VADER sentiment

analysis model is effective for processing user feedback and generating ranking scores. This combination makes the AI layer technically feasible and easy to integrate with the main application.

These technologies are well-documented, widely adopted, and compatible with each other. The team possesses the required programming knowledge, and the architecture supports future scalability. The only challenge may involve tuning the AI ranking model due to limited training data, but this can be improved gradually using user feedback.

2.3.2 Economic Feasibility

The economic feasibility of the Intelligent Ranking and Chat System for Property Rental is evaluated by analysing the financial requirements, long-term operational sustainability, and cost–benefit outcomes of implementing the platform. One of the major strengths of the proposed system is its reliance on open-source technologies, which significantly reduces development and maintenance costs. Frameworks and tools such as React.js, Spring Boot, MySQL, Redis, Python, Flask, and VADER Sentiment Analysis are freely available and well-documented. This eliminates licensing fees, making the solution highly affordable for academic institutions, small businesses, or startups exploring real-estate technology solutions.

Hardware costs are minimal, as the system can run efficiently on mid-range machines during development and testing. For deployment, the platform can be hosted on cost-effective cloud services that offer flexible pricing models, including pay-as-you-go or shared hosting. The modular architecture also enables selective scaling—meaning only the components experiencing high load, such as the backend or AI service, need additional resources. This prevents unnecessary spending and ensures that financial investments grow gradually with actual user demand.

Maintenance costs are controlled as well. Since the system is built using widely adopted technologies, support resources, developer communities, and documentation are readily available, reducing the expenditure on specialized expertise. Furthermore, Redis caching helps reduce database load, thereby lowering infrastructure usage and indirectly reducing hosting costs.

The economic benefits gained from implementing the system outweigh the initial setup costs. Improved personalization, faster search experiences, verified listings, and enhanced communication can attract more users, increase platform reliability, and potentially generate revenue through property listings, subscription models, or advertisement placements. Thus, the overall financial outlook is positive, making the project economically feasible both in the short term and as a scalable long-term solution.

2.3.3 Operational Feasibility

Operational feasibility assesses whether the system can be used effectively in real-world conditions, addressing user needs while ensuring smooth day-to-day functioning. The Intelligent Ranking and Chat System for Property Rental has been designed with ease of use, accessibility, and practical workflow integration as top priorities. The user interface is simple and intuitive, developed using React.js to ensure smooth navigation, fast updates, and responsive layouts. Both property seekers and owners can interact with the system without requiring technical expertise, as common actions—such as searching properties, submitting details, chatting, or giving feedback—are performed through familiar web elements.

The system's operational workflow mirrors how users naturally explore rental options. Seekers can browse listings, apply filters, read details, chat instantly with owners, and view recommendations that update based on their interactions. Owners can upload property details, manage chats, and track listing performance. These operations are logically organized within clean dashboards, reducing confusion and improving overall user experience.

Behind the scenes, the backend ensures that all processes—like data storage, ranking requests, real-time messaging, and verification checks—run smoothly. Since the architecture is modular, each service operates independently, reducing the chances of system-wide failures. Redis caching improves performance by minimizing delays, while MySQL handles structured data reliably. The AI module works automatically in the background, refining ranking outputs without affecting the user's normal workflow.

Only minimal staff involvement is required beyond initial deployment. Content moderation, verification checks, and occasional system monitoring can be managed easily through administrative tools. As the system scales, operational workloads can be supported by automated scripts, cloud monitoring services, or load-balancing tools.

Overall, the platform is practical, user-friendly, and capable of handling real-world usage scenarios, making it operationally feasible for continuous deployment in rental-search environments.

2.4 Software and Hardware Specification

Software Requirements

- Frontend: React.js
- Backend: Java, Spring Boot
- Database: MySQL, Redis
- AI/ML: Python, Flask, VADER
- Development Tools: VS Code, GitHub, Postman
- Operating System: Windows/Linux

Hardware Requirements

- Processor: Minimum Intel i5 or equivalent
- RAM: 8 GB or higher (recommended for running backend + AI services)
- Storage: At least 20 GB free space

Stable Internet Connection for database, testing, and API interactions

2.5 Data Flow Diagram

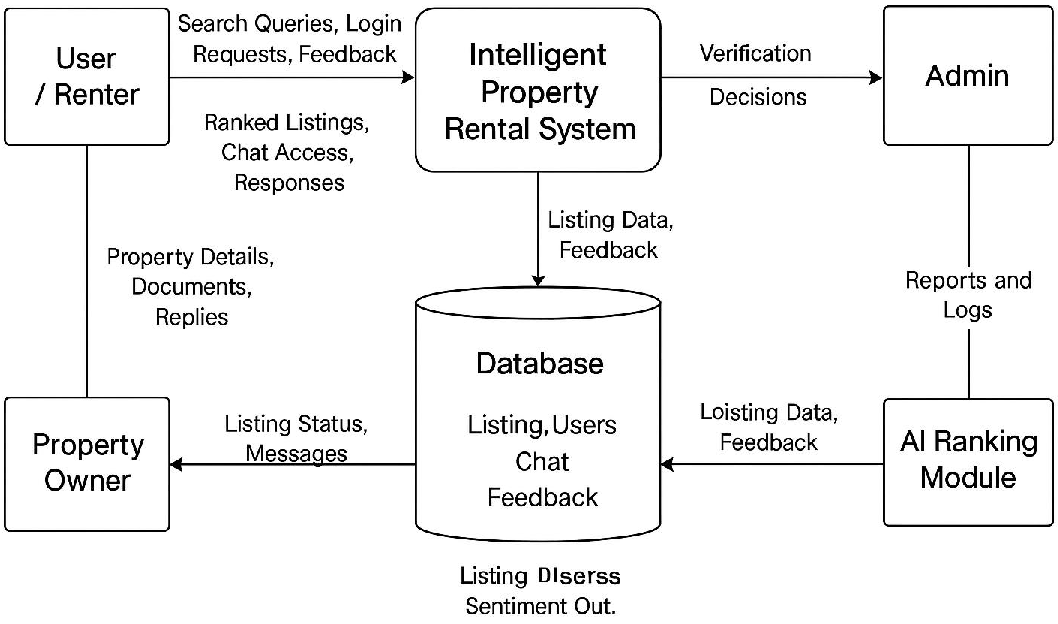

Figure 2.1: Data Flow Diagram

System Design

3.1 Design Methodology

The design phase outlines how the system components are structured and how different modules interact to achieve the desired functionality. Since the goal of the project is to build an intelligent, responsive, and scalable rental platform, the design methodology focuses on modularity, clear separation of concerns, and smooth integration between the AI layer and the core web services.

The system follows a hybrid design approach, combining traditional architectural models with AI-driven components:

a) Layered Architecture

The system is divided into distinct layers, each responsible for a specific function:

Presentation Layer (Frontend – React.js)
Handles user interaction, search input, chat interface, listing display, and dynamic content updates.

Application Layer (Backend – Spring Boot)
Manages business logic such as authentication, listing management, user feedback, ranking requests, and communication handling.

AI/ML Layer (Python + Flask)
Processes ranking algorithms, performs sentiment analysis, interprets conversational cues, and refines recommendations.

Data Layer (MySQL + Redis)
Stores structured data such as user profiles, listings, chat history, and feedback. Redis accelerates frequently accessed data.

b) API-Based Integration

Communication between the frontend, backend, and AI component is conducted through REST APIs.
This ensures:

Low coupling between components

Easy maintenance

Future scalability (e.g., mobile app integration)

c) Iterative Development

The system is developed in cycles, allowing each core module—search, ranking, chat, verification—to be built, tested, and improved progressively. User feedback collected during testing helps refine the ranking algorithm over time.

Overall, the methodology ensures a reliable, flexible, and intelligent system capable of expanding with additional AI features in the future.

3.2 User Interface Design

The user interface is designed with simplicity, clarity, and usability as the highest priorities. Since users frequently browse through multiple listings and interact with owners, the UI must be quick, responsive, and easy to navigate. React.js is used to build a component-based interface that updates content without page reloads, offering a smooth user experience.

Key UI Components

1. Home Page

Provides a clean search bar for entering location and basic filters.

Displays recommended listings generated from ranking API responses.

2. Listing View Page

Shows detailed information about each property: images, price, location, amenities, and verification status.

Includes buttons for saving, sharing, or contacting the owner.

3. Chat Interface

Real-time messaging window built using WebSocket-based communication.

Allows users to message property owners directly.

Messages load instantly to maintain conversational flow.

4. User Dashboard

Displays user activity such as saved properties, chat history, and feedback options.

Provides an organized view for easy decision-making.

5. Owner Dashboard

Allows property owners to add new listings, upload documents, and manage posted properties.

Includes a simple verification upload section.

UI Design Principles Followed

Consistency: Uniform colors, typography, and layout across pages.

Responsiveness: Works effectively on desktops, tablets, and mobile screens.

Minimal Navigation: Clearly visible links and buttons to reduce user confusion.

Accessibility: Adequate color contrast, readable text size, and intuitive icons.

The UI is crafted to support the intelligent nature of the system by making data easy to explore and AI-powered suggestions easy to understand. Combined, these elements produce an efficient and engaging user experience.

Testing

4.1 Testing Techniques & Strategies

Testing is a crucial phase of software development, ensuring that each module of the system performs as expected and functions smoothly when integrated. Since the platform involves multiple layers—frontend, backend, AI services, and real-time communication—different testing strategies were applied to validate reliability, accuracy, and performance.

a) Unit Testing

Unit tests were conducted on individual components to ensure that each part of the system behaved as expected before integrating them together.

• API endpoints in the Spring Boot backend: Each REST API endpoint was tested independently using tools like Postman and JUnit. Tests verified correct input handling, response status codes, validation errors, and data returned from service and repository layers. This ensured that backend operations such as authentication, listing management, and messaging worked correctly in isolation.

• React components for rendering listings, chat windows, and dashboards : Frontend unit tests checked whether components rendered correctly with different props and states. Tests were used to validate UI behavior, ensure proper event handling, and confirm that components updated correctly when receiving new data from APIs. This reduced UI bugs and improved overall interface reliability.

• Python functions related to ranking and sentiment analysis : The AI module was tested by passing sample feedback, listing data, and mock sentiment scores to verify that ranking logic operated correctly. Individual Python functions were tested to ensure that tokenization, sentiment scoring, and sorting mechanisms produced expected results.

• Mock data was used to isolate each unit and confirm that each function produced correct outputs. : Mock objects and sample datasets were used to avoid dependencies on external systems such as the real database or live API responses. This allowed each unit

to be tested independently, ensuring accurate output validation and making debugging easier.

b) Integration Testing

Integration testing focused on verifying how different modules interacted with one another after unit testing was completed.

• **React frontend and Spring Boot backend:** Tests ensured that the frontend correctly sent requests and received appropriate JSON responses from the backend. API endpoints for login, listing retrieval, chat loading, and feedback submission were verified to work seamlessly with the UI.

• **Backend services and the Python AI module:** Integration tests checked the communication between Spring Boot and the Flask-based AI module. The backend was tested to confirm that it correctly sent listing data to the AI API, received ranking scores, and integrated them into the final response shown to the user.

• **Backend services and MySQL/Redis databases:** Tests validated that the backend correctly stored, retrieved, and updated information in MySQL. Redis caching was also tested to ensure quick access to frequently requested data. These tests confirmed consistency between cached and persistent data.

These integration tests ensured that data moved accurately between all layers of the system. API calls, AI ranking responses, and real-time chat updates were checked for consistency and correctness.

c) Functional Testing

Functional testing focused on checking whether the system performed all the intended operations according to the project requirements.

• User registration and login: Tests verified that new users could sign up, existing users could log in, and authentication was handled securely with proper validation. Incorrect inputs were also tested to ensure proper error messages.

• Viewing and filtering property listings: Tests checked that users could browse listings, apply filters, and receive correct search results. The system was validated for proper display of verified status, amenities, and listing details.

• Real-time messaging between users and owners: Functional tests ensured that the chat system delivered messages instantly, updated conversations without reloading, and maintained a correct chat history for both users and owners.

• Submission and handling of feedback: Tests confirmed that users could submit feedback and that the backend correctly stored it. The AI module's ability to process this feedback for ranking adjustments was also validated.

• Ranking adjustment based on user interaction: Test cases verified that the system updated listing rankings based on sentiment scores, clicks, and user preferences. This ensured that recommendations responded dynamically to user behavior.

Test cases were designed around real user scenarios to confirm that each feature behaved as expected and aligned with the system's functional requirements.

d) Performance Testing

Since the system involves ranking and chat functionality, load tests were conducted to check responsiveness. Redis caching was evaluated to ensure faster retrieval of frequently accessed data.
Special focus was placed on:

API response time under multiple requests

Ranking model latency when generating personalized results

Message delivery delays in the real-time chat module

Testing showed that performance remained stable, though high loads may still introduce minor delays.

e) Usability Testing

The user interface was tested with peers to evaluate clarity, ease of navigation, and intuitiveness. Feedback from informal user sessions helped refine component layouts, button placements, and error messages.

4.2 Debugging & Code Improvement

Throughout development, debugging played a major role in enhancing system stability. Issues identified during testing were analyzed and resolved using tools such as the browser console, Spring Boot logs, and Python debugging utilities.

a) Frontend Debugging

Common issues included:

- Improper state updates in React components
- UI elements not rendering due to incorrect props
- Chat window not refreshing after new messages
- These were resolved by improving component lifecycle handling, restructuring API calls, and refining the WebSocket event listeners.

b) Backend Debugging

The Spring Boot backend required fixes for:

- Incorrect API mappings
- Validation errors during user input
- Slow queries affecting performance
- Optimizations such as indexing frequently queried fields, restructuring service methods, and improving exception handling helped stabilize backend operations.

c) AI Module Debugging

The Python-based ranking system occasionally returned unexpected results due to:

- Inconsistent input formatting
- Missing or incomplete user feedback
- Limited initial training data
- Adjustments were made to normalize inputs, fallback to default rankings when data was sparse, and include more sample data to improve accuracy.

d) Database and Caching Improvements

- Slow MySQL queries were optimized through indexing.
- Redis cache keys were reorganized to reduce retrieval conflicts.
- Database connection issues were resolved by fine-tuning Spring Boot configurations.

e) Code Refactoring

To enhance maintainability, several parts of the project were refactored:

To enhance maintainability, several parts of the project were refactored:

• **Reusable functions were extracted in React:** Reusable logic such as repetitive API calls, form handlers, and state-update patterns were moved into separate utility functions or custom hooks. This reduced code duplication and made components simpler and easier to update.

• **Service layers in Spring Boot were reorganized for cleaner logic:** Business logic was separated into well-structured service classes, reducing the load on controller methods and improving readability. Redundant code was removed, and responsibilities were clearly divided across controllers, services, and repositories.

• **AI ranking code was modularized for easier modification:** The Python-based AI module was broken into smaller files handling preprocessing, sentiment scoring, ranking logic, and routing separately. This modular approach allows quick modifications or model upgrades without affecting the entire system.

• **The continuous cycle of testing, debugging, and refining helped develop a stable and efficient system capable of supporting intelligent property search and real-time communication.:** Regular testing and debugging across all modules ensured that issues were found and fixed early. This iterative refinement resulted in a more dependable architecture with improved performance, smoother communication, and reliable AI-driven ranking.

Implementation

5.1 System Implementation

The implementation phase translates the system design into a working software product. For this project, the development process was carried out in modular stages to ensure smooth integration between the user interface, backend services, AI components, and database layers.

The system follows a multi-tier architecture, allowing each module to be developed, tested, and deployed independently:

a) Frontend Implementation

The frontend of the system has been developed using React.js, a component-based JavaScript library widely used for building interactive user interfaces. The modular nature of React enables the entire application to be divided into small, reusable components that manage their own logic and rendering. This enhances maintainability, scalability, and performance. The major frontend components developed in this project are described below:

- **Search Bar Component:** The search bar is one of the primary user interaction elements. It allows users to input location, price preferences, and basic filtering parameters. Real-time input handling using React state. Auto-triggered updates to fetch relevant property listings. Integration with backend APIs to retrieve results dynamically. Input validation to avoid incorrect or empty queries. The search bar acts as the entry point for the recommendation flow, ensuring that the system receives well-structured search parameters for further AI-based ranking.

- **Listing cards:** Each property displayed on the home page or search results page is represented through a listing card. These cards present summarized information about a property in a visually appealing format. Typical information shown includes Property title and a brief description, Price, location, and basic amenities, verification status (if the listing is verified),

thumbnail images, a “View Details” action button, the listing card component is designed to be highly reusable across multiple views—such as recommended listings, search results, and saved properties. It supports dynamic rendering, ensuring that updates on the backend (e.g., price changes or new listings) are reflected immediately on the client side, and Chat window

- **Chat window -** The chat window provides real-time communication between property owners and prospective renters. Functional characteristics include: WebSocket-based connectivity for instant messaging, automatically updating message threads without page reloads, Scrollable conversational interface, Indicators for message timestamps and sender identity, The chat interface significantly increases engagement and reduces delays in communication that typically occur when users rely on phone calls or external messaging services.

- **User dashboard -** The user dashboard acts as a personalized control centre for renters. Major functionalities include: Viewing saved or bookmarked listings, accessing previous chat histories, managing personal profile information, viewing property recommendations generated by the AI ranking API, submitting feedback that contributes to sentiment analysis. The dashboard allows users to conveniently revisit properties or conversations without having to search again, improving the overall user experience.

- **Owner dashboard -** The owner dashboard provides property owners with tools to manage their listings efficiently. Key features include: Adding new property listings with images and details, Editing or deleting existing listings, tracking chats or inquiries from potential renters, viewing verification status for each listing, managing account and profile settings. This section gives property owners a clean and centralized interface to keep their listings updated.

- **React Hooks and State Management -** React’s built-in hooks such as **useState**, **useEffect**, and **useContext** were used extensively to manage data flow and UI

behavior. Examples of usage include: useState (Handling component-level data like form inputs, search fields, or toggle states), useEffect (Fetching data from backend APIs when a component is mounted or updated), useContext (Managing global states such as logged-in user details or authentication tokens). This approach ensures modularity and avoids unnecessary re-renders.

- **API Integration Using Axios -** Axios was chosen as the HTTP client for API requests due to its simplicity and promise-based architecture. The frontend interacts with the backend via REST endpoints to perform operations such as: Fetching search results, Sending messages through the chat interface, Retrieving user dashboards, Submitting or retrieving feedback, Triggering the AI ranking service**.** Axios also handles authorization headers (JWT tokens) to ensure secure communication.

b) Backend Implementation

The backend was implemented using Spring Boot, focusing on clear separation of controllers, services, and repositories. Key backend functionalities include:

• **User Authentication** – The backend manages the complete authentication workflow using JWT (JSON Web Tokens) to ensure secure access to system resources. When a user logs in, the credentials are verified, and a token is issued that must accompany every subsequent request to protected APIs. This mechanism prevents unauthorized access, maintains user identity across sessions, and supports role-based permissions for different types of users within the platform.

• **Listing Management** – All property-related operations are handled by the backend, which includes adding new listings, updating existing information, and removing outdated entries. Each submission from property owners is validated, stored in the database, and made available to seekers through dedicated APIs.

Backend logic ensures that the listings remain consistent, searchable, and properly categorized for efficient retrieval.

• **Chat Message Handling** – The backend processes every message exchanged between users and property owners. Incoming messages are received through REST endpoints, time-stamped, and saved in the database so that complete conversation histories can be fetched whenever required. This ensures smooth communication, preserves chat continuity, and supports future analytical improvements.

• **Data Validation** – Before any request is processed or stored, the backend performs strong validation checks. These include verifying field completeness, checking data types, preventing invalid or harmful input, and ensuring the overall accuracy of submitted information. Effective validation safeguards the database, reduces system errors, and helps maintain high-quality data throughout the application.

• **Request Routing to the AI Module** – The backend acts as the connecting layer between the main system and the AI ranking service. It gathers necessary inputs such as listing attributes, usage patterns, and feedback, then forwards them to the Python-based AI module through REST communication. After receiving the computed ranking scores, the backend organizes the results and returns them to the frontend for personalized display.

• **Feedback Processing** – Feedback provided by users is received and validated by the backend before being stored in the database. This information plays an important role in refining the ranking logic, as the backend periodically sends collected feedback to the AI module. By integrating user responses into the system, the platform continuously improves the accuracy and relevance of its recommendations.

REST APIs were developed to connect the frontend with backend logic and to communicate with the AI service.

c) AI/ML Module Implementation

The ranking and sentiment analysis components were developed using Python and deployed via a Flask API.

Major tasks performed by the AI module:

- **Generating ranking scores based on listing attributes** – The AI module analyzes various property attributes such as price, location, amenities, room type, and user engagement patterns. These attributes are processed through the ranking logic to compute a numerical score for each listing. This score reflects how closely a listing aligns with user preferences, helping the system display the most relevant options first.

- **Adjusting ranking using user feedback** – The AI component continuously refines ranking outputs by incorporating user feedback. Ratings, comments, and interaction behavior are evaluated to identify patterns of user satisfaction or dissatisfaction. Based on this information, the system adjusts future ranking scores so that listings with consistently positive feedback appear higher, while less preferred listings are ranked lower.

- **Using VADER to evaluate sentiment from chat or comments** – The system employs the VADER sentiment analysis model to interpret the tone of user-generated text, such as chat messages or written feedback. VADER classifies text as positive, negative, or neutral, allowing the AI to understand emotional cues and user attitudes. These sentiment insights help refine the relevance of listings and improve personalized recommendations.

- **Returning personalized recommendations to the backend** – After analyzing listing attributes, user behavior, and sentiment results, the AI module generates personalized ranking outputs and sends them back to the backend through REST communication. The backend then organizes the response and delivers tailored property suggestions to the frontend, ensuring that each user receives results that match their unique requirements.

The AI module was structured to allow additional models or algorithms to be integrated later without major changes.

d) Database & Caching Layer

• **MySQL was used to store users, listings, feedback, and chat history** - MySQL functions as the primary relational database for the system, responsible for preserving structured and permanent data. It stores essential information such as user profiles, property listings, chat conversations, and feedback entries. The relational structure ensures data consistency, supports foreign keys for linking users and listings, and provides efficient querying for operations like filtering listings or retrieving user history. MySQL's ACID compliance ensures reliability, which is crucial for real-world property rental platforms where data integrity cannot be compromised.

• **Redis was used as a cache for frequent queries** (e.g., popular listings), helping reduce API latency. Redis was integrated as an in-memory caching layer to store data that is frequently requested by users—such as trending properties or repeated search queries. Since Redis operates directly from RAM, it provides extremely fast read and write speed compared to disk-based databases. This allows the system to respond quickly during high-traffic periods without placing unnecessary load on MySQL. By caching computed results or repeated API responses, Redis significantly reduces latency and enhances the overall user experience.

• **SQL queries were optimized, and database schemas were normalized** to improve performance. To ensure the backend performs efficiently under increasing data size, multiple SQL optimizations were implemented. Indexing was applied on columns like price, location, and listing ID to speed up search operations. Schemas were normalized (up to 3NF) to eliminate redundant data, improve consistency, and reduce storage overhead. Queries were rewritten using optimized JOINs, prepared statements, and pagination (LIMIT/OFFSET) to

prevent full-table scans. These optimizations resulted in faster response times and more predictable performance under load.

5.2 Software Implementation

The software implementation focuses on converting functional requirements into working code using the selected tools and technologies.

a) User Authentication and Access Control

JWT (JSON Web Token) authentication was implemented in the Spring Boot backend to ensure secure access. Login, registration, and session management were integrated with the frontend.

b) Listing Management

Owners can add, edit, or remove listings. The backend validates every input, stores verified data securely, and retrieves listings efficiently for display on the frontend. This module ensures that the property data shown to users is accurate, well-structured, and consistently updated.

Each listing includes:

- **Basic details** - Every listing contains essential information such as the property title, description, price, location, available amenities, room type, and occupancy details. These details allow users to quickly understand the key features of the property and determine its suitability. The system ensures that mandatory fields are provided and properly formatted before the listing is saved to the database.

- **Photos** - High-quality images form an important part of the listing, giving users a visual representation of the property. Owners can upload multiple photos, which are stored securely and linked to the listing. The backend performs validations such as file size checks, format support (e.g., JPEG/PNG), and safe storage procedures. These photos are displayed in the UI through optimized image rendering for faster loading.

- **Verification status** - Each listing has an associated verification flag that indicates whether the property has been authenticated by the platform or admin. This verification status builds user trust by differentiating authentic, reviewed

listings from unverified ones. Owners can upload documents or proofs, which are validated before verification status is updated. The system visually highlights verified listings on the frontend.

• **Owner information** - Details about the property owner—such as name, contact information, and profile status—are stored and linked to each listing. This information allows renters to contact owners directly through the built-in chat system. The backend ensures that only authorized owners can modify their listings, and all interactions are logged securely. Displaying owner data enhances transparency and improves user confidence during communication.

c) Ranking Algorithm Integration:

The backend communicates with the AI module through structured REST requests, sending detailed listing information and user-related data to the Flask-based ranking service. Once the data is received, the AI system processes it using predefined ranking logic and returns a score for each listing. These scores are calculated by examining multiple factors, including:

• **Relevance** – The AI evaluates how well a property matches the user's stated preferences such as budget, location, and room type. Listings that closely fit the user's requirements receive higher priority in the ranking list.

• **User interactions –** The system considers how the user has interacted with similar listings, including clicks, views, time spent on certain properties, and saved items. These behavioral cues help the model understand user intent and refine future results.

• **Sentiment patterns –** Using sentiment analysis, the AI interprets the emotional tone in user comments, chat responses, or feedback. Positive sentiments can boost a listing's score, while negative sentiments may reduce its rank.

• **Feedback history –** Past ratings and opinions provided by users are incorporated into the ranking model. Listings with consistently strong feedback are placed higher, enabling the system to deliver results that reflect real user experiences and satisfaction trends.

These rankings are then sorted and displayed to the user in real time.

d) Real-Time Chat Implementation:

The chat module is implemented using WebSocket communication integrated with Spring Boot, allowing users and property owners to exchange messages instantly. The WebSocket connection provides a continuous, bidirectional communication channel, ensuring smooth and real-time interaction between both parties. Key features of the chat system include:

- **Instant message delivery** – Messages are transmitted in real time without requiring page reloads, enabling seamless and interactive conversations.
- **Typing indicators –** The system displays when the other user is typing, making the chat experience more natural and intuitive.
- **Chat history storage –** All messages are saved systematically in the database, allowing users to view previous conversations whenever needed.
- **Notifications for new messages –** Users receive alerts when new messages arrive, ensuring timely responses and improved engagement.

This ensures smooth communication between property owners and potential renters.

e) Feedback Handling

Users can provide feedback about listings, which is stored in MySQL. The AI module periodically uses this feedback to refine ranking recommendations.

5.3 Software Installation

The installation process outlines how the complete system can be set up on a local machine or deployed on a server.

• **Node.js & npm (for React frontend):** Node.js provides the runtime environment required to execute JavaScript outside the browser, which is essential for building and running the React frontend. npm (Node Package Manager) is bundled with Node.js and is used to install dependencies, manage project libraries, and execute frontend build scripts. React's development server, component-based builds, and production bundling all rely heavily on Node.js and npm.

• **Java JDK 17+ (for Spring Boot backend) :** The Spring Boot backend requires Java Development Kit (JDK) version 17 or higher for compiling and running server-side code. JDK includes essential tools such as the Java compiler, runtime environment, and debugging utilities. Using JDK 17+ ensures compatibility with modern Spring Boot versions, improved performance, and access to updated language features.

• **Python 3.8+ (for Flask AI module):** Python is required to run the AI module responsible for ranking, sentiment analysis, and intelligent recommendations. Version 3.8 or above ensures compatibility with libraries such as Flask, VADER Sentiment, NumPy, and other data-handling packages. The Flask server exposes API endpoints that the backend calls to retrieve ranking results.

• **MySQL Server:** MySQL acts as the main relational database system where user data, property listings, feedback history, and chat conversations are stored. Running MySQL Server locally or on a hosted service allows seamless communication with the Spring Boot backend. Proper configuration of users, schemas, and permissions is required before the backend can perform CRUD operations.

• **Redis:** Redis is used as an in-memory data store to cache frequently accessed results, improving overall system performance. It reduces database load and speeds up API response times for operations like fetching popular listings or repeated searches. Redis must be installed and running before the backend can use it for caching.

• **Git:** Git is required for version control, enabling developers to track changes, manage branches, and collaborate efficiently. It also allows seamless cloning of repositories for the frontend, backend, and AI modules. Using Git ensures proper project organization and easy rollback in case of issues.

b) Backend Setup (Spring Boot)

• **Clone the backend repository using Git:** The first step is to clone the backend source code from the remote Git repository to your local machine. This ensures that you have the complete project structure, including all Java files, configuration files, and Maven dependencies. Cloning also enables easy synchronization with future updates or bug fixes from the repository.

• **Import the project into any Java IDE (IntelliJ / Eclipse):** Once downloaded, the project should be opened in a Java-integrated development environment such as IntelliJ IDEA or Eclipse. The IDE automatically detects the Maven project and loads all necessary dependencies defined in the pom.xml file. Using an IDE provides built-in support for editing, debugging, and running Spring Boot applications efficiently.

• **Configure application.properties with MySQL and Redis credentials:** The backend requires proper database configuration to connect with MySQL for persistent storage and Redis for caching. In the application.properties file, credentials such as database URL, username, password, Redis host, and port must be set. Additional Spring Boot configurations—like server port, Hibernate dialect, and connection pooling—may also be customized based on system needs. Correct configuration ensures that the backend can interact smoothly with external services during runtime.

• **Run the application using Maven or IDE's run option:** After configuration, the Spring Boot application can be started directly from the IDE using the built-in run option, or via the Maven command mvn spring-boot:run. When executed, Spring Boot initializes all beans, loads configurations, establishes database connections, and starts the embedded server (typically Tomcat). Any startup warnings or errors (e.g., incorrect credentials or service unavailability) are displayed in the console for troubleshooting.

• **Verify API availability via Postman:** Once the backend is running, the exposed REST APIs should be tested using Postman or a similar API testing tool. Developers can send different types of requests (GET, POST, PUT, DELETE) to verify that features such as user authentication, listing management, chat system, and feedback submission work correctly. Checking the response status codes, headers, and JSON outputs ensures that the backend is functioning properly before integrating with the frontend or AI module.

c) Frontend Setup (React.js)

- Open the project folder in VS Code.
- Run npm install to install dependencies.
- Use npm start to launch the development server.
- Configure API endpoints to point to the backend URL.

d) AI Module Setup (Python + Flask)

- Install required packages using pip install -r requirements.txt.
- Run Flask server using python app.py.
- Confirm endpoint availability using a browser or Postman.

e) Database Initialization

- Create necessary schemas in MySQL.
- Import the SQL script to generate tables.
- Start Redis server for caching.

f) Final Deployment

Once all services are running:

- Frontend communicates with Spring Boot backend
- Backend interacts with MySQL, Redis, and Flask AI service
- Users can access the system from a web browser

This installation procedure ensures that the system operates as a cohesive and fully functional web application.

5.4 Result

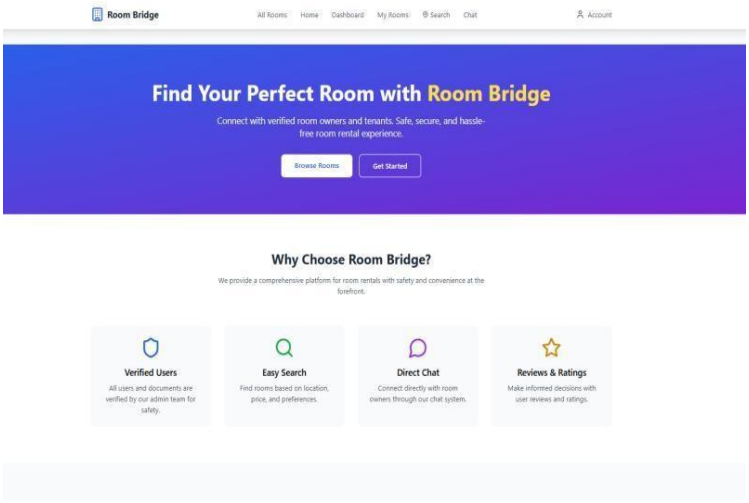

Figure 5.4.1: Home Page

A visual representation of the website's main interface, the layout and key navigation elements

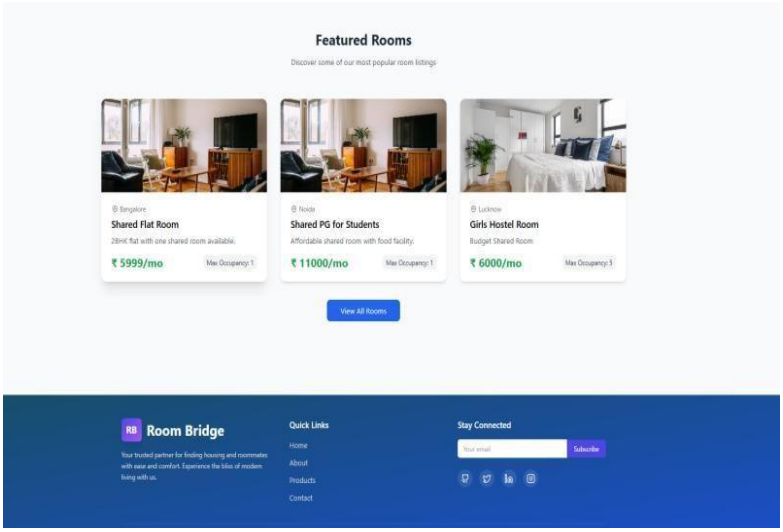

Figure 5.4.2: All Listing Page

An overview of a all listings, displaying relevant information, and options for interaction

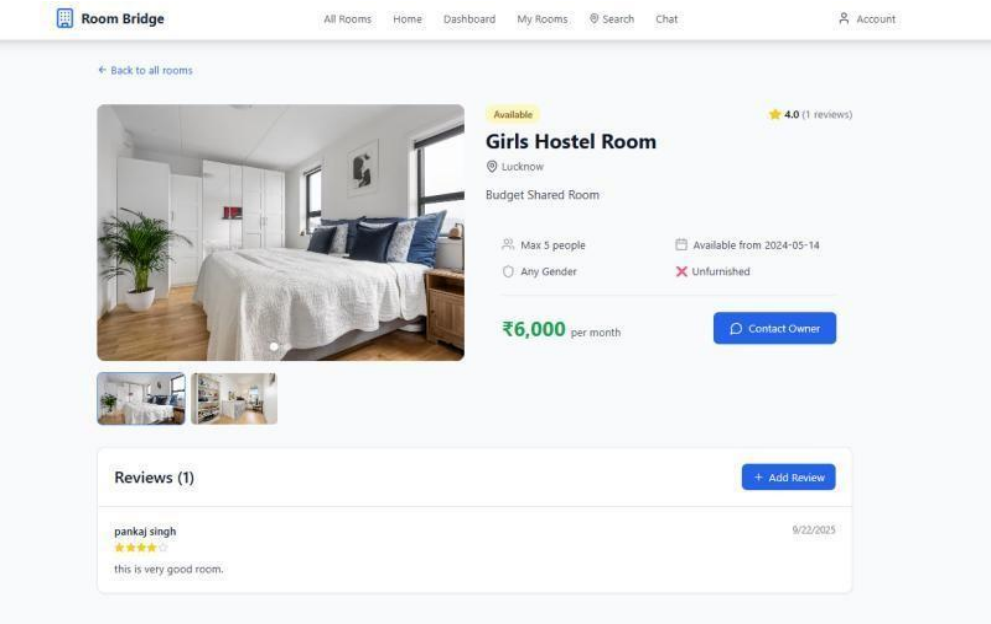

Figure 5.4.3: Listing Details Page

A detailed view of a specific listing, displaying detailed information, images, and options for interaction or purchase

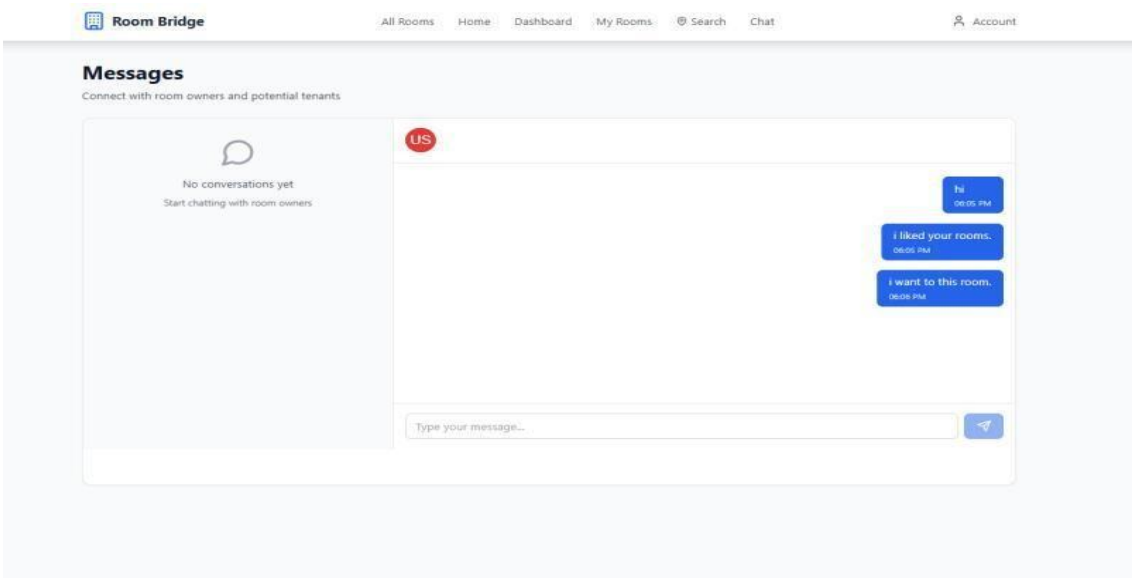

Figure 5.4.4: Real-Time Chat Interface

A dynamic communication window allowing users to send and receive messages instantly within the platform

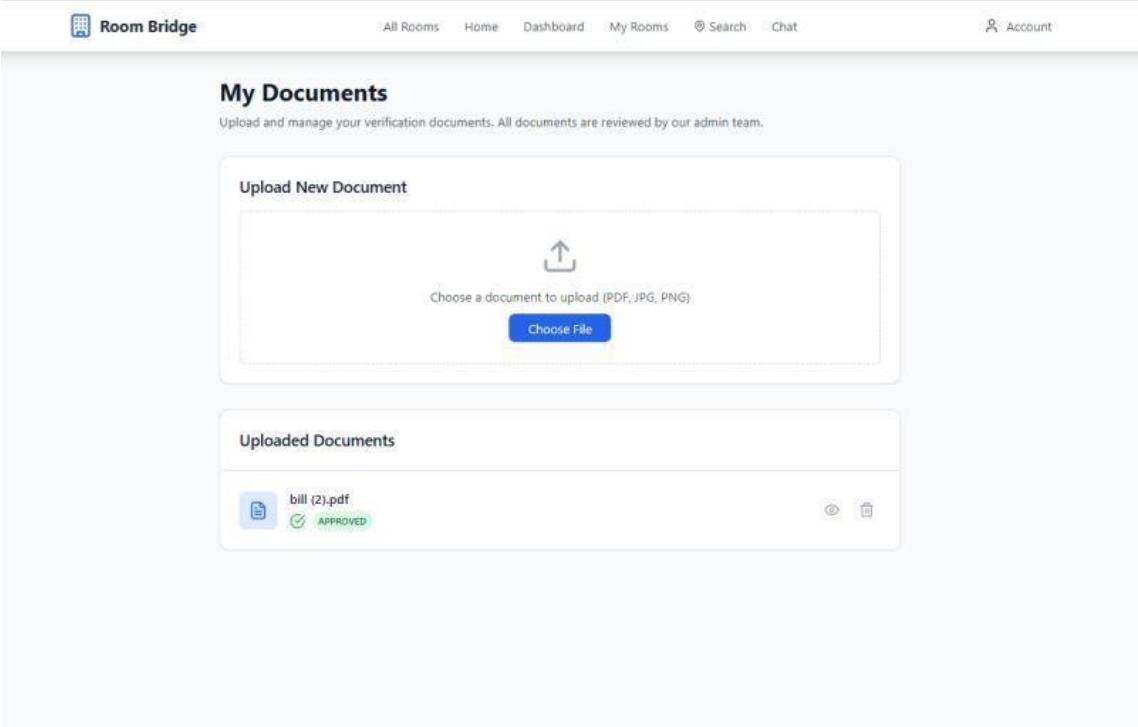

Figure 5.4.5: Document Upload

An interface feature enabling users to easily select and upload files to the platform for processing or sharing.

Conclusion & Future Scope

6.1 Conclusion

The Intelligent Ranking and Chat System for Property Rental has been successfully designed and implemented to address the limitations of traditional rental platforms. The system provides users with a more personalized, reliable, and efficient experience by integrating advanced AI-based ranking, verified listings, and real-time messaging.

Key achievements of the project include:

Personalized Search and Ranking: The AI module evaluates user preferences, behavior, and feedback to display relevant property listings.

Verified Listings: By incorporating verification checks, the system improves user trust in the information provided.

Real-Time Communication: A chat system allows users and property owners to interact instantly, enhancing engagement and reducing response delays.

Feedback Integration: User feedback is actively used to refine the ranking algorithm and improve overall accuracy over time.

Scalable Architecture: The modular, layered design ensures that future upgrades, such as adding mobile support or new AI features, can be implemented with minimal disruption.

Through careful system analysis, iterative development, and rigorous testing, the platform has demonstrated its practical feasibility, operational efficiency, and usability in metropolitan property rental scenarios.

6.2 Future Scope

While the current system successfully achieves its core objectives, there remains substantial room for further enhancement and expansion. These improvements can significantly elevate user experience, system intelligence, and overall platform reliability.

Mobile Application Development: Developing dedicated mobile applications for Android and iOS would make the platform more accessible and convenient for users. A mobile app would support features like push notifications, real-time chat alerts, location-based recommendations, and offline browsing capabilities. This expansion would help reach a wider audience, especially users who primarily rely on smartphones for searching rental properties.

Advanced AI and Machine Learning: The current sentiment-based ranking model can be extended with more advanced AI techniques. Implementing collaborative filtering, content-based filtering, or deep learning models can significantly enhance recommendation accuracy. Predictive analytics can be introduced to forecast property availability, price trends, or user interest patterns, helping both renters and property owners make informed decisions. A reinforcement learning approach could help the system continuously refine rankings based on real-time feedback.

Enhanced Verification Mechanisms: Integrating automated document verification can streamline the property approval process and reduce manual checks. AI-driven image verification models can detect manipulated photos or confirm whether uploaded images match the property description. Such mechanisms would strengthen platform authenticity and build greater trust among users.

Payment and Booking Integration: Introducing in-app payment gateways can allow users to book rooms, pay deposits, or secure properties directly through the platform. This would transform the system into a fully integrated rental ecosystem, reducing users' reliance on external applications for transactions. Features such as booking confirmations, refund processing, and automated receipts can add more professionalism to the platform.

Enhanced Chat Features: AI-powered chatbots can be incorporated to instantly answer common questions, guide users through the search process, or help owners manage their listings. Chat translation and multi-language support would make the system more inclusive for users from different regions and linguistic backgrounds. Additional features like message read indicators, file sharing, and voice notes could further improve communication.

Performance Optimization: As user traffic scales, database operations may need further optimization through advanced indexing, sharding, or query restructuring. Redis caching strategies can be expanded to include more frequently accessed datasets to reduce backend load. Reducing AI model inference time can improve real-time recommendations and ensure a smoother user experience.

Analytics Dashboard: A dedicated analytics dashboard for property owners and administrators can provide insights such as:

- Most viewed or most contacted listings
- Feedback trends and user satisfaction metrics
- Market demand patterns in specific areas

These insights can help owners make data-driven decisions and optimize their listings to attract more users.

Conclusion:

Overall, the current implementation provides a strong and functional foundation for an intelligent rental platform. However, the proposed enhancements—ranging from mobile apps to advanced AI integration—can substantially increase the system's usability, scalability, and market relevance. With continuous improvement, the platform can evolve into a fully automated, highly intelligent, and user-centric rental solution.

References

[1] Mozilla Developer Network (MDN), "HTML5 Guide," *MDN Web Docs*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTML. Accessed: 2025.

[2] Mozilla Developer Network (MDN), "CSS3 Documentation," *MDN Web Docs*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/CSS. Accessed: 2025.

[3] ECMA International, *ECMAScript® 2024 Language Specification*. [Online]. Available: https://tc39.es/ecma262/. Accessed: 2025.

[4] React Team, "React.js Official Documentation," *React.dev*. [Online]. Available: https://react.dev/. Accessed: 2025.

[5] Google Developers, "Web Vitals Metrics," *web.dev*. [Online]. Available: https://web.dev/vitals/. Accessed: 2025.

[6] Bootstrap Team, "Bootstrap 5 Documentation," *getbootstrap.com*. [Online]. Available: https://getbootstrap.com/. Accessed: 2025.

[7] Axios Team, "Axios HTTP Client," *axios-http.com*. [Online]. Available: https://axios-http.com/. Accessed: 2025.

[8] W3C, "Web Content Accessibility Guidelines (WCAG) 2.2," *World Wide Web Consortium (W3C)*. [Online]. Available: https://www.w3.org/TR/WCAG22/. Accessed: 2025.

[9] Redux Team, "Redux Toolkit Documentation," *redux.js.org*. [Online]. Available: https://redux.js.org/. Accessed: 2025.

[10] J. Duckett, *HTML and CSS: Design and Build Websites.* Indianapolis, IN, USA: Wiley, 2011.

[11] Y. Katz, *Learning React: Modern Patterns for Developing React Apps*, 2nd ed. O'Reilly Media, 2023