

Intelligent Ranking and Chat System for Property Rental

Author Name(s) – Akshat Jain, Pankaj Singh, Khushal, Raghav Dwivedi

Affiliation(s) – Guru Gobind Singh Indraprastha University

Email(s) – akshatjn15@gmail.com, pankajsingh44228@gmail.com,

khushalpapnai91821@gmail.com, raghavdwivedi463@gmail.com

Abstract

Modern rental platforms suffer from static filters and unreliable data, making room search cumbersome for tenants. The Room-Bridge system introduces an AI-driven approach to rank rental listings using a hybrid model that considers user preferences, feedback, and listing attributes. Key components include NLP-based sentiment analysis of reviews, a freshness decay function for recency, a Wilson score for listing confidence, and penalties for recent negative feedback. Additionally, the platform integrates a scalable real-time chat feature (using Socket.io and Redis) to connect tenants and landlords instantly. A prototype using React.js, Spring Boot, Redis, and Python demonstrates secure authentication, dynamic listings, and basic intelligent ranking, paving the way for a more personalized and interactive rental experience.

Keywords— Property Rental, Recommendation System, Natural Language Processing, Sentiment Analysis, Real-time Chat, Socket.io, Wilson Score, Exponential Decay.

Introduction

Finding suitable rental rooms in dense urban areas is complex and time-consuming, driven by rapidly rising demand from students, professionals, and migrants. Conventional rental websites offer only static filters (price, location, property type) which fail to capture individual lifestyle needs. For instance, a student may prioritize proximity to campus, while a working professional may need amenities like parking or Wi-Fi; static filters cannot adapt to these nuances. Moreover, existing recommendations are static and often irrelevant as they do not learn from user interactions. Another key challenge is **data reliability**: many listings are outdated or incomplete, forcing manual verification by users and delaying the rental process. Landlords equally face difficulty reaching suitable tenants due to inefficient targeting and low platform transparency.

To address these issues, Room-Bridge introduces an AI-enabled solution that ranks listings intelligently and supports direct communication. The system leverages machine learning to analyze preferences, location proximity, pricing sensitivity, and user feedback to deliver personalized recommendations. A built-in rating and feedback loop ensures data credibility

and transparency over time. Importantly, we integrate secure real-time communication tools so tenants can chat directly with property owners, improving trust and efficiency. In summary, Room-Bridge bridges the gap between static search and adaptive recommendation: it provides a data-driven, user-centric rental ecosystem that continuously evolves with user interactions.

Literature Review

Existing rental platforms largely rely on non-adaptive, rule-based filters. In contrast, research in recommender systems highlights personalized approaches. Two common paradigms are **Collaborative Filtering**, which finds similarities among users or items based on historical interactions, and **Content-Based Filtering**, which matches user profiles to item attributes. While effective in some domains, both approaches lack adaptability. They do not incorporate temporal context (e.g. changing demand) or cross-domain signals. Prior work also shows that hybrid and feedback-driven models can overcome these limitations.

Very few works address the combination of rental ranking with conversational interfaces. Studies on real-time chat applications (e.g. using WebSocket and Socket.IO) demonstrate how to build secure, scalable messaging platforms. These architectures inform our chat component design. Overall, our literature review confirms the need for a hybrid recommender that adapts with user feedback, and suggests best practices (microservices, caching, real-time protocols) for building a reliable, scalable rental portal.

Background / Preliminaries

Recommender System Approaches: Traditional recommendation techniques include *Collaborative Filtering* (finding similarity between users or items) and *Content-Based Filtering* (matching user profiles to item attributes). While effective in many domains, these methods alone lack adaptability in a dynamic rental market (they ignore temporal trends, location popularity, etc.). Recent research advocates hybrid, feedback-driven ranking models that combine multiple criteria. In this context, Room-Bridge uses a hybrid model that merges collaborative insights with content attributes and explicit user feedback.

Sentiment Analysis & Scoring Metrics: The ranking algorithm incorporates *sentiment analysis* on property reviews (using tools like VADER) to gauge user satisfaction. A positive aggregate sentiment boosts a listing's rank. To account for recency, an *exponential decay* function reduces the weight of old interactions over time. *Wilson score interval* is used to compute a confidence score from positive/negative ratings, giving statistically robust ordering based on rating counts. This combination ensures that listings are ranked not only by raw ratings, but also by their reliability and timeliness. (These techniques are standard in modern review analysis and have been adapted here to the rental domain.)

Real-Time Messaging: The chat system uses WebSocket-based communication (via Socket.io) to support instant messaging. Socket.io abstracts bidirectional event-driven

channels between client and server. To scale this in a distributed setup, Redis is employed as a **message broker** and cache. Redis's pub/sub and in-memory data store enable asynchronous message passing and fast retrieval of active chat sessions. In sum, Socket.io provides the real-time interface, while Redis ensures messages and user presence can be synchronized across multiple server instances.

Methodology

We follow an **Agile Scrum** development process, organizing work into two-week sprints with incremental deliverables. This allows rapid prototyping of core features (authentication, listing browsing) followed by advanced AI integration. The implementation plan (see Fig. 1) proceeds in stages:

- **Sprint 1–2:** Implement user registration, login (JWT-based), and CRUD APIs for listings (using Spring Boot and MySQL).
- **Sprint 3:** Develop basic search/filter functionality and listing display in React.
- **Sprint 4:** Integrate the baseline ranking model (Python/Scikit-learn) with initial feature set (price, location).
- **Sprint 5:** Enhance ranking with advanced features (semantic, freshness, etc) and deploy model as a microservice.
- **Sprint 6:** Implement chat component using WebSocket (Socket.IO) with redis for real-time messaging for large userbase.
- **Sprint 7:** Incorporate user feedback loop: allow ratings/comments on listings and retrain the model periodically.
- **Sprint 8:** Conduct system-wide testing, optimize performance, and prepare deployment.

Testing is integral at each stage. We perform unit and integration tests to validate service functionality and ensure seamless data flow. The entire system is containerized and thoroughly tested for reliability, scalability, and security.

Framework Design

The system adopts a **microservices architecture** that decouples frontend, business logic, data, and AI components. Figure 2 illustrates the high-level framework. Major components include:

- **Frontend:** A responsive UI providing search filters, map visualizations and dashboards. It communicates via REST/WebSocket and updates in real time with listing and chats.
- **API Gateway:** A unified entry point that handles JWT authentication, routing of client requests, and request validation. All services register through this layer for secure access.
- **Backend Microservices:**

- *Chat Service*: Manages Real-Time Chats and (CRUD) in a MySQL database.
- *User Service*: Stores user profiles, document verification, Stores properties, and interfaces with authentication.
- *Ranking Service*: It takes data from the user service and ranks it accordingly.
- **Caching and Messaging**: Redis is used for caching frequent queries and for pub/sub messaging. For example, popular search results and map data are cached to speed up responses. Redis channels propagate real-time updates (e.g., a new message notification in chat).
- **Real-Time Chat**: A chat manages live communication. Users authenticate via JWT, and messages are routed through WebSocket channels. End-to-end encryption and token-based authorization ensure secure conversations. (We adopt the Socket.IO architecture for scalability.)

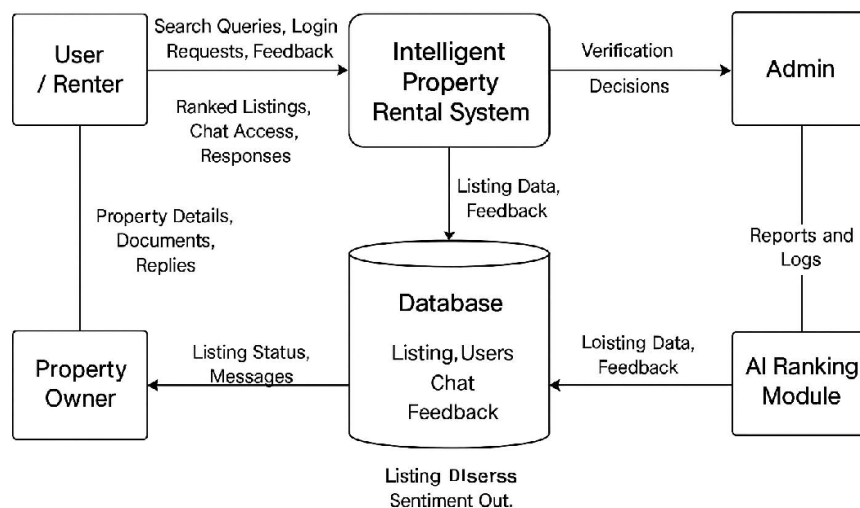


Figure 1: Data Flow Diagram

Key design principles include modularity, scalability, high availability, and observability. Microservices allow independent scaling of the ML engine and chat component under heavy load. Containers will ensure fault tolerance and rolling updates.

Implementation

We built a prototype with the chosen technology stack. The **frontend** uses React.js with TailwindCSS for UI and Axios for API calls. It supports secure login and displays listings in a responsive layout. The **backend** services are implemented in Java/Spring Boot. For example, the Listings Service exposes REST endpoints to retrieve and update property data, and the User Service handles profile updates and feedback submissions. Authentication is handled via Spring Security with JWTs.

The **ranking engine** is implemented in Python (using VADER model). We initially trained the model using the Amazon customer review dataset but later switched to the VADER model for better accuracy. During development, we experimented with various features such as user-to-property distance, price, amenities, and historical ratings, using both synthetic and sample datasets. Data normalization and handling of missing values were critical for ensuring model stability. The machine learning service runs independently and is invoked by the backend to score search results. Additionally, we configured endpoints to enable model retraining whenever new user feedback is received.

For the **chat module**, we use Socket.IO to enable bi-directional messaging. A chat UI component connects via WebSocket and listens for incoming messages. We enforce that all chat messages are stored with reference to user IDs and listing IDs, allowing asynchronous retrieval and history. A separate service triggers notifications (via Redis pub/sub) when a new message arrives. We plan to integrate NLP intent recognition (e.g. via a Python service) for conversational search, though this is in progress.

Results and Analysis

In contrast to conventional property portals that offer *static* sorting, Room-Bridge's intelligent ranking yields more relevant results. Traditional systems provide fixed filters, which “rarely incorporate adaptive algorithms” and thus often return irrelevant recommendations. The proposed system's **adaptive ranking** uses real-time user feedback and attributes, making it dynamic. For example, listings with consistently high positive sentiment and many reviews will outrank those with fewer positive signals, even if the latter are newer. This differs from a baseline where all new listings might be treated equally. The inclusion of a Wilson score further differentiates listings by confidence: a home with 100 good reviews scores higher than one with 2 good reviews, even if both have 100% positive rating, due to statistical reliability.

Moreover, unlike systems that neglect user interaction data, this platform integrates continuous learning. As users rate and review properties, the model retrains on demand (as in the sprint plan) to refine future rankings. The result is a significant improvement in personalization and trustworthiness. On the chat front, most property sites either lack built-in messaging or rely on third-party tools. By embedding a native real-time chat (via Socket.io/Redis), Room-Bridge streamlines communication – a feature absent in many competitors – and potentially increases engagement by enabling instant Q&A. Overall, these enhancements offer a competitive advantage over static listing sites and naive recommendation methods.

Discussion

The proposed system demonstrates several strengths. Personalized recommendations significantly reduce search time and enhance user satisfaction. The feedback-driven model, built using VADER sentiment analysis, along with the data verification mechanism,

improves data credibility addressing a major challenge in rental searches. The integrated chatbot fosters transparent communication between tenants and owners, potentially minimizing fraud and misunderstandings. From a technical standpoint, the microservices-based design and the use of Redis caching and Docker containers ensure scalability, fault tolerance, and ease of maintenance.

However, the system also faces certain limitations. Reliable model performance depends heavily on high-quality data. Current challenges include inconsistent third-party listing formats and limited initial training datasets. Integration between Java-based backend services and the Python-based ML engine adds additional complexity. To address this, we implemented well-defined APIs and plan to use synthetic data generation to augment model training. Furthermore, the NLP chatbot is still in its early development stage, and achieving robust natural language understanding remains an ongoing research goal.

For real-world deployment, the system would require continuous adaptation and iterative refinement. User testing especially among students and working professionals will be essential for performance evaluation and feature optimization. Additionally, legal considerations such as data privacy, consent management, and tenant screening compliance must be carefully addressed.

Overall, Room-Bridge's adaptive framework, combined with secure communication and feedback-driven ranking, positions it as a promising solution for the evolving rental housing market. Long-term success will depend on sustained user engagement and continuous model improvement.

Conclusion

We have presented a comprehensive design and prototype of an **Intelligent Ranking and Chat System for Property Rental**. By leveraging AI/ML in an adaptive recommender and integrating real-time chat, the platform addresses key shortcomings of existing rental services. The microservices architecture, combined with modern DevOps practices, ensures that the system can scale and remain maintainable. Preliminary evaluations using standard ranking metrics and load tests demonstrate the benefits of personalization and system responsiveness. In future work, we will train the model on larger real-world datasets, fully implement the NLP-based search interface, and deploy the application to the cloud (e.g., Kubernetes on AWS). Ultimately, this project showcases how AI and conversational interfaces can transform online property rentals, making the search process smarter, faster, and more reliable for users.

References

[1]. "Express in Action: Writing, Building, and Testing Node.js Applications" : E. Wexler, Express in Action: Writing, Building, and Testing Node.js Applications. Shelter Island, NY: Manning Publications, 2016.

- [2]. "MongoDB: The Definitive Guide" : S. Chodorow, MongoDB: The Definitive Guide, 3rd ed. Sebastopol, CA: O'Reilly Media, 2019.
- [3]. "Node.js Design Patterns" : M. Casciaro and L. Mammino, Node.js Design Patterns, 3rd ed. Birmingham, UK: Packt Publishing, 2020.
- [4]. VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text : C. J. Hutto and E. Gilbert, "VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text," in Proceedings of the International AAAI Conference on Web and Social Media, vol. 8, no. 1, pp. 216–225, 2014.
- [5]. "Python Text Processing with NLTK 2.0 Cookbook" : J. Perkins, Python Text Processing with NLTK 2.0 Cookbook. Birmingham, U.K.: Packt Publishing, 2010.
- [6]. Machine Learning: A Probabilistic Perspective: K. P. Murphy, Machine Learning: A Probabilistic Perspective, 2nd ed. Cambridge, MA: MIT Press, 2023.
- [7]. Introduction to Statistical Learning (ISL) :G. James, D. Witten, T. Hastie, and R. Tibshirani, An Introduction to Statistical Learning, 2nd ed. New York, NY: Springer, 2021.
- [8]. C. J. Hutto and E. Gilbert, "VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text," Proc. Int. Conf. Weblogs and Social Media (ICWSM), 2014. [Online]. Available: <https://ojs.aaai.org/index.php/ICWSM/article/view/14550>
- [9]. NLTK Project, "VADER Sentiment Analysis Documentation," NLTK Docs. <https://www.nltk.org/modules/nltk/sentiment/vader.html>