DS Journal of Digital Science and Technology
Volume 3 Issue 4, 41-48, Oct - Dec 2024
ISSN: 2583-5416 / https://doi.org/10.59232/DST-V3I4P105

Review Article

House Rental Application: A MERN Stack-Based Solution for Efficient Property Listings and Management

Santhosh Kumar[1], Sathish[2], Kaarthika[3]

[1,2,3]*Artificial Intelligence and Data Science, Sri Shakthi Institute of Engineering and Technology, Tamilnadu, India.*

[1] santhoshkumars21ads@srishakthi.ac.in

Received: 09 October 2024; Revised: 08 November 2024; Accepted: 02 December 2024; Published: 24 December 2024;

Abstract - The increasing demand for online property listings has led to the development of several web-based applications aimed at simplifying property rental processes. This paper presents a house rental application developed using the MongoDB, Express, React, Node.js (MERN) stack, which facilitates property owners to list their houses while providing tenants with a convenient platform to search for rental properties based on various criteria. The application supports functionality for user authentication, a searchable property database, a Wishlist feature, and the ability to view property owners' contact details. The system uses JSON Web Tokens (JWT) for secure user authentication and Material UI for an intuitive, user-friendly interface. This paper discusses the core features of the application, its architecture, and the technologies employed, focusing on how it contributes to streamlining the house rental process and enhancing user experience.

Keywords - Real estate management, Web application development, Tenant-seller interaction, Dynamic property search, Full-stack development.

1. Introduction

Online property rental platforms have revolutionized how property owners and tenants connect, offering convenient ways to list and search for rental properties. Despite widespread adoption, these platforms often fail to meet user expectations in several critical areas. First, many platforms suffer from cumbersome or overly complex user interfaces, making navigation and property management challenging, especially for non-technical users. Second, the search functionalities in existing solutions are often limited, lacking the ability to filter results based on nuanced preferences such as specific amenities or proximity to essential services. Finally, security concerns, including weak user authentication mechanisms, pose risks to both property owners and tenants, potentially exposing sensitive data to unauthorized access. This paper introduces a house rental application developed using the MongoDB, Express.js, React.js, Node.js (MERN) stack, designed to address these challenges. The application provides an intuitive user interface for property owners to list properties. It offers tenants advanced search options, including location-based filters, number of bedrooms and bathrooms, and desired amenities. A Wishlist feature allows users to save preferred listings for future reference, enhancing convenience and personalization. Additionally, the application implements robust user authentication via JSON Web Tokens (JWT), ensuring a secure platform for all users. Several online rental platforms, such as Airbnb, Zillow, and Realtor.com, have set the benchmark for property listings by providing diverse features tailored to global audiences. Airbnb excels in short-term rental services, offering a wide range of filters and user-generated reviews. Zillow and Realtor.com, primarily catering to long-term rentals and real estate sales, provide extensive property data but often prioritize data quantity over user-friendly design.

However, these platforms exhibit notable limitations. For instance, Airbnb focuses on short-term stays and lacks specific tools for long-term rental seekers. Zillow and Realtor.com, while offering detailed property descriptions, lack personalized features such as wish lists or tailored filtering by niche amenities. Furthermore, many platforms rely on traditional user authentication methods, which are less secure than modern token-based approaches like JWT. By addressing these gaps, the proposed MERN-based house rental application combines the best practices of existing platforms with enhanced usability, security, and personalization features to cater to both property owners and tenants effectively. This application integrates JWT for robust security, Material UI for an intuitive interface, and a modular architecture designed for scalability and customization.

2. Materials and Methods

2.1. Security Implications and Best Practices for JSON Web Tokens (JWT)

2.1.1. Understanding JWT in Authentication

JSON Web Tokens (JWT) is a widely adopted method for securely transmitting information between a client and a server as a JSON object. JWTs are compact, self-contained, and cryptographically signed, making them suitable for authentication in modern web applications. In a house rental application, JWTs verify user identity and manage sessions securely, allowing authenticated users to perform actions such as listing properties, saving properties to their Wishlist, and viewing sensitive data. JWT consists of three parts:

- Header: Specifies the token type (e.g., JWT) and signing algorithm (e.g., HS256 or RS256).
- Payload: Contains claims such as user ID, roles, and expiration time.
- Signature: A cryptographic hash created using the header, payload, and a secret key. This ensures the token's integrity and authenticity.

2.1.2. Security Implications of JWT

While JWTs provide a robust mechanism for authentication, they are not without risks. Key security implications include:

- Token Tampering: Attackers may attempt to alter the payload or header of a JWT. Without proper validation, such tampering could compromise application security.
- Token Theft: If a JWT is intercepted during transmission or stored insecurely, attackers could use it to impersonate the legitimate user.
- Token Replay Attacks: Since JWTs are stateless, a valid token stolen from a user session can be reused by an attacker until it expires.
- Excessive Token Lifetime: Tokens with extended expiration times increase the risk of misuse if compromised.

2.1.3 Best Practices for Securing JWTs

To mitigate these risks, the following best practices should be implemented:

- Use Secure Communication Channels: Always transmit JWTs over HTTPS to protect against Man-in-The-Middle (MITM) attacks. Avoid sending tokens over insecure channels like HTTP or unencrypted WebSocket connections.
- Implement Token Expiration and Refresh Tokens: Set short expiration times for JWTs (exp claim) to minimize the exploitation window in case of token compromise. Pair this with refresh tokens to enable secure, seamless re-authentication.

- Store Tokens Securely: Avoid storing JWTs in insecure storage mechanisms, such as browser local or session storage, as these are susceptible to Cross-Site Scripting (XSS) attacks. Use secure, HTTP-only cookies with the Secure and Same Site attributes enabled.
- Validate Tokens Properly: Verify the token signature using the server's secret or public/private key pair (e.g., RSA) to ensure the token's integrity. Always check the claims, such as exp (expiration), iat (issued at), and aud (audience), to validate their legitimacy.
- Adopt Strong Signing Algorithms: Use secure algorithms, such as RS256 (RSA with SHA-256), instead of less secure options like HS256. RS256 ensures asymmetric encryption, with a private key for signing and a public key for verification.
- Revoke Compromised Tokens: Maintain a token blacklist or invalidate tokens after logout or suspected compromise. This can be achieved by checking the token against a server-stored database of revoked tokens.
- Mitigate Cross-Site Scripting (XSS) Risks: Prevent XSS vulnerabilities in the application to ensure malicious scripts cannot access JWTs stored on the client side.

2.1.4. JWT in the Context of the House Rental Application

In the proposed application, JWT is central to secure user authentication. Key security features include:

- Encrypting tokens with RS256 for enhanced security.
- Using refresh tokens to balance short-lived access tokens with usability.
- Storing tokens in HTTP-only, Secure cookies to prevent XSS exploitation.
- Implementing server-side token invalidation to enhance control over session management.
- By adhering to these best practices, the application ensures robust security, protecting sensitive user data and minimizing risks associated with authentication mechanisms.

2.2. Database Design and Relationships

A well-structured database is critical for efficiently operating any web application, especially in a property rental platform with complex relationships between users, properties, and other entities. The database design for this house rental application employs MongoDB, a NoSQL database, to provide scalability, flexibility, and support for dynamic data schemas. The following schemas are central to the application:

2.2.1. User Schema

- *Purpose:* Stores information about the platform users, including tenants and property owners (sellers).
- *Fields:*
  - Email: Unique identifier for each user.
  - Password: Securely hashed password for authentication.
  - Role: Defines whether the user is a tenant or seller.
  - Wish list: Array storing references to properties marked as favorites by the tenant.
- *Relationships:*
  - A tenant can have multiple properties on their Wishlist.
  - A seller can list multiple properties.

2.2.2. Property Schema

- *Purpose:* Contains the details of each property listed by sellers.
- *Field:*
  - Title: Name or headline for the property listing.
  - Description: Detailed description of the property.
  - Price: Monthly rent or sale price.
  - Bedrooms: Number of bedrooms available.

  - Bathrooms: Number of bathrooms available.
  - Location: Address, including city, state, and postal code.
  - Amenities: List of amenities such as parking, balcony, or Wi-Fi.
  - Owner: Reference to the seller (user) who listed the property.
- *Relationships:*
  - Each property belongs to one seller.
  - Properties can be associated with multiple tenants through their Wishlist.

2.2.3. Wishlist Schema

- *Purpose:* Tracks the properties a tenant has marked for future reference.
- *Fields:*
  - User: Reference to the tenant (user) who created the Wishlist.
  - Property: Reference to the property added to the Wishlist.
- *Relationships:*
  - A tenant can have multiple Wishlist entries, each corresponding to a property.

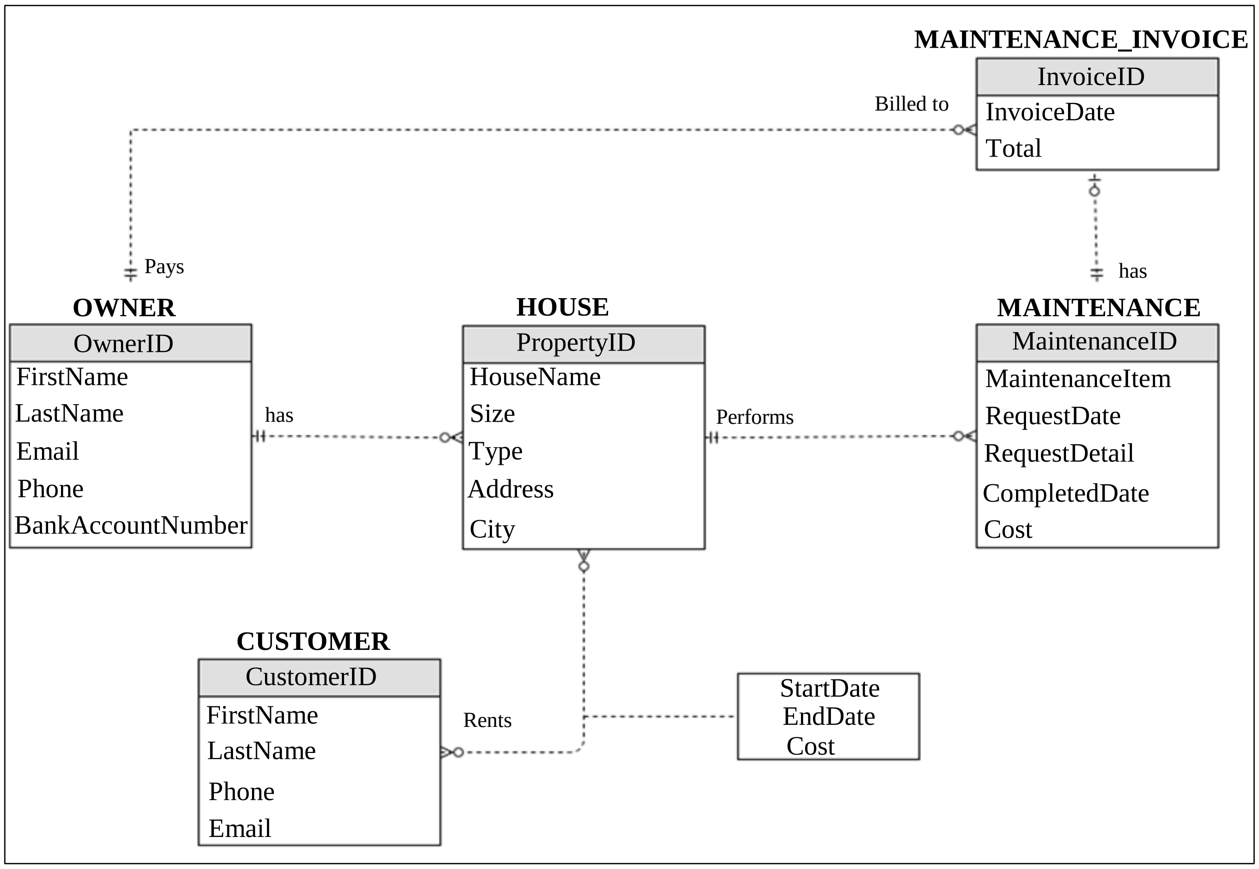

Fig. 1 Database architecture

3. Results and Discussion

The house rental application successfully enables sellers to list properties and tenants to easily search, filter, and add properties to their Wishlist. The key features, such as property listing, advanced search filters, and direct contact with property owners, enhance the user experience. By using JWT for authentication, the application ensures secure access to sensitive features, while Material UI provides a modern and responsive design.

3.1. User Feedback and Usability Testing Results

To ensure that the house rental application meets user needs, we conducted usability testing and gathered feedback from diverse participants, including property owners, prospective tenants, and developers. Below are the key insights and corresponding design improvements:

3.1.1. Feedback on Navigation and Interface

- Issue: Users found the initial navigation unintuitive, especially when switching between listing properties and browsing available rentals.
- Solution: A streamlined navigation bar was implemented with clear labels and categorized sections (e.g., “My Listings,” “Browse Properties,” and “Wishlist”). Dropdown menus and a breadcrumb trail were added to improve discoverability.
- UX Principle Applied: Consistency and Standards – Ensuring interface elements like buttons and navigation links align with common design patterns to reduce cognitive load.

3.1.2. Wishlist Feature Concerns

- Issue: Testers highlighted the difficulty in managing saved properties due to a lack of sorting or filtering options in the Wishlist.
- Solution: Added sorting options (e.g., by location, price, and date added) and implemented filters to refine the Wishlist view.
- UX Principle Applied: Flexibility and Efficiency of Use – Supporting advanced users with sorting and filtering for a more tailored experience.

3.1.3. Search Functionality Limitations

- Issue: Search results often lacked relevance due to inadequate filtering options.
- Solution: Enhanced the search engine with multi-parameter filters (e.g., location, price range, bedrooms, bathrooms, and amenities). Introduced predictive search to suggest relevant terms based on user type.
- UX Principle Applied: User Control and Freedom – Giving users control over refining their searches while providing helpful suggestions.

3.1.4. Security and Authentication Process Feedback

- Issue: Users expressed concerns about their data security during login and property management.
- Solution: Implemented secure login via JWT with HTTPS, added two-factor authentication, and provided clear messaging about the security protocols in use.
- UX Principle Applied: Help Users Recognize, Diagnose, and Recover from Errors – Ensuring users understand what is happening behind the scenes and feel secure while interacting with the application.

3.1.5. Mobile Responsiveness

- Issue: Testers accessing the platform on mobile devices reported challenges with viewing property details and navigating menus.
- Solution: Redesigned the interface using a mobile-first approach, incorporating responsive design techniques such as collapsible menus, touch-friendly controls, and adaptive grids for property listings.
- UX Principle Applied: Responsive Design – Ensuring the application is fully usable on various devices and screen sizes.

3.1.6. Feedback on Visual Design

- Issue: The initial design was perceived as too plain lacking visual hierarchy.
- Solution: Updated the color palette for better contrast, incorporated whitespace to improve readability, and used typography and icons to guide user attention to critical elements

- UX Principle Applied: Aesthetic and Minimalist Design – Maintaining simplicity while emphasizing key elements for a visually appealing interface.

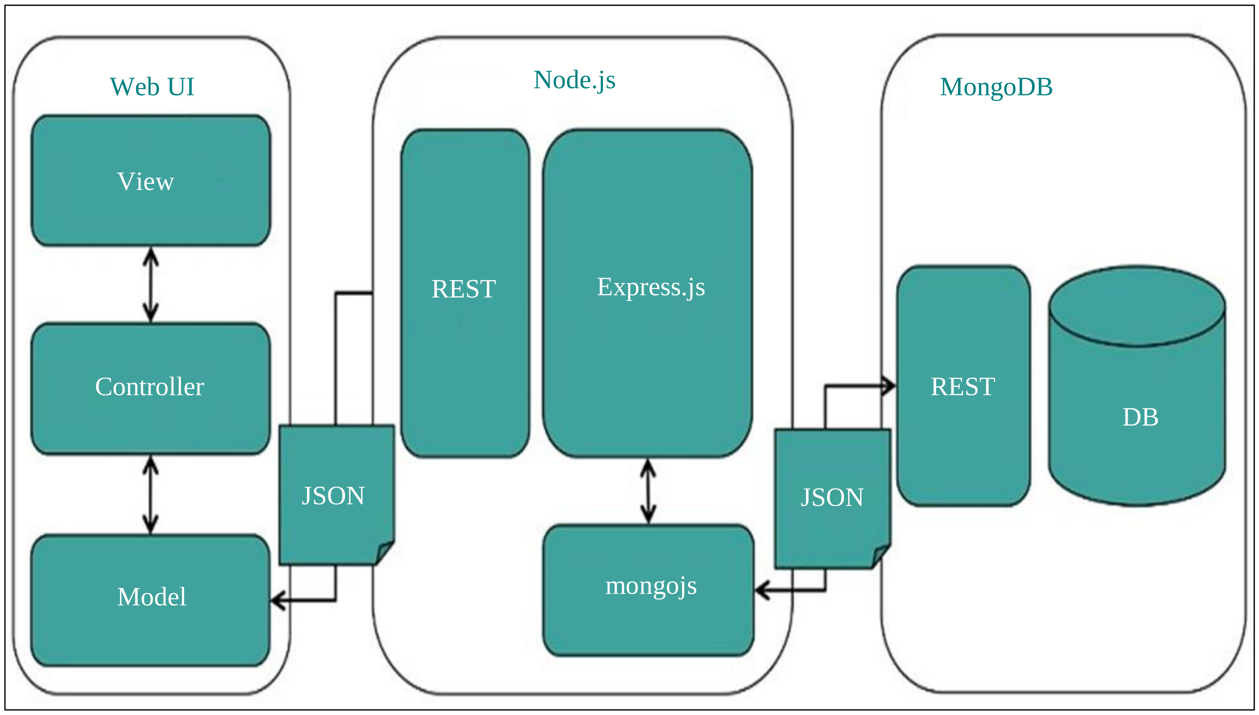

Fig. 2 Architecture of how the application works

3.2. Deployment Practices

3.2.1. Codebase Management

- Version control is implemented using Git, with branches for development, testing, and production.
- Continuous Integration/Continuous Deployment (CI/CD) pipelines are set up to automate testing and deployment.

3.2.2. Containerization

- Docker packages the application into containers, ensuring consistency across environments.
- Multi-container setups include services for the backend (Node.js + Express), database (MongoDB), and frontend (React).

3.2.3. Testing Environments

- Staging servers replicate the production environment for testing and validation before deployment.
- Automated test scripts check for functional and integration issues.

3.2.4. Security

- HTTPS is enforced to secure data transmission.
- Secrets like API keys and database credentials are managed using tools like AWS Secrets Manager or Azure Key Vault.

3.3. Hosting Considerations

3.3.1. Hosting Platforms

- The application is deployed on cloud-based platforms like AWS, Azure, or Google Cloud Platform (GCP).
- MongoDB Atlas manages the database, offering features like auto-scaling and backup.

3.3.2. Load Balancing

- Load balancers distribute traffic across multiple server instances to ensure high availability and responsiveness.

3.3.3. Auto-Scaling

- Auto-scaling mechanisms adjust server capacity based on traffic, ensuring cost-efficiency and consistent performance.

3.3.4. Monitoring and Logging

- Monitoring tools like New Relic or Datadog provide insights into application performance and detect issues.
- Logs are managed using tools like Elasticsearch, Logstash, Kibana (ELK) Stack for troubleshooting and analytics.

3.4. Operational Environments

3.4.1. Development

- Local machines with Node.js, React, and MongoDB setups are used for development.
- Hot-reloading is enabled to ensure fast updates during code changes.

3.4.2. Staging

- Mimics the production environment for QA testing.
- Hosted on cloud environments with restricted access to simulate real-world usage.

3.4.3. Production

- Runs on high-performance instances with global Content Delivery Networks (CDNs) like Cloudflare to reduce latency.
- Backup systems ensure data availability even in cases of failure.

4. Conclusion

The proposed house rental application represents a significant step forward in addressing existing platforms' challenges in the property rental market. By leveraging the MERN stack, the application combines a robust backend with a dynamic and user-friendly frontend, creating an efficient solution for tenants and property owners. Its advanced property search functionality, equipped with precise filters for location, price, amenities, and property features, allows tenants to find properties that closely match their needs, saving time and effort. Additionally, implementing a secure authentication mechanism using JSON Web Tokens (JWT) ensures that sensitive user data and platform features are well-protected, enhancing user trust and platform credibility.

One of the standout features is the Wishlist functionality, which enables tenants to save properties for easy access later. This feature simplifies decision-making and adds a layer of convenience for users managing multiple potential rentals. For property owners, the platform offers seamless property listing and management tools, allowing them to showcase their properties effectively and reach a larger audience. Using Material UI in the frontend design ensures the application is modern, intuitive, and responsive, providing a high-quality experience across devices.

The platform's scalability is another major contribution achieved through a modular architecture and an optimized database design. This ensures that the application can handle increasing volumes of users and listings without performance degradation. Moreover, including planned features such as multilingual support, mobile applications, and push notifications positions the platform as a comprehensive solution for the global rental market. Sellers are further empowered with analytics dashboards and tools to monitor property performance, helping them make informed decisions to attract tenants.

The market impact of this application is substantial, as it bridges critical gaps in accessibility, efficiency, and security within the property rental ecosystem. By improving the experience for tenants and property owners, the platform addresses the inefficiencies of traditional methods and current digital solutions. Future enhancements like integrated messaging, payment processing, and AI-driven property recommendations will further amplify its usability and appeal. As the application evolves, it is poised to become a market leader in real estate technology, setting new benchmarks in functionality, user experience, and scalability.

References

[1] React Documentation. [Online]. Available: https://legacy.reactjs.org/

[2] Node Documentation. [Online]. Available: https://nodejs.org/docs/latest/api/

[3] Express Documentation. [Online]. Available: https://expressjs.com/en/api.html

[4] What is MongoDB?, MongoDB. [Online]. Available: https://www.mongodb.com/docs/manual/

[5] JWT Authentication Guide. [Online]. Available: https://jwt.io/

[6] Rental House Management System: Thesis Documentation Chapter One, Sourcecodetester, 2020. [Online]. Available: https://www.sourcecodester.com/blog/14103/rental-house-management-system-thesis-documentation-chapter-one.html

[7] Wendy Usrey, "*The Rental Next Door: The Impact of Rental Proximity on Home Values*," Thesis, Colorado State University, 2012. [Google Scholar] [Publisher Link]

[8] Alycia Lucio, Rental Application Form, Zillow, 2019. [Online]. Available: https://www.zillow.com/rental-manager/resources/rental-application-form/