

Rentify – Full Stack Project Documentation

1. Abstract :

This document provides a comprehensive overview and technical documentation for the House Rental Web Application. The system is a full-stack project designed to connect property owners and renters, allowing for secure property management, inquiries, and role-based access control. It leverages modern technologies like React, Vite, TailwindCSS, Node.js, Express, MongoDB, JWT, and Cloudinary for image storage.

2. Introduction :

The House Rental Web App addresses the need for a streamlined platform that facilitates communication and transactions between property owners and potential renters. The application provides user authentication, role-based features, property listings, and an inquiry system.

Finding rental properties and managing communications between owners and renters can be inefficient. The House Rental App solves this by providing a single platform where:

- Owners can list and manage properties.
- Renters can view properties, send inquiries, and track responses.
- Authentication and authorization ensure secure data handling.

3. Objectives:

- Provide a secure multi-role platform for Owners and Renters.

- Enable property listing creation, editing, and deletion.
 - Allow renters to view properties and send inquiries.
 - Implement strong authentication and data security.
 - Offer a scalable architecture for deployment.
-
- Build a secure, role-based property rental system.
 - Provide an intuitive frontend for property management.
 - Implement RESTful APIs with JWT-based security.
 - Enable cloud-based deployment for scalability.

4. System Requirements

Functional Requirements:

- User registration/login
- JWT-based authentication
- Property CRUD operations
- Inquiry management
- Role-based access control

Non-Functional Requirements:

- Performance and scalability
- Data security
- Responsive design
- Cloud deployment capability

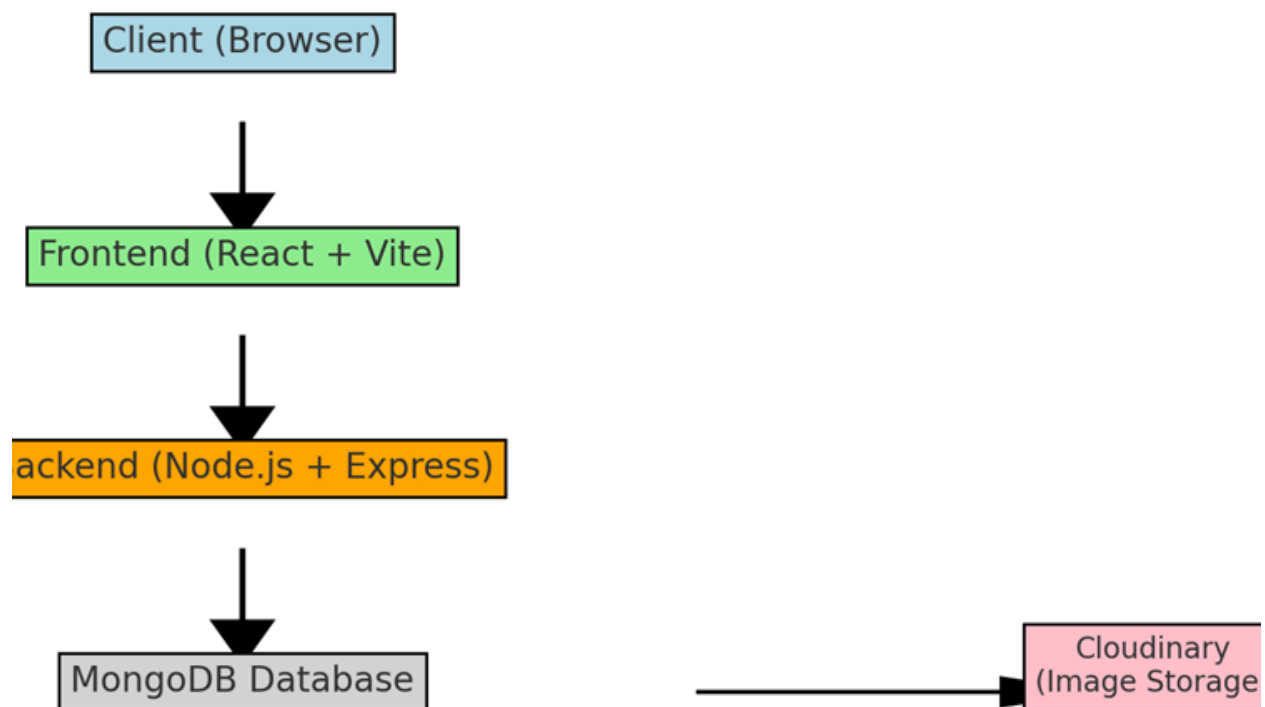
Hardware/Software Requirements

- Node.js >= 18
- MongoDB Atlas
- Vite, React 19
- Cloudinary account
- Deployment on Render

5. System Architecture

The application follows a client-server model with separate frontend and backend layers communicating via RESTful APIs. The backend connects to a MongoDB database and integrates with Cloudinary for image storage.

High-level Architecture Diagram:



6. Module Descriptions

The system consists of the following core modules:

1. Authentication Module – Manages user registration, login, JWT tokens, and role-based access.
2. Property Management Module – Allows owners to add, edit, delete, and view property listings.

- 3. Inquiry Management Module – Enables renters to send inquiries and owners to manage responses.
- 4. UI Module – Provides frontend components for dashboards, forms, and listings.

7. Database Design

The project uses **MongoDB**, a NoSQL database, which stores data in flexible, JSON-like documents. Its schema-less nature allows for scalability and easy modifications. The database is organized into several key **collections**:

1. Users Collection

Purpose: Stores information about renters, property owners, and admins.

Fields:

- `_id` (ObjectId): Unique identifier.
- `name` (String): Full name of the user.
- `email` (String, unique): Used for login.
- `password` (String): Bcrypt-hashed password.
- `role` (String): Defines user type – "owner", "renter", or "admin".
- `createdAt` / `updatedAt` (Date): Track account creation and modifications.
- `refreshToken` (String, optional): Stores the latest refresh token for session management.

2. Properties Collection

Purpose: Holds property listing details created by owners.

Fields:

- `_id` (ObjectId): Unique property ID.
- `ownerId` (ObjectId, reference to Users): Identifies the property owner.
- `title` (String): Property name.
- `description` (String): Detailed property information.
- `location` (String): City/area address.
- `price` (Number): Rental price per month.
- `images` (Array of Strings): Cloudinary URLs.
- `availabilityStatus` (Boolean): Indicates if the property is currently available.
- `createdAt` / `updatedAt` (Date).

3. Inquiries Collection

Purpose: Stores messages or booking requests from renters to owners.

Fields:

- `_id` (ObjectId).
- `propertyId` (ObjectId, reference to Properties): Links inquiry to a property.
- `renterId` (ObjectId, reference to Users): The renter making the inquiry.
- `message` (String): User-submitted message.
- `status` (String): "pending", "approved", or "rejected".
- `createdAt` (Date).

4. Optional Collections (Future Enhancements)

- **Transactions:** For storing payment details.
- **Notifications:** For push notification history.
- **Admin Logs:** For system activity tracking.

Relationships

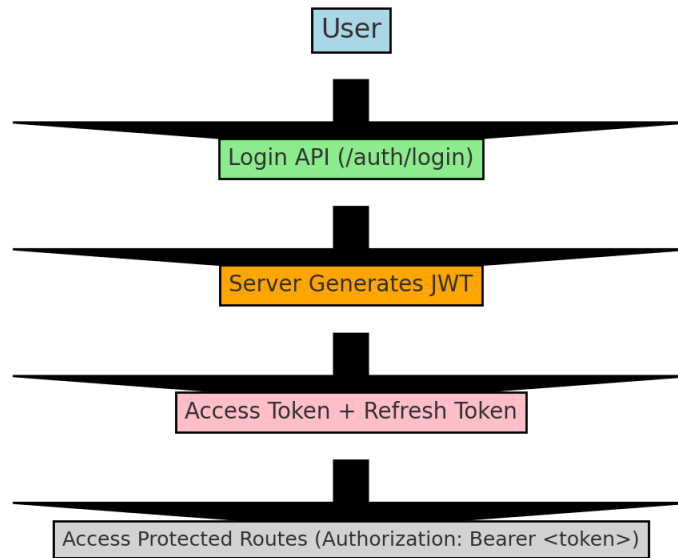
- **One-to-Many:**
 - One owner can have many properties.
 - One property can have many inquiries.
- **Many-to-One:**
 - Many inquiries link back to one renter.

8. API Design

Key API endpoints include:

- POST /auth/register
- POST /auth/login
- GET /properties
- POST /properties
- PUT /properties/:id
- DELETE /properties/:id
- POST /inquiries
- GET /inquiries/:ownerId

9. Security – JWT Flow



The system uses JWTs for authentication. Access tokens expire quickly, while refresh tokens are used to request new access tokens. Passwords are hashed using bcrypt.

10. API Endpoints

- POST /auth/register
- POST /auth/login
- GET /properties
- POST /properties
- PUT /properties/:id
- DELETE /properties/:id
- POST /inquiries
- GET /inquiries/:ownerId

11. Deployment

The frontend is built with Vite and deployed to Render/Netlify. The backend is deployed to Render, and MongoDB Atlas is used for the database. Backend Deployment

Steps:

1. **Push code to GitHub** (make sure package.json, .env.example, and backend files are included).
2. **Create a MongoDB Atlas cluster:**
 - Sign up at [MongoDB Atlas](#).
 - Create a cluster, get the **connection string**, and add your database name.
3. **Set environment variables** (in Render/Railway dashboard):
 - MONGO_URI
 - JWT_SECRET
 - CLOUDINARY_URL (if using Cloudinary)
4. **Deploy backend:**
 - In Render: *New Web Service* → *Connect GitHub* → *Select repo* → *Build Command*: npm install, *Start Command*: npm start.
 - Expose API at https://your-backend.onrender.com.
5. **Verify:**
 - Test APIs using Postman.

2. Frontend Deployment (Netlify or Vercel)

Steps:

1. **Build frontend:**
2. `bash`
3. `CopyEdit`
4. `cd frontend`
`npm install`
`npm run build`
5. This generates a `dist/` folder.
6. **Deploy to Netlify:**
 - Drag and drop `dist/` to Netlify, or connect GitHub repo.
7. **Set environment variables:**
 - Example: `VITE_API_URL=https://your-backend.onrender.com`.
8. **Update `vite.config.js`** to proxy or point to backend.
9. **Verify:**
 - Visit the Netlify URL, check API calls.

3. Domain & SSL

- Use a custom domain via Netlify/Vercel.
- Both Netlify and Render provide automatic HTTPS.

12. Testing

Testing includes unit tests for backend APIs (via Jest/Postman) and frontend tests using React Testing Library.

14. Security Implementation

The House Rental Web App must enforce strong security measures to protect user data, maintain trust, and prevent unauthorized access.

1. Authentication and Authorization

- Use **JWT (JSON Web Tokens)** for secure session handling.
- Include **access tokens** with short lifetimes and **refresh tokens** to obtain new access tokens without re-login.
- Apply **role-based access control (RBAC)** to ensure owners, renters, and admins can only access permitted features.

2. Password Protection

- Store passwords using **bcrypt hashing** with a strong salt factor (e.g., 10–12).
- Never store plain-text passwords in the database.

3. Data Validation & Sanitization

- Validate all incoming data on both frontend and backend.
- Sanitize inputs to prevent **NoSQL injection**, **cross-site scripting (XSS)**, and malformed requests.

4. HTTPS Everywhere

- Enforce HTTPS for all communications to protect data in

transit.

- Use TLS certificates (Netlify/Render provide free SSL).

5. **CORS Configuration**

- Restrict allowed domains for API calls.
- Prevent unauthorized third-party access.

6. **Secure File Handling**

- Limit file types and sizes for property images.
- Scan files to prevent malicious uploads.

7. **Rate Limiting & Brute-Force Protection**

- Limit login attempts to prevent automated credential stuffing.
- Use libraries like express-rate-limit.

8. **Environment Variable Protection**

- Store secrets (JWT keys, DB URIs, API keys) in .env files.
- Never commit sensitive information to GitHub.

9. **Database Security**

- Use MongoDB Atlas IP whitelisting.
- Enable user authentication for database access.

10. **Logging & Monitoring**

- Log suspicious activities (e.g., failed logins).
- Use monitoring tools to detect anomalies

The House Rental Web App incorporates multiple layers of security to ensure data protection, user privacy, and safe application operation. Authentication is managed using JSON Web Tokens (JWT), where access tokens are short-lived to minimize risk, and refresh tokens are used to issue new access tokens without requiring repeated logins. Role-based access control (RBAC) restricts actions based on whether a user is a renter, property owner, or administrator, ensuring that only authorized users can perform sensitive operations like property modifications or data management. Password security is enforced by hashing all stored passwords using bcrypt with a strong salt factor, preventing the storage of plain-text credentials and reducing vulnerability to data breaches.

Input validation and sanitization occur on both the frontend and backend to prevent NoSQL injection, cross-site scripting (XSS), and malformed data attacks. All communications are secured over HTTPS using TLS certificates, protecting data in transit from interception or tampering. Cross-Origin Resource Sharing (CORS) policies are configured to allow API access only from trusted domains. File uploads, such as property images, are restricted by type and size, and validation ensures that malicious files cannot be stored on the server or Cloudinary.

To prevent brute-force login attempts and denial-of-service attacks, rate limiting is implemented on authentication endpoints. Environment variables, including database credentials, JWT secrets, and API keys, are stored securely in .env files and never exposed in the code repository. The database itself is secured using MongoDB Atlas features like IP whitelisting and mandatory authentication. Finally, activity logging and monitoring tools help track failed login attempts, suspicious requests, and other anomalies, enabling timely detection and mitigation of potential security threats.

15. Conclusion


This documentation describes a robust, scalable, and secure full-stack House Rental Web App. It can be deployed to cloud platforms like Render, with future enhancements including payment integration, chat features, and advanced search filters. House Rental Web App represents a comprehensive solution designed to address the challenges of property management and rental interactions in a secure, scalable, and user-friendly manner. By combining a modern frontend built with React and Vite, a robust backend developed using Node.js and Express, and a reliable

database powered by MongoDB, the project achieves a balance between performance and maintainability. Its modular structure—covering authentication, property management, inquiry handling, and image storage—ensures that each component can evolve independently, allowing for future scalability and easier feature integration.

Security considerations, including JWT-based authentication, bcrypt password hashing, and role-based access control, demonstrate a strong focus on safeguarding user data and preventing unauthorized access.

Additionally, implementing responsive design with TailwindCSS ensures the application works seamlessly across devices, while deployment strategies using services like Render, Netlify, and MongoDB Atlas provide a stable and cloud-ready environment.


The app is not only functional in its current state but also designed with extensibility in mind. Future improvements such as payment integration, AI-driven recommendations, real-time communication, and mobile application support can further enhance its value. Overall, this project provides a strong foundation for a professional-grade platform capable of serving both local and large-scale rental markets.



[Home](#) [Login](#) [Register](#)


Create Your Account

Select Role


 Renter

Register


Already have an account? [Sign In](#)



[Home](#) [Login](#) [Register](#)



Cozy Living Room Retreat

 Kolkata


₹20000

per month


Description

"Charming historic cottage featuring original woodwork and a cozy fireplace."

Property Features


 Bedrooms

3


 Bathrooms

2


Additional Info

 Monthly Rent

₹20000

 Property Type

Residential



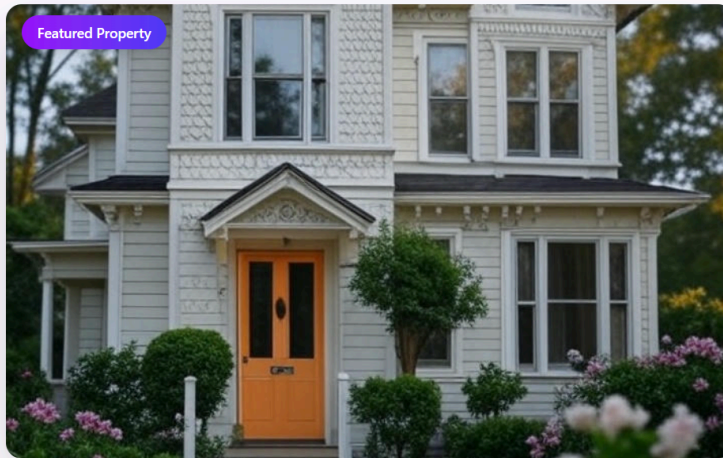
Interested?

Send an inquiry to the owner

Your Message

Hi, I'm interested in this property...

Send Inquiry

[Home](#) > [Properties](#) > [Property Details](#)


Featured Property



Interested?

Send an inquiry to the owner

Your Message

Hi, I'm interested in this property...

[Send Inquiry](#)

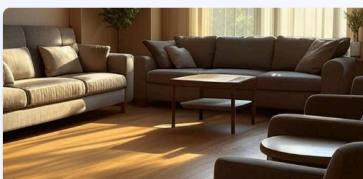
Cozy Living Room Retreat

[Kolkata](#)

₹20000
per month

Available Properties for Rent

Find your perfect home today



3BHK near Greater Noida
Delhi

[View Details](#)



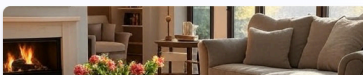
Serene Master Bedroom Haven
Santacruz

[View Details](#)



Cozy Living Room Retreat
Kolkata

[View Details](#)



Login to RentiFi

Email Address

Password

Log In

Don't have an account? [Register](#)