HouseHunt (RentEase)

A Full-Stack Property Rental Management Platform

Project Report

Technology Stack: MERN (MongoDB, Express.js, React.js, Node.js)

Date: February 2026

Table of Contents

1. Introduction

HouseHunt (RentEase) is a comprehensive, full-stack web application designed to streamline the process of renting and managing residential properties. The platform bridges the gap between property owners seeking tenants and renters looking for suitable accommodation, all within a single, intuitive digital ecosystem.

In the modern real estate landscape, finding the right rental property or managing multiple listings can be a time-consuming and fragmented process. HouseHunt solves this problem by providing a centralized platform where owners can list properties with rich media (images, videos, amenity details), renters can browse, filter, and book properties, and administrators can oversee the entire ecosystem.

The application supports three distinct user roles — Admin, Property Owner, and Renter (Tenant) — each with a tailored dashboard and feature set. Key capabilities include property listing and management, advanced search and filtering, booking management, integrated rent payment processing, real-time chat between owners and renters, a notification system, reviews and ratings, analytics dashboards, and comprehensive admin oversight.

Built using the MERN stack (MongoDB, Express.js, React.js, Node.js), the application follows a RESTful API architecture, ensuring scalable, maintainable, and responsive user experience across devices.

2. Project Objectives

The primary objectives of the HouseHunt project are:

1. **Simplify Property Discovery:** Enable renters to effortlessly browse, search, and filter rental properties using advanced criteria such as price range, property type, furnishing status, BHK configuration, and location.
2. **Empower Property Owners:** Provide owners with a comprehensive dashboard to list properties with rich media (multiple images, videos), manage bookings, track property views, monitor revenue, and communicate with tenants.
3. **Streamline Booking Workflow:** Implement an end-to-end booking lifecycle — from request submission by renters, to acceptance/rejection by owners, to payment processing and cancellation support.
4. **Enable Secure Payments:** Integrate a rent payment system allowing tenants to pay rent for booked properties, with order creation, verification, and payment history tracking.
5. **Facilitate Communication:** Build an in-app real-time messaging system that allows renters and owners to communicate directly, with unread message tracking and conversation management.
6. **Provide Feedback Mechanism:** Allow tenants to rate and review properties after booking, helping future renters make informed decisions and owners to improve service quality.
7. **Deliver Actionable Analytics:** Offer owners a rich analytics dashboard with booking trends, revenue breakdown, property performance metrics, and growth indicators.

8. **Ensure Administrative Control:** Equip administrators with tools to manage all users, approve/revoke owner access, monitor all properties and bookings across the platform.
9. **Build a Notification System:** Keep all users informed of important events (new bookings, status changes, payments, reviews) through an in-app notification center with real-time polling.
10. **Adopt Modern Architecture:** Build a scalable, maintainable application using MERN stack with clean separation of concerns, RESTful API design, JWT-based authentication, and modular component structure.

3. Technology Stack

3.1 Frontend

Technology	Version	Purpose
React.js	18.2.0	UI library for building component-based single-page application
React Router DOM	6.15.0	Client-side routing and navigation
Material UI (MUI)	5.14.5	Pre-built UI components (Tables, Tabs, Modals, Buttons, TextFields)
MUI Joy	5.0.0-beta.2	Extended component library for enhanced UI
MUI Icons Material	5.14.3	Icon library for visual elements
Ant Design	5.8.3	Additional UI components (Carousel, message alerts)
Bootstrap / React-Bootstrap	5.3.1 / 2.8.0	Responsive grid layout, modals, forms
Axios	1.4.0	HTTP client for API communication
Lucide React	0.563.0	Modern icon set for UI elements
XLSX	0.18.5	Excel file generation for data export
File Saver	2.0.5	Client-side file download utility

3.2 Backend

Technology	Version	Purpose
Node.js	Latest LTS	JavaScript runtime for server-side logic
Express.js	4.18.2	Web framework for building RESTful APIs
MongoDB	Latest	NoSQL database for flexible data storage

Technology	Version	Purpose
Mongoose	7.4.3	MongoDB ODM for schema modeling and validation
JSON Web Token (JWT)	9.0.1	Token-based stateless authentication
bcryptjs	2.4.3	Password hashing and comparison
Multer	1.4.5	Multipart form handling for image/video uploads
CORS	2.8.5	Cross-Origin Resource Sharing configuration
dotenv	16.3.1	Environment variable management
Nodemon	3.0.1	Development auto-restart on file changes
Razorpay SDK	2.9.6	Payment gateway integration (order and verification)

3.3 Development Tools

Tool	Purpose
VS Code	Primary code editor / IDE
Postman	API testing and debugging
MongoDB Compass	Database visualization and management
Git	Version control
npm	Package management
Browser DevTools	Frontend debugging and inspection

4. System Architecture

HouseHunt follows a three-tier client-server architecture with clear separation between the presentation layer (React frontend), business logic layer (Express.js backend), and data layer (MongoDB database).

4.1 Architecture Layers

Layer	Component	Responsibility
Presentation Layer	React.js Frontend	Renders UI components, handles user interactions, manages client-side routing, communicates with backend via Axios HTTP calls. Uses Context API for global state (user session).

Layer	Component	Responsibility
API / Business Layer	Express.js REST API	Defines route endpoints, applies authentication middleware (JWT), executes business logic in controllers, validates input, handles file uploads via Multer, and returns JSON responses.
Data Access Layer	Mongoose ODM	Defines data schemas with validation rules, provides an abstraction over MongoDB operations (CRUD), enforces data integrity with model constraints, and supports indexing.
Database Layer	MongoDB	Stores all application data in flexible JSON-like documents across collections: users, properties, bookings, chats, notifications, payments, reviews, recent views.
Static File Server	Express Static Middleware	Serves uploaded property images and videos from the /uploads directory via static file serving.

4.2 Authentication Flow

11. User submits login credentials (email + password) to POST /api/user/login.
12. Server validates credentials using bcryptjs password comparison.
13. On success, server generates a JWT token (signed with JWT_SECRET, 1-day expiry) and returns it with user data.
14. Client stores the token and user data in localStorage.
15. For protected API calls, client sends the token in the Authorization: Bearer <token> header.
16. Auth middleware (authMiddleware.js) intercepts protected routes, verifies the JWT, extracts the user ID, and injects it into req.body.userId.
17. If the token is invalid or missing, the request is rejected with a 401 status.

4.3 Project Directory Structure

Directory	Description
code/backend/	Node.js server application root
code/backend/config/	Database connection configuration
code/backend/controllers/	Business logic handlers for each module (7 controllers)
code/backend/middlewares/	JWT authentication middleware
code/backend/routes/	API endpoint definitions (7 route files)
code/backend/schemas/	Mongoose data models (8 schemas)
code/backend/uploads/	Static file storage for uploaded images/videos

Directory	Description
code/frontend/src/	React application source root
code/frontend/src/modules/common/	Shared components (Home, Login, Register, Chat, Notifications)
code/frontend/src/modules/admin/	Admin dashboard components (4 components)
code/frontend/src/modules/user/Owner/	Owner dashboard components (8 components)
code/frontend/src/modules/user/renter/	Renter dashboard components (3 components)

5. Features Implemented

5.1 Authentication & Authorization

- **User Registration:** Support for three roles — Admin, Owner, and Renter. Passwords are securely hashed using bcryptjs with salt rounds.
- **Login with JWT:** Stateless token-based authentication with 1-day token expiry. Role-based redirection to Admin, Owner, or Renter dashboard.
- **Forgot Password:** Email-based password reset with new password confirmation and hash update.
- **Auth Middleware:** Protects all sensitive API routes by validating JWT tokens and extracting user identity.
- **Owner Access Control:** Admin must grant permission before owners can access their dashboard (granted/ungranted status).

5.2 Property Management (Owner)

- **Add Property:** Rich property listing form with property type, listing type (Rent/Sell/Lease), furnished status, address, area, bedrooms, bathrooms, contact, price, and additional information.
- **Multi-Image Upload:** Upload up to 10 property images with inline preview and removal. Uses Multer for server-side handling.
- **Video Upload:** Supports property video upload (mp4, mov, webm, avi, mkv) up to 100MB.
- **Amenities Selection:** Toggle-based selection of amenities (WiFi, AC, Gym, Parking, Pool, Security, Power Backup, etc.).
- **Edit & Delete Properties:** Inline editing via modal with all property fields. Delete with confirmation prompt.
- **Property Availability:** Automatic availability status update when bookings are accepted or revoked.

5.3 Property Discovery (Renter)

- **Enhanced Property Cards:** Visually rich property cards displaying cover image, price, location, bed/bath count, amenities, ratings, and view count.
- **Advanced Filtering:** Filter by text search (address), property type, ad type, minimum/maximum price, BHK configuration, and furnished status with clear-all and result count.
- **Property Detail Modal:** Full property details with image carousel, specifications, amenity list, and booking form.
- **Favorites:** Heart toggle to mark favorite properties, persisted in browser localStorage.
- **View Tracking:** Automatic property view count increment when renters view a property. Recently viewed properties list.

5.4 Booking Management

- **Booking Request:** Renters submit booking requests with their name and phone number. Dual notifications sent to both owner and renter.
- **Booking Lifecycle:** Owners can Accept (status -> booked) or Revoke (status -> revoked) bookings. Property availability updates automatically.
- **Cancel Booking:** Renters can cancel pending or booked bookings. Owner is notified of cancellation.
- **Booking Dashboard:** Both owners and renters see booking statistics — total, active, completed, and cancelled counts.

5.5 Payment System

- **Order Creation:** Server creates a payment order with the property rent amount, verifying booking ownership and preventing duplicate payments.
- **Payment Verification:** Simulated Razorpay-style flow with order creation, payment ID generation, and status update to paid.
- **Payment History:** Both payer and owner can see completed payment records with transaction ID, amount, date, and property details.
- **Payment Status Check:** API to verify if rent has been paid for a specific booking.
- **Dual Notifications:** Owner notified of payment received; renter notified of successful payment with transaction ID.

5.6 In-App Chat / Messaging

- **Direct Messaging:** Renters and owners can chat directly from the booking screen. Messages stored in MongoDB with sender, receiver, and property reference.
- **Conversation List:** Sidebar showing all conversations with user avatar initials, last message preview, timestamp, and unread badge count.
- **Message Interface:** Chat bubbles with sent/received distinction, timestamps, and read receipt indicators.
- **Real-time Polling:** Messages refresh every 5 seconds. Unread count updates via periodic API polling.

- **Chat Notifications:** Receiver gets an in-app notification for every new message.

5.7 Notification System

- **Bell Icon with Badge:** Persistent notification bell showing unread count across all pages.
- **Event Types:** Notifications for bookings, booking status changes, property listings, payments, and reviews.
- **Notification Center Modal:** Full list of notifications with title, message, and relative timestamps.
- **Mark Read / Delete:** Individual mark-as-read, mark-all-as-read, and individual delete functionality.
- **Polling:** Notifications refresh every 30 seconds automatically.

5.8 Reviews & Ratings

- **Submit Review:** Tenants can rate properties (1-5 stars) and write text reviews after booking. Duplicate reviews prevented per user-property pair.
- **Average Rating:** Property's average rating is automatically recalculated and stored on the property document.
- **Owner Reviews Dashboard:** Owners see all reviews with summary stats, average rating, total count, and rating distribution.
- **Search & Filter:** Search reviews by tenant name or content. Filter by star rating.
- **Owner Notification:** Owners receive a notification when a new review is submitted.

5.9 Owner Analytics Dashboard

- **KPI Cards:** Total Revenue, Total Bookings, Active Properties, Average Booking Value — each with growth percentage indicators.
- **Monthly Booking Trends:** Horizontal progress bars visualizing booking distribution across months.
- **Revenue Breakdown:** Property sales vs. rental income distribution.
- **Top Performing Properties:** Ranked table of top 5 properties by booking count with views, revenue, and status.

5.10 Admin Panel

- **User Management:** View all registered users with ability to Grant or Ungrant owner access.
- **Property Oversight:** Read-only view of all properties across all owners with key details.
- **Booking Oversight:** System-wide view of all bookings with color-coded status badges.
- **Tab Navigation:** Clean tab-based interface switching between Users, Properties, and Bookings management.

5.11 Data Export

- **Excel Export:** Booked Houses page allows exporting data to .xlsx file with data sheet and summary sheet using the SheetJS (xlsx) library and FileSaver.

5.12 Additional Features

- **Toast Notifications:** Custom toast notification system for real-time user feedback on actions.
- **Responsive Design:** UI built with Bootstrap grid and MUI components for cross-device compatibility.
- **Mock Data Fallback:** Backend gracefully handles MongoDB disconnection by serving mock data for development.
- **Static File Serving:** Uploaded images and videos served via Express static middleware at /uploads path.

6. Database Design

The application uses MongoDB, a NoSQL document database, with Mongoose ODM for schema definition and validation. The database consists of 8 collections, each representing a core entity.

6.1 Entity-Relationship Overview

The data model centers around Users who can be Admins, Owners, or Renters. Owners create Properties, which Renters book via Bookings. Bookings trigger Payments. Renters leave Reviews on properties. All users exchange Chat messages and receive Notifications. Recent Views track renter browsing history.

6.2 Collection Schemas

6.2.1 Users Collection (users)

Field	Type	Description
_id	ObjectId	Auto-generated unique identifier
name	String (required)	User's full name (auto-capitalized)
email	String (required)	User's email address (unique identifier for login)
password	String (required)	bcrypt-hashed password
type	String (required)	User role: Admin, Owner, or Renter
granted	String	Owner access status: granted or ungranted

6.2.2 Properties Collection (propertyschemas)

Field	Type	Description
_id	ObjectId	Auto-generated unique identifier
ownerId	ObjectId (ref: user)	Reference to the property owner
propertyType	String (required)	Type: Apartment, House, Villa, PG, etc.
propertyAdType	String (required)	Listing type: Rent, Sell, Lease
propertyAddress	String (required)	Full address of the property
ownerContact	Number (required)	Owner's contact number
propertyAmt	Number	Monthly rent / price amount
propertyImage	Object (Array)	Array of image objects {filename, path}
propertyVideo	Object	Video object {filename, path, originalName, size, mimetype}
additionalInfo	String	Extra property description
ownerName	String	Name of the owner
bedrooms	Number (default: 1)	Number of bedrooms
bathrooms	Number (default: 1)	Number of bathrooms
area	String	Property area (sq. ft.)
parking	Boolean (default: false)	Parking availability
furnished	String (enum)	unfurnished / semi-furnished / fully-furnished
amenities	String (JSON)	JSON-encoded array of amenity names
views	Number (default: 0)	Total view count
rating	Number (default: 0)	Average rating from reviews
isFeatured	Boolean	Featured listing flag
virtualTourUrl	String	URL for virtual property tour

6.2.3 Bookings Collection (bookingschemas)

Field	Type	Description
_id	ObjectId	Auto-generated unique identifier
propertyId	ObjectId (ref: propertyschema)	Reference to the booked property
ownerID	ObjectId (ref: user)	Reference to the property owner
userID	ObjectId (ref: user)	Reference to the renter/tenant

Field	Type	Description
userName	String (required)	Tenant's name
phone	Number (required)	Tenant's contact number
bookingStatus	String (required)	Status: pending, booked, cancelled, revoked
createdAt	Date (auto)	Booking creation timestamp
updatedAt	Date (auto)	Last update timestamp

6.2.4 Chats Collection (chats)

Field	Type	Description
_id	ObjectId	Auto-generated unique identifier
senderId	ObjectId (ref: user, required)	Message sender
receiverId	ObjectId (ref: user, required)	Message recipient
propertyId	ObjectId (ref: propertyschema)	Optional property reference for context
message	String (required)	Message content
isRead	Boolean (default: false)	Read status indicator
createdAt	Date (auto)	Message timestamp

6.2.5 Notifications Collection (notificationschemas)

Field	Type	Description
_id	ObjectId	Auto-generated unique identifier
userId	ObjectId (ref: user, required)	Notification recipient
type	String (enum, required)	booking, property, booking_status, review
title	String (required)	Notification title
message	String (required)	Notification body text
isRead	Boolean (default: false)	Read status
relatedId	ObjectId	Reference to related entity (booking, review, etc.)
createdAt	Date (auto)	Notification timestamp

6.2.6 Payments Collection (paymentschemas)

Field	Type	Description
_id	ObjectId	Auto-generated unique identifier

Field	Type	Description
bookingId	ObjectId (ref: bookingschema, required)	Associated booking
propertyId	String (required)	Associated property ID
payerId	ObjectId (ref: user, required)	Tenant who paid
ownerId	ObjectId (ref: user, required)	Property owner receiving payment
amount	Number (required)	Payment amount in INR
currency	String (default: INR)	Payment currency
razorpayOrderId	String (required)	Generated order identifier
razorpayPaymentId	String	Verified payment identifier
status	String (enum)	created, paid, failed
payerName	String	Tenant name for display
propertyAddress	String	Property address for display
createdAt	Date (auto)	Payment timestamp

6.2.7 Reviews Collection (reviewschemas)

Field	Type	Description
_id	ObjectId	Auto-generated unique identifier
propertyId	ObjectId (ref: propertyschema, required)	Reviewed property
ownerId	ObjectId (ref: user, required)	Property owner
userId	ObjectId (ref: user, required)	Reviewer (tenant)
userName	String (required)	Reviewer's display name
bookingId	ObjectId (ref: bookingschema)	Associated booking
rating	Number (1-5, required)	Star rating
review	String (required)	Review text content
propertyAddress	String	Property address for display
propertyType	String	Property type for display
createdAt	Date (auto)	Review timestamp

6.2.8 Recent Views Collection (recentviews)

Field	Type	Description
_id	ObjectId	Auto-generated unique identifier

Field	Type	Description
userId	ObjectId (ref: user, required)	Viewing user
propertyId	ObjectId (ref: propertyschema, required)	Viewed property
viewedAt	Date	Last viewed timestamp

Note: Compound unique index on (userId, propertyId) ensures one record per user-property pair (upsert pattern).

7. User Interface Screens

Below is a description of the major application screens organized by user role.

7.1 Public Pages

Screen 1: Home / Landing Page

The landing page showcases the HouseHunt (RentEase) brand with a navigation bar containing Home, Login, and Register links. A property image carousel rotates through featured property images. Below, the EnhancedPropertyCards component displays all available properties in a visually rich card grid with cover images, prices, locations, bed/bath counts, amenities, and rating badges. The page includes a call-to-action section encouraging users to register as property owners, and a footer with copyright information.

Screen 2: Login Page

A clean authentication form with email and password text fields built using Material UI components. The form validates credentials against the backend and redirects users to their respective dashboards based on role (Admin -> AdminHome, Owner -> OwnerHome, Renter -> RenterHome). Includes links to Forgot Password and Register pages. Ungranted owners see an appropriate error message.

Screen 3: Registration Page

A user registration form with fields for Name, Email, Password, and a User Type dropdown (Renter/Owner). Submits to the backend API which hashes the password and creates the user account. Navigates to login page on success.

Screen 4: Forgot Password Page

A simple password reset interface with Email, New Password, and Confirm Password fields. Validates that passwords match before submitting. Updates the password hash in the database and redirects to login.

7.2 Admin Dashboard

Screen 5: Admin Home — All Users Tab

A tabbed interface with All Users, All Properties, and All Bookings tabs. The Users tab displays a comprehensive MUI table showing user ID, name, email, type, and granted status. For Owner-type users, Grant/Ungrant action buttons allow the admin to control platform access.

Screen 6: Admin — All Properties Tab

A read-only data table showing all properties across the platform with columns for Property ID, Owner ID, Property Type, Ad Type, Address, Contact, and Amount.

Screen 7: Admin — All Bookings Tab

A read-only table of all system bookings with color-coded status badges (green for booked, orange for pending, red for cancelled). Columns include Booking ID, Owner ID, Property ID, Tenant ID, Tenant Name, Contact, and Status.

7.3 Owner Dashboard

Screen 8: Owner Dashboard Overview

An overview page with stat cards showing Total Properties, Active Bookings, Total Revenue, and Property Views. Features a property performance table (top 5 properties), recent bookings sidebar with status badges, payment history table, and quick-action buttons for navigation.

Screen 9: Add Property Form

A multi-section form with: property type dropdown, listing type selection, furnished status, address input, area specification, bedroom and bathroom counts, contact number, price, amenity toggle badges (WiFi, AC, Gym, Pool, etc.), multi-image upload with previews (10 max), video upload, and a submit button with client-side validation.

Screen 10: Owner — All Properties

A table listing all of the owner's properties with Type, Ad Type, Address, Contact, Amount, and Availability status. Each row has Edit and Delete action buttons. Edit opens a modal with editable fields; Delete shows a confirmation prompt.

Screen 11: Owner — All Bookings

A table of booking requests for the owner's properties with Accept and Revoke buttons for managing booking status. Status badges indicate current state (pending, booked, cancelled).

Screen 12: Owner — Booked Houses

A detailed view of currently booked properties with summary cards (Total Booked, Total Tenants, Monthly Revenue), a search bar for filtering, and an Export to Excel button. The table shows tenant name, phone, property type, address, rent, bedrooms, furnished status, status, and booking date.

Screen 13: Owner — Reviews

A reviews management page with overview cards (Average Rating with star display, Total Reviews, Rating Distribution progress bars). Below, review cards show reviewer avatar, name, date, star rating, review text, and property information. Includes search and star-rating filter.

Screen 14: Owner — Analytics

An analytics dashboard with KPI cards (Total Revenue, Total Bookings, Active Properties, Avg Booking Value with growth indicators), monthly booking trend bars, revenue breakdown, and a top 5 performing properties table.

7.4 Renter Dashboard

Screen 15: Renter Dashboard Overview

A stat summary page showing Total Bookings, Active Bookings, Completed, and Cancelled counts with recent booking cards. Each card shows property details and status with a Cancel Booking option for active bookings.

Screen 16: Browse Properties

The main property discovery page with an advanced filter panel (text search, property type, ad type, price range, BHK, furnished status) and a grid of enhanced property cards. Each card features a cover image, favorite heart toggle, view count, rating badge, amenities pills, price, and location. Clicking a card opens a detail modal with image carousel, full specs, and booking form.

Screen 17: My Bookings

A table of the renter's bookings with columns for Booking ID, Property ID, Name, Phone, Status, Date, and Actions. Actions include Cancel, Pay Rent (opens payment modal), Write Review (opens star-rating modal), and Chat with Owner.

7.5 Shared Components

Screen 18: Chat / Messaging Interface

A split-panel messaging interface. Left panel shows a conversation list with avatar initials, user name, last message preview, timestamp, and unread badge. Right panel shows the active chat with message bubbles (sent/received), timestamps, read receipts, and a message input bar.

Screen 19: Notification Center

A bell icon with unread count badge (visible in the navigation bar). Clicking opens a modal with a scrollable list of notifications grouped by type with colored icons, title, message, relative timestamp, and mark-read/delete actions.

8. REST APIs Overview

The backend exposes 7 API modules with a total of 27 endpoints. All protected routes require a JWT token in the Authorization: Bearer <token> header.

8.1 User APIs (/api/user)

Method	Endpoint	Auth	Description
POST	/api/user/register	No	Register a new user (Renter/Owner)
POST	/api/user/login	No	Authenticate user, return JWT token
POST	/api/user/forgotpassword	No	Reset user password
POST	/api/user/getuserdata	Yes	Get authenticated user's profile data
GET	/api/user/getAllProperties	No	Fetch all properties (public)
POST	/api/user/bookinghandle/:propertyid	Yes	Submit a booking request for a property
GET	/api/user/getallbookings	Yes	Get all bookings for the logged-in renter
PATCH	/api/user/cancelbooking/:bookingId	Yes	Cancel a booking
POST	/api/user/submitreview	Yes	Submit a review for a property
GET	/api/user/getreviews/:propertyId	No	Get all reviews for a property

8.2 Admin APIs (/api/admin)

Method	Endpoint	Auth	Description
GET	/api/admin/getallusers	Yes	Fetch all registered users
POST	/api/admin/handlestatus	Yes	Grant or ungrant owner access
GET	/api/admin/getallproperties	Yes	Fetch all properties system-wide
GET	/api/admin/getallbookings	Yes	Fetch all bookings system-wide

8.3 Owner APIs (/api/owner)

Method	Endpoint	Auth	Description
POST	/api/owner/postproperty	Yes	Add a new property listing (multipart: images + video)
GET	/api/owner/getallproperties	Yes	Fetch all properties owned by the logged-in user
GET	/api/owner/getallbookings	Yes	Fetch all bookings for owner's properties
POST	/api/owner/handlebookingstatus	Yes	Accept or revoke a booking request
GET	/api/owner/getbookedproperties	Yes	Get currently booked properties with tenant details
GET	/api/owner/getreviews	Yes	Get all reviews for owner's properties
PATCH	/api/owner/incrementview/:propertyId	No	Increment property view counter
DELETE	/api/owner/deleteproperty/:propertyid	Yes	Delete a property listing
PATCH	/api/owner/updateproperty/:propertyid	Yes	Update property details (with optional image)

8.4 Notification APIs (/api/notifications)

Method	Endpoint	Auth	Description
GET	/api/notifications/	Yes	Fetch all notifications for logged-in user
PATCH	/api/notifications/read/:id	Yes	Mark a notification as read
PATCH	/api/notifications/readall	Yes	Mark all notifications as read
DELETE	/api/notifications/:id	Yes	Delete a notification

8.5 Payment APIs (/api/payment)

Method	Endpoint	Auth	Description
POST	/api/payment/create-order	Yes	Create a payment order for a booking
POST	/api/payment/verify	Yes	Verify and complete a payment
GET	/api/payment/history	Yes	Get payment history for user (as payer or owner)
GET	/api/payment/check/:bookingId	Yes	Check if rent has been paid for a booking

8.6 Chat APIs (/api/chat)

Method	Endpoint	Auth	Description
POST	/api/chat/send	Yes	Send a message to another user
GET	/api/chat/conversations	Yes	Get conversation list with last message & unread count
GET	/api/chat/messages/:otherUserId	Yes	Get message history with a specific user
GET	/api/chat/unread-count	Yes	Get total unread message count

8.7 Recent Views APIs (/api/recent-views)

Method	Endpoint	Auth	Description
POST	/api/recent-views/track/:propertyId	Yes	Track a property view (upsert pattern)
GET	/api/recent-views/list	Yes	Get last 10 recently viewed properties

9. Challenges Faced & Solutions

#	Challenge	Solution
1	**Multi-file Upload Handling:** Uploading multiple property images and a video simultaneously required handling different file types with proper validation.	Used Multer's upload.fields() method with separate field configurations for images (max 10) and video (max 1). Implemented custom fileFilter to validate file extensions by field name, and set a 100MB size limit.
2	**Role-Based Access & Owner Approval:** Owners needed admin approval before accessing their dashboard, requiring a multi-step authorization flow.	Implemented a granted field on the User schema. Login controller checks this status and blocks ungranted owners. Admin panel provides Grant/Ungrant toggle buttons to control access.
3	**Real-Time Messaging Without WebSockets:** Building a chat system required near-real-time message delivery without the complexity of a WebSocket server.	Implemented a polling-based approach where the frontend fetches new messages every 5 seconds and unread counts periodically. Messages stored in MongoDB with read status tracking.
4	**Cross-Origin Resource Sharing (CORS):** React dev server (port 3000) and Express server (port 8001) run on different ports, causing CORS issues.	Configured Express CORS middleware with explicit origin whitelist including http://localhost:3000, allowed methods, and permitted headers (Content-Type, Authorization).
5	**Amenities Data Format Inconsistency:** Amenities needed to be stored as a JSON string but could	Implemented robust parsing in both the addPropertyController and getAllPropertiesController — validating/parsing JSON strings, stringifying arrays, and falling back to [] for malformed data.

#	Challenge	Solution
	arrive as an array or malformed string from the frontend.	
6	**MongoDB Disconnection Resilience:** During development, the app needed to function even when MongoDB was unavailable.	Implemented a isMongoConnected() helper function that checks mongoose.connection.readyState. Controllers conditionally use MongoDB or a mock data fallback.
7	**Notification System Scalability:** Creating notifications for every action across multiple user types without tightly coupling the code.	Created a centralized createNotification() helper function in the notification controller, imported and called from other controllers (booking, payment, property, chat).
8	**Payment Flow Without External Gateway:** Integrating a full Razorpay payment flow in development without live API keys.	Implemented a simulated payment flow that mirrors the Razorpay pattern (order creation -> payment verification) using crypto-generated IDs. Can be upgraded to live Razorpay SDK by replacing ID generation with actual API calls.
9	**Excel Export with Summary Data:** Exporting booked properties to Excel required both detailed data rows and a summary sheet.	Used the SheetJS (xlsx) library to create a multi-sheet workbook — one sheet with detailed booking data and a second summary sheet with aggregate statistics. FileSaver triggers the client-side download as .xlsx file.
10	**Property Rating Aggregation:** Maintaining an accurate average rating on each property as reviews are added.	After each review submission, the system queries all reviews for that property, calculates the arithmetic mean, rounds to one decimal place, and updates the property's rating field.

10. Conclusion

The HouseHunt (RentEase) project has been successfully developed as a comprehensive, full-stack property rental management platform using the MERN stack. The application demonstrates a practical implementation of modern web development practices including RESTful API design, JWT-based authentication, role-based access control, and component-based frontend architecture.

The platform effectively addresses the core challenges of the property rental process by providing:

- **A unified platform** connecting property owners and renters with intuitive, role-specific dashboards.
- **Rich property listing capabilities** with multi-image/video uploads, amenity tagging, and advanced search filtering to help renters find the perfect property.
- **A complete booking lifecycle** from request to acceptance, payment, and review — keeping all parties informed through the notification system.
- **In-app communication** enabling direct, context-rich conversations between owners and renters.
- **Data-driven insights** through analytics dashboards, view tracking, and rating aggregation to help owners optimize their listings.

- **Administrative oversight** ensuring platform integrity with user management and content monitoring.

The project involved designing 8 database collections, implementing 27 REST API endpoints across 7 modules, and building 23+ React components organized by user role. Key technical challenges — including multi-file uploads, real-time messaging via polling, cross-origin configuration, and payment simulation — were systematically addressed with practical, scalable solutions.

This project has served as an excellent learning experience in full-stack development, covering frontend design with React and Material UI, backend development with Express.js, NoSQL database modeling with MongoDB/Mongoose, and integration of third-party libraries for file handling, data export, and payment processing.

11. Future Enhancements

The following enhancements can be considered for future iterations of the platform:

18. **WebSocket-Based Real-Time Chat:** Replace the current polling-based chat with Socket.IO or native WebSockets for instant message delivery, typing indicators, and online status.
19. **Live Payment Gateway Integration:** Replace the simulated payment flow with a live Razorpay or Stripe integration, enabling actual online rent payments, automatic receipt generation, and recurring payment schedules.
20. **Map-Based Property Search:** Integrate Google Maps or Mapbox to display property locations on an interactive map, enabling geographic search, distance-based filtering, and neighborhood exploration.
21. **Email & SMS Notifications:** Extend the notification system to send email and SMS alerts for critical events (booking confirmations, payment receipts, password resets) using Nodemailer, SendGrid, or Twilio.
22. **Virtual 360° Property Tours:** Integrate a 360° image viewer (e.g., Pannellum) allowing owners to upload panoramic photos for immersive virtual property tours.
23. **AI-Powered Property Recommendations:** Implement a recommendation engine that suggests properties based on user search history, recently viewed properties, saved favorites, and budget preferences.
24. **Lease Agreement & Document Management:** Add a module for digital lease agreement creation, e-signing, and document storage — enabling paperless rental transactions.
25. **Multi-Language Support (i18n):** Internationalize the application to support multiple languages using React i18n libraries, expanding the platform's reach.
26. **Progressive Web App (PWA):** Convert the frontend into a PWA with offline caching, push notifications, and home screen installation for a native app-like mobile experience.
27. **Advanced Admin Analytics:** Build a comprehensive admin analytics dashboard with user growth charts, platform revenue metrics, property listing trends, and geographic distribution heat maps.

28. **Tenant Verification System:** Implement identity verification for renters (Aadhaar/PAN validation, background checks) to build trust and improve tenant application quality.
29. **Maintenance Request Module:** Add a feature allowing tenants to submit maintenance requests to owners, track request status, and manage service history.