**Real Estate Management System**

*Project Report*

Generated: 2025-09-23T18:18:22.910Z

# 1. Overview

This document summarizes the application, technology stack, features, API, and configuration based on the repository README and package manifests.

# 2. Technology Stack

## Frontend (client)

axios: ^1.10.0

leaflet: ^1.9.4

react-leaflet: ^5.0.0

react-router-dom: ^7.6.2

### Frontend Dev Dependencies

vite: ^6.3.5

## Backend (server)

bcryptjs: ^3.0.2

cloudinary: ^1.41.3

cors: ^2.8.5

dotenv: ^16.5.0

express: ^5.1.0

jsonwebtoken: ^9.0.2

mongoose: ^8.15.2

multer: ^2.0.1

multer-storage-cloudinary: ^4.0.0

nodemailer: ^7.0.5

# 3. Documentation (from README.md)

# 🏠 Real Estate Management System

A full-stack real estate application built with React.js frontend and Node.js/Express.js backend with MongoDB database. Users can list properties, browse listings, manage their properties, and handle property bookings with email notifications.

## 📋 Table of Contents

- [Features](#-features)

- [Tech Stack](#-tech-stack)

- [Application Flow](#-application-flow)

- [Project Structure](#-project-structure)

- [Prerequisites](#-prerequisites)

- [Installation](#-installation)

- [Environment Configuration](#-environment-configuration)

- [Running the Application](#-running-the-application)

- [API Endpoints](#-api-endpoints)

- [Database Schema](#-database-schema)

- [Contributing](#-contributing)

## ✨ Features

### 🔐 Authentication & User Management

- User registration and login with JWT authentication

- Secure password hashing with bcrypt

- Protected routes for authenticated users

### 🏘️ Property Management

- Add, edit, and delete properties

- Upload property images

- View all properties with filtering options

- Property availability status

- Property booking system

### 🔎 Advanced Search & Discovery

- Filter by city, price range (dual-range slider), BHK (1/2/3/4 via pill buttons), and area/locality

- Optional nearby suggestions when no exact matches are found (geospatial search by radius)

- Quick-select radius pills (1/2/5/10/20 km) + custom radius

- "Use my location" geolocation button with reverse geocoded city hint

- Nearby-only mode when coordinates are provided (see `nearbyOnly` query)

### 📧 Notifications & Communication

- Real-time email notifications for property inquiries

- In-app notification system

- Contact property owners directly

- Booking viewings with email confirmations

### 🎨 Modern UI/UX

- Responsive design with modern styling

- Interactive property cards

- Clean and intuitive navigation

- Gradient backgrounds and smooth animations

- Add Property page with two-column layout, image previews, geolocation button, and inline validation

- Smooth scrolling improvements: image lazy-loading, component memoization, and content-visibility

## 🛠️ Tech Stack

### Frontend

- \*\*React.js\*\* - UI framework

- \*\*React Router\*\* - Client-side routing

- \*\*Vite\*\* - Build tool and development server

- \*\*Axios\*\* - HTTP client for API calls

- \*\*Leaflet\*\* - Interactive maps

- \*\*CSS3\*\* - Styling with modern features

### Backend

- \*\*Node.js\*\* - Runtime environment

- \*\*Express.js\*\* - Web framework

- \*\*MongoDB\*\* - NoSQL database

- \*\*Mongoose\*\* - MongoDB object modeling

- \*\*JWT\*\* - Authentication tokens

- \*\*bcryptjs\*\* - Password hashing

- \*\*Multer\*\* - File upload handling

- \*\*Nodemailer\*\* - Email functionality

- \*\*CORS\*\* - Cross-origin resource sharing

## 🔄 Application Flow

### User Journey

1. \*\*Landing Page\*\* → Browse available properties

2. \*\*Registration/Login\*\* → Create account or sign in

3. \*\*Property Listings\*\* → View all properties with filters (city, price slider, BHK pills, area, radius)

4. \*\*Property Details\*\* → View detailed property information

5. \*\*Contact Owner\*\* → Send inquiries via email

6. \*\*Book Viewing\*\* → Schedule property viewings

7. \*\*My Properties\*\* → Manage own property listings

8. \*\*Add Property\*\* → Two-column form, BHK pills, image previews, optional geolocation

### Data Flow

```

User Action → Frontend (React) → API Request → Backend (Express) → Database (MongoDB) → Response → Frontend Update

```

### Authentication Flow

```

Login/Register → JWT Token Generation → Token Storage → Protected Route Access → Token Verification → User Data

```

## 📁 Project Structure

```

my-real-estate-app/

├── client/ # Frontend React application

│ ├── public/

│ │ └── vite.svg

│ ├── src/

│ │ ├── components/ # Reusable UI components

│ │ │ ├── Navbar.jsx # Navigation component

│ │ │ ├── Navbar.css

│ │ │ ├── NotificationBell.jsx # Notification component

│ │ │ ├── NotificationBell.css

│ │ │ ├── PropertyCard.jsx # Property display card

│ │ │ └── PropertyCard.css

│ │ ├── pages/ # Page components

│ │ │ ├── Home.jsx # Landing page

│ │ │ ├── Home.css

│ │ │ ├── Login.jsx # Login page

│ │ │ ├── Register.jsx # Registration page

│ │ │ ├── PropertyList.jsx # Property listings

│ │ │ ├── PropertyList.css

│ │ │ ├── PropertyDetails.jsx # Property detail view

│ │ │ ├── PropertyDetails.css

│ │ │ ├── AddProperty.jsx # Add new property

│ │ │ ├── EditProperty.jsx # Edit existing property

│ │ │ ├── MyProperties.jsx # User's properties

│ │ │ └── MyProperties.css

│ │ ├── utils/ # Utility functions

│ │ │ ├── auth.js # Authentication utilities

│ │ │ └── ProtectedRoute.jsx # Route protection

│ │ ├── styles/ # Global styles

│ │ │ └── AuthPage.css

│ │ ├── App.jsx # Main app component

│ │ ├── App.css # Global styles

│ │ └── main.jsx # App entry point

│ ├── index.html # HTML template

│ ├── package.json # Frontend dependencies

│ └── package-lock.json

├── server/ # Backend Node.js application

│ ├── models/ # Database models

│ │ ├── Property.js # Property schema

│ │ └── UserNew.js # User schema with notifications

│ ├── routes/ # API routes

│ │ ├── auth.js # Authentication routes

│ │ └── properties.js # Property management routes

│ ├── middleware/ # Custom middleware

│ │ └── auth.js # JWT verification middleware

│ ├── utils/ # Utility functions

│ │ ├── cloudinary.js # Cloudinary configuration

│ │ └── storage.js # File storage utilities

│ ├── controllers/ # Route controllers (empty)

│ ├── uploads/ # File upload directory (served at /uploads)

│ ├── server.js # Main server file

│ ├── package.json # Backend dependencies

│ └── package-lock.json

└── README.md # Project documentation

```

## 📋 Prerequisites

Before running this application, make sure you have the following installed:

- \*\*Node.js\*\* (v16 or higher)

- \*\*npm\*\* or \*\*yarn\*\*

- \*\*MongoDB\*\* (local installation or MongoDB Atlas account)

- \*\*Git\*\*

## 🚀 Installation

### 1. Clone the Repository

```bash

git clone <your-repository-url>

cd my-real-estate-app

```

### 2. Install Dependencies

#### Backend Dependencies

```bash

cd server

npm install

```

#### Frontend Dependencies

```bash

cd ../client

npm install

```

## ⚙️ Environment Configuration

### Backend Environment Variables (.env)

Create a `.env` file in the `server/` directory:

```env

# Server Configuration

PORT=5000

NODE\_ENV=development

# Database Configuration

MONGO\_URL=mongodb://localhost:27017/real-estate-app

# OR for MongoDB Atlas:

# MONGO\_URL=mongodb+srv://<username>:<password>@<cluster>.mongodb.net/real-estate-app

# JWT Configuration

JWT\_SECRET=your-super-secret-jwt-key-here

# Email Configuration (for notifications)

SMTP\_HOST=smtp.gmail.com

SMTP\_PORT=587

SMTP\_SECURE=false

SMTP\_USER=your-email@gmail.com

SMTP\_PASS=your-app-password

SMTP\_FROM=your-email@gmail.com

# Cloudinary Configuration (for image uploads)

CLOUDINARY\_CLOUD\_NAME=your-cloud-name

CLOUDINARY\_API\_KEY=your-api-key

CLOUDINARY\_API\_SECRET=your-api-secret

# CORS Configuration

CLIENT\_URL=http://localhost:5173

```

### Frontend Environment Variables (.env)

Create a `.env` file in the `client/` directory:

```env

# API Configuration

VITE\_API\_URL=http://localhost:5000

# Map Configuration (if using maps)

VITE\_MAP\_API\_KEY=your-map-api-key

# Other Configuration

VITE\_APP\_NAME=Real Estate App

```

## 🏃‍♂️ Running the Application

### 1. Start the Backend Server

```bash

cd server

npm start

```

The server will start on `http://localhost:5000`

Notes:

- Ensure the directory `server/uploads` exists for image uploads. Static files are served from `/uploads`.

- Geospatial queries require MongoDB 2dsphere index (already defined on `Property.location`).

### 2. Start the Frontend Development Server

```bash

cd client

npm run dev

```

The client will start on `http://localhost:5173`

### 3. Access the Application

Open your browser and navigate to `http://localhost:5173`

## 📡 API Endpoints

### Authentication Routes (`/auth`)

| Method | Endpoint | Description | Auth Required |

|--------|----------|-------------|---------------|

| POST | `/register` | Register new user | No |

| POST | `/login` | User login | No |

### Property Routes (`/properties`)

| Method | Endpoint | Description | Auth Required |

|--------|----------|-------------|---------------|

| GET | `/` | Get all properties (filters: `city`, `minPrice`, `maxPrice`, `bhk`, `area`, `suggestNearby=true`, `lat`, `lng`, `radiusKm`, `nearbyOnly=true`) | No |

| GET | `/my-properties` | Get user's properties | Yes |

| GET | `/:id` | Get single property | No |

| POST | `/` | Add new property | Yes |

| PUT | `/:id` | Update property | Yes |

| DELETE | `/:id` | Delete property | Yes |

| POST | `/:id/contact` | Contact property owner | No |

| POST | `/:id/book` | Book property viewing | No |

Notes:

- When no exact matches are found and `suggestNearby=true` with coordinates provided, the API returns nearby results and sets the response header `X-Suggested: true`.

- If `nearbyOnly=true` is provided along with `lat`, `lng`, and `radiusKm`, only nearby results are returned.

## 🗄️ Database Schema

### User Schema (`UserNew`)

```javascript

{

name: String (required),

email: String (required, unique),

password: String (required, hashed),

notifications: [{

message: String (required),

type: String (required),

link: String,

read: Boolean (default: false),

createdAt: Date (default: now)

}]

}

```

### Property Schema (`Property`)

```javascript

{

title: String (required),

description: String (required),

price: Number (required),

city: String (required),

bhk: Number,

areaName: String,

location: {

type: { type: String, enum: ['Point'] },

coordinates: [Number] // [lng, lat]

},

image: String,

images: [String],

available: Boolean (default: true),

bookings: [{

name: String,

email: String,

date: String,

time: String,

message: String,

createdAt: Date (default: now)

}],

userId: ObjectId (ref: 'UserNew', required),

createdAt: Date,

updatedAt: Date

}

```

## 🔧 Configuration Details

### Email Setup (Gmail)

1. Enable 2-factor authentication on your Gmail account

2. Generate an App Password

3. Use the App Password in `SMTP\_PASS`

### MongoDB Setup

1. \*\*Local MongoDB\*\*: Install MongoDB locally and start the service

2. \*\*MongoDB Atlas\*\*: Create a free cluster and get the connection string

### Cloudinary Setup (Optional)

1. Create a free Cloudinary account

2. Get your cloud name, API key, and API secret

3. Add them to the backend `.env` file

## 🐛 Troubleshooting

### Common Issues

1. \*\*MongoDB Connection Error\*\*

- Ensure MongoDB is running locally

- Check your connection string in `.env`

- Verify network connectivity for Atlas

2. \*\*Email Not Sending\*\*

- Verify SMTP credentials

- Check if 2FA is enabled for Gmail

- Use App Password instead of regular password

3. \*\*CORS Errors\*\*

- Ensure `CLIENT\_URL` is set correctly in backend `.env`

- Check if frontend is running on the correct port

4. \*\*JWT Token Issues\*\*

- Verify `JWT\_SECRET` is set in backend `.env`

- Check token expiration time