# Nest Finder Platform Design Document

**Table of Contents**

1. **Introduction**
2. **System Overview**
3. **Architecture**
   * High-Level Architecture
   * Component Interactions
4. **Backend Design**
   * Technologies Used
   * Project Structure
   * API Design
5. **Database Schema**
6. **Relational Database Schema**
7. **Frontend Design**
   * Technologies Used
   * Project Structure
   * Routing
   * State Management
   * Key Components
8. **Security Considerations**
9. **Deployment Plan**
10. **Future Plans**
11. **Conclusion**

## 1. Introduction

This **Nest Finder** (Real Estate Platform) is a comprehensive web application designed to revolutionize the property market by seamlessly connecting users looking to buy, sell, or rent properties. The platform caters to a diverse audience, offering a user-friendly interface, secure transactions, and advanced communication features, such as real-time messaging powered by Socket.IO. Built on the MERN stack (MongoDB, Express.js, React, Node.js) and utilizing modern web development practices, the system is architected to ensure scalability, maintainability, and high performance.

The platform's backend employs Node.js with Express to deliver a robust RESTful API for data management, with MongoDB serving as the database to store user information, property listings, and messages. For security, the system leverages JWT for authentication and Bcrypt for password encryption, ensuring user data is protected. On the frontend, React is combined with Tailwind CSS and Flowbite to create a responsive and visually appealing interface. State management is handled by Redux, ensuring efficient and consistent data flow across the application.

Key features include property management functionalities for listing, browsing, and searching properties, a real-time messaging system for user interaction, and secure user authentication. The platform is hosted on Vercel, offering global accessibility with a streamlined CI/CD pipeline.

This document serves as a detailed blueprint, outlining the system architecture, backend and frontend designs, security considerations, deployment strategies, and future plans for enhancement. It aims to guide developers, stakeholders, and contributors in building and scaling a platform that delivers a seamless and engaging experience to users in the real estate market.

## 2. System Overview

The platform is designed to offer a comprehensive and user-friendly experience with the following core functionalities:

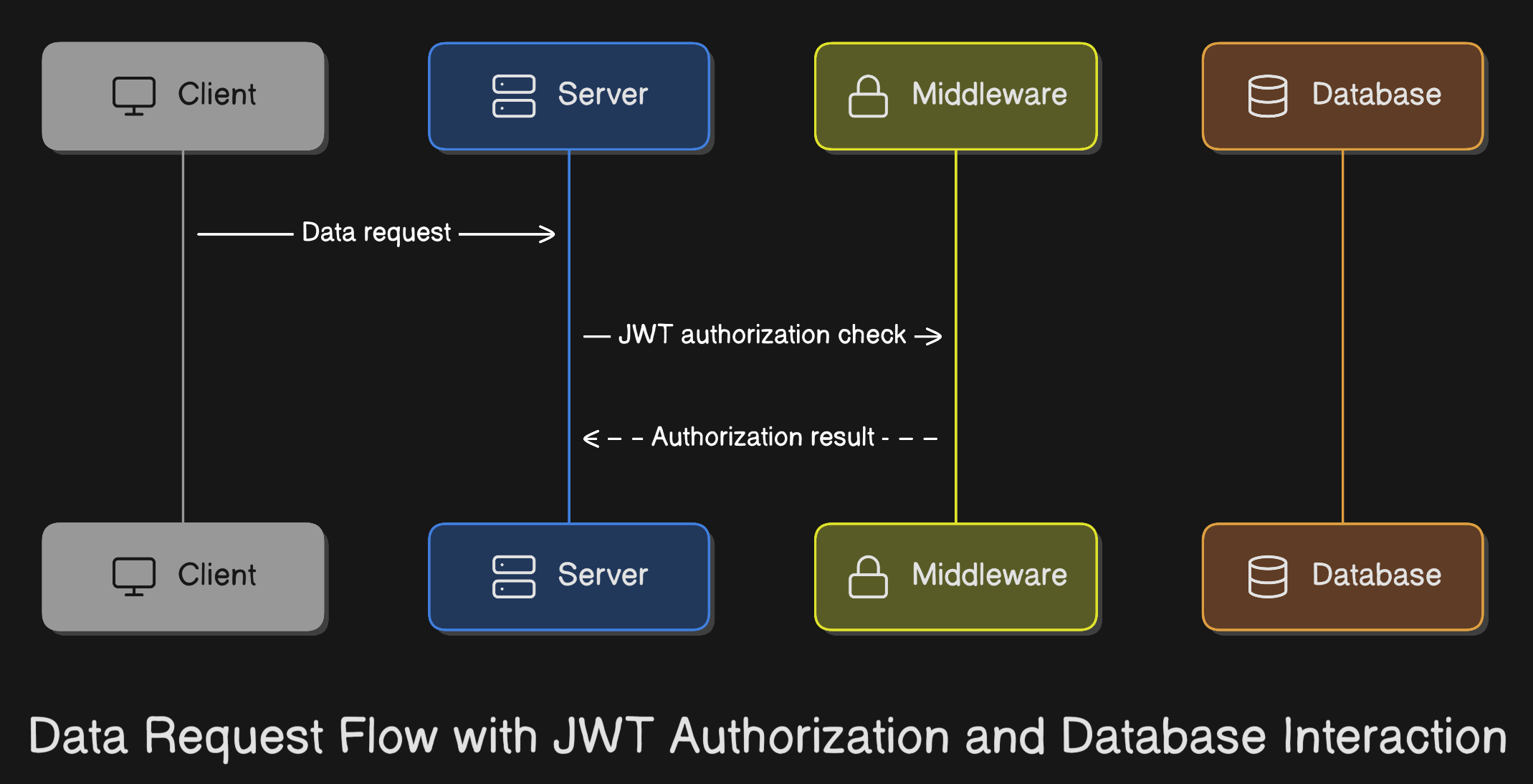
* **Property Listing and Search:** Users can easily browse and search for available properties, utilizing advanced filters and search criteria to find their ideal listings.
* **User Registration and Authentication:** All users must register and authenticate to access the platform's features. This includes a secure sign-up process and login system, ensuring personalized and protected access.
* **Real-Time Messaging:** Integrated WebSocket’s technology enables real-time messaging capabilities between users, providing instant communication and collaboration on property inquiries, updates, and negotiations.
* **Responsive and Intuitive User Interface (UI):** The platform is designed to be fully responsive, offering a seamless experience on desktops, tablets, and mobile devices. Its intuitive interface ensures that users can easily navigate and access features.

**Constraints and Assumptions**

* **User Registration Requirement:** All users must register and log in to access the full set of platform features, ensuring a secure and personalized experience for each individual.
* **Real-Time Messaging Dependency:** Real-time messaging functionality is dependent on a stable WebSocket connection. A reliable internet connection is necessary to maintain continuous communication.
* **Global Availability with Vercel Deployment:** The platform will be deployed on Vercel, ensuring its global availability, scalability, and fast content delivery, providing users around the world with consistent and reliable access.

## 3. Architecture

**High-Level Architecture:**  
 The system is built on a client-server model with the following components:  
 1. Frontend: React with Tailwind CSS and Flowbite for a responsive UI.  
 2. Backend: Node.js with Express for RESTful APIs.  
 3. Database: MongoDB for storing data such as users, properties, and messages.  
 4. Messaging: Socket.IO for real-time communication.  
 5. Deployment: Hosted on Vercel with CI/CD pipelines.  
   
**Component Interactions:**  
 - Frontend communicates with the Backend using REST APIs for data retrieval and updates.  
 - Backend interacts with MongoDB for data persistence.  
 - Socket.IO establishes a WebSocket connection for in-app messaging.  
 - Vercel ensures continuous deployment and scalability.

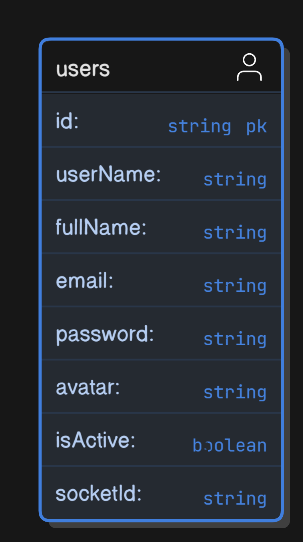


## 4. Backend Design

**Technologies Used:**  
 - Node.js: Backend runtime.  
 - Express.js: Web framework for RESTful APIs.  
 - MongoDB: Database for storing application data.  
 - Socket.IO: Real-time messaging framework.  
 - JWT: For secure authentication.  
 - Bcrypt: For password hashing.  
   
 **Project Structure:**  
 backend/  
 ├── controllers/  
 ├── models/  
 ├── routes/  
 ├── middlewares/  
 ├── utils/  
 └── index.js  
   
 **API Design:**  
 Key Endpoints:  
 1. User Management:  
 - POST /api/register: User registration.  
 - POST /api/login: User authentication.  
 2. Property Management:  
 - GET /api/listAllProperty: Fetch properties.  
 - POST /api/ listAllProperty: Add a new property.  
 - PUT /api/editProperty/:id: Update property details.  
 - DELETE /api/deleteProperty/:id: Remove a property.  
 3. Messaging:  
 - GET /api/messages/:conversationId: Fetch messages.  
 - POST /api/messages: Send a message.

## 5. Database Schema

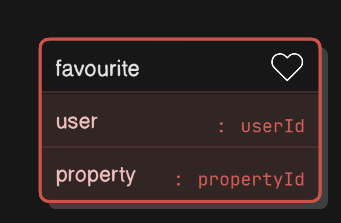
1. **User**

****

1. **Property**

****

1. **Favorite**

****

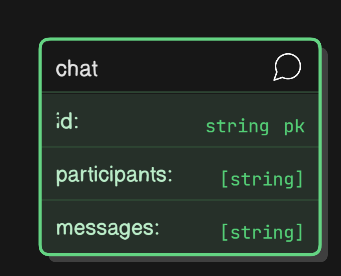
1. **Request**

****

1. **Message**

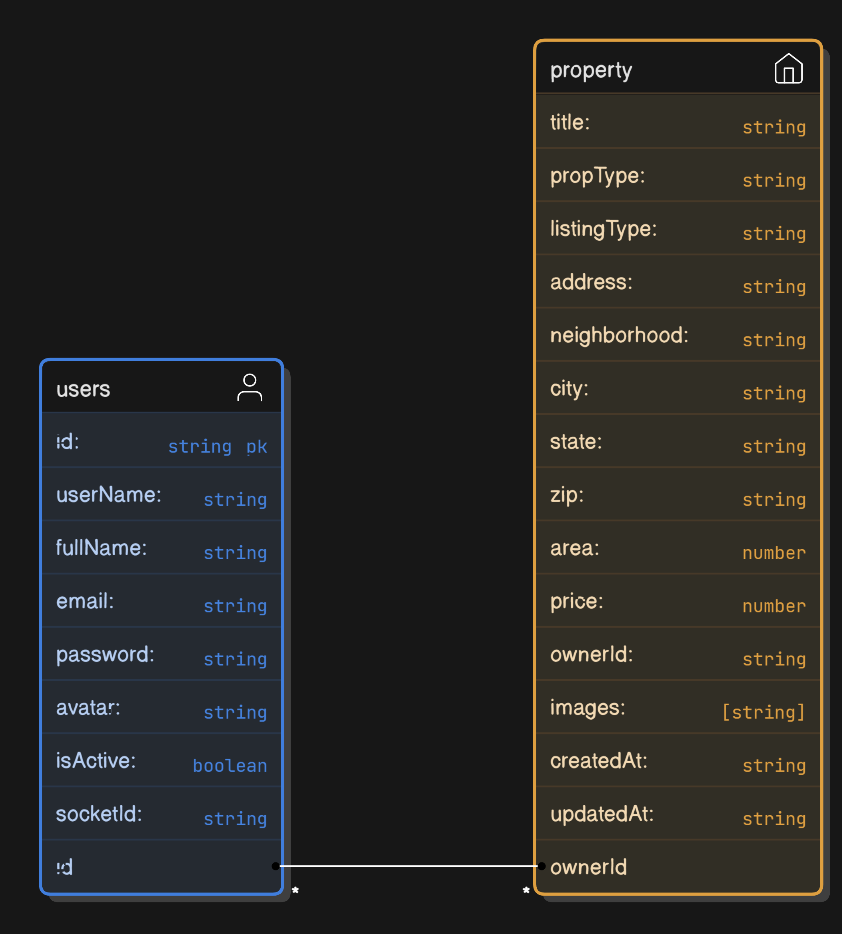
****

1. **Chat**

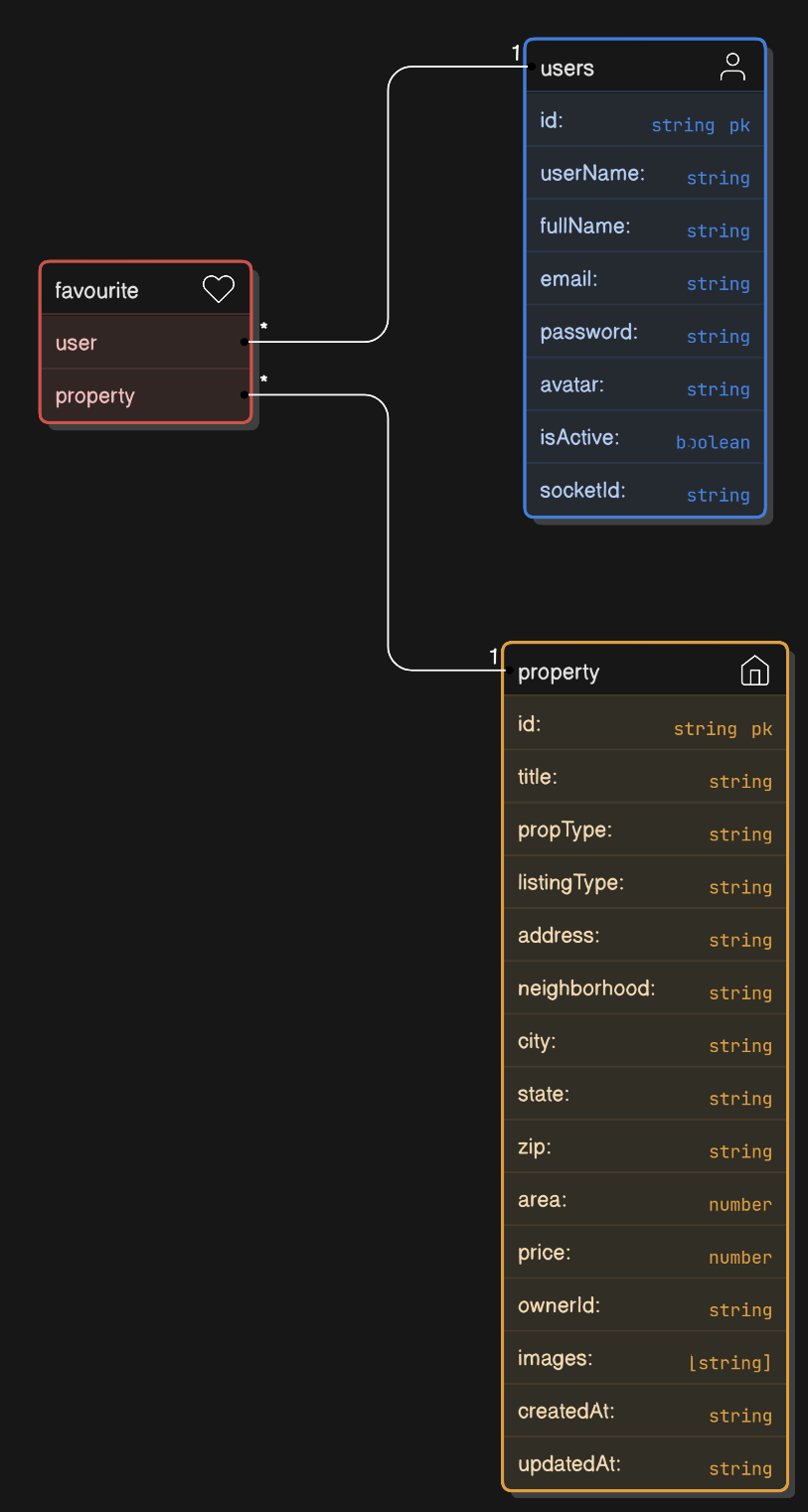
****

## 6. Relational Database Schema

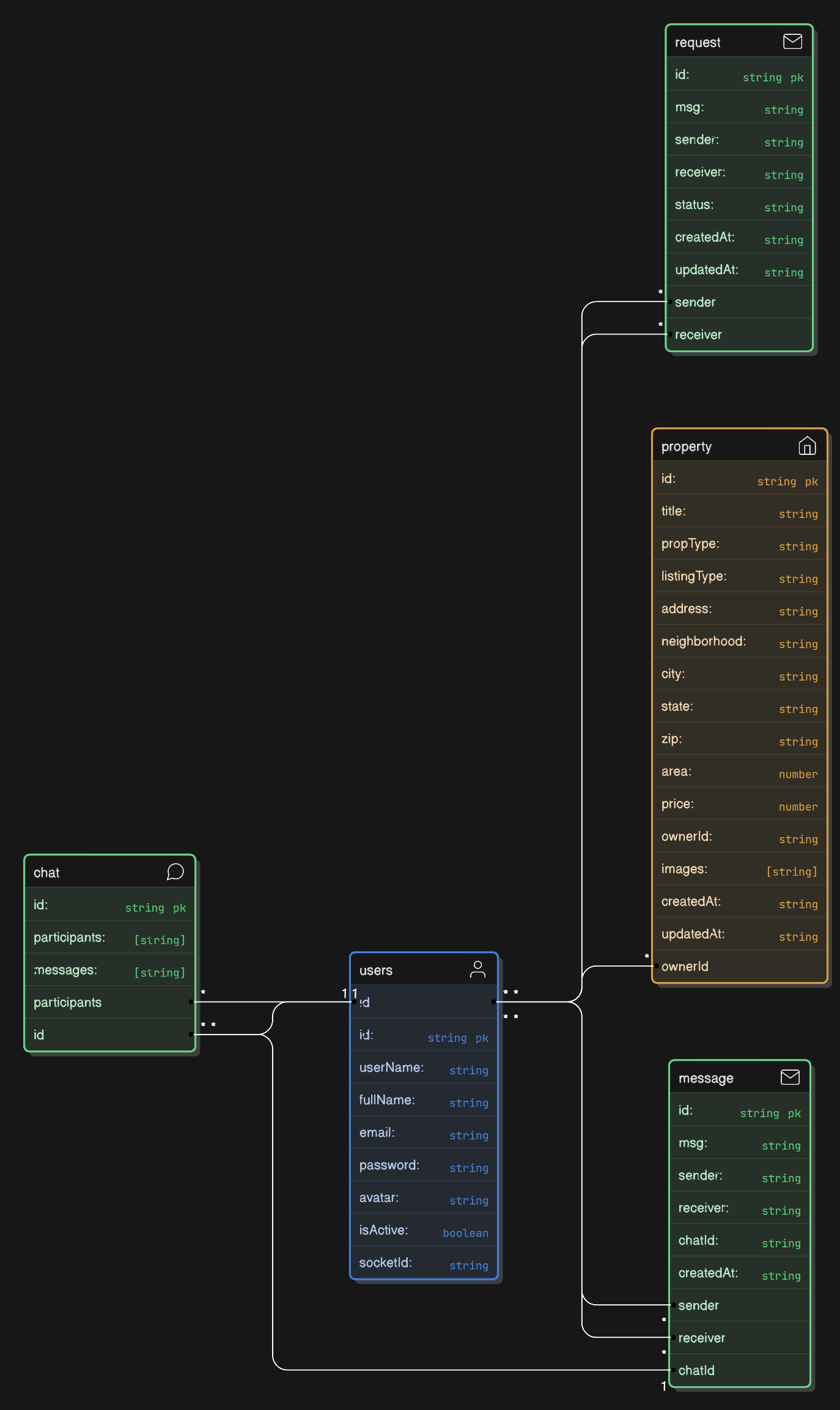
1. **User Property**

****

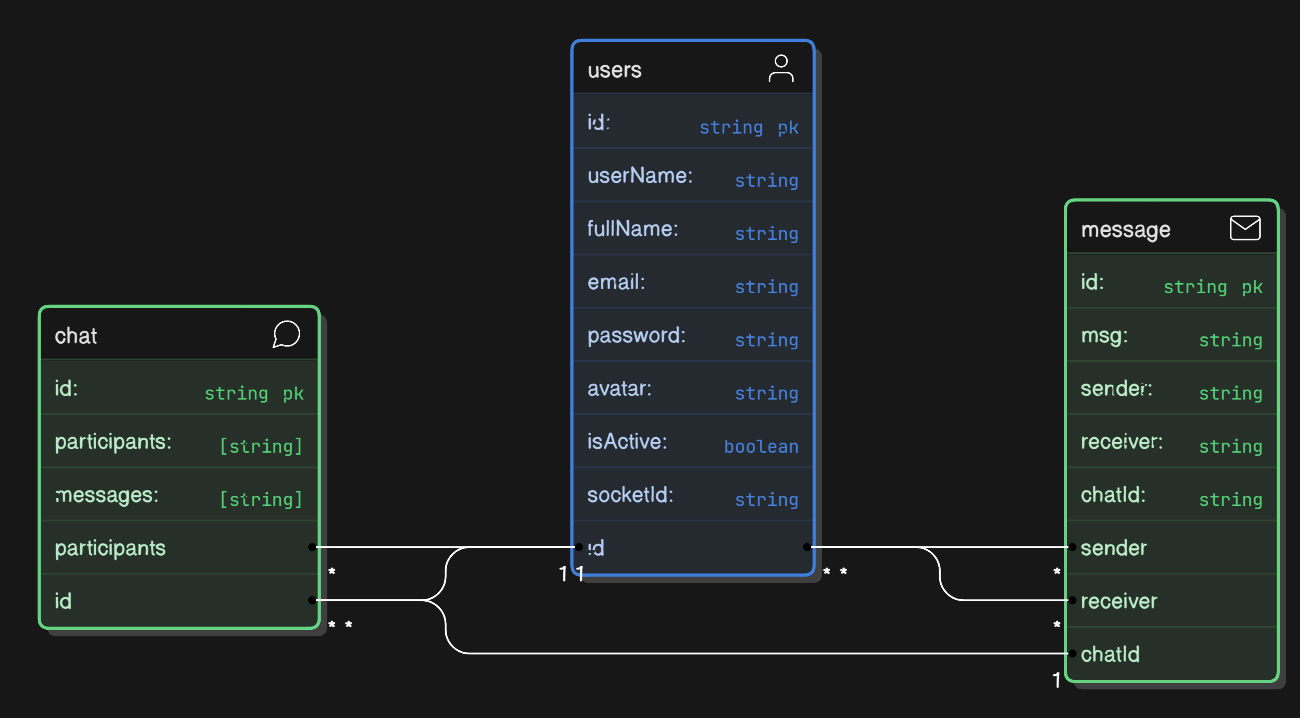
1. **User Property Favorite**

****

1. **Request Chat User Property**

****

1. **Chat Message User**

****

## 7. Frontend Design

**Technologies Used:**  
 - React: Component-based UI framework.  
 - Tailwind CSS: Utility-first CSS framework for responsive design.  
 - Flowbite: Pre-built components for faster development.  
 - Redux: State management library.  
   
 **Project Structure:**  
 frontend/  
 ├── src/  
 │ ├── components/  
 │ ├── pages/  
 │ ├── redux/  
 │ ├── utils/  
 │ ├── App.js  
 │ └── index.js  
   
 **Routing:**  
 - /: Home page with featured properties.  
 - /login: User login.  
 - /register: User registration.  
 - /properties: Browse properties.  
 - /properties/:id: Property details.  
 - /messages: Messaging interface.  
   
 **State Management**:  
 - Redux Store:  
 - user: User authentication and profile.  
 - properties: List of properties.  
 - messages: Chat messages.  
   
 **Key Components:**  
 1. Navbar: Navigation bar with links to major sections.  
 2. PropertyCard: Displays property details.  
 3. MessageBox: Real-time chat interface.

## 8. Security Considerations

* **Authentication (JWT-based):**  
  The platform uses **JSON Web Tokens (JWT)** for user authentication. Upon successful login, users receive a token that grants them access to protected resources. The token has a set expiry time to ensure session validity, after which re-authentication is required.
* **Password Protection (Bcrypt Hashing):**  
  User passwords are stored securely using **Bcrypt hashing**. This algorithm ensures that passwords are never stored in plain text, providing strong protection against data breaches and unauthorized access.
* **Data Encryption (HTTPS):**  
  All data transmitted between users and the platform is encrypted using **HTTPS**. This ensures that sensitive information, such as login credentials and payment details, is securely transmitted and protected from interception by third parties.

## 9. Deployment Plan

## Frontend Deployment:

## Platform: Vercel

## Role: Vercel will be used to host the frontend application. It’s a serverless platform designed for fast deployments, automated scaling, and performance optimization. Vercel will automatically deploy the frontend from GitHub on each commit, and it supports static site generation and server-side rendering (SSR) for dynamic content.

## Backend Deployment:

## Platform: Render

## Role: Render will host the backend services. It’s a platform for deploying full-stack apps, APIs and databases. The backend code will be deployed to Render using GitHub Actions, which will enable continuous deployment (CD). Render offers a variety of services, such as managed PostgreSQL databases and Docker support for custom backends.

## Database Management:

## Service: MongoDB Atlas

## Role: MongoDB Atlas is a fully-managed cloud database platform for MongoDB. It simplifies database operations such as scaling, backups, and monitoring. MongoDB Atlas will be used for managing your application’s data, providing a flexible schema that’s ideal for rapid development and iteration.

## Image Storage and Optimization:

## Service: Cloudinary

## Role: Cloudinary will be used for image storage and optimization. It’s a cloud-based media management platform that helps automate tasks like image and video transformations (e.g., resizing, cropping, compressing) and delivery via a Content Delivery Network (CDN). Cloudinary integrates seamlessly with your frontend to serve optimized media for different devices and screen sizes.

## 10. Future Plans

1. **Payment Integration:**

* **Secure Transactions:** Integrate **Stripe**, **PayPal**, or **Razorpay** for property payments and deposits.
* **Benefit:** Ensures smooth, secure transactions and builds trust.

1. **Localization:**

* **Multi-Language Support:** Offer multilingual UI, currency conversion, and localized listings.
* **Benefit:** Expands global reach and accessibility.

1. **Mobile App:**

* **Platform Reach:** Develop **iOS** and **Android** apps to browse listings and make transactions.
* **Benefit:** Increases engagement and accessibility on mobile.

1. **Video Call Feature:**

* **Virtual Tours:** Add video calls for property viewings.
* **Benefit:** Enables remote property tours, reducing physical visits.

## 11. Conclusion

In conclusion, this Nest Finder (Real Estate Platform) is designed to provide a comprehensive, secure, and user-friendly experience for individuals looking to buy, sell, or rent properties. Key features include a **Property Listing and Search** system that allows users to easily browse and filter available properties based on their preferences. A secure **User Registration and Authentication** process ensures that only registered users can access personalized features, protecting user data and privacy.

The platform is equipped with robust security measures, ensuring **Secure Transactions** for all user interactions. Real-time communication is made possible through **Real-Time Messaging** powered by WebSocket technology, enabling instant communication between users for property inquiries, updates, and negotiations. Finally, the platform offers a **Responsive and Intuitive User Interface (UI)**, ensuring a smooth experience across all devices, from desktops to mobile phones, allowing users to effortlessly navigate and access features.

This design document serves as a strategic roadmap for the platform’s development, focusing on core functionalities that will deliver an engaging, efficient, and secure experience for all users, ensuring the platform’s success and scalability in the real estate market.