



INTERNATIONAL ISLAMIC UNIVERSITY CHITTAGONG

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Final Project Report

THE REAL ESTATE PLATFORM

Group - 05



Course Title : Software Engineering Lab

Course Code ; CSE-3638

Submitted by

- Nahian Subah Ishma_C223286
- Rehnuma Tasneem_C223288
- Saima Kawsar_C223297
- Sakaratul Ara Tasmia_C223298
- Tahsin Islam Nafisa_C223311

Semester : 6th

Section : 6CF

Submitted to

Sultana Tasnim Jahan
Assistant Lecturer
Dept. of CSE , IIUC

Acknowledgement

We would like to express our heartfelt gratitude to our respected course instructor, **Ms. Sultana Tasnim**, Assistant Lecturer, Department of Computer Science & Engineering, International Islamic University Chittagong, for her continuous guidance, encouragement, and insightful feedback throughout the development of this project. Her support has been instrumental in shaping our ideas and turning them into a complete system.

We are also deeply thankful to our dedicated team members, whose collaboration and tireless efforts made this project possible. Each member brought their own strengths and skills, contributing meaningfully to various phases including requirement analysis, system design, implementation, and testing.

This project is the result of the collective efforts of Nahian Subah Ishma, Rehnuma Tasneem, Saima Kawsar, Sakaratul Ara Tasmia, and Tahsin Islam Nafisa. Throughout this journey, we have gained valuable experience in teamwork, time management, and problem-solving. Most importantly, we learned how to apply our theoretical knowledge to develop a real-world solution. This project has greatly enriched our understanding of software engineering and collaborative development.

Abstract

In the real estate sector, key stakeholders such as buyers, agents, and administrators often face challenges including inefficient property management, lack of transparency, poor communication, and absence of secure digital solutions. Traditional systems are typically fragmented, outdated, and do not support modern features like online payments or role-specific functionalities.

To address these issues, we developed a web-based real estate platform that facilitates seamless interaction and secure transactions between all user roles. The platform includes essential features such as property listing, wishlist management, offer submission, user authentication via Firebase, and secure payment processing through Stripe. It is built using the MERN (MongoDB, Express.js, React, Node.js) stack, ensuring high performance, responsiveness, and scalability.

The development process followed the complete software engineering life cycle. Starting with a feasibility study, we proceeded through requirement analysis, system design (including various UML diagrams), and implementation. The architecture is modular and follows a three-tier structure for better maintainability and scalability.

The final system provides a reliable, user-friendly, and secure solution for managing real estate operations. It meets all project objectives and enhances user experience while offering a long-term, cost-effective alternative to traditional real estate management systems.

Table of Contents

Acknowledgement	1
Abstract.....	2
1. Introduction	6
1.1 Background of the Project	6
1.2 Problem Statement.....	6
1.3 Objectives	6
1.4 Scope of the Project.....	6
2. Project Planning.....	7
2.1 Timeline / Gantt Chart	7
2.2 Milestones	7
2.3 Work Distribution Among Team Members	7
2.4 Software Development Model Used	7
3. Literature Review / Related Work	8
3.1 Existing Systems or Studies.....	8
3.2 How Our System is Different.....	8
4. Methodology	9
4.1 Software Development Model Used	9
4.2 Tools and Technologies Used	9
4.3 Overview of Planning.....	10
5. Requirement Analysis	11
5.1 Functional Requirements.....	11
5.2 Non-Functional Requirements.....	11
5.3 Use Case Diagram(s)	12
6. Feasibility Study.....	13
6.1 Technical Feasibility	13
6.2 Economic Feasibility	13
6.3 Operational Feasibility	14
7. System Design.....	15
7.1 System Architecture.....	15
7.2 ER Diagram.....	16
8. Implementation	17
8.1 Tools & Technologies Used	17
8.2 Modules Developed.....	17

8.3 Sample Code Snippets / Screenshots	18
8.4 Screenshots of working software	20
9. Testing	21
9.1 Test Plan	22
9.2 Types of Testing Done	22
9.3 Sample Test Cases	22
9.4 Bug Log	23
9.5 Final Testing Result.....	23
10. Deployment.....	24
10.1 Deployment Environment.....	24
10.2 Deployment Steps.....	24
10.3 Installation Guide (Local)	24
10.4 Deployment Challenges	24
11. Maintenance.....	25
11.1 Types of Maintenance Expected.....	25
11.2 Future Upgrade Plan.....	25
11.3 Known Issues or Limitations	25
12. Result and Discussion	26
12.1 Project Outcome	26
12.2 Key Functionalities Achieved	26
12.3 Performance and Limitations	26
13. Challenges Faced.....	27
13.1 Technical Challenges.....	27
13.2 Team-Related Challenges	27
13.3 Summary.....	27
14. Personal Reflection & Learning Outcomes	28
14.1 What Was Learned by Each Team Member.....	28
14.2 Technical Knowledge Gained	28
14.3 Teamwork, Communication & Problem-Solving Reflection	28
14.4 Improvement Areas	28
15. Conclusion	29
15.1 Overall Summary of the Project Outcome	29
15.2 Were the Objectives Met?	29
15.3 What Went Well / Could Be Improved?	29

16. Future Work	30
16.1 Suggested Improvements	30
16.2 Additional Features.....	30
16.3 Scaling Possibilities	30
17 References	30

1. Introduction

1.1 Background of the Project

In recent years, the real estate world has gone digital. Most people now prefer to search for homes, apartments, or commercial spaces online rather than physically visiting places or meeting agents directly. But even though there are plenty of real estate websites out there, many of them don't meet user expectations. Some are slow, outdated, or just too complicated to use. Others don't offer key features like secure payments or proper dashboards for agents and admins. That's why we decided to build the real estate platform—a modern web-based solution that brings everything together in one place. It connects buyers, agents, and admins through a clean and responsive design, built using powerful technologies like React, Node.js, Express, and MongoDB (the MERN stack).

1.2 Problem Statement

Even though many real estate websites exist, most of them offer limited features and lack the flexibility users expect today. Buyers often find it hard to search for properties efficiently because of poor filters or outdated listings. Important details like pricing, images, or property features are sometimes missing, which creates confusion and lack of trust. Agents do not have complete control over their listings or communication tools, and admins have very few options for platform monitoring. In addition, many platforms fail to provide secure login systems, real-time updates, or proper support for online payments. These limitations reduce user satisfaction and make property transactions more difficult. The real estate platform was created to solve these problems by building a complete digital solution that is easy to use, highly secure, and supports role-based dashboards, online transactions, and real-time property management for all users.

1.3 Objectives

The main objective of the real estate platform is to build a complete and user-friendly property management system that allows buyers, agents, and admins to perform their roles smoothly. It focuses on helping buyers find properties easily, save their favorite ones, and purchase them securely. Agents are provided with tools to manage their listings and engage with interested buyers. Admins have full control over all users, listings, and platform activities. The platform ensures high performance by using modern technologies like React for the frontend and Node.js for the backend. It also includes Firebase and JWT for secure logins and Stripe for smooth payment processing. Real-time data updates and responsive design ensure the system works effectively across all devices. The goal is to make the property transaction process faster, safer, and easier for everyone involved.

1.4 Scope of the Project

This platform includes everything needed to handle real estate activities online. It supports three main user roles: buyer, agent, and admin. Each role has its own dashboard with different features. Buyers can browse properties, view details, add items to their wishlist, and pay online. Agents can add new listings, update existing ones, and monitor their performance. Admins have access to all parts of the platform—they can manage users, approve or remove listings, and view reports. We've also added secure logins, real-time updates, and a smooth interface that works on any device. In the future, more features like live chat support, push notifications,

or AI-based suggestions could be added. Overall, the platform is designed to be flexible, scalable, and ready for real-world use.

2. Project Planning

2.1 Timeline / Gantt Chart

Date	Task Description	Completion (%)
30-06-2025	Project setup and basic frontend layout	70%
01-07-2025	Backend API structure and database schema initialized	80%
03-07-2025	User authentication and property listing completed	90%
05-07-2025	Dashboards, property edit/delete functionality added	95%
07-07-2025	Final testing, bug fixing, UI polish, and deployment	100%

2.2 Milestones

Milestone	Target Date	Status
Initial project setup and design	30-06-2025	Completed
Backend and database development	01-07-2025	Completed
Core features (auth, listing) ready	03-07-2025	Completed
Dashboard and property management	05-07-2025	Completed
Final deployment and delivery	07-07-2025	Completed

2.3 Work Distribution Among Team Members

Team Member	Responsibilities
Nahian Subah Ishma	Frontend UI design using React and Tailwind CSS
Rehnuma Tasneem	Documentation, coordination, and API testing
Tahsin Islam Nafisa	Backend development using Express.js and MongoDB
Sakaratul Ara Tasmia	Firebase authentication, JWT setup, and backend deployment
Saima Kawsar	Stripe payment integration and user dashboard features

2.4 Software Development Model Used

For developing the Real Estate Platform, we used a Modified Agile Software Development Model. This model helped our team work well together, quickly change plans when needed, and keep improving the project all the time. We chose Agile because it works best for projects where things change often and teamwork is important.

Why Agile?

Traditional models like Waterfall follow strict steps, so it's hard to make changes once the project starts. But in real projects like the Real Estate Platform, the design, user feedback, and features keep changing. Agile lets us change and improve things as we go.

3. Literature Review / Related Work

3.1 Existing Systems or Studies

In recent years, many online platforms have been developed to make buying, selling, or renting properties easier. Popular examples include **Bikroy.com** (Bangladesh), **Zillow** (USA), **99acres** (India). These platforms let users browse property listings, view basic details, and contact sellers or agents. Most of them mainly focus on listing and searching properties. Some also provide simple map views and contact forms for inquiries.

However, these systems have some limitations that affect user experience. For example, Bikroy.com works like a classified ads site, where users post property ads just like used items. It lacks real-time updates, payment integration, and role-based dashboards. Zillow, while more advanced, is mostly location-based and not suitable for global or highly customizable expansion. It also lacks deeper features for agents and admin monitoring. Platforms like 99acres offer property listings and contact features, but users still need to manage bookings, payments, and communication outside the platform.

3.2 How Our System is Different

In contrast, our real estate platform offers a more complete and modern solution. This system supports three distinct user roles: buyer, agent, and admin, each with their own dashboards. It uses secure login through Firebase and JWT authentication, and real-time data fetching ensures users always get the most updated property information.

A key difference is the integration of payment processing using Stripe, allowing buyers to pay directly within the platform, a feature missing in most local systems.

The platform is built on the MERN stack (MongoDB, Express.js, React, Node.js), which enables better performance, faster rendering, and easy scalability for future features such as chat support, push notifications, and recommendation engines.

Additionally, it has a clean, responsive design that works well on mobile devices, tablets, and desktops. Overall, while existing platforms provide basic services, our system delivers an all-in-one real estate experience with advanced features, improved security, and room for future growth.

4. Methodology

4.1 Software Development Model Used

We followed the Agile Software Development Model for the real estate platform project. Agile helped us work flexibly, deliver features step by step, and respond to changes quickly. Below are the key characteristics of our Agile approach:

Iterative and Incremental Development was a major feature of our approach. We broke the project into smaller, manageable tasks called "sprints." Each sprint focused on implementing a specific set of features such as UI setup, authentication system, user dashboards, and more. After completing each sprint, we tested and improved the features before moving on to the next sprint. This process helped us maintain quality and stability throughout the project.

Frequent Team Meetings and Collaboration played an important role in our success. We communicated regularly using online platforms like Messenger, WhatsApp, and Google Meet. During these meetings, team members updated each other on progress, discussed any challenges, and planned the next steps together. This ongoing communication helped us work more efficiently and make decisions faster.

Continuous Integration and Testing was used to keep the system stable. We often merged new code into the main project branch to avoid big problems later on. After adding each new feature such as login, property listing, or payment module we carefully tested it. This way, we could find and fix errors early before they affected other parts of the system.

Client-Oriented Flexibility helped us focus on user experience. Even though we didn't have a real external client, we treated our users buyers, agents, and admins as our clients. We gathered their feedback through internal testing and used it to improve the interface and responsiveness. For example, we redesigned parts of the dashboard when users found them confusing.

Progress Tracking with Real Deadlines was another important part of our plan. Each sprint had a clear deadline, shown in our Gantt Chart. We tracked progress by measuring how much work was done and only marked tasks as complete after proper testing. This helped us stay organized, motivated, and ensured the project finished on time.

4.2 Tools and Technologies Used

We used a variety of modern tools and technologies to build the real estate platform efficiently and ensure good performance:

Frontend

- React.js – For building user interfaces.
- Tailwind CSS – For fast and responsive styling.

Backend

- Node.js & Express.js – For building the server and handling API requests.

- MongoDB – As the main database to store property listings, users, and transactions.

Authentication & Security

- Firebase Authentication – For easy and secure login/signup.
- JWT (JSON Web Tokens) – For protecting routes and securing user roles.

4.3 Overview of Planning

We planned the real estate platform project by dividing it into several important phases to keep the work organized and smooth.

Requirement Gathering

In this phase, we focused on understanding what the platform needed to offer. We clearly defined our target users buyers, agents, and admins and thought from their perspectives. We listed down all the essential features that the platform must include, such as property browsing, user dashboards, wishlist options, secure authentication, and an online payment system. This phase gave us a clear direction for design and development.

Design Phase

Once we had our requirements, we moved on to designing the platform. We created wireframes and basic layouts for each page, including the homepage, login/signup, dashboards, and property listings. While designing, we paid special attention to keeping the user interface clean, easy to use, and mobile responsive. Our goal was to make sure users could navigate the platform smoothly on any device.

Development Phase

In this phase, we started building the actual platform. We first set up the backend using Node.js and Express, along with a MongoDB database. Then we developed the frontend using React.js and Tailwind CSS, step by step. Features like secure login, property listing, role-based dashboards, and payment integration were added one by one. We also used Firebase for authentication and JWT for securing protected routes, while Stripe was added for handling payments.

Testing & Debugging

After building the main features, we moved on to testing. We tested each module manually for all user roles admin, agent, and buyer. This helped us find bugs and fix them early. We also improved performance and user experience based on our testing results. This phase was important to ensure everything worked smoothly and securely before launch.

Deployment & Final Review

Once the platform was fully functional and tested, we deployed the frontend to Netlify and the backend to Railway. After deployment, we reviewed all features, ensured the live version worked correctly, and prepared demo credentials for testing and presentation. This final review ensured everything was ready for users.

Overall, this planning approach helped us work step by step, stay organized, and complete the project successfully within the planned time.

5. Requirement Analysis

5.1 Functional Requirements

Functional requirements define the core functionalities that the system must offer to satisfy user needs. For the real estate platform, the system must fulfill the following:

1. **User Registration and Login**
Users (Buyer, Agent, Admin) must be able to register and log in using email and password. Authentication is handled via Firebase and secured using JWT.
2. **Role-Based Dashboards**
The system must provide separate dashboards for Buyers, Agents, and Admins with features tailored to each role.
3. **Property Listings**
Agents must be able to list, edit, and remove property listings. Buyers can browse, search, and view property details.
4. **Wishlist Management**
Buyers should be able to add properties to their wishlist and remove them when desired.
5. **Property Purchase**
Buyers can initiate and complete a property purchase through an integrated Stripe payment system.
6. **Admin Management**
Admins can view platform statistics, manage users, and monitor all property listings.
7. **Agent Requests and Approval**
Users can request to become an agent, and the Admin must approve or reject these requests.
8. **Reviews and Ratings (Optional/Future Scope)**
Buyers may leave reviews and ratings for properties or agents to improve platform credibility.

5.2 Non-Functional Requirements

Non-functional requirements ensure the quality and constraints under which the system operates. The real estate platform must meet the following:

1. **Performance**
The platform must respond to user actions (e.g., property search or wishlist update) within 2 seconds under normal load.
2. **Scalability**
The system should handle growth in user base and data volume without performance degradation. The MERN stack and TanStack Query support efficient scaling.
3. **Security**
The system must protect user data using Firebase Authentication and JWT. Stripe ensures secure payment processing, and HTTPS must be enforced on all web traffic.
4. **Usability**
The interface must be intuitive and user-friendly for all types of users (Buyers, Agents, Admins), following modern UI/UX design principles.

5. **Availability**

The system must maintain a 99.9% uptime with reliable hosting (Netlify for frontend, Railway for backend).

6. **Maintainability**

The codebase should follow clean architecture and be modular, allowing for future updates and bug fixes with minimal effort.

7. **Compatibility**

The platform must be fully responsive and accessible from major browsers (Chrome, Firefox, Edge) and devices (desktop, tablet, mobile).

8. **Data Integrity**

All transactions and property updates must be reliably recorded without data loss or inconsistency.

5.3 Use Case Diagram(s)

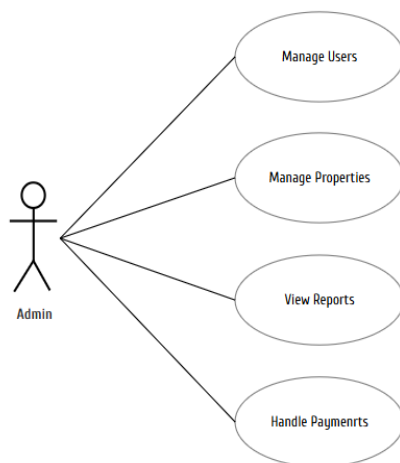


Fig: Admin Use case

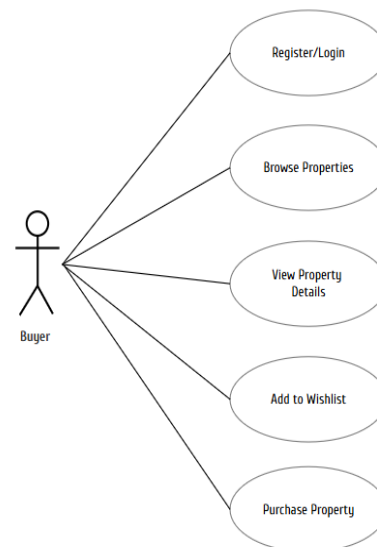


Fig: Buyer Use case

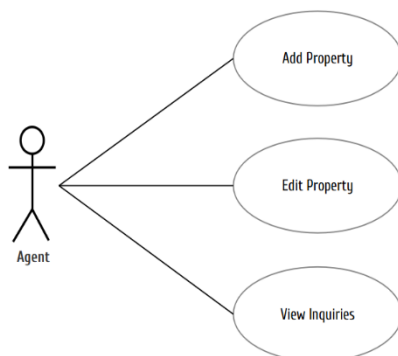


Fig: Agent Use case

6. Feasibility Study

6.1 Technical Feasibility

The Real-Estate-Management Platform can be developed using three distinct approaches:

- **In-House Solution:**
This approach requires the organization to have or hire experienced developers, designers, and system administrators who will build, deploy, and secure the application on dedicated servers or a private cloud. Every layer of the stack including the codebase, hosting environment, backups, intrusion detection, and ongoing technical support must be managed internally. This makes the approach technically feasible only for teams with deep expertise or sufficient funding to acquire it.
- **Third-Party Platform:**
This approach shifts most responsibilities to the vendor. Hosting, security patches, and feature upgrades are handled by the service provider, requiring only minimal local configuration. This option is technically feasible even for organisations with limited IT resources.
- **Hybrid Solution:**
A commercial platform delivers the core services, while in-house or contracted engineers integrate custom features and manage bespoke interfaces. This approach is suitable for teams with moderate development capacity and the ability to coordinate with an external provider on platform-level issues.

6.2 Economic Feasibility

Each development option has a different cost structure and long-term financial impact:

- **In-House Solution:**
Requires the highest initial investment of 1,950,000 BDT and an annual operating cost of 550,000 BDT. However, it generates the highest net benefit, approximately 1,450,000 BDT per year, leading to a discounted cumulative value of about 8,909,613 BDT over ten years.
- **Third-Party Platform:**
Has the lowest upfront cost of 800,000 BDT. Its yearly subscription and support fees nearly offset its benefits, resulting in a small annual surplus of 30,000 BDT and a ten-year present value of approximately 123,229 BDT.
- **Hybrid Solution:**
Falls between the two, with an initial cost of 1,325,000 BDT, recurring annual expenses of 500,000 BDT, and a yearly net return of 570,000 BDT. Its ten-year present value is about 3,502,405 BDT.

These figures show that while the in-house solution requires more capital investment, it offers the strongest long-term economic benefit.

6.3 Operational Feasibility

Operational demands differ according to the chosen approach:

- **In-House Solution:**
The organization must manage day-to-day system health, apply security updates, monitor performance, and troubleshoot issues continuously. This requires strong internal IT governance and operational capacity.
- **Third-Party Platform:**
This option reduces operational overhead significantly since the vendor handles backups, updates, and infrastructure monitoring. Local staff focus mainly on content management and basic administration.
- **Hybrid Solution:**
Reduces but does not eliminate operational responsibilities. The vendor manages core services, but custom modules, integrations, and security controls need to be supervised internally or by an outsourced partner.

We chose the hybrid approach because it gave us the best mix of flexibility, cost, and ease of managing the project. Making the platform completely in-house would need a lot of technical skills, more money, and more work to run it, which we didn't have. Using only a third-party platform would not let us add important custom features that our users need. The hybrid way let us use a trusted commercial platform for the basic parts, but also allowed us to create and add our own special features. This made it a smart and practical choice to build a strong, easy-to-use, and flexible real estate platform.

7. System Design

7.1 System Architecture

The system architecture of the Real Estate platform follows a **3-tier architecture** model, which separates the application into three main layers: the Presentation Layer, the Business Logic Layer, and the Data Layer.

The **Presentation Layer** is the user interface of the application, developed using React.js. It handles all user interactions, displaying property listings, dashboards, and forms, and communicates with the backend via API calls. This layer ensures responsiveness and smooth user experience across devices.

The **Business Logic Layer** sits in the backend and is built using Node.js with the Express.js framework. This layer processes incoming requests, applies the core logic, such as authentication, role-based access control, and payment processing via Stripe. It validates data, manages workflows, and enforces security before interacting with the database.

The **Data Layer** consists of the MongoDB database, where all data about users, properties, transactions, and system configurations are stored. This layer is responsible for data persistence, retrieval, and ensuring data integrity.

This separation of concerns allows the system to be scalable, maintainable, and easier to debug, as each layer focuses on a specific responsibility.

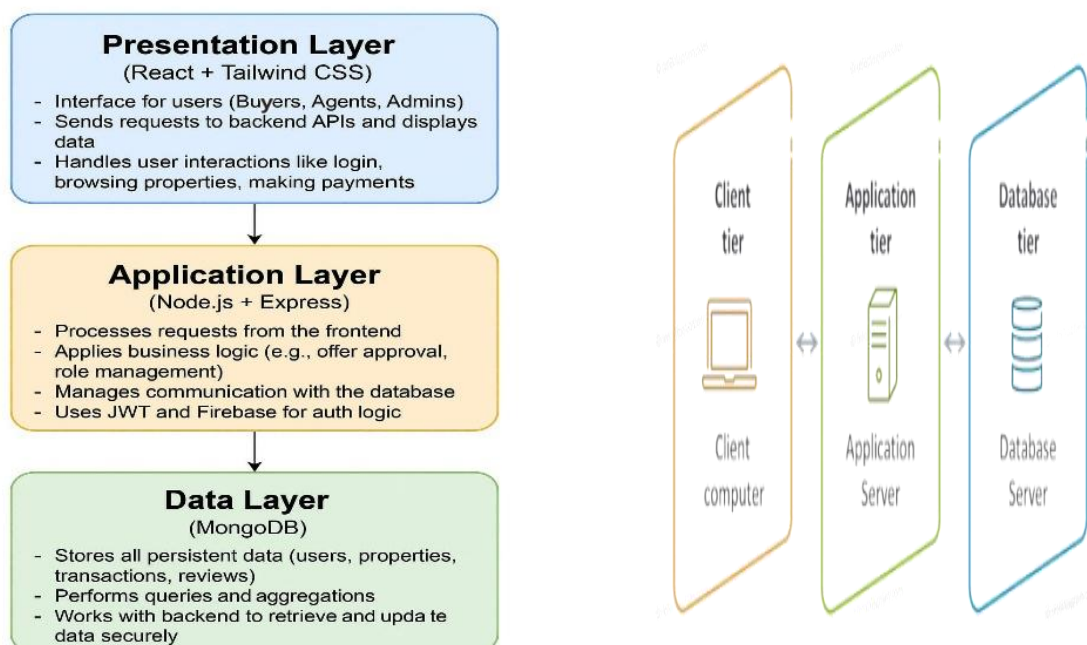


Fig: 3-Layer Architecture

7.2 ER Diagram

This ER diagram shows how users interact in the the real estate platform. There are three main users: Admin, Agent, and Buyer. Agents can list properties, buyers can view, purchase, and review them. Each property can have images. Buyers can also save properties to wishlists. Purchases lead to payments, and buyers can leave reviews after buying.

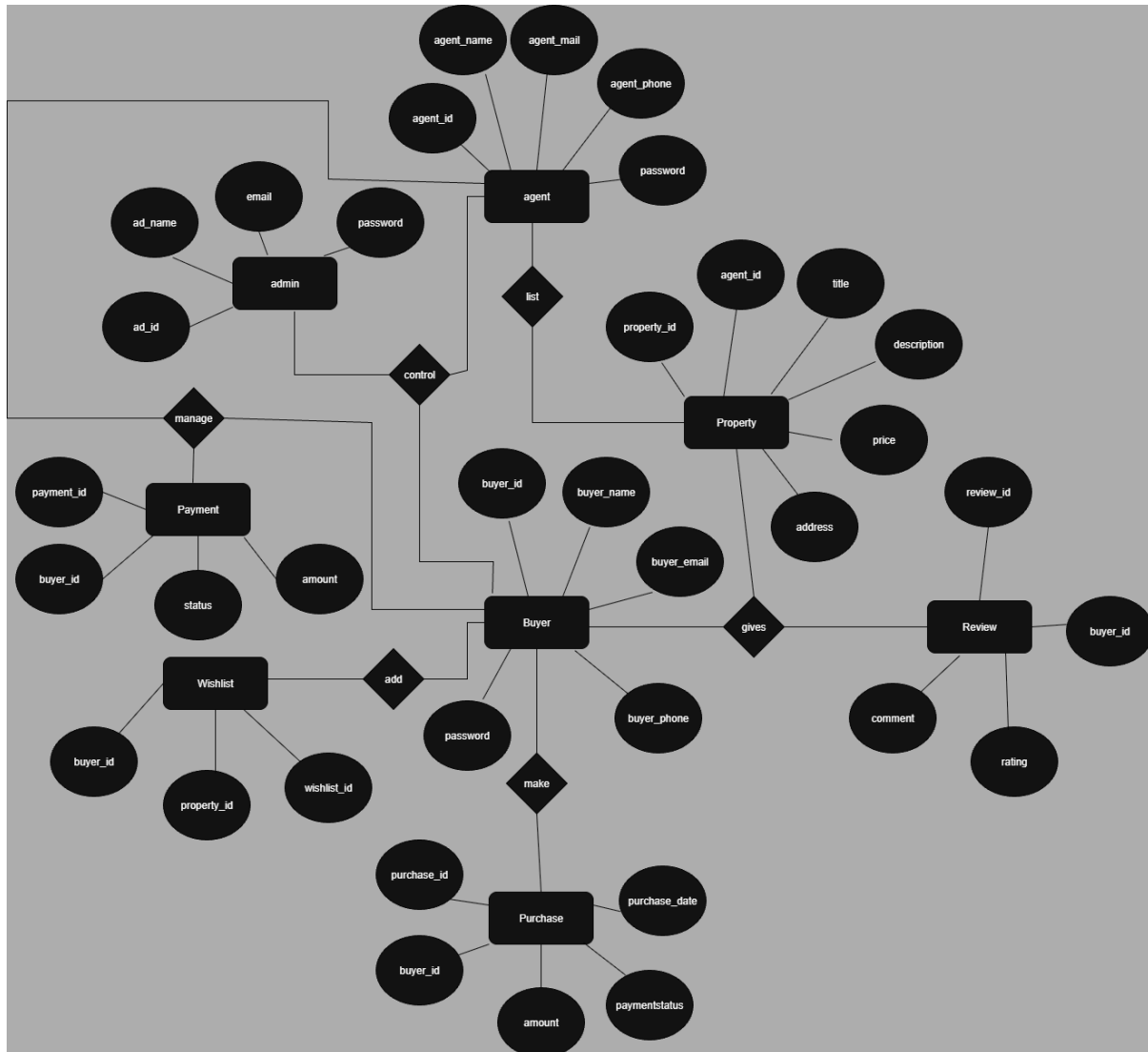


Fig: ER Diagram

8. Implementation

8.1 Tools & Technologies Used

Tool/Technology	Purpose/What We Used It For
React.js	Frontend development — creating UI, forms, dashboards, etc.
Tailwind CSS	Styling the frontend — responsive and modern design
Node.js	Backend runtime — handling server-side logic
Express.js	Backend framework — managing APIs and routing
MongoDB	Database — storing user data, property listings, transactions
Stripe	Payment gateway — secure online payments
JWT (JSON Web Token)	Authentication — securing access, role-based login system
Postman	API testing — testing backend routes and responses
Git & GitHub	Version control — managing code changes and team collaboration
VS Code	Code editor — writing and debugging code

8.2 Modules Developed

The Real Estate Platform consists of the following key modules, each playing a specific role in ensuring smooth functionality:

1. User Module

- This module handles user registration, login, and logout.
- It supports role-based access control, allowing different roles: Admin, Agent, and Buyer.
- Each role sees a different dashboard and has specific permissions.
 - Admin can manage everything.
 - Agents can list and manage their own properties.
 - Buyers can browse, book, and pay for properties.

2. Property Module

- This module allows agents to post new property listings with descriptions, prices, and images.
- Agents can update or delete their properties.
- Buyers can browse available properties, filter by location, price, or type.
- All properties are stored and fetched from the MongoDB database.

3. Booking Module

- Allows buyers to book a property they are interested in.
- The booking is stored and shown in the user's dashboard.
- Agents/Admins can view bookings and either approve or decline them.
- Prevents duplicate bookings and ensures a smooth flow from booking to payment

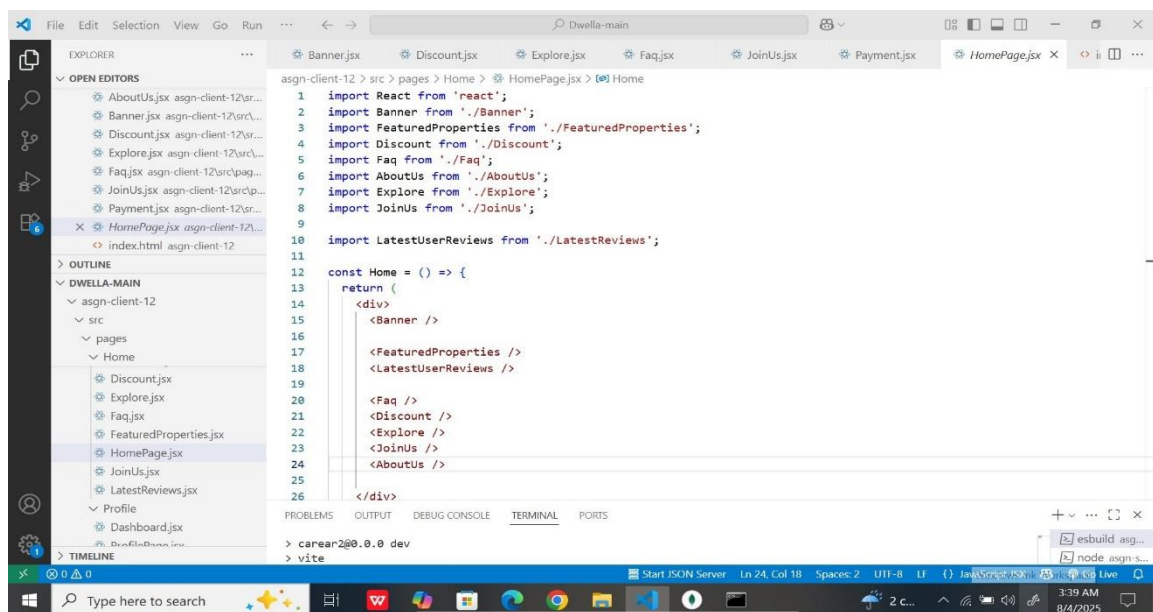
4. Payment Module

- This module is integrated with Stripe API to enable secure online payments.
- Once a booking is approved, the buyer can pay using Stripe.
- Payment confirmation updates the booking status and generates a transaction record.

5. Admin Dashboard

- A dedicated area for Admins to manage the entire system.
- Admin can view and manage:
 - All Users (approve/restrict agents, manage buyers)
 - All Properties
 - System Statistics (number of bookings, revenue, active users, etc.)
- Helps maintain full control and smooth operation of the platform.

8.3 Sample Code Snippets / Screenshots



The screenshot displays a Visual Studio Code editor window with the file explorer on the left showing the project structure. The main editor area shows the code for `HomePage.jsx`. The code imports various components and uses them in a functional component `Home`.

```
1 import React from 'react';
2 import Banner from './Banner';
3 import FeaturedProperties from './FeaturedProperties';
4 import Discount from './Discount';
5 import Faq from './Faq';
6 import AboutUs from './AboutUs';
7 import Explore from './Explore';
8 import JoinUs from './JoinUs';
9
10 import LatestUserReviews from './LatestReviews';
11
12 const Home = () => {
13   return (
14     <div>
15       <Banner />
16       <FeaturedProperties />
17       <LatestUserReviews />
18       <Faq />
19       <Discount />
20       <Explore />
21       <JoinUs />
22       <AboutUs />
23     </div>
24   );
25 }
26 export default Home;
```

The terminal at the bottom shows the command `carear2@0.0.0 dev` and the output `> vite`.

Fig: Home Page Code

```
14 const UserDashboardNav = () => {
15   const location = useLocation();
16   const { currentUser, loading, error } = useAuth();
17   const [activeNav, setActiveNav] = useState(location.pathname.split('/')[1] || 'dashboard');
18   const [graphData, setGraphData] = useState({});
19   const [currentTime, setCurrentTime] = useState(new Date());
20   const [totalProperty, setTotalProperty] = useState(0);
21   const [soldItems, setSoldItems] = useState(0);
22   const [requestItems, setRequestItems] = useState(0);
23   const [fetchError, setFetchError] = useState(null);
24
25   const navigationItems = [
26     { name: "Home", path: "/", icon: Home },
27     { name: "All Properties", path: "/allprop", icon: Building },
28     { name: "Agent Dashboard", path: "/dashboard", icon: LayoutDashboard },
29     { name: "Agent Profile", path: "/profile", icon: User },
30     { name: "Add Property", path: "/agent/add", icon: PlusCircle },
31     { name: "My added properties", path: "/agent/myadded", icon: List },
32     { name: "My sold properties", path: "/sold", icon: CheckCircle },
33     { name: "Requested properties", path: "/request", icon: Inbox },
34     { name: "Messages", path: "/contact", icon: MessageSquare },
35     { name: "Settings", path: "/setting", icon: Settings },
36   ];
37
38   useEffect(() => {
```

Fig: Agent Dashboard Code

```
3 import React, { useState, useEffect } from 'react';
4 import { useLocation } from 'react-router-dom';
5 import { motion } from 'framer-motion';
6 import { Settings, MessageSquare, Home, Building, LayoutDashboard, User, Heart, ShoppingCart, Star, Tag } from 'lucide-react';
7 import NavCard from '../components/CardComponent/NavCard.jsx';
8 import StatsCard from '../components/CardComponent/StatsCard.jsx';
9 import WeatherChart from '../pages/User/chart/WeatherChart.jsx';
10 import TimeWidget from '../pages/User/chart/TimeWidget.jsx';
11 import axios from 'axios';
12 import { useAuth } from '../context/AuthContext';
13
14 const UserDashboardNav = () => {
15   const location = useLocation();
16   const { currentUser, loading, error } = useAuth();
17   const [activeNav, setActiveNav] = useState(location.pathname.split('/')[1] || 'dashboard');
18   const [graphData, setGraphData] = useState({});
19   const [currentTime, setCurrentTime] = useState(new Date());
20   const [totalReviews, setTotalReviews] = useState(0);
21   const [wishlistItems, setWishlistItems] = useState(0);
22   const [offerItems, setOfferItems] = useState(0);
23   const [fetchError, setFetchError] = useState(null);
24
25   const navigationItems = [
26     { name: "Home", path: "/", icon: Home },
27     { name: "All Properties", path: "/allprop", icon: Building },
```

Fig: User Dashboard Code

```
12 import { useAuth } from '../context/AuthContext';
13
14 const UserDashboardNav = () => {
15   const location = useLocation();
16   const { currentUser, loading, error } = useAuth();
17   const [activeNav, setActiveNav] = useState(location.pathname.split('/')[1] || 'dashboard');
18   const [graphData, setGraphData] = useState({});
19   const [currentTime, setCurrentTime] = useState(new Date());
20   const [totalReviews, setTotalReviews] = useState(0);
21   const [userItems, setUserItems] = useState(0);
22   const [propertyItems, setPropertyItems] = useState(0);
23   const [fetchError, setFetchError] = useState(null);
24
25   const navigationItems = [
26     { name: "Home", path: "/", icon: Home },
27     { name: "All Properties", path: "/allprop", icon: Building },
28     { name: "Admin Dashboard", path: "/dashboard", icon: LayoutDashboard },
29     { name: "Admin Profile", path: "/profile", icon: User },
30     { name: "Manage Properties", path: "/admin/manageprop", icon: Settings },
31     { name: "Manage Users", path: "/admin/manage", icon: Users },
32     { name: "Manage reviews", path: "/admin/managereview", icon: Star },
33     { name: "Messages", path: "/contact", icon: MessageSquare },
34     { name: "Settings", path: "/setting", icon: Settings },
35   ];
36
37   useEffect(() => {
```

Fig: Admin Dashboard Code

8.4 Screenshots of working software

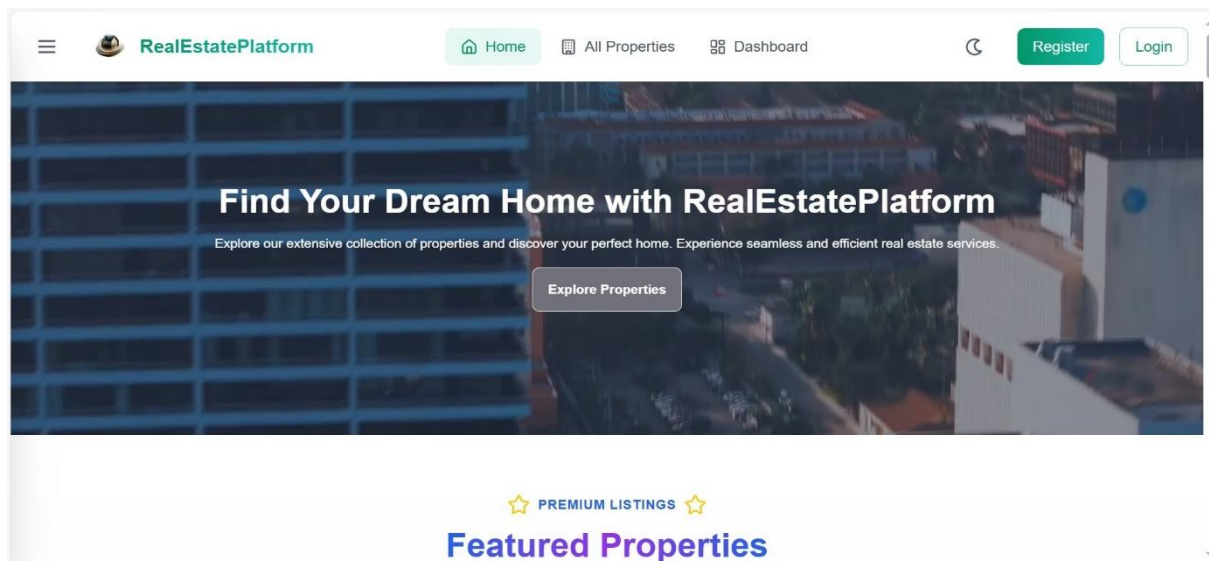


Fig: Home Page

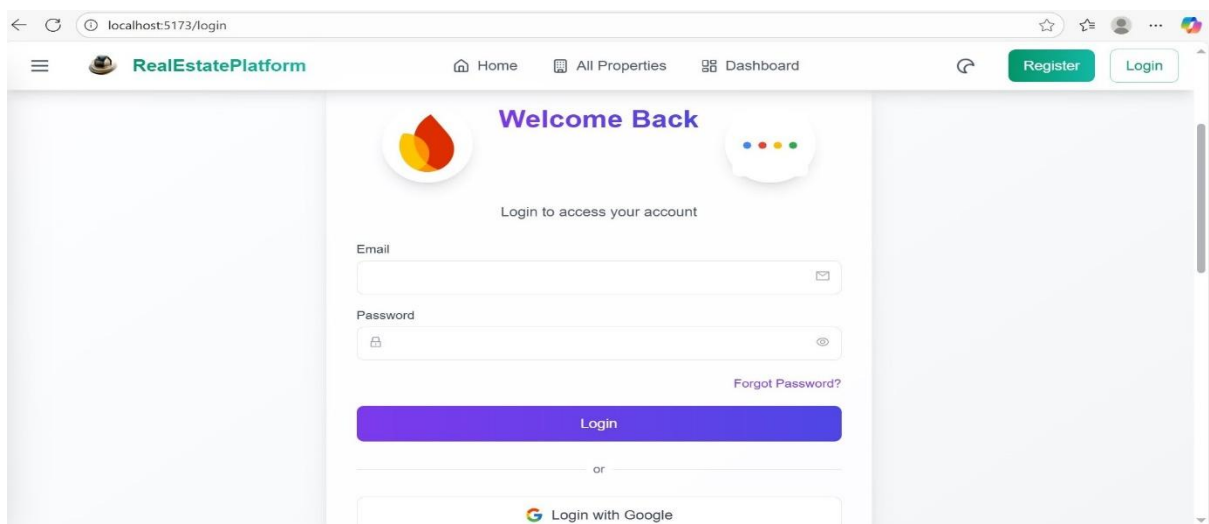


Fig: Login Page

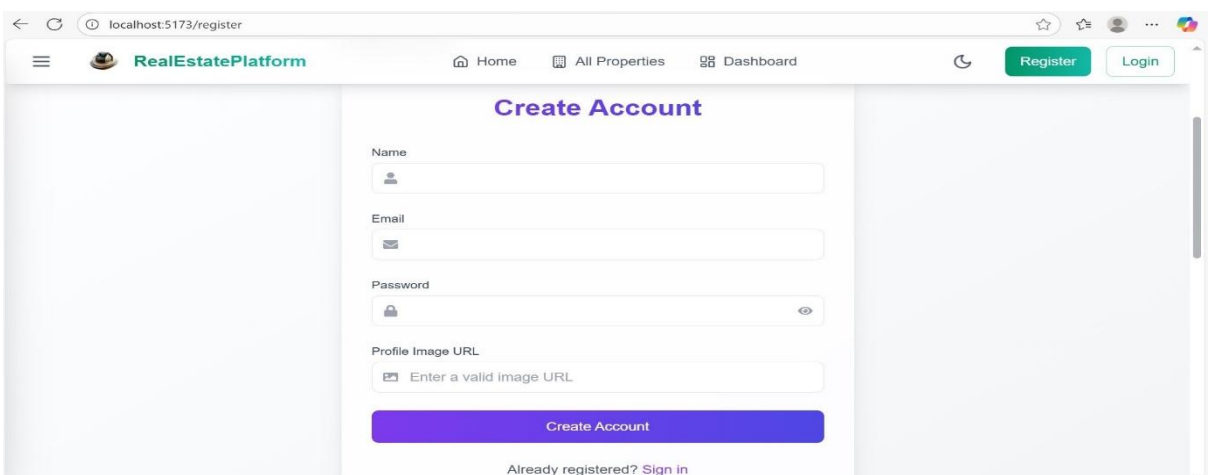


Fig: Registration Page

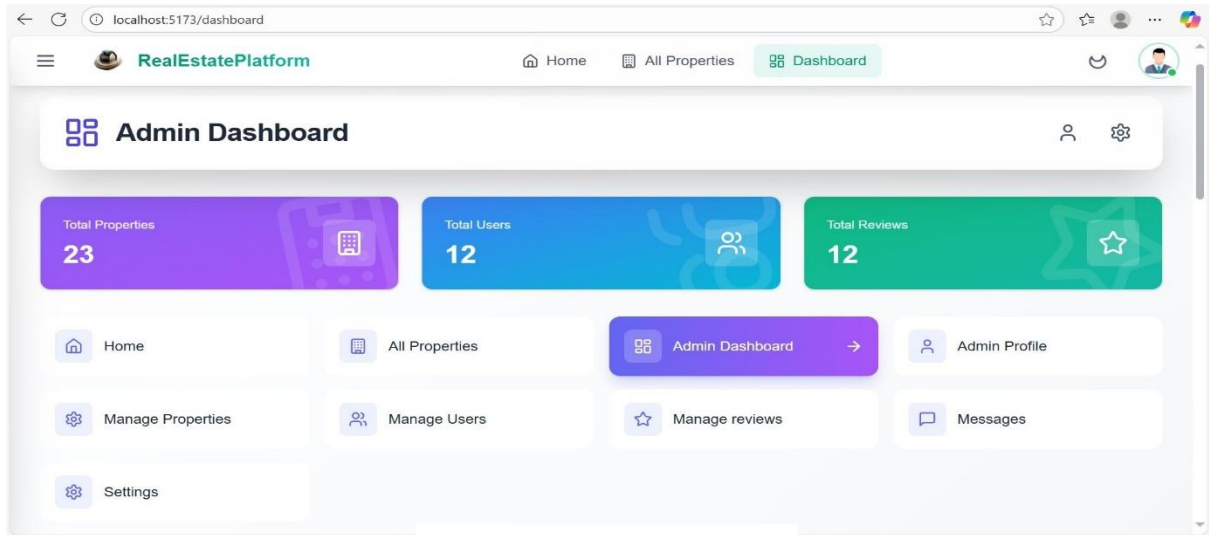


Fig: Admin Dashboard

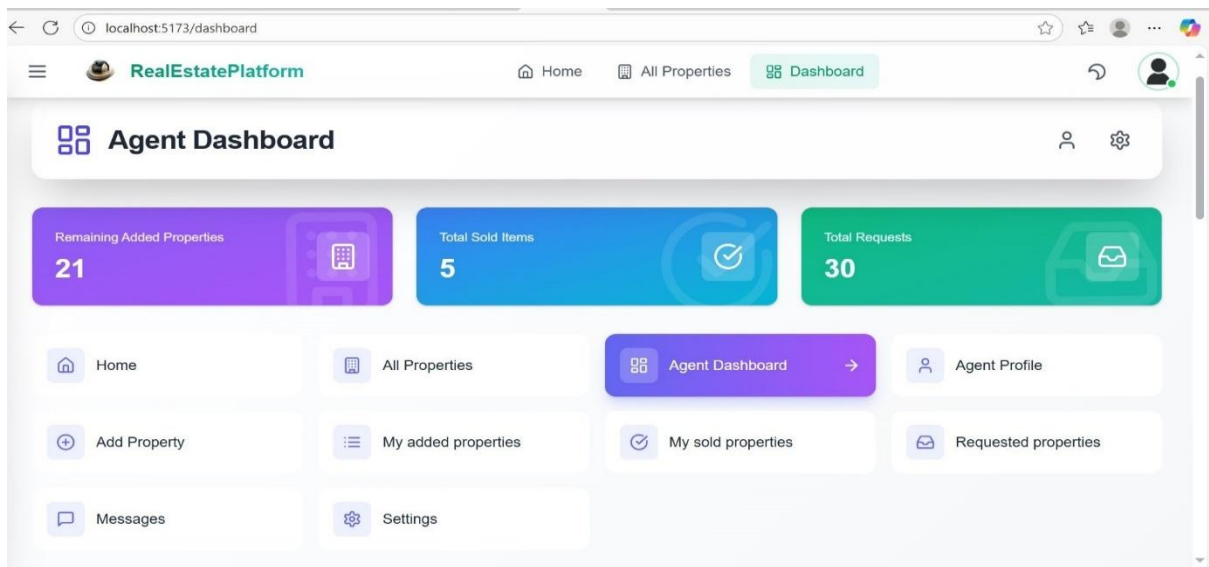


Fig: Agent Dashboard

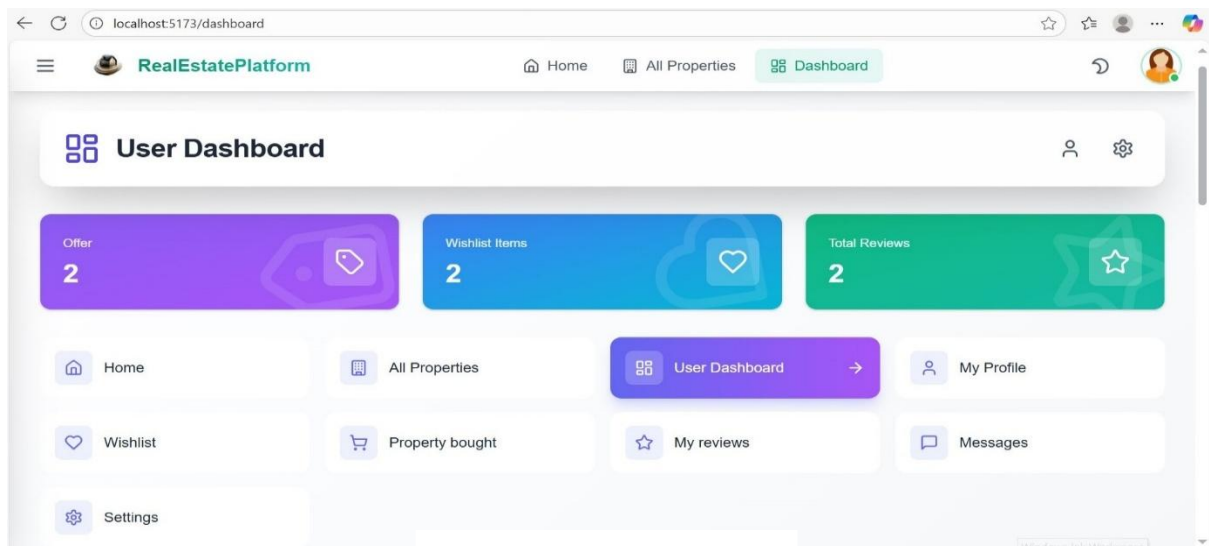


Fig: User Dashboard

9. Testing

9.1 Test Plan

The goal of testing was to make sure that all main features of the real estate platform worked properly for all user roles (admin, agent, buyer). We tested things like login, property add/view, wishlist, and payment. We also checked user access control, design responsiveness on different devices, and connection between frontend and backend.

Testing was done step-by-step during development using manual testing and debugging tools. Each feature was tested after development and bugs were fixed along the way.

9.2 Types of Testing Done

Unit Testing:

Each function or component was tested alone to make sure it works correctly by itself.

Integration Testing:

Different parts of the system, like frontend and backend, were tested together to see if they communicate and work well as a group.

System Testing:

The entire application was tested from start to finish to verify that all features work correctly in real use.

Acceptance Testing:

We checked if the final system met the user requirements and behaved as expected before deployment.

9.3 Sample Test Cases

Test Case ID	What Was Tested	Input	Expected Result	Result
TC-001	Login with correct details	Email: <u>admin@gmail.com</u> Password: Admin1234	Goes to Admin Dashboard	Passed
TC-002	Login with wrong password	Email: <u>agent@gmail.com</u> Password: wrongpass	Shows “Invalid Credentials”	Passed
TC-003	Add new property as Agent	Filled-up correct form	Property added and shown	Passed
TC-004	Payment using Stripe	Valid payment info	Goes to payment success page	Passed
TC-005	Wrong role dashboard access	Buyer tries to open Agent page	Access denied or redirected	Passed

9.4 Bug Log

Bug ID	Problem Found	Fixed?	Fixed During
B-001	Property images didn't show on Firefox	Yes	Sprint 4
B-002	Stripe checkout error for zero payment amount	No	Still unresolved
B-003	Buyer could open Agent-only pages	Yes	Sprint 5
B-004	UI broke on small mobile screens	Yes	Sprint 9
B-005	Wishlist not updating in database	Yes	Sprint 7
B-006	Duplicate entries created on form resubmission	Yes	Sprint 8

9.5 Final Testing Result

All 5 test cases passed successfully. Testing covered user authentication, property listing, payment, and access control. The system behaved as expected in all tested scenarios. From the 6 reported bugs, 5 were fixed, and 1 issue (B-002) related to Stripe payment error for zero amount is still unresolved. It will be fixed in a future sprint.

Overall, the system is stable, secure, and ready for deployment.

10. Deployment

This section describes how the system was made live, the tools used, and any deployment-related issues.

10.1 Deployment Environment

- Frontend Hosting: Firebase
- Backend: Not deployed (currently runs on local server)
- Database: MongoDB Atlas (cloud-based)

10.2 Deployment Steps

A. Frontend Deployment (Firebase)

1. Project was built using: `npm run build`
2. Firebase CLI was initialized with: `firebase init`
3. Firebase project was linked and deployed using: `firebase deploy`
4. Added `firebase.json` file to handle SPA routing.

B. Backend (Local)

- Backend server is currently tested and run locally using: `npm run dev`
- Backend requires `.env` file for environment variables (MongoDB URI, JWT, Stripe).

C. Database

- MongoDB Atlas is used.
- Collections were created for users, properties, wishlist, offers and payments.

10.3 Installation Guide (Local)

1. Clone the project: `git clone https://github.com/therealestate`
`cd therealestate`
2. Install required packages: `npm install`
3. Set environment variables in `.env` file:
 - `MONGO_URI=...`
 - `JWT_SECRET=...`
 - `STRIPE_SECRET=...`
4. Run frontend and backend servers: `npm run dev`

10.4 Deployment Challenges

- Firebase required additional setup for routing.
- Stripe keys needed careful management to avoid exposure.
- CORS issues between local backend and Firebase frontend were handled by backend CORS configuration.

11. Maintenance

11.1 Types of Maintenance Expected

Corrective Maintenance:

This involves fixing any bugs or defects discovered after deployment. These may include unexpected crashes, errors in data processing, or UI issues that affect user experience. Prompt corrective actions ensure the Real Estate platform remains stable and reliable for all users.

Adaptive Maintenance:

As external technologies and environments evolve, the platform must adapt accordingly. This includes updating integrations with services like Stripe and Firebase, ensuring compatibility with new browser versions, and adjusting to changes in device capabilities to maintain seamless operation.

Perfective Maintenance:

To continuously improve user satisfaction, perfective maintenance focuses on enhancing the platform's features and performance. This can involve refining the UI/UX design, optimizing speed, and adding new functionalities based on user feedback and changing market demands.

Preventive Maintenance:

Preventive activities aim to avoid future problems by regularly updating software dependencies, applying security patches, and monitoring system health. This proactive approach helps maintain security standards and reduces the risk of unexpected failures.

11.2 Future Upgrade Plan

The project has several planned upgrades aimed at expanding functionality and improving user experience. Developing mobile applications for iOS and Android is a key future goal to increase accessibility. Additionally, the introduction of advanced search filters and AI-powered recommendations will help users find properties more efficiently. The platform will also incorporate notification systems, including email and push alerts, to keep users informed about property updates and purchase confirmations. Multi-language support is planned to broaden the platform's reach internationally. Enhancing payment options by integrating additional gateways and offering installment plans will provide greater flexibility for buyers. Lastly, an analytics dashboard for agents and administrators will offer valuable insights into user engagement and sales performance.

11.3 Known Issues or Limitations

Despite its strong feature set, the Real Estate platform has some known limitations. Currently, the platform does not support offline usage, which requires users to have a stable internet connection at all times. Scalability may become a challenge as the user base grows, potentially requiring backend improvements to handle higher loads. Occasional payment gateway errors have been observed in rare edge cases, indicating a need for further robustness in transaction handling. The user interface, while functional and responsive, could benefit from more modern and intuitive design enhancements. Additionally, the platform's role management system is limited to three roles—Buyer, Agent, and Admin and may need finer permission controls for larger organizations or more complex workflows.

12. Result and Discussion

12.1 Project Outcome

The Real Estate platform was successfully developed and deployed as a fully functional real estate application. It effectively connects buyers, agents, and administrators through role-based dashboards, offering a seamless experience for browsing, listing, managing, and purchasing properties. The integration of secure authentication and payment systems ensures a trustworthy and smooth transaction process. The project was delivered on schedule, meeting all initial requirements and providing a scalable foundation for future enhancements.

12.2 Key Functionalities Achieved

- **Role-Based Access Control:** Distinct dashboards and permissions for Buyers, Agents, and Admins to maintain security and clarity of functions.
- **Property Listings Management:** Agents and Admins can add, edit, and delete property listings efficiently.
- **Wishlist and Purchase Features:** Buyers can save favorite properties and complete purchases via Stripe integration.
- **Secure Authentication:** Combined Firebase and JWT-based authentication ensures secure login and data protection.
- **Responsive User Interface:** The platform adapts flawlessly across devices, providing an optimal experience on both desktop and mobile.
- **Optimized Data Fetching:** Utilization of TanStack Query improves frontend performance and user experience.

12.3 Performance and Limitations

The platform performs reliably under typical user loads, with fast response times and smooth UI interactions. However, there are some limitations to consider:

- **Scalability:** Current backend architecture is suitable for moderate traffic but may require optimization or scaling strategies for high user volumes.
- **Offline Usage:** The application lacks offline support, requiring continuous internet connectivity.
- **UI Enhancements:** While functional and responsive, the user interface could be further refined for a more modern and intuitive feel.
- **Role Granularity:** The system currently supports three user roles; more granular permission settings could benefit larger organizations.
- **Edge Case Handling:** Rare payment gateway or API errors need more robust handling to ensure complete fault tolerance.

13. Challenges Faced

13.1 Technical Challenges

1.Payment Integration:

Implementing Stripe for real-time payment processing was initially complex, especially handling payment intents securely. To resolve this, detailed documentation was followed, and test environments were used extensively before deployment.

2.Role-Based Access Management:

Ensuring different user experiences for Buyers, Agents, and Admins through protected routes and dashboards posed architectural difficulties. The team resolved this using JWT-based token validation and conditional rendering based on role data fetched from the backend.

3.Firebase & JWT Syncing:

Combining Firebase Authentication with custom JWT tokens for API authorization was tricky. After researching best practices, middleware was implemented in the backend to validate Firebase tokens and issue secure JWTs.

4.Backend Deployment:

Railway deployment introduced issues related to environment variables and database connection timeouts. The team resolved these by adjusting .env configuration and monitoring server logs to troubleshoot effectively.

13.2 Team-Related Challenges

1.Task Distribution and Deadlines:

Coordinating schedules among 5 members with varying availability was a hurdle. This was managed using Trello boards and weekly progress check-ins to ensure clarity and accountability.

2.Version Control Conflicts:

While using Git, multiple pull requests caused merge conflicts, especially during UI updates. Team members resolved this by clearly communicating code responsibilities and conducting regular Git merges with conflict resolution meetings.

3.Communication Gaps:

Initial misunderstandings around certain feature implementations caused delays. These were addressed through quick team meetings (via Discord/Google Meet) and better documentation in shared docs.

13.3 Summary

Despite facing several technical and coordination-related hurdles, the team handled them effectively through collaboration, clear communication, and proactive research. These experiences significantly contributed to the learning outcomes of the project.

14. Personal Reflection & Learning Outcomes

14.1 What Was Learned by Each Team Member

Throughout the development of the Real Estate platform, each team member gained valuable knowledge and hands-on experience. The team leader, who also handled the frontend, learned to manage a full-fledged React application by organizing component hierarchies, implementing dynamic routing, and utilizing global state management through TanStack Query. The backend developer improved their skills in creating and securing RESTful APIs, designing MongoDB schemas, and integrating Firebase authentication combined with JWT-based user access control. The UI/UX designer concentrated on mastering responsive design techniques using Tailwind CSS and crafted user-friendly dashboard interfaces with a focus on usability. Another member, working across both frontend and backend, tackled full-stack challenges such as connecting APIs to the frontend, implementing Stripe payment integration, and managing role-based authorization. The team member responsible for testing and deployment grew more confident in bug tracking, version control workflows, and deploying the frontend to Netlify and the backend to Railway, while overcoming common deployment challenges.

14.2 Technical Knowledge Gained

The project offered extensive practical experience with the MERN stack, particularly in handling real-world functionalities like authentication, role-based dashboards, and payment processing. The team became proficient in React.js, including the use of hooks and asynchronous data fetching. On the backend, they enhanced their understanding of Express.js routing, MongoDB data modeling, and API security. Integrating Firebase authentication and Stripe payment systems expanded their knowledge of third-party service integration. Additionally, deploying the application on Netlify and Railway provided valuable insights into environment configuration, build optimization, and live production testing.

14.3 Teamwork, Communication & Problem-Solving Reflection

This project underscored the critical role of teamwork and communication. Working remotely required the team to consistently use collaboration tools like GitHub, Trello, and Google Meet. They learned to break down complex tasks into manageable components, resolve version control conflicts, and support each other when technical obstacles arose. Regular stand-up meetings kept everyone aligned on progress, while brainstorming sessions fostered creative problem-solving. Difficult moments, such as debugging deployment errors or synchronizing frontend and backend work, were overcome through collective effort, demonstrating strong cooperation and responsibility within the team.

14.4 Improvement Areas

Despite successful completion, the project revealed areas where improvement is possible. More detailed initial planning with clearer timelines and task assignments could have helped avoid delays. Documentation of APIs and workflows was inconsistent, making some onboarding and troubleshooting processes more difficult. Although manual testing was effective, the lack of automated tests and detailed logging made bug detection challenging at times. Finally, time management in the final stages, especially during deployment, proved difficult. These lessons are valuable for structuring future projects more efficiently and sustainably.

15. Conclusion

15.1 Overall Summary of the Project Outcome

The Real Estate platform was successfully developed and deployed as a comprehensive solution for property buyers, agents, and administrators. It offers role-based dashboards, secure authentication, efficient property management, wishlist functionality, and integrated Stripe payments. The system provides a seamless user experience across devices and lays a solid foundation for future growth.

15.2 Were the Objectives Met?

Yes, all the primary objectives were met. The team implemented the core features outlined in the project plan, including role-based access control, responsive UI, secure payment processing, and backend API integration. The project was delivered on time with a stable and functional platform, fulfilling the initial requirements.

15.3 What Went Well / Could Be Improved?

The project benefited from effective team collaboration, clear communication, and thorough testing, which contributed to smooth development and deployment. The use of modern technologies like React, Firebase, and Stripe was successful in delivering a high-quality product. Areas for improvement include more detailed initial planning, better documentation practices, the introduction of automated testing, and enhancements in UI design to further improve usability and scalability.

16. Future Work

16.1 Suggested Improvements

In future versions, the Real Estate platform can be enhanced by refining the user interface with more intuitive layouts and smoother interactions. Improving error handling and providing clearer loading indicators would contribute to a better user experience. Additionally, implementing real-time data updates and optimizing API performance will make the platform more responsive and efficient.

16.2 Additional Features

Introducing new features like property reviews and ratings, live chat between buyers and agents, and map-based property search would increase interactivity and user trust. The addition of notifications for property visits, approval statuses, and personalized recommendations could further boost user engagement.

16.3 Scaling Possibilities

To accommodate growing user numbers and increased traffic, the backend can be optimized through load balancing and caching strategies. Migrating to a cloud-based infrastructure and incorporating server-side rendering may improve scalability and performance. Developing a mobile app using React Native would also expand accessibility to a wider audience.

17 References

- W3Schools. (n.d.). *HTML Tutorial*. Retrieved from <https://www.w3schools.com/html/>
- MongoDB. (n.d.). *MongoDB Documentation*. Retrieved from <https://www.mongodb.com/docs/>
- React. (n.d.). *React – A JavaScript library for building user interfaces*. Retrieved from <https://react.dev/>
- Stripe. (n.d.). *Online Payment Processing for Internet Businesses*. Retrieved from <https://stripe.com/docs>
- Node.js. (n.d.). *Node.js Documentation*. Retrieved from <https://nodejs.org/en/docs/>
- Express.js. (n.d.). *Express - Node.js web application framework*. Retrieved from <https://expressjs.com/>
- GitHub Repository - *Real Estate Project*.