CS6P05ES – Final Project

BEng (Hons) in Software Engineering

BlockEstate – Blockchain Powered Real Estate Platform

Final Report

Coursework Details:

Name : Ushan Kaushalya Warnakulasooriya

Student ID : E229299

Program : BEng (Hons) in Software Engineering – London Met

Module : CS6P05ES – Final Project

Coursework : Final Report

Date Submitted : 25/04/2025

Abstract

Real estate investment has always been an aspiration for many, but it's a feat only a small number of individuals can manage. The process is expensive, complicated, and filled with legal procedures and risks. BlockEstate is a venture that wants to tear down these barriers, and it provides a solution where anyone, regardless of geographical location, can quickly invest in real estate without the necessity to purchase a whole property. A share of a property is the idea. In return, the investor will get the rental, rather than the whole amount being purchased by a single person.

For extra security, this system uses the Algorand blockchain technology, to handle digital assets in a way that is safe and easily accessible to all. Key issues like trust deficiency, fraud, and undisclosed matters in conventional real property transactions are thus resolved. In it, the MERN stack, together with Firebase, is used for a secured infrastructure of the project through user authentication and media storage. The platform involves three primary roles and assigns different tasks to each one: investors (who buy shares); agencies (who list properties after admin approval); and system admins (who manage and verify the system).

Customers are eligible to sign up manually or by integrating Google and Apple accounts. Upon the completion of the process, the verified agencies can enlist a new listing, and likewise, investors can search for the properties and invest if they are interested. A soft update of this system is meant for users to be able to navigate easily and in addition to secure and complete backend processes that will take care of security, data management, and role-dependent access.

BlockEstate is a scheme that is intended to make property investment straightforward, secure, and inclusive. The approach focuses on dismantling the obstacles of cost, location, and complexity while it also installs a paperless system making the process very easy to the users and trustworthy. In doing so, the project would also like to shift people's popular belief about real estate and make it possible for everyone, not just a relatively small number to invest.

Table of Contents

Table of Figures

Table of Tables

Abbreviations

1. Chapter 01 – Introduction

BlockEstate is a fresh and innovative real estate investment platform that is based on blockchain technology to deal with real life problems in the property market. To this end, we can say that BlockEstate is a solution that helps in solving these types of problems because among other activities.

Investing in property, in general, needs a large amount of money, geographical problems, legal work and trust in many different people. So, due to the above-mentioned reasons, most ordinary people are not capable of investing that money or do not dare to do so. BlockEstate comes up with a better solution because through its people can invest in property shares not complete properties and use a simple and safe online platform. Such a concept implies that one can advance some money and receive rental income without going through the stress phase of being the owner of a whole building.

This platform has been empowered by the Algorand blockchain for the purpose of effecting management of property shares as well as making all transactions transparent, safe and unchangeable. The registration and signing in can be done through Google or Apple accounts so that the access may be quick and uncomplicated.

The whole project has a well-documented workflow with the MERN stack. Initially, the UI/UX designs were finished, and then, the frontend was developed with the help of React. The backend was built using Express.js with a monolithic structure. User authentication and token management are satisfied by using Firebase. The platform is intended to offer various roles such as normal users, agencies, and system admins.

BlockEstate is the tool that has been designed to facilitate the process for anyone who wants to become a real estate investor, from every part of the world, and it does not matter if a person is pro in it. BlockEstate is a technological solution and everyone, not only the wealthy, in any part of the globe, can invest in a property and earn more money.

1.1 Goals

The primary aim of the BlockEstate project is to remove the obstacles in traditional real estate investment by letting even non-wealthy investors to invest in shares of real property through a blockchain-driven system. Instead of being required to have a large amount of money to purchase the entire property, users can perform the transaction of a small stake in the property by means of the secure and transparent digital transactions of the Algorand blockchain. This, in turn, not only democratizes the real estate investment that becomes more accessible to average people but also decreases the fraud risk.

A key goal of the BlockEstate project is to eliminate geographical limitations and make high-value property investment accessible to everyone. Traditionally, investing in properties located in premium areas especially those with strong foreign interest or tourism appeal—has only been possible for locals or those with significant resources. With BlockEstate, even individuals from completely different regions can invest in such properties and earn valuable rental income. This opens the door for more inclusive participation in profitable real estate markets and allows people to benefit from rental opportunities in top-tier locations, regardless of where they live.

Moreover, the platform aspires to be the most convenient and reliable tool for both investors and real estate agencies. Through a user-friendly and intuitive interface, investors will be able to browse the offered properties, know relevant details, as well as conduct shares and rental income management without hustle. On the one side, real estate agencies can perform such activities as they list the properties and carry them through the process of sale after being authorized by the system. The platform, furthermore, makes it its business to automatically handle/transfer distributions of rental income and lets customers not only transfer shares but also sell them.

All in all, the focus of BlockEstate is to make real estate investing effortless, worldwide, and secure for anyone, including newcomers to the property market.

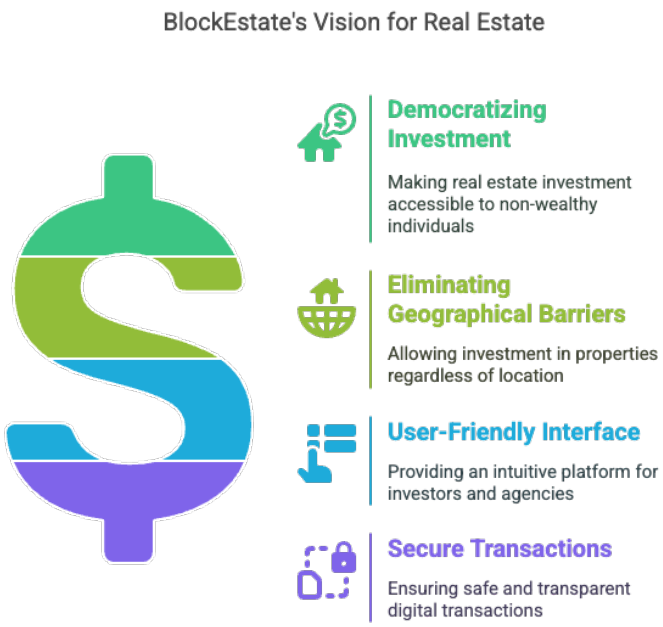

Figure 1 – Goals

1.2 Motivation

Investing in real estate is a common way for people to increase their wealth over time but for most people, quite a new concept, especially to get started in the field. Typically, buying property involves a big investment and the whole process is quite confusing as well as full of legal things to do or steps to take, middlemen and risks. In some cases, people even come across the kind of properties, which they wish to buy just due to the difficulty of the same, especially when the said people are not physically present in the target area or not even in the same country. These obstacles create a perception in the large public's minds that real estate is off-limits for them and especially for the small investors.

In many areas, property owners can gain a steady source of income by renting out their properties to people from all over the globe. There are, however, some areas which attract more foreign travelers and thus rent out more. But if a site has a property and is not easily accessible or not known by many, it may not be profitable. Therefore, a large proportion of the best homes remain latent. That's when a solution came into my mind – a solution that would link people from anywhere to the rest of the world to invest or rent. Among these people are those who are keen to acquire a rental return on their investment and at the same time increase their saving while on the

other hand; there are others who would like to sell a part of their property to get the money. The two groups are affected in the same way by the current system.

The main reason started BlockEstate was to make property investment easier, solve geographical problems, more secure, and more inclusive by using blockchain. With this application, people can invest in fractional ownership of property wherever they may be without being in a financial tight corner or worrying about fraud. It is my goal to ensure that regardless of whether one is a local or a citizen living abroad, one can participate in property investment and enjoy the benefits of rental income without all the usual complications and limitations.

1.3 Method

To go about developing the BlockEstate project, adhered to a very systematic approach that involved design and planning, followed by development, version control, and testing.

As a first step, sketched out the UI/UX of the system on Figma. Doing so helped me to clarify the platform's structure, user flow, and features and to get a visual picture of these elements before. Moreover, it was simply a saving of time during the development phase as had already got a clear vision from there.

After the UI/UX designing, it was time to dig into Algorand blockchain which preferred to the one for the handling of digital property shares. The research was on the underlying technology of the Algorand platform, and the confirmation of the information based on features such as fast transactions, low gas fees, and security. This knowledge was beneficial in my effort to establish the connection between blockchain features. However, the activity was prescribed in accordance with the Web platform in its transition process.

1.3.1 Waterfall Model

After planned and researched, started the development phase by MERN stack (MongoDB, Express, React, Node.js). Choose the Waterfall model for this project because it was well aligned with the project requirements and the plan was detailed from the start. It enabled me to concentrate on each stage singularly without distractions.

1.3.1.1 Benefits of Waterfall Model

- Clear structure and timeline
- Easy to manage stages
- Good for solo developers like me, who develop step by step

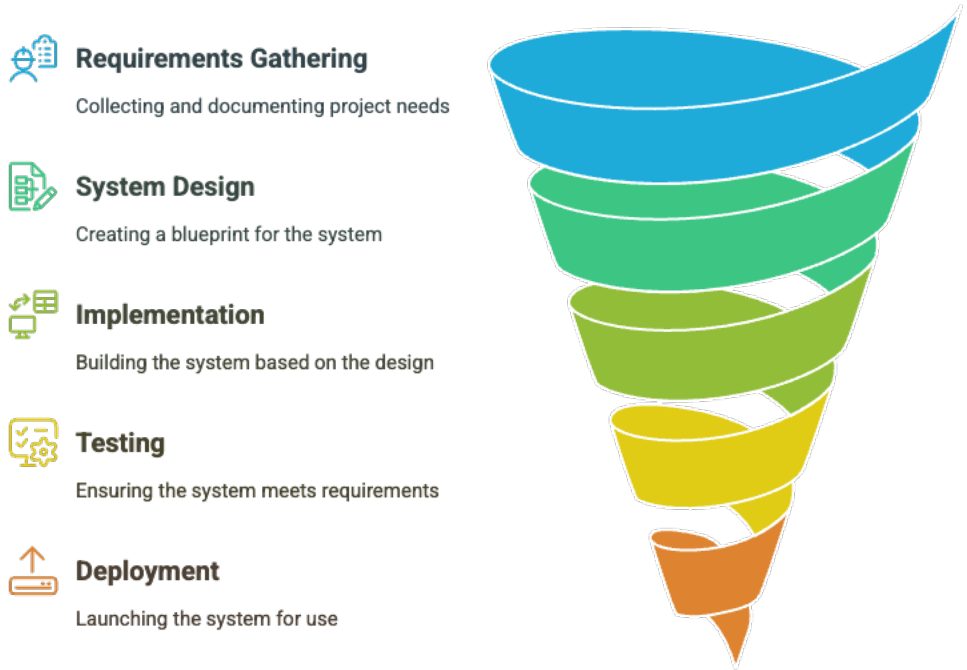

Figure 2 - Waterfall Model

As for the front end, worked in React and used the component-based architecture. This helped me in writing the code clean and making it reusable. Additionally, added a light and dark theme toggle to allow the users to choose the style that user want.

1.3.2 Monolithic Architecture

Once the UI development phase was finished, began work on the backend part by using Express.js and implemented the monolithic architecture. This approach implies that a single, consolidated application contains all the backend logic (APIs, database models, authentication, and so forth).

1.3.2.1 Benefits of Monolithic Architecture

- Easy and faster develop for solo developers
- Easier and debug since everything is in one place
- Better performance for small and medium applications

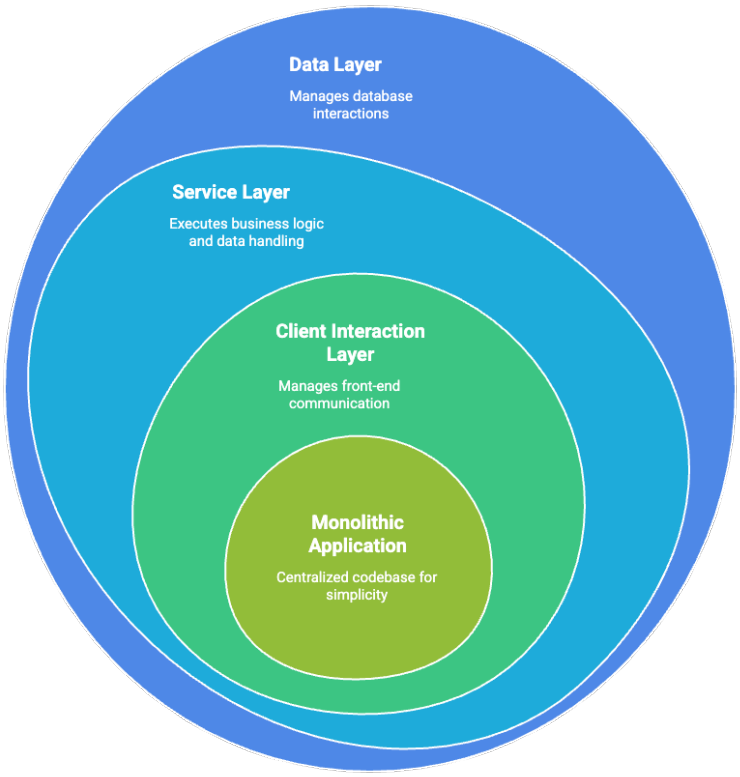

Figure 3 - Monolithic Architecture

1.3.3 MongoDB

Setup MongoDB as the main database and modeled the database schema using Mongoose. User, property, transaction, and token are examples of tables in MongoDB, and the Model Classes also provide the validation to the data.

1.3.4 Firebase

For the identification of users, combined Firebase Authentication. It permits users to securely register and log in user with the possibility to use either email/password or social media logins. Turned to Firebase to produce tokens which would be used for user verification before granting access rights to them. This way, we could manage more simple tasks without compromising security.

1.3.5 GitHub

To stay on track with the project the author has been using Git for version control and GitHub for the organization storage. This way, the author was able to locate the code backup, track, submit changes to it safely, and make changes in the project without any problems. Above all, the author found this service very useful, especially when the author had the opportunity to remember previous functionalities.

1.4 Overview

The platform lets its members invest in real estate property through shared ownership owning only part of the property, thus it's open to everyone, not only professional investors. It's not just about the financial aspect of it but also making the process clear and trustworthy via blockchain technology.

The platform was designed on the MERN stack (MongoDB, Express.js, React, and Node.js) hence it is a good choice for a new modern web application. The user interface was completed with a component-based structure in React and two themes – the dark theme and the light theme to make it simple and joyful for the user. The backend was done in a monolithic architecture which made the application easy to handle and structured logically, especially in the initial stages of the development.

In addition to this, Firebase Authentication is the server to which they registered your presence and checked for identification. The data is arranged in the form of MongoDB model classes which is a system function just like tables are in relational databases. The project was always treated with the help of the Git and GitHub tools which provided the control of the versions and expanded the code stability cycle.

The key technical highlights are,

- UI/UX designed in Figma before development began.
- Followed the Waterfall Model – planning, design, development, and testing done in order.
- Firebase Authentication used for secure token-based login and OTP verification.
- Developed with a component-based frontend structure using React.
- Integrated light/dark mode toggle for better user accessibility.
- MongoDB model classes created to manage and validate structured data.
- Monolithic architecture used for the backend to simplify internal communication and debugging.
- Version control handled using Git and GitHub.

- Conducted research and selected Algorand blockchain for secure and scalable asset management.
- Technical documentation and planning:

1. UML Diagrams – visualizing system use cases and flows.
2. ER Diagram – defining data structure and relationships.
3. System Architecture Diagram – showing how components interact with each other

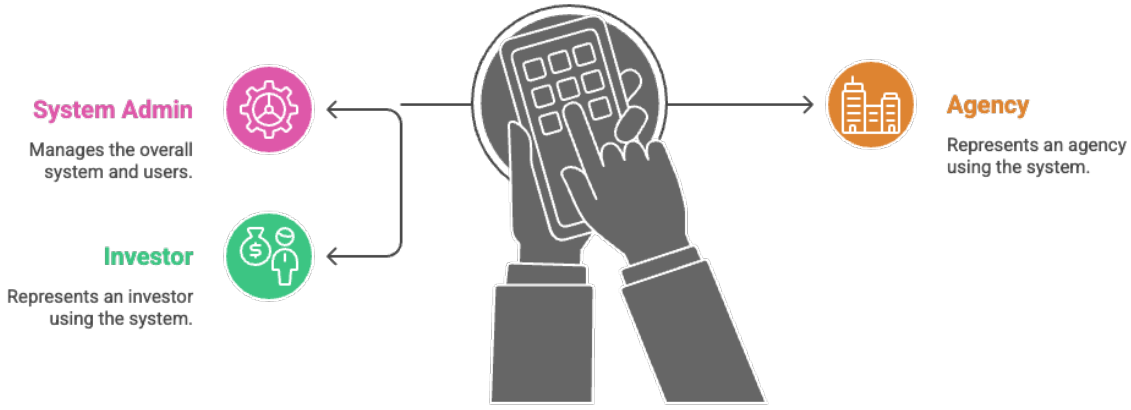

Figure 4 - System Users

2. Chapter 02 – Background and Problem Statement

BlockEstate was inspired by a real-world issue in the real estate industry despite being one of the most trusted and rewarding investment options, it's still inaccessible for many due to high costs, complicated processes, and fraud risks. On the other hand, property owners also face difficulties reaching potential investors, especially beyond their local area. This section continues from the introduction by providing deeper context into this issue, reviewing existing solutions in the space, and identifying the exact problem that BlockEstate is designed to solve. It sets the stage for understanding why this project is important, and how it aims to make real estate investment easier, safer, and more inclusive for everyone.

2.1 Introduction

It's always been known that real estate is a great investment, and therefore people seem to agree, but not all of them consider it simple to start. Among the different challenges is the fact that the purchase of a house comes with a huge cost that most people do not seem to be able to afford. Moreover, the transaction itself is a daunting task, being very time-consuming and entangled with all sorts of legal aspects, paper works, and intermediaries, such as, for instance, agents or lawyers. This kind of situation becomes one of the many reasons why people are turned off from investments. Also, the non-disclosure of information about transactions has an even more negative effect on the situation. The buyers very often are not aware of the history of ownership of the property, the legal disputes, or the presence of hidden fees, and this certainly boosts the chances of fraud or scams.

The idea of this project, BlockEstate came into existence when the founders found out they could solve these real problems. Without a doubt, the aim of this project is to set new standards of real estate transactions by giving people the possibility to become investors of property in simple, secure, and shared ways using the technology of blockchain. Instead of having to buy the entire property, they just need to buy a part and through that, they will be able to enjoy the perks of being a rental owner without the worries carried by the ownership burden. This method of sharing is trustless because the transactions were secured and transparent by using blockchain.

2.2 Literature Review

Before commencing my platform BlockEstate, the author decided to learn about the existing systems and platforms in the real estate domain, particularly those that target the same core problem: making property investment simpler and more accessible. Through this research, several projects were found that have significantly influenced both the direction and inspiration of BlockEstate.

Lofty was one of the first platforms that enticed me. Lofty is set up on blockchain and provides tokenized real estate, which enables the users to become the owners of the revenue-sharing part of the properties. What was quite intriguing for me was the minimalism and the availability it went together with to real estate investors. It was an eye-opener for me regarding the change of the old traditional property ownership to a new model that is more inclusive with blockchain. However, the author has found a major drawback in Lofty: consent agreements do not come with a feature of auto-distribution of rental income. According to my opinion, this means that personal financial contribution is a crucial part of investment in real estate, because regular profit generation is a huge motivator for most of the investors.

Website link - www.lofty.com

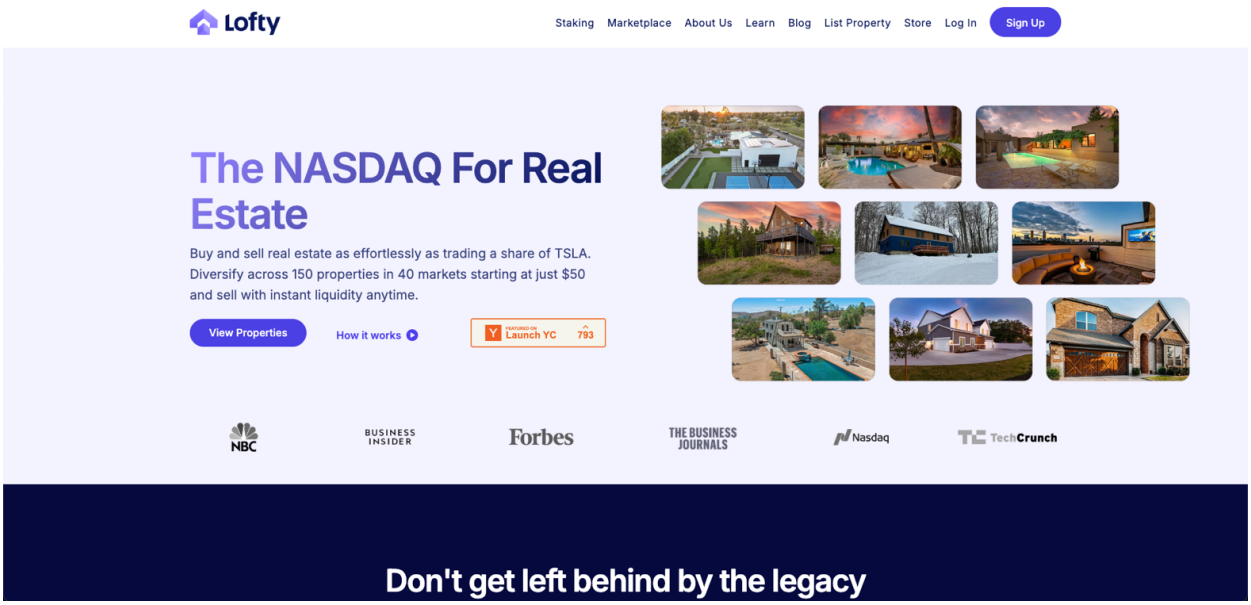

Figure 5 - Lofty Homepage

There was also another platform that gave me the inspiration of Landshare Token. As well as Lofty, it employs blockchain technology to extend the real estate tokenization process. Nevertheless, Landshare has one more feature: it concentrates on the creation of income from real estate and provides the users with the opportunity to receive profit through tokens in a Defi manner. Now the author saw the possibility of fusion between real estate and the new blockchain finance tools, but conversely, there was a feeling of complexity in their model that could be a stumbling block for less technical or new investors. It encouraged me to come up with something simpler and still powerful - a platform that everyone can use with ease and confidence.

Website link - www.landshare.io

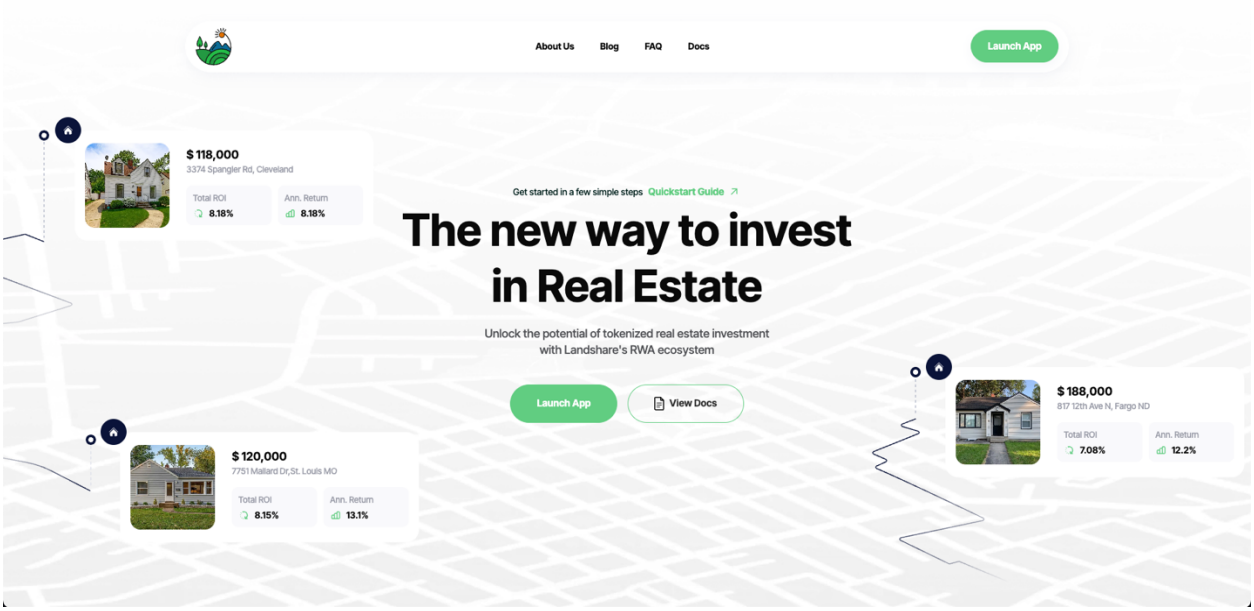

Figure 6 - Landshare Homepage

On the flip side, Mulberry Residence, unlike Factom, is not a blockchain-based centralized real estate market but is more focused on applying traditional methods of property listing buying, and ownership. The fact that Mulberry has a polished, user-friendly interface and still lacks transparency, decentralized control, and fractional ownership options caught my eye. The main debating point features the slick experience as an attractor to users, but on the other hand, also shows the limits of centralization regarding trust and investment flexibility.

Website link – www.mulberryresidence.com

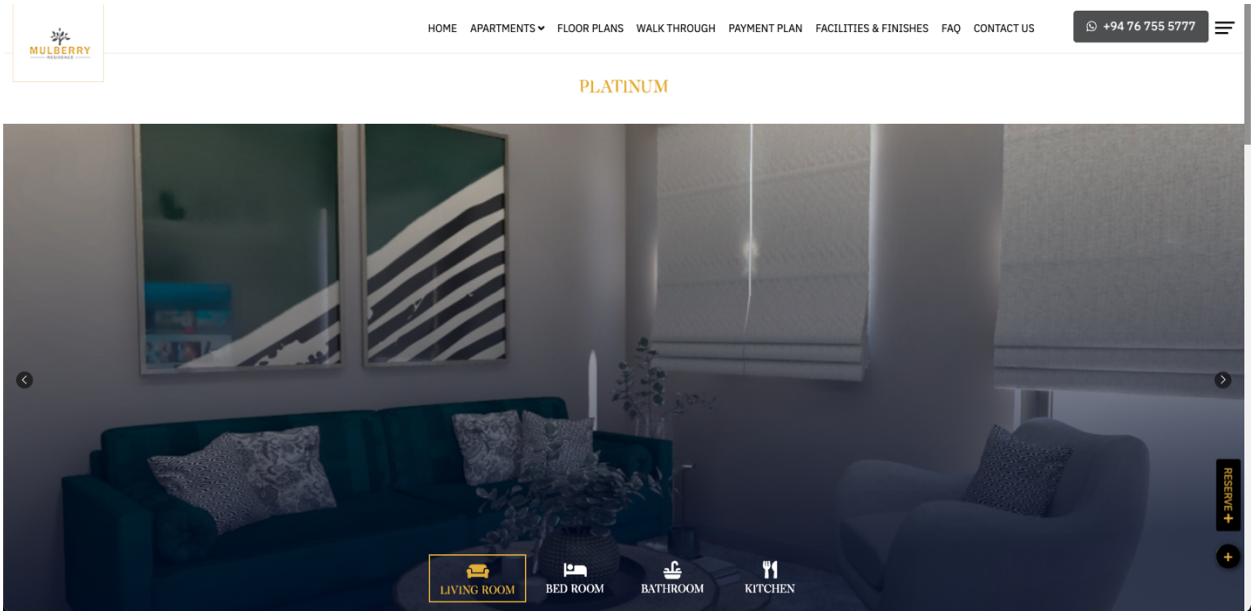

Figure 7 - Mulberry Residence Homepage

Next, the author investigated RealT, a platform based in the United States that employs Ethereum blockchain to tokenize real estate. RealT gives token holders fractional ownership and shares revenue from the properties' weekly rental. What the author found attractive at RealT is its ability to elegantly fuse the legal and blockchain aspects to create a frictionless investment mechanism. Though mainly concentrated in one country, RealT, however, mainly targets the US market where the platform operates under strict laws, a factor that might not facilitate the system to be sustainable or relevant in countries like Sri Lanka.

Website link – www. realt.co

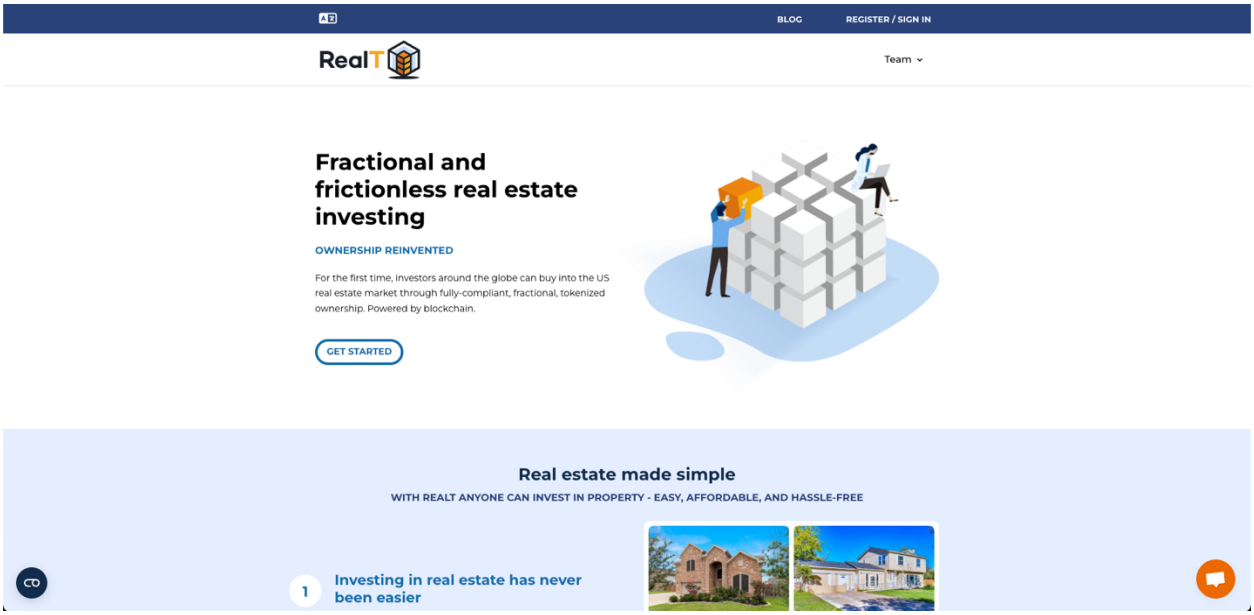

Figure 8 - RealT Homepage

Prime Lands being a thriving real estate company in Sri Lanka, not only introduced me to land parting in practice but also taught me about savvy promotion skills by the locals of the property. They are a classic example of an organization that focuses solely on the development of the lands, land division, marketing of the land, and carrying it out the manual way without any use of the blockchain. Seeing the way they go through their methods; the author understood the significance of making property on the local level both available and attractive to the regular folks. As such, the manual division approach gave me a hint of how the land in the real world would be divided while the same principles of blockchain share digital assets would be used, thus enabling anyone to become a landlord without having to own the entire asset.

All these platforms brought valuable ideas that defined the BlockEstate project. Although all of them have their good sides, some of them also showed that there were some weaknesses like the inability to share rental income, complexity and less market coverage. BlockEstate aspires to fill those gaps by introducing a simple and easy to use blockchain-powered link that will give the investors the intersectional ownership, rental income distribution, transparent tracking of the investments and allow the users from any part of the world to invest in the properties that are high in demand or those with massive tourist flows. It is crucial to mention that there was no rental income sharing.

2.3 Problem Statement

For a long time now, putting money into real estate has been acknowledged as the most dependable way of making some nice gains but not everyone is as lucky as to be able to play in this field. One of the most serious obstacles that prevent small investors from entering the real estate market is the high cost of purchasing a property. Subsequently, the more demanding part of people is to buy a place of their own and this is the point at which it becomes almost impossible for small investors to make their entry. Along with that, the conventional property buying procedure is entangled with a lot of legal issues, and paperwork and it, moreover, is a very time-consuming job that most of the potential buyers want to avoid. Even if someday, they could buy a property, the management responsibilities of establishing it, finding renters, and keeping it in a good state will finally be a load on their shoulders, particularly for the inexperienced ones.

One of the major concerns in real estate is that residents of low-value or rural areas rarely become aware of the profitable investment opportunities. Although these areas could be excellent sources of rental income, particularly from abroad, such places are inaccessible due to the constraining geographical and legal factors, and therefore the high rental potential remains untapped. The need for a solution is eminent to provide people with the possibilities of investment and the corresponding properties irrespective of their location.

Another critical reason is the opaqueness and absence of reliability in transactions regarding real estate. It is only very often that the buyer is not told exactly the property's history of ownership, legal status, or the conditions under which the property was purchased that have not yet been officially hidden. There is a great risk of forging, tampering, and fraud because of the lack of regulations, too many scams, and too many counterfeit documents, which may lead to monetary losses for the victim. Furthermore, there is no such a way that allows easy and secure investing in property as well as earning a part of the property's revenue if it is in place. The existing fractional ownership platforms are unsatisfactory since they still have some shortcomings as to becoming a trustable, secure, and user-friendly system. Thus, there is a real necessity for an organized and decentralized platform such as BlockEstate, that can assure the investment of a part of a property in a trustworthy way. It is good to mention that this is a blockchain premise which besides the fractional ownership enables real estate transactions to be highly protected, automated and open to the public. This basically nullifies all the previous obstacles that the common man used to face.

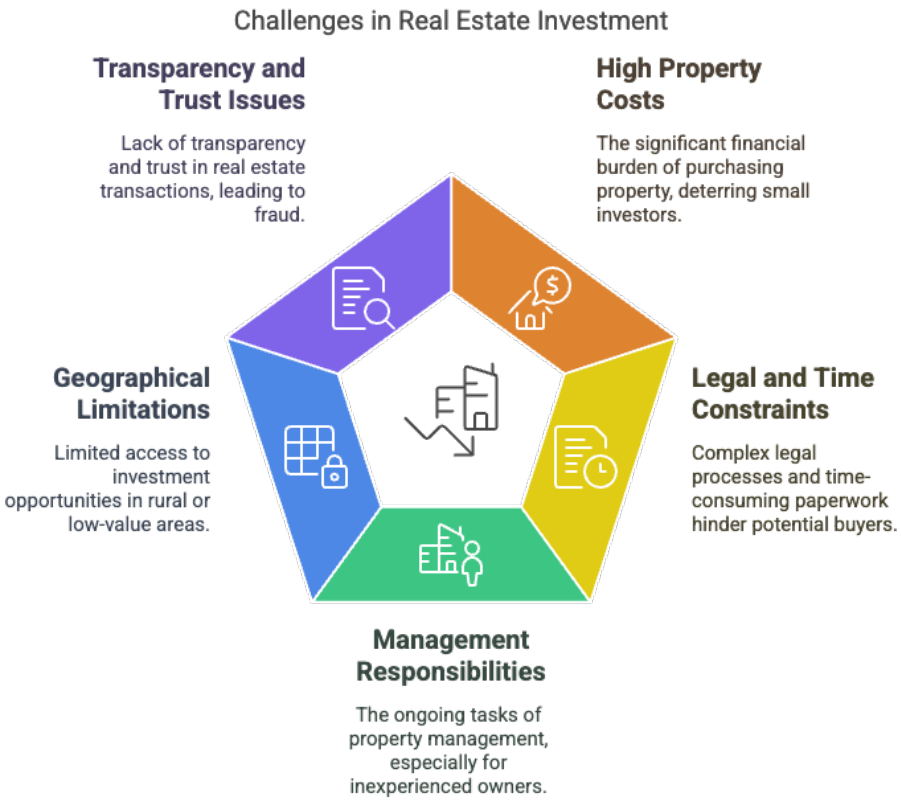

Figure 9 - Problems Statement

3. Chapter 03 – Project Management

The effective completion of any software project is primarily based on the quality of its planning, organization, and management. This chapter sets out the main project management strategies that were employed while developing BlockEstate. It explains the methods used to break down the work, control time, keep up with how things are moving, and to see that all the necessary tasks are performed in a timely fashion and without much waste. In this part, we will go through the topics like the way the development was conducted, the version control tools that were used, and how tasks were managed and scheduled at different stages of the project. The intention was to give a complete view of the process of the project from the stage of the idea to the actual implementation in a logical and well-arranged manner.

3.1 Approach

To ensure that BlockEstate project was completed successfully, the author used a systematic and methodological approach that helped me utilize the resources most efficiently right from the design phase to the development phase. Being the sole participator in the project, the author first set out to explicitly process the entire project by going through the stages of requirement analysis, UI/UX design, and development. the author followed the Waterfall model as my main software development methodology because it provided a crystal-clear structure, where each phase had defined outputs before moving to the next. This model was most useful to me because of its capability to coordinate the project tasks, particularly since the scope and features were easily known at the beginning of the project. By using the version control system through Git and GitHub, the author was able to handle changes, maintain code history, and keep the development process safe, and in good condition. The second step was to further break the tasks down into smaller parts, which ensured it to be less cumbersome and easier to monitor the project's progress. This method enabled me to stay focused, cut down rework, and carry on with the platform in an easy and effective way.

3.1.1 Project Information

This project is intended to take away the common barriers of real estate investment such as a lot of money, difficult legal procedures and risks of being cheated. The platform, engineered using the MERN stack with Algorand blockchain integration, is a tool that supplies the user with a new, easy, and clear way to acquire, sell, and get rental income on a single property that is not entirely for personal use.

Topic	**Description**
Project Name	BlockEstate – Blockchain-based Real Estate Investment Platform
Project Objectives	• Allow fractional ownership of properties • Enable rental income sharing • Reduce fraud and increase transparency • Simplify investment process through tech
Duration	January 2025 – May 2025 (5 months)
User Roles	• Investor: Buys property shares, earns rental • Agency: Lists property, manages assets • Admin: Handles user management, verifies properties
Highlights	• Built with MERN Stack • Integrated Algorand blockchain • Firebase Authentication • Light/Dark Theme toggle • Fully responsive frontend • Git-based version control

Table 1 - Project Information

3.1.2 Work Breakdown Structure (WBS)

1. **Project Initialization - Starting point of the project**
   - 1.1 Requirement Gathering
     - 1.1.1 Understand requirements
     - 1.1.2 Define use cases
   - 1.2 Feasibility Study
     - 1.2.1 Evaluate technical, financial, and operational feasibility
   - 1.3 Technology Research
     - 1.3.1 Study blockchain (Algorand), Firebase, and MERN stack
2. **System Design - Designing the system structure and user experience**
   - 2.1 UI/UX Design
     - 2.1.1 Create wireframes
     - 2.1.2 Create UI designs in Figma
   - 2.2 UML & ER Diagrams
     - 2.2.1 Use Case
     - 2.2.2 Class Diagram
     - 2.2.3 ER Diagram
     - 2.2.4 System Architecture
   - 2.3 Database Modeling
     - 2.3.1 Create MongoDB schema models using Mongoose
3. **Frontend Development - Build the client-side using React and Tailwind**
   - 3.1 Project Setup
     - 3.1.1 React app initialization
     - 3.1.2 Setup Folder structure, routing
   - 3.2 Component Development
     - 3.2.1 Create reusable components (buttons, cards, inputs, etc.)
   - 3.3 Page Development
     - 3.3.1 Create static + dynamic pages (home, auth, investor, agency)
     - 3.3.2 Light/Dark mode toggle using context or state
   - 3.4 Firebase Auth Integration

3.4.1 Connect Firebase for user login/register/token auth

4. Backend Development - Build the server-side using Node.js + Express

4.1 Backend Setup

4.1.1 Setup Express server

4.1.2 Define project structure

4.2 REST API Development

4.2.1 Build CRUD APIs for properties, users, agencies

4.3 Firebase Token Validation

4.3.1 Secure API using Firebase ID tokens

4.4 Blockchain Integration

4.4.1 Use Algorand for property share minting, transfer

4.5 Role-Based Access

4.5.1 Protect routes based on user roles (investor, agency, admin)

5. Testing & Validation - Ensure functionality, security, and stability

5.1 Unit Testing

5.1.1 Test frontend and backend units separately

5.2 Integration Testing

5.2.1 Ensure frontend and backend work together properly

5.3 Blockchain Transaction Testing

5.3.1 Validate minting, transfers on Algorand testnet

5.4 Firebase Auth Testing

5.4.1 Simulate different login scenarios

6. Finalization

6.1 Documentation

6.1.1 Write final report, user guide, and system documentation

3.2 Initial Project Plan

To be most productive while working on the BlockEstate project, the author have designed a project plan at the beginning of the project using a Gantt chart. The project plan clearly states the time to be taken by each main task from gathering the requirements to the final development. Time has also been taken out from the calendar of each phase that will give me the decision to compare the development done to the development that was supposed to be done on time. Doing the chart was helpful for me because it made me see the relationship among all the jobs and the space-time needed, author was sure that working would make sense.

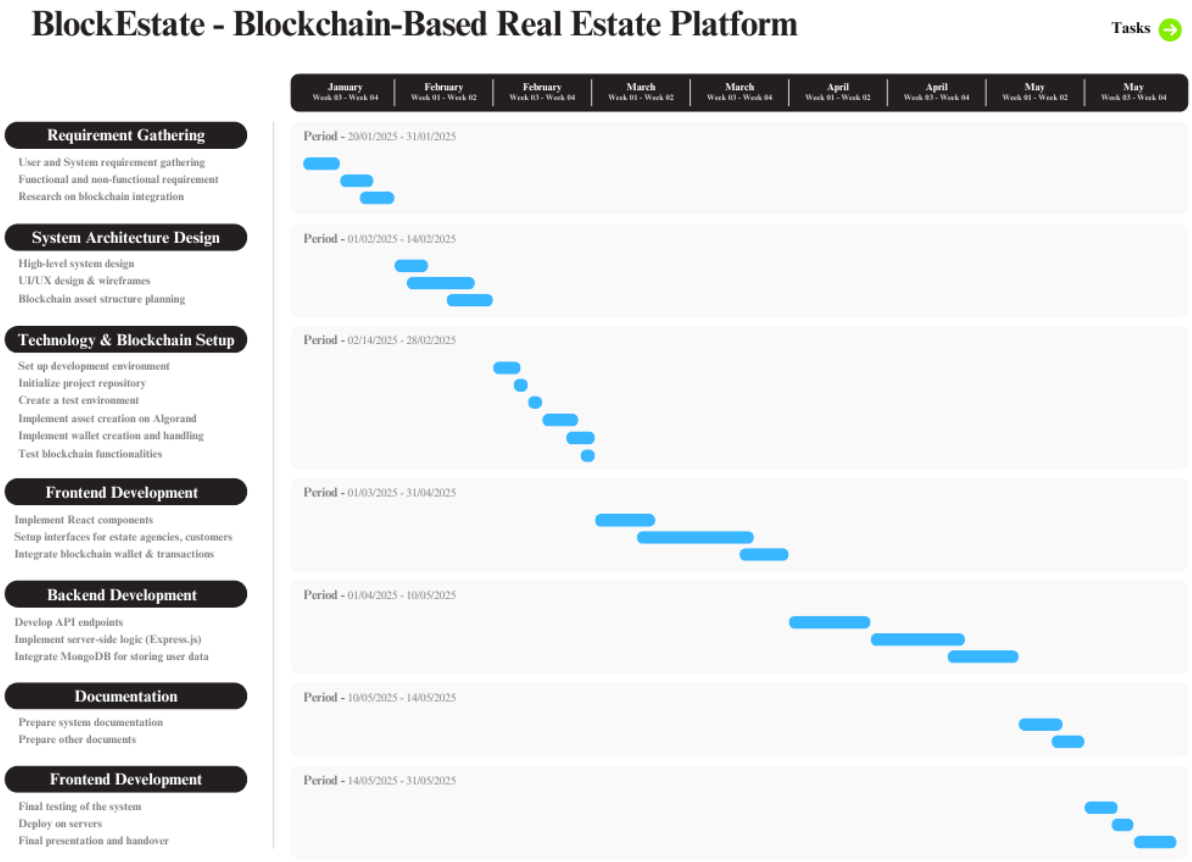

Figure 10 - Gannet Chart

3.3 Problems and Changed to the Plan

One of the methods the author employed for managing the BlockEstate project effectively was setting up a project plan with a Gantt chart. The chart illustrates the expected duration for each of the major tasks, starting from requirement gathering and ending with final development. Each stage was designated to dates, which enabled me to follow the progress and stay on track. This diagram helped me in the rationalization of the work process through visualization of the tasks and dependencies. That way the author was sure to have a smooth flow of work.

One of my experiences with BlockEstate project development was that some difficulties the author did not expect made me change the project plan original. the author had planned the backend development not to start until all the front-end work was completed, but when the author was doing the UI, the author figured that the author was in a need of real-time data responses to test the user flows and API interactions. Thus, the author had to change my plan so that the author could start working on the backend earlier. That's why the author was into the process of building the Express server and defining MongoDB schemas the same time the user-facing pages were being developed.

The critical among the issues the author was faced with is the real talk importance of: Firebase Authentication integration. To be honest, in the beginning, the author had no correct idea about exactly how complex this issue was, from making the whole token-based auth thing to linking it with MongoDB users. Hence, the author had to spend more time on it than the author expected and to reassign some days to fixing and debugging. Moreover, at the first draft of my plan, the author didn't think of role-based route protection at an early stage, however, while we were testing the system, it was a must to be put in place momentarily for better simulation of user behaviors like Investor and Agency flows.

Overall, the changes did not postpone the final milestone but consisted of moving tasks around the calendar and having some overlapping in the development stages. These choices made development crisper and accomplished a more solid synchronization between the frontend features and backend logic.

3.4 Final Project Record

While carrying out the BlockEstate project, the development team had to come across several practical problems and dynamic needs which were the root cause of the project plan modifications. The changes included bringing forward the backend development up to the level of testing frontend components in real-time, going deeper with the integration of Firebase Authentication than planned. The modifications made in that way required a part of the work to be performed simultaneously, and the schedule to be rearranged. As a result, the Gantt chart has been updated to reflect the final timeline of the project, which is reconfigured in a way that it can be compared with the original plan and proves to be the actual implementation of the tasks.

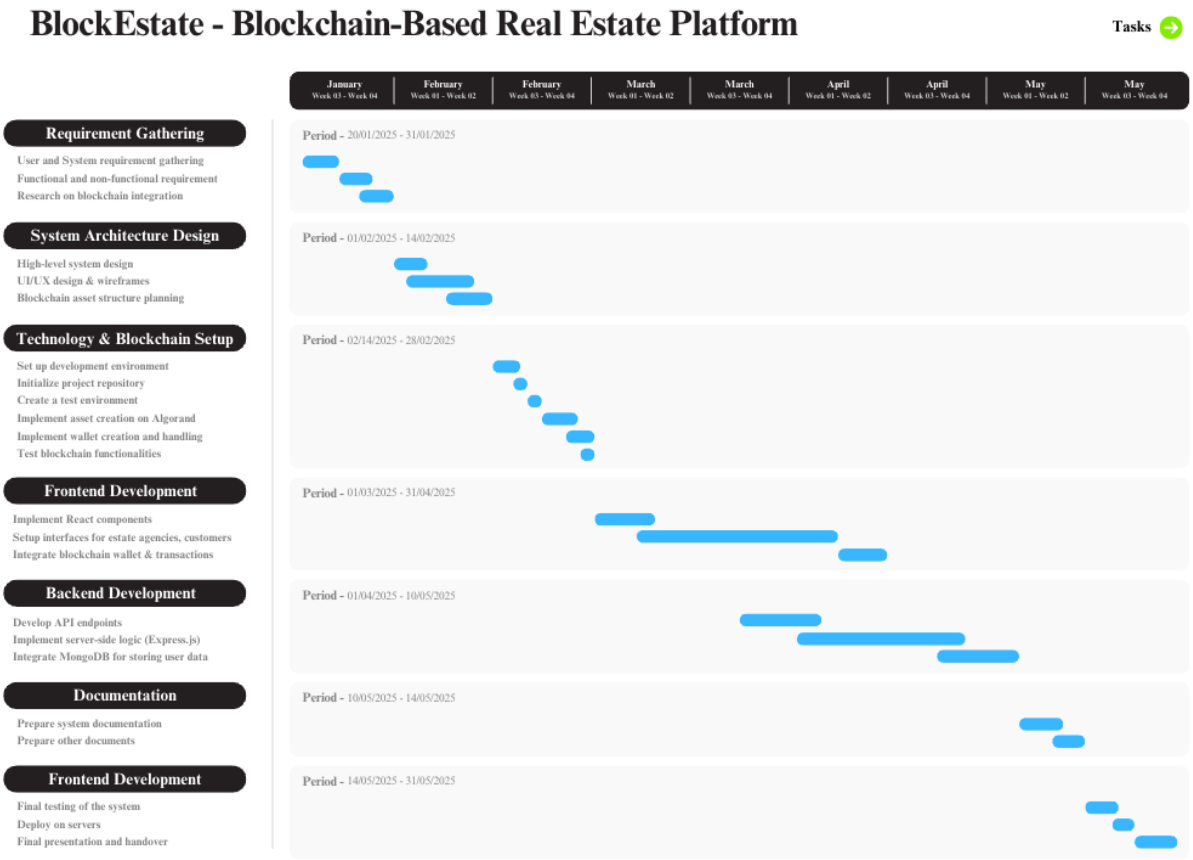

Figure 11 - Final Gannett Chart

4. Chapter 04 – Feasibility Study

Conducting a feasibility study aimed at evaluating the potential of realizing the project with the given resources, skills, and time in the first place was a reasonable procedure. The feasibility report can cover different issues like technical problems, finances, user feedback, as well as the realization levels of the system goals. Here is where we discuss if the project will result through the utilized tools and methods, and at the same time, reveal the risks and constraints that can be seen at an early stage of the project. After doing this, the author was able to see that the project was not only attractive on paper but also do-able in the concrete world with a clear road map and confidence for further actions.

4.1 Time Feasibility

The most essential aspect while conducting the project is the consideration of the time factor. No matter how great the concept is, the project will fail if it is not completed in the expected length of time. In the case of BlockEstate, the key was to evaluate the tasks, time restriction, and personal agenda in detail so that everything could be done in a systematic manner and without pressure or the lack of critical points. the author used a carefully constructed Gantt chart and a proper Work Breakdown Structure (WBS) for each phase of development conducted- from research, design, front-end and back-end development, testing, and finally deployment. Of course, several unexpected situations emerged which led to minor delays, however, in general, the schedule was not tight, the time-limit was reasonable and allowed me to complete the job during the stipulated period. This way the author was able to avoid stress, stayed focused, and delivered a quality outcome because the author planned my time accordingly.

4.2 Cost Feasibility

Cost feasibility is of major importance in determining the financial viability of a project. It serves to clarify the question of whether the returns (transactional, social and ecological) the project is expected to create are worth the costs incurred. For BlockEstate project commencement, the author did a financial overview of all the tools, services and platforms required. Working on a student project as a solo developer had me targeting at the least cost possible by employing freely accessible or open-source tools. Nevertheless, for features like secure authentication and hosting, the author had to spend some money. The provision of a table containing all the tools and services needed and their corresponding costs can be found below.

Requirement/Tool	**Cost**	**Notes**
WebStorm IDE	Free	Community Version
HTTPie	Free	Open-source API testing tool
Figma	Free	Free version used for UI/UX design
Firebase	$10	Used for secure token-based user authentication and image storage
Git & GitHub	Free	Free version control and code collaboration
MongoDB Atlas	Free	Use free tier
Internet and Utilities	$10	Monthly cost for basic utilities
Total Cost	**$35**	**Per Month**

Table 2 - Cost Feasibility

4.3 Scope Feasibility

When we talk about the BlockEstate project, the intended scope was very ambitious and realistic. It included creating user registration through Firebase authentication, using Algorand blockchain to tokenize real estate, modeling the property, making the investments, and adding some features and control for the admin. the author broke the scope into modules that the author was able to manage easily, thus, keeping the development smooth and easy to implement.

As a one-man band in this project, the author have checked if the scope is coherent with my abilities and the time the author have. the author used the new React, Node.js, Firebase, and MongoDB to build the system that can be easily scaled. Besides, some features have been saved for future releases such as advanced rental income analytics or full property management tools. In other words, the core was done within scope and there was space if it grew after the initial version.

4.4 Technical Feasibility

The project is based on the MERN stack (MongoDB, Express.js, React.js, and Node.js), which is a well-accepted, latest, and highly scalable technology stack for creating a web application from the front and back end. The app's core technologies are all open-source and have a large, active community of developers who help to make the whole process of systematic more efficient and sustainable.

Moreover, the author oversaw setting up Firebase for user authentication and image storage. It was great because the author was not only able to build the login and image storing parts securely and reliably but also do it in the fastest time possible through the chosen service. And for the required blockchain functionalities the author made use of the Algorand blockchain, which is capable of handling, in a very quick, secure and inexpensive manner, transactions, best-suited for property tokenization and the transfer of ownership, all of these were compatible, and with my technical knowledge and practice, they did not sound alien to me, so the author was okay working with them.

The system was technically feasible to implement and deploy because no highly advanced or exploratory technologies were employed. In addition, both Git and GitHub were used for version can very well, be said control, collaboration and project tracking.

4.5 Economic Feasibility

Economic feasibility is the process of evaluating the expense and projected value of the BlockEstate system to achieve the financial condition of development and maintenance. The main aim is to assure that expected benefits and results of a project are worth the resources that are being used for it.

This project was part of my academic work; therefore, most of the tools and services which the author used had no cost or had a low initial cost. I, for example, used free tools like Figma for UI/UX designing, Visual Studio Code for front-end development, and Firebase's free plan for the initial authentication and testing. The only paid service was Firebase's Blaze Plan, which the author upgraded to manage storage and user data more efficiently during testing. Also, because the author was the only developer involved, there were no labor costs, and no server costs.

Additionally, the release of the system as a real-world solution, in terms of a generation of income through property investment of a part or service charges, clearly indicates that the system is not only financially possible to build but also has prospects for future returns, which means that it is economically feasible.

5. Chapter 05 – Design

The Design phase is the essence of transforming conceptual ideas into a visual and structural representation of the BlockEstate system, this is. The part outlines all the core design facets embracing UML diagrams and user interface (UI) mockups. The design of each component was crucially handled to bring forth the system that would be easily expandable, secure, and user-friendly. For the UML diagrams, they were designed to model system interactions and class behaviors, whereas ER diagram was utilized to define data structure in the MongoDB database. With the use of flowcharts, it was very easy to conceive the process where numerous user activities and operations are handled and processed by the system. The user interface (UI) part gave birth to beautiful, fluid, and modern designs, with an emphasis on the user experience, which led to fewer complaints from both investors and agencies. The entire design process basically was the starting point for the formation of a coherent and properly ordered platform.

5.1 System Design Overview

The BlockEstate system is a web application that follows a client-server architecture model, which is designed to be scalable and modular. The frontend is built using React with Tailwind CSS that provides a user interface which is clean and responsive. Here it interacts with a RESTful API that is developed in Node.js and Express, which is the backends' core logic and is also easier for the client's application to be built. The primary database used here is MongoDB which is utilized for storing and managing all user, property, and transaction data. Firebase Authentication, in addition to the secured login and registration of the user, also provides e-mail verification, and user account management to manage the access control based on the user's roles like Investors and Agencies effectively.

The system itself incorporates the use of the Algorand blockchain for tokenized property shares management, and this without any intermediary (i.e., makes it possible for transparent, non-changeable, and secure handling of the asset). The manner of media files like a profile picture or a real-estate document has been considered through Firebase Storage. The design's central feature is modularity which, by the present, leads to the easier maintenance and development of the system. Thus, it supports smooth data flows among components, instantaneous interaction testing during development, and a base already configured to be deployed on cloud platforms like AWS

or Vercel after the end of the development stage. The system is now being worked on with security, scalability, and user-friendly functionality as the main priorities.

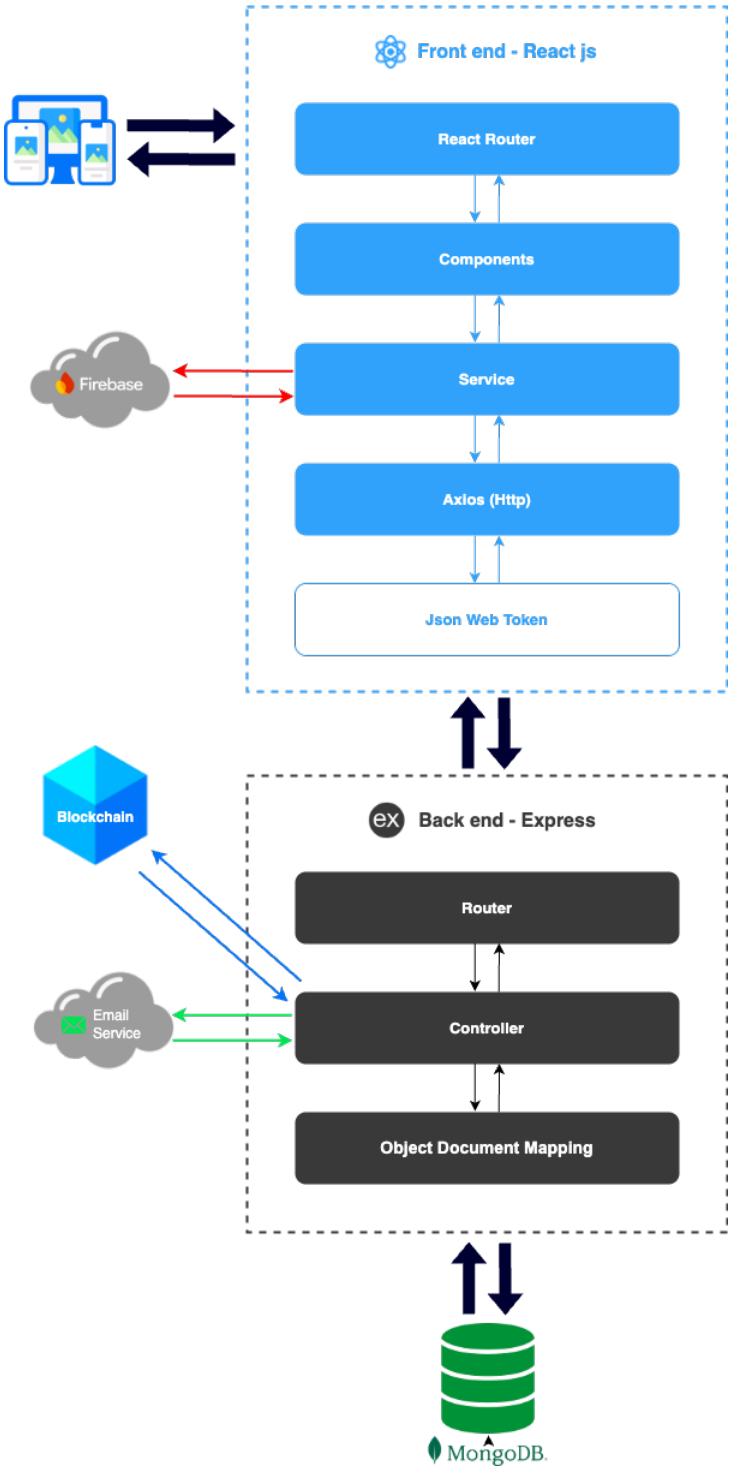

Figure 12 - System Architecture

5.2 Hardware and Software Requirements

Software Requirements

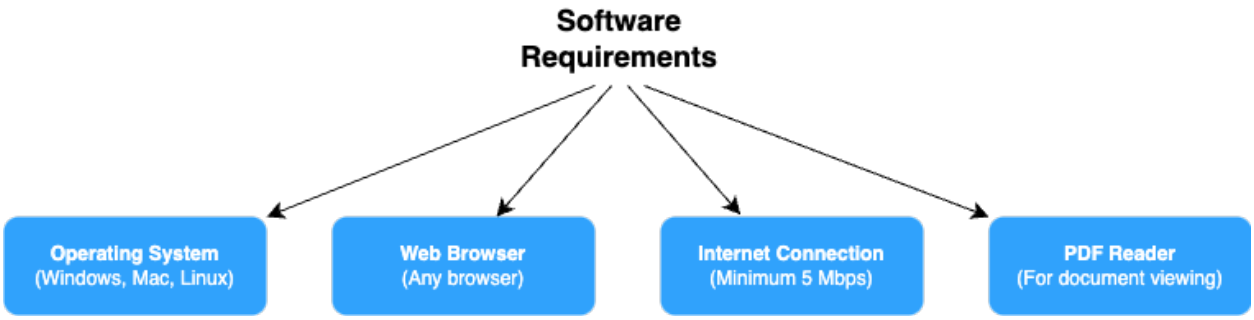

Figure 13 - Software Requirements

Hardware Requirements

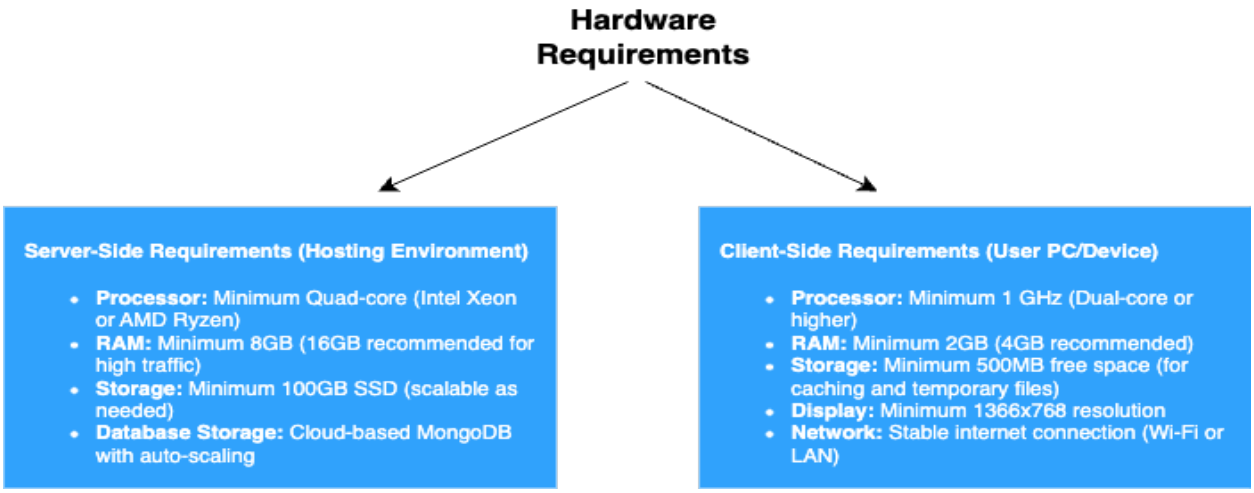

Figure 14 - Hardware Requirements

5.2.1 Design

5.2.1.1 UML Diagrams

Use case Diagram

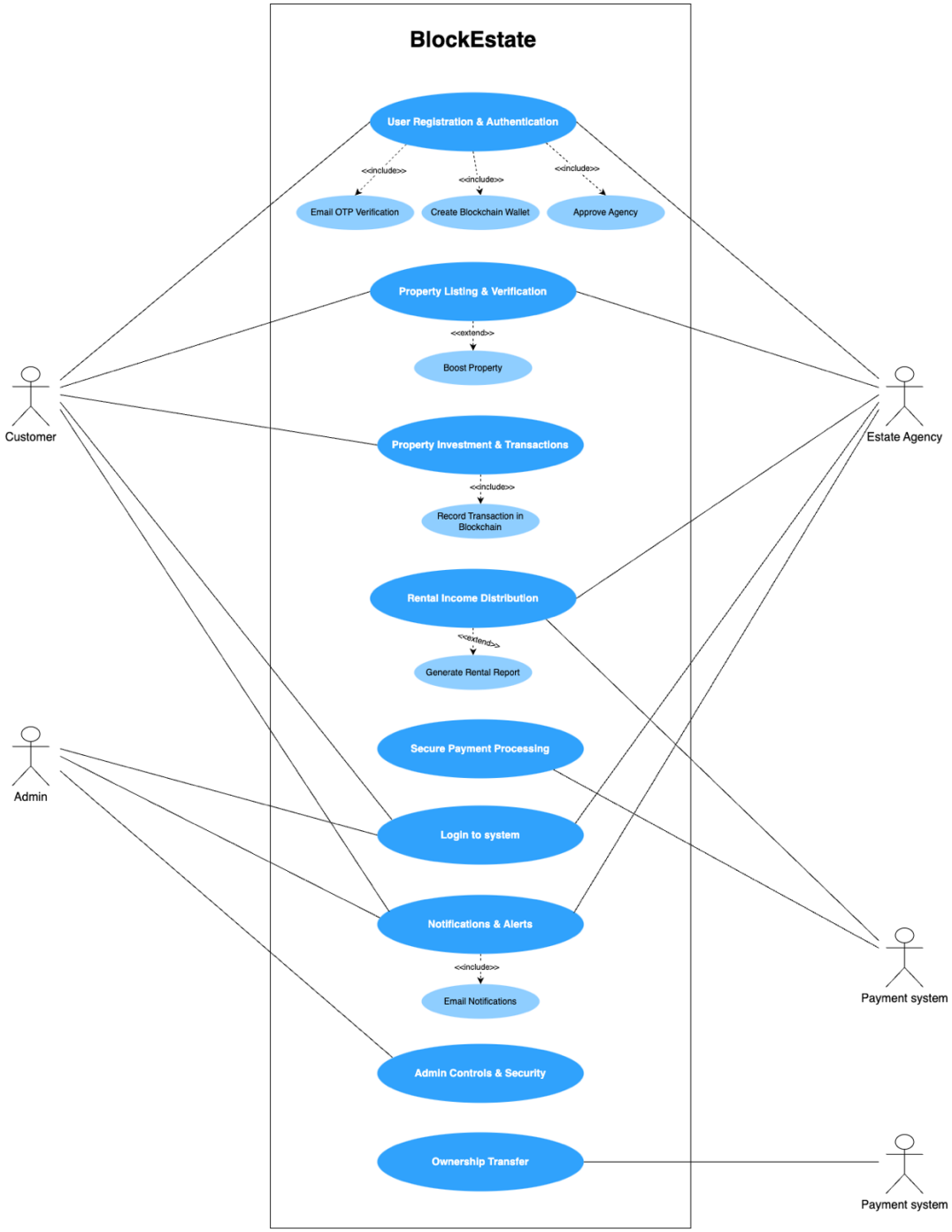

Figure 15 - Use Case Diagram

Flowchart of a Blockchain Asset

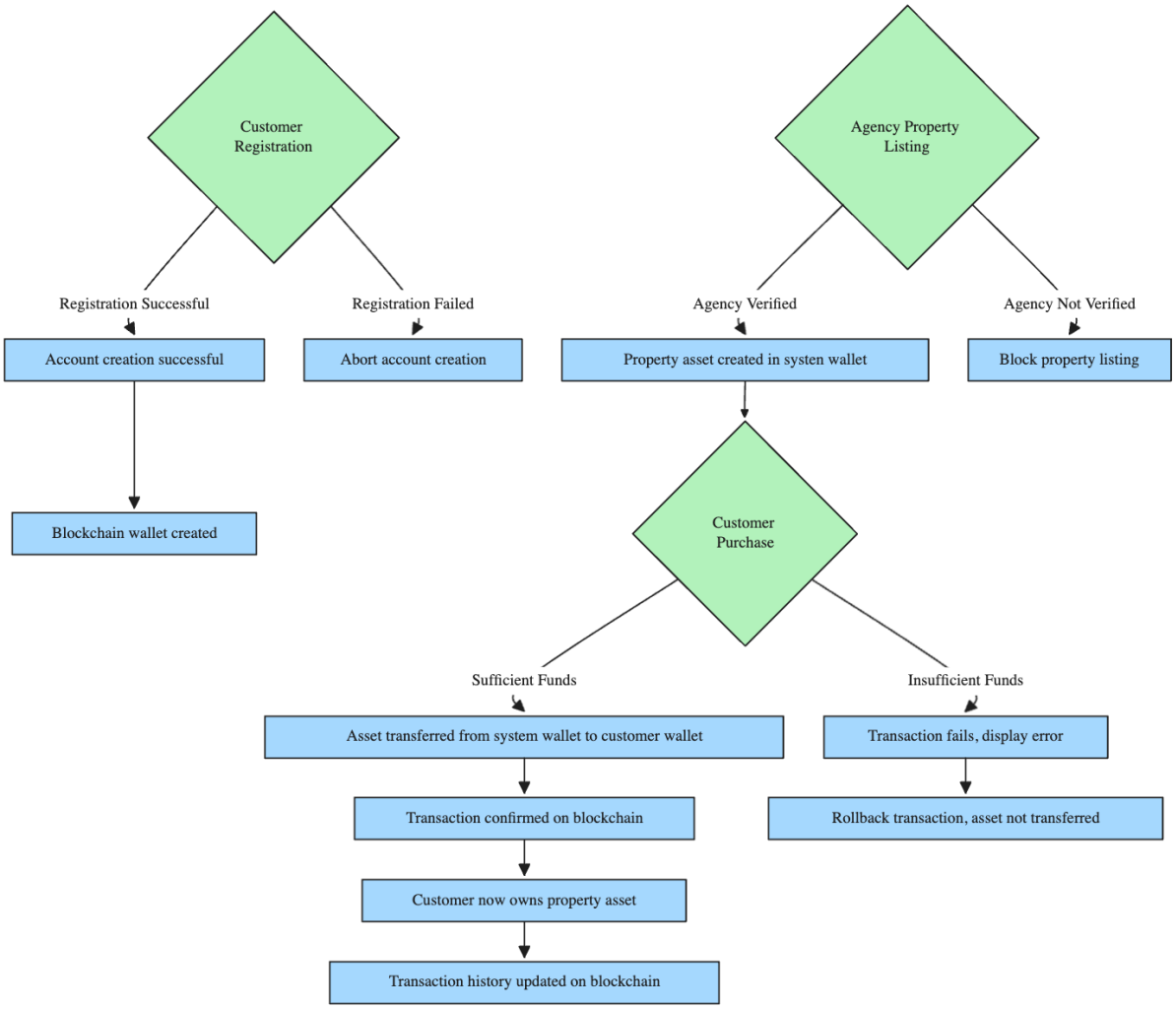

Figure 16 - Flow Chart of a Blockchain Asset

Class Diagram

Below is a class diagram that demonstrate main functionalities of the project.

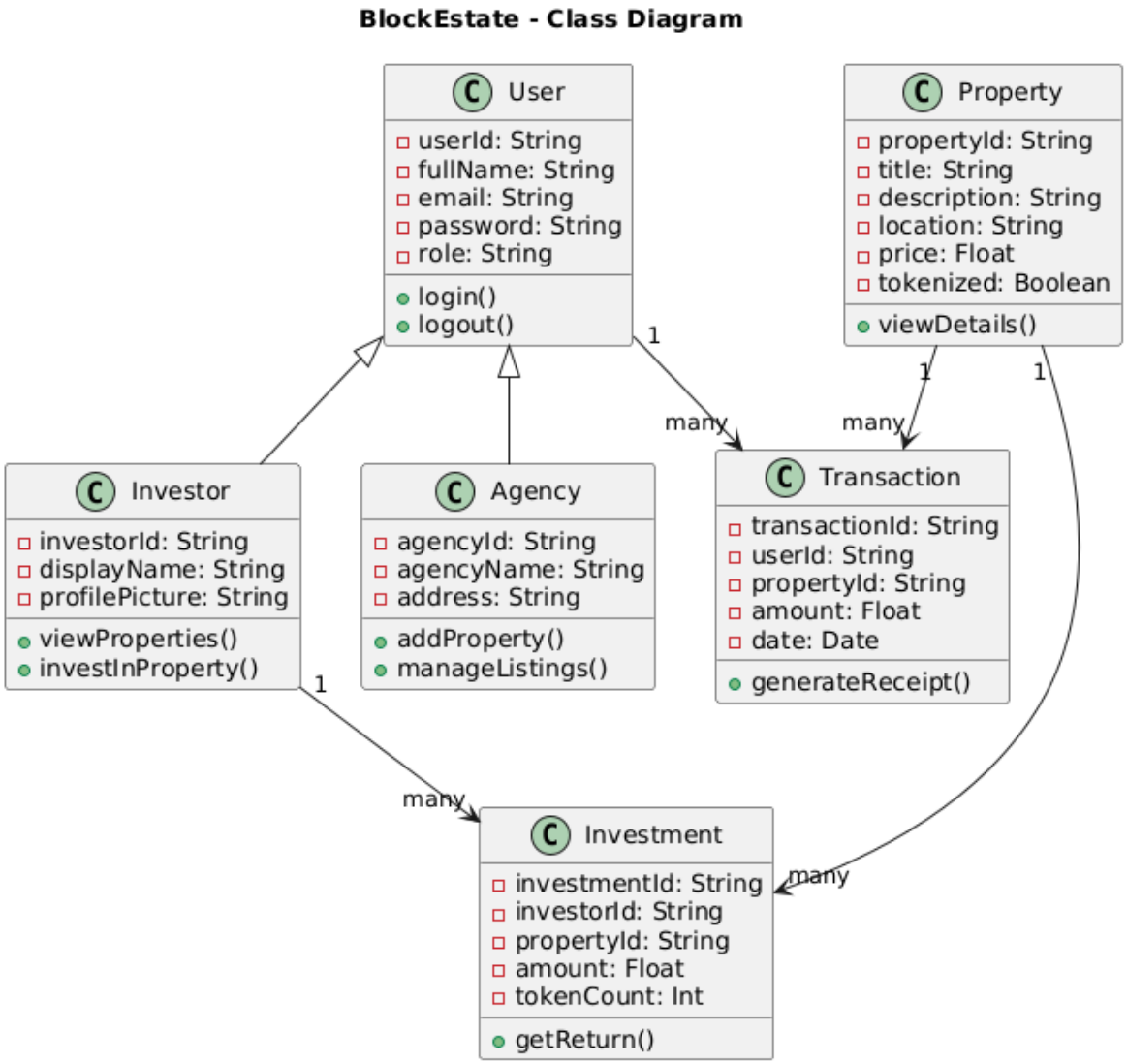

Figure 17 - Class Diagram

5.2.2 Database Design

5.2.2.1 ER Diagram

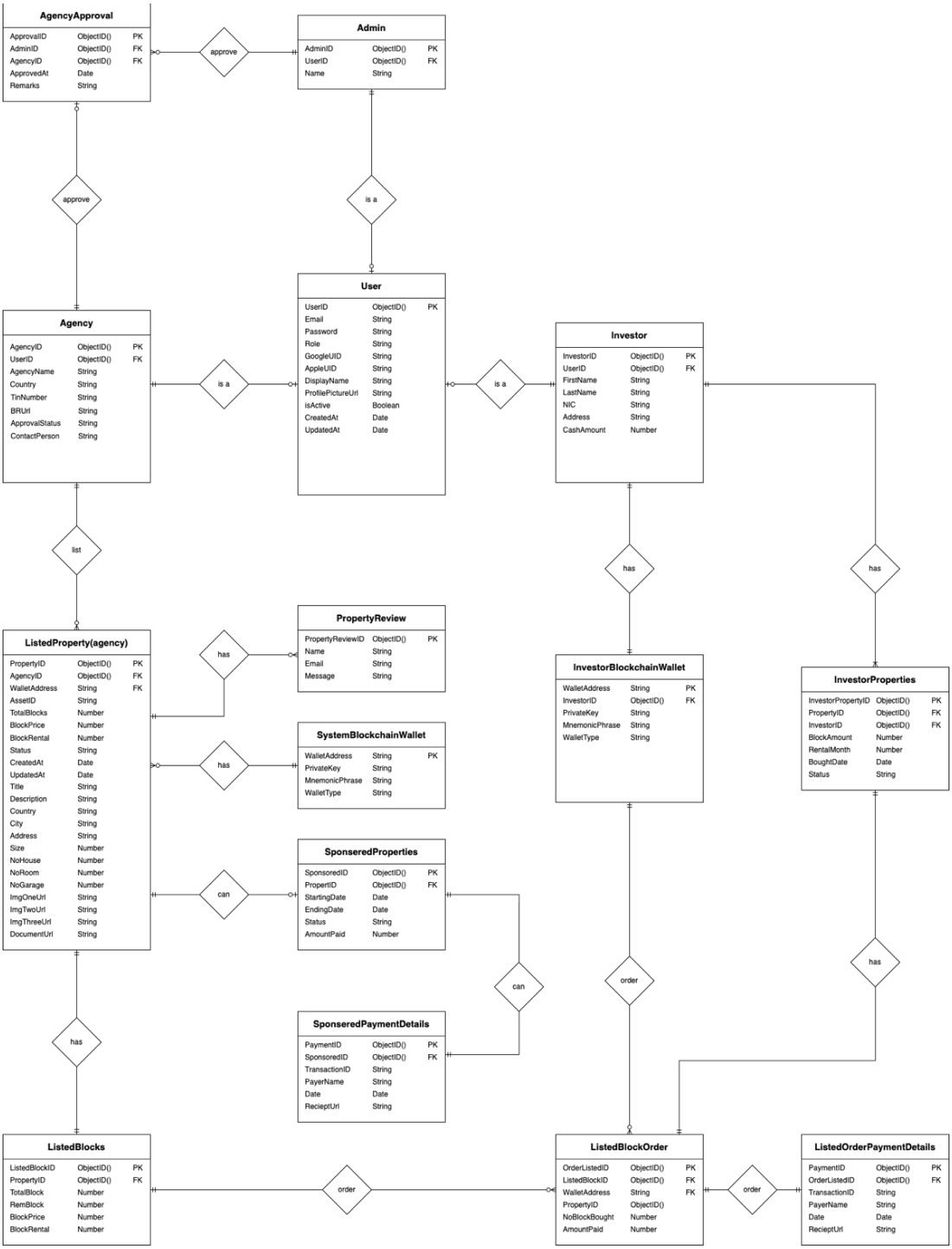

Figure 18 - ER Diagram

5.2.2.2 Database Mapping

BlockEstate platform is centered around a user-friendly database that manages the users, property listings, transactions, and ownership records as efficiently as possible. The next are the most critical database mappings that allow the various entities to link to one another in the system. The mapping is intended to provide the validated access of investors, user authentication, blockchain based transactions and the like, thus data consistency, security, and the system being one of the keys to the core features being major benefits of the latter.

- **ListedProperty** (PropertyId, AgencyID, WalletAddress, AssetID, TotalBlocks, BlockPrice, BlockRental, Status, CreatedAt, UpdatedAt, Title, Description, Country, City, Address, Size)

- **User** (UserID, Email, Password, Role, GoogleUID, AppleUID, DisplayName, ProfilePicURL)

- **Investor** (InvestorID, UserID, FirstName, LastName, NIC, Address, CashAmount)

- **Admin** (AdminID, UserID, Name)

- **Agency** (AgencyID, UserID, AgencyName, Country, TINNum, BRurl, ApprovalStatus, ContactPerson)

- **InvestorBlockWallet** (WalletAddress, InvestorID, PrivateKey, MnemonicPhrase, WalletType)

5.2.2.3 Database Connection

The system has created a database connecting code as below,

mongodb.js

```
1  import mongoose from 'mongoose';
2  import { DB_URI } from "../config/env.config.js";
3
4  if(!DB_URI){
5      throw new Error("MongoDB URI doesn't exist");
6  }
7
8  const connectToDatabase = async () : Promise<void> => {  Show usages
9      try {
10         await mongoose.connect(DB_URI);
11         console.log("MongoDB Connected successfully");
12     } catch(err){
13         console.error("Error connecting database");
14         process.exit( code: 1);
15     }
16 }
17
18 export default connectToDatabase;  Show usages
```

Figure 19 - Mongo DB file

5.2.3 User Interfaces

Home Page

The design of the BlockEstate homepage has been done with a lot of care and it is designed to create a first impression of modernity and reliability for the users of the website. It is bright and cheerful and at the same time, it helps the audience to understand the main purpose, to know the property investment opportunities that are available and to have instant access to critical sections like registration, how it works, and contact. Dark mode on the website is a matter of having an uncomplicated, polite look and ensuring eye protection when visiting it late at night. What's more, this design choice is in line with the current user interface guidelines and that means it can be accessed easily by more people. Below are the dark and light themes of heroic section.

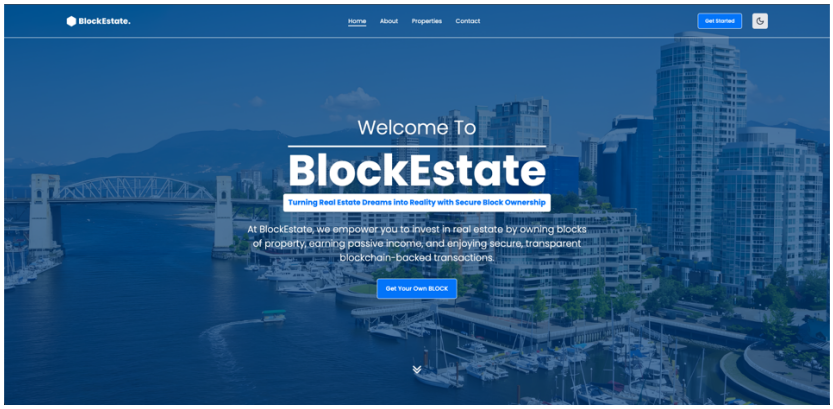

Figure 20 - Homepage UI Light

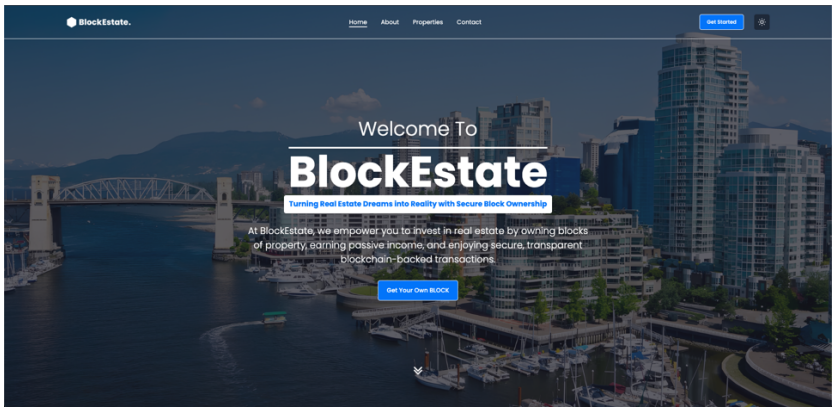

Figure 21 - Homepage UI Dark

Contact Page

BlockEstate's contact page has been created in such a way so as to enable users, investors, and prospects to communicate directly without hassle. The page offers not only the most important contact information such as the phone number or email address but also the location for conducting all the business operations. And to make the user experience complete, there is a simple and ready contact form which enables anyone to communicate easily and quickly even without staying on the web page. The color decision has great coherence too, the dark theme is kept throughout the page, hence, the visual harmony and smoother user experience are maintained.

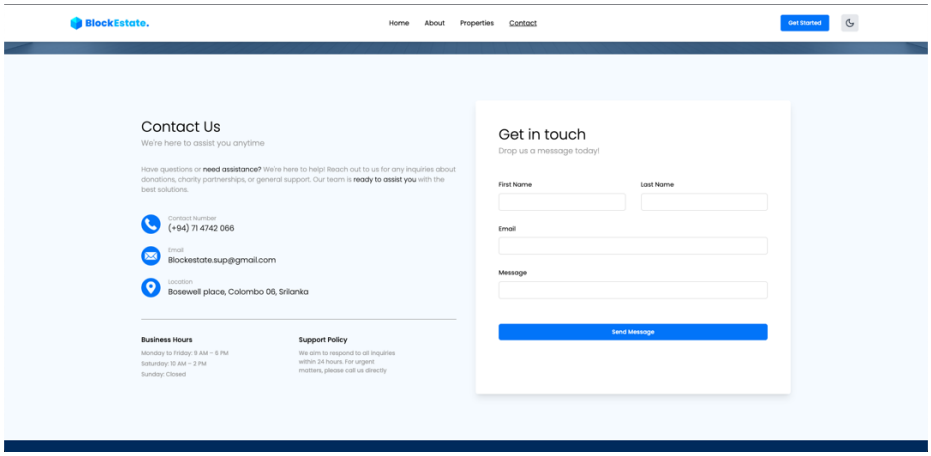

Figure 22 - Contact Page Light

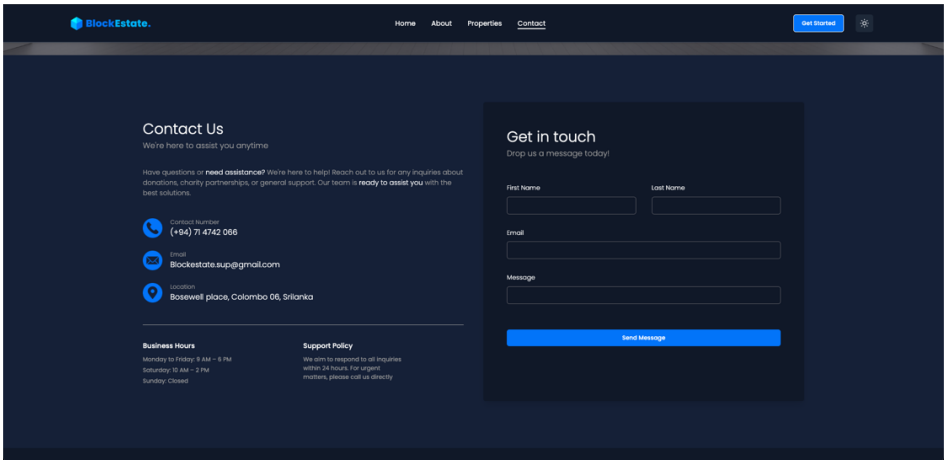

Figure 23 - Contact Page Dark

Property Listing Page

The Property Listing Page showcases the properties that are available for purchase with marked details such as the location, price, rental income, and the ownership details. The users can look through, filter, and search for properties depending on their conditions. Every item provides photos and a short summary, as well as a link to further details.

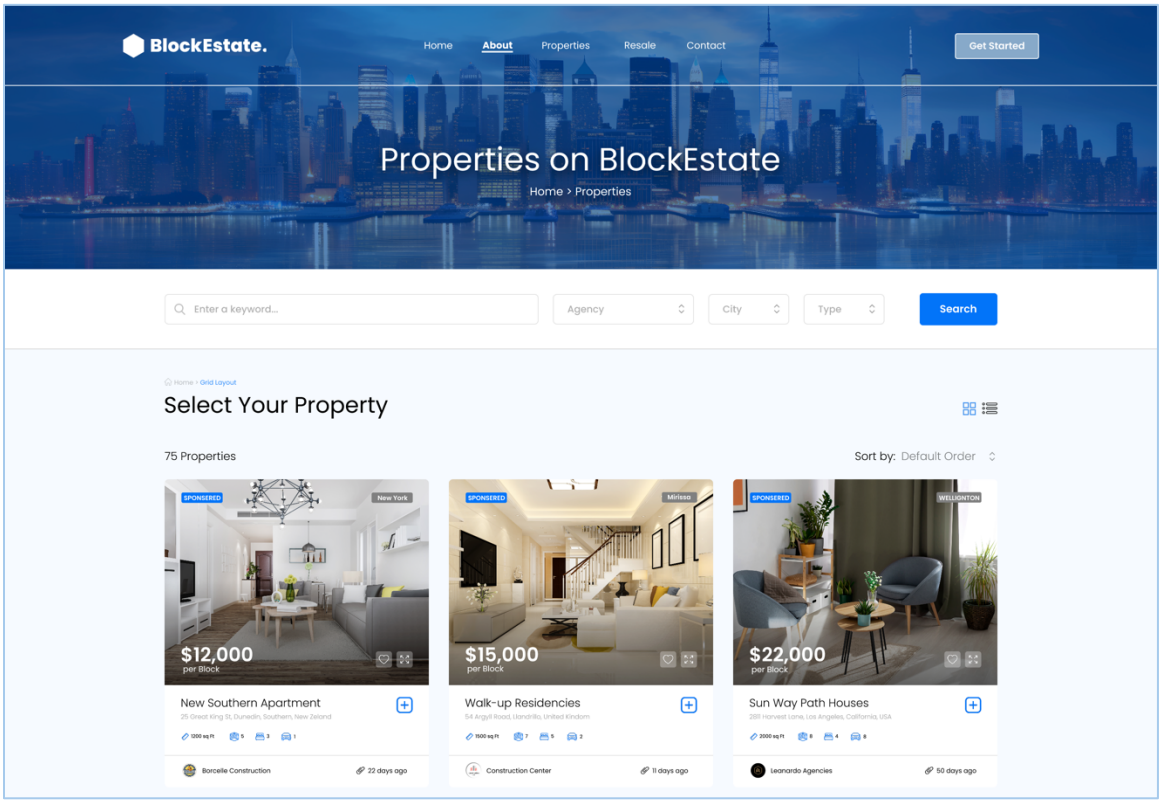

Figure 24 - Property Listing Page

The Property Listing Page has two view options: Grid View and List View.

- Grid View shows properties in a grid format with pictures that is easy to scan several properties at a time.
- List View displays properties in an extended list, with more details per property such as price, rental income, and ownership, for those users who like more structure and comprehensiveness in their view.

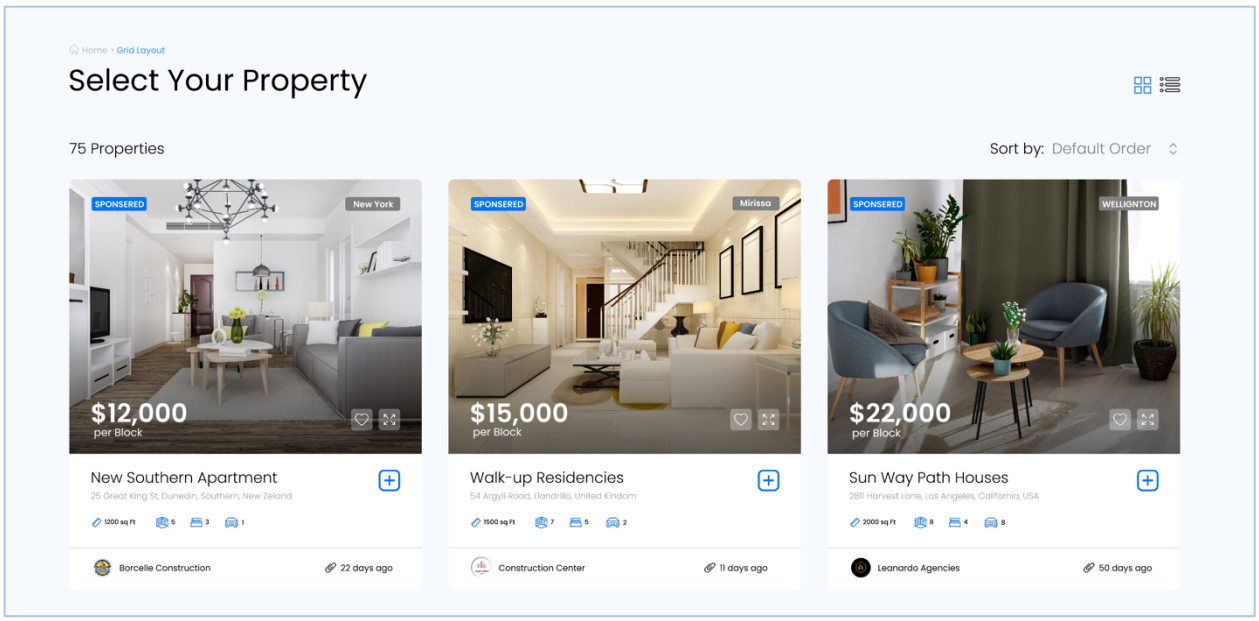

Figure 25 - Property Listing Grid View

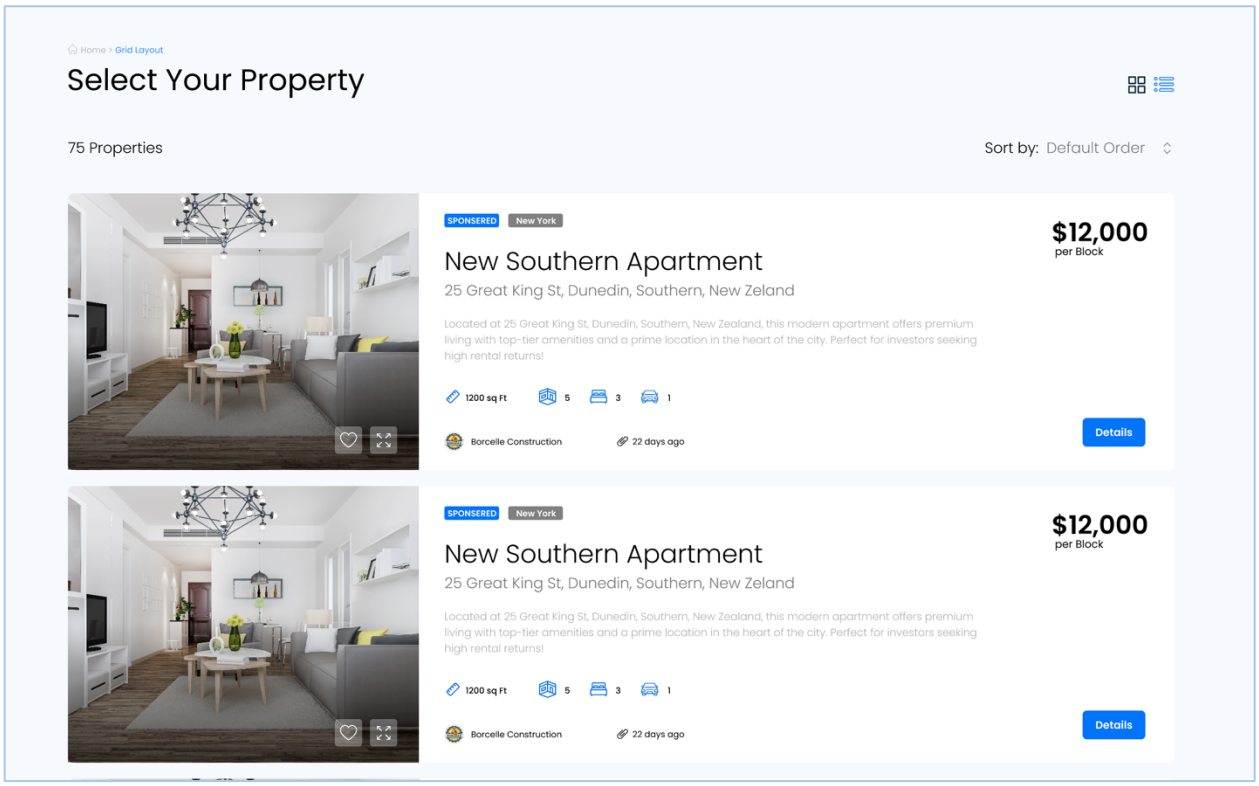

Figure 26List View - Property Listing

Property Listing Page

The Property Details Page is a place that offers a wealth of information on a specific property. People will find high-resolution pictures, a description that is both easy to understand and informative, and all the necessary figures, like the price per block, the total number of blocks, the rent for each block, and the corresponding ownership availability details. A clear display allows users to just grab the number of blocks they want to book and complete a single payment step from the page, thus smoothing the whole process of making a transaction. The web page is very clear, responsive, and fully in line with the dark theme of the platform, thus providing an interesting and professional experience to all the users.

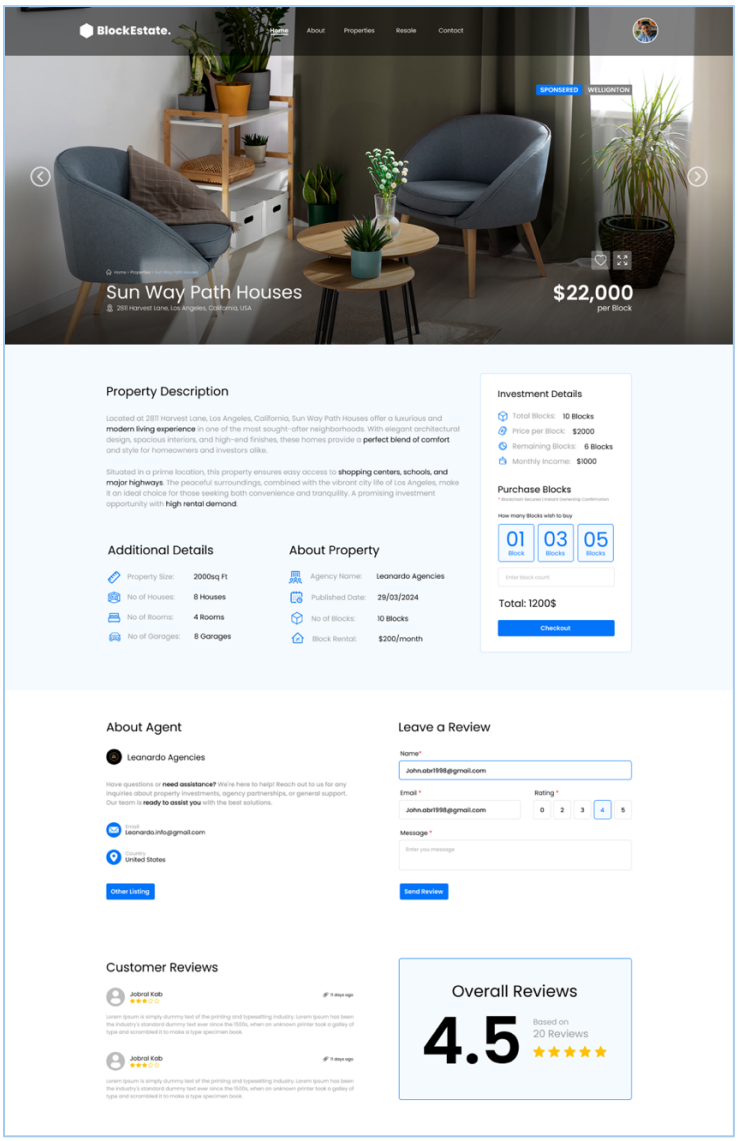

Figure 27 - Single Property Page

Login Page

The Login Page is used by users for logging into their accounts safely through their password and email. It is an uncomplicated, very much user-friendly page with alluring text fields for both input details. Furthermore, clients can log in with the help of third-party login services like Google or Apple, allowing it to have a variety and convenience.

The "Forgot Password" link is the one that will give the option to the user, who has forgotten the login credentials, to recover them. The one-pager is characterized by a competent and well-designed process that not just offers a clean and user-friendly login but also gives success or failure feedback with a step-by-step wizard. This approach not only makes it possible for customers to accomplish their goals promptly but also makes it easy for them to interact further with the system.

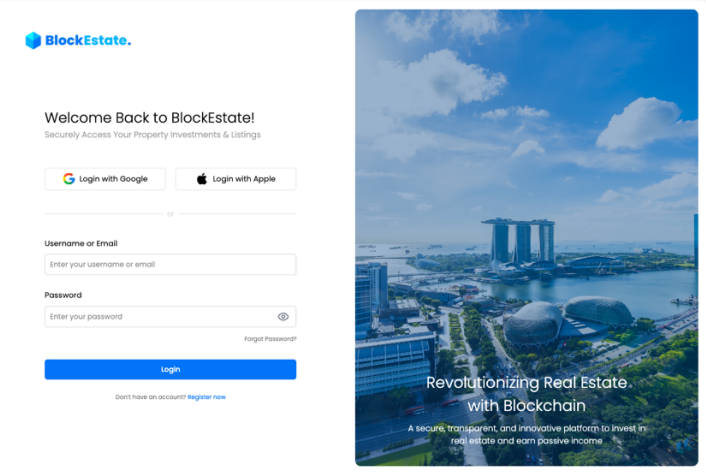

Figure 28 - Login Page Light

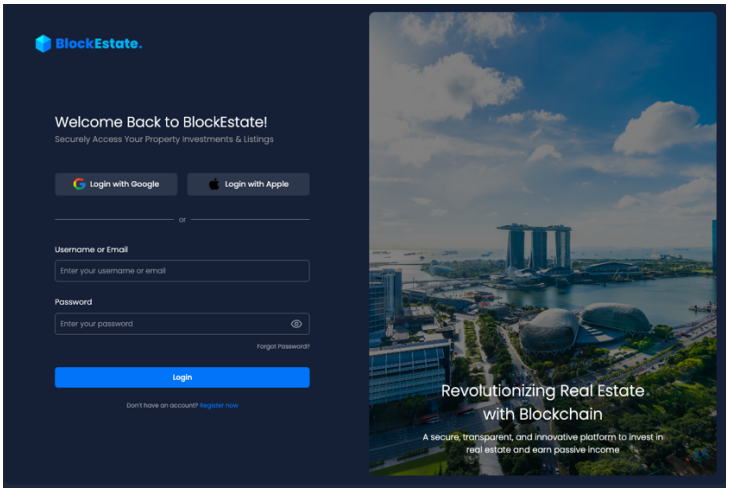

Figure 29 - Login Page Dark

Register Pages

The page is initialized by asking the user to choose their role type: an Investor or were Agency. The purpose of this request is to direct the registration process according to the user's needs and access levels within the platform.

- Investor: This group is made for the users who are going to invest in property blocks and manage their portfolio by themselves. Investors are eligible to access and purchase the available properties, keep them for rent, and do the transactions safely and securely by a blockchain.
- Agency: This option is relevant for the real estate agencies who want to list and handle the properties on the website. Agencies have the authority to manage their portfolio, add property listings, and talk to both investors and the admin team.

Upon the user's choice of the role, he/she is required to proceed further with all the rest of the registration form where the details like their email, password, and other role-specific information are to be provided. This is how the platform knows how to deal with them in the most suitable way.

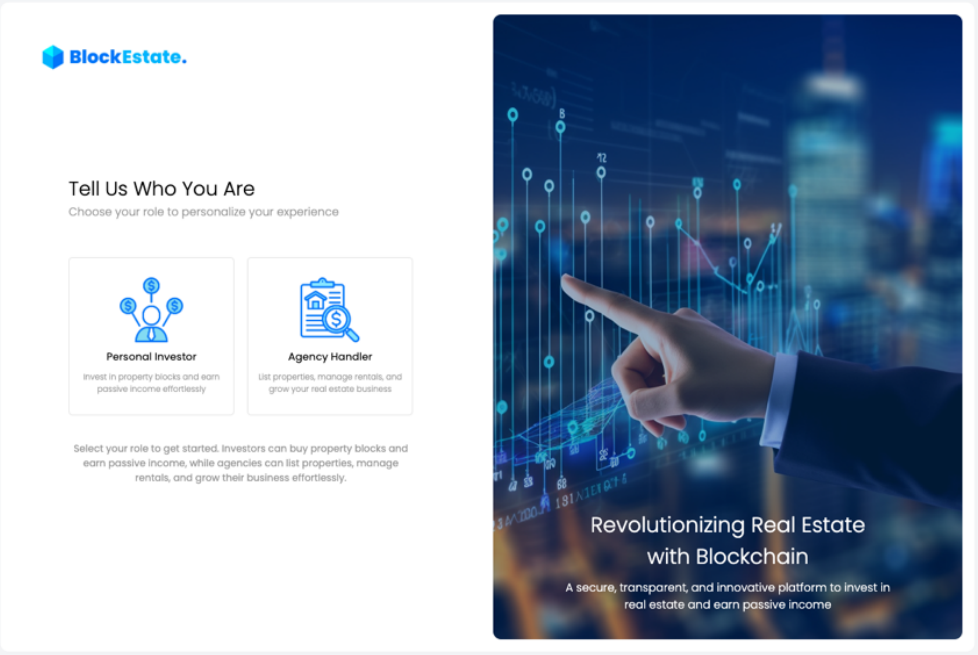

Figure 30 - Role Selection Page Light

The OTP Verification Page is very important to secure the registration process. When the user puts in their email address, the system instantly sends a 6-digit OTP to the given email for verification. This OTP can only be used for the period of 3 minutes and then it becomes null and void, thus it guarantees that the process is secure yet also fast. The page has a very simple design where users can type the code, see the countdown of the remaining time, and send a request to resend if they need to do so. This procedure checks the email's validity first and then the user can go ahead with the registration flow.

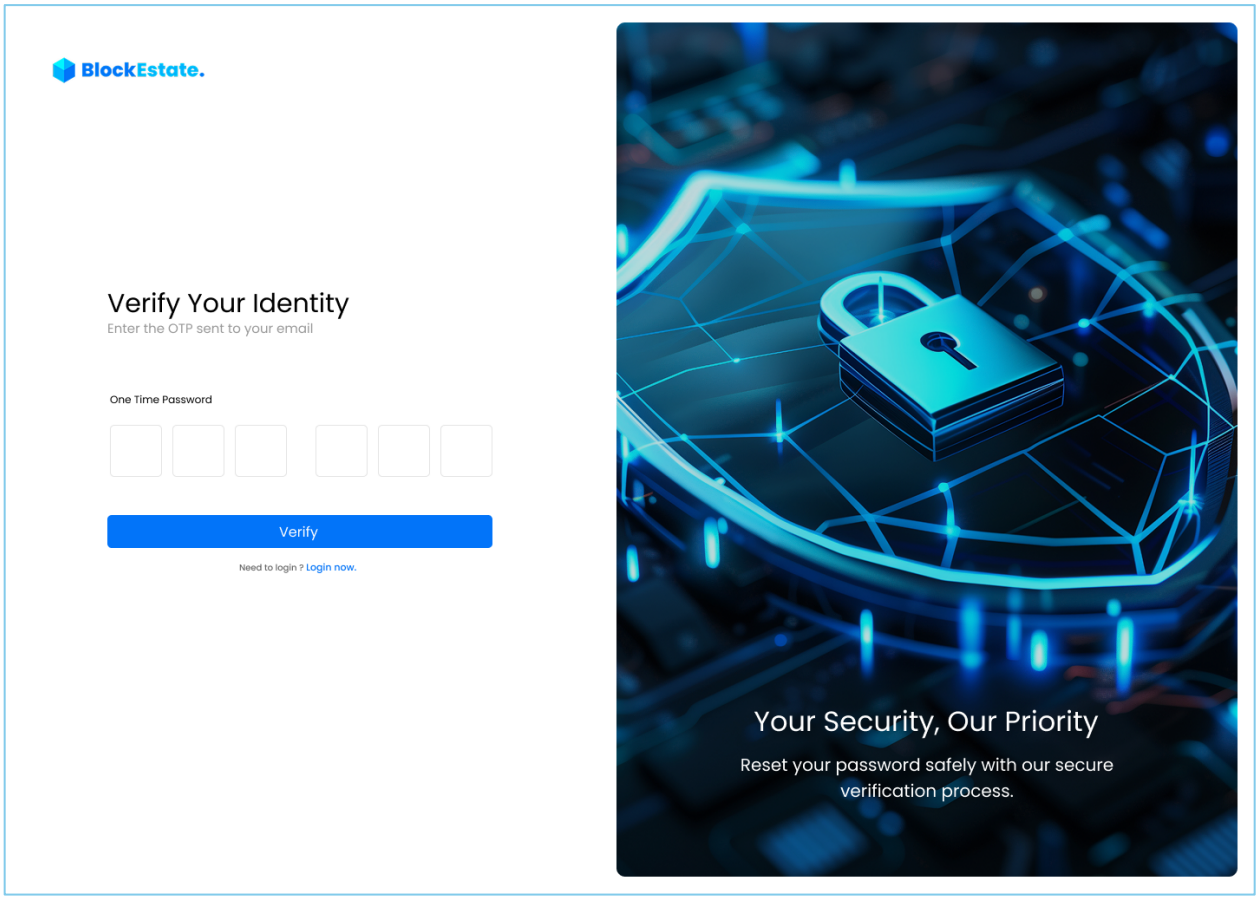

Figure 31 - OTP Verification Page

The OTP Verification Page is set up to validate the email address of the user that allows the user to go ahead with registration. As soon as the user writes his or her email, a 6-digit OTP is instantly sent and will last for 3 minutes. They must key in this code within the established time frame to prove their identity. After the OTP is verified successfully from the user's side, they are then redirected to the account setup page. There they can proceed with their registration by adding their personal details, uploading a profile image, and setting a secure password.

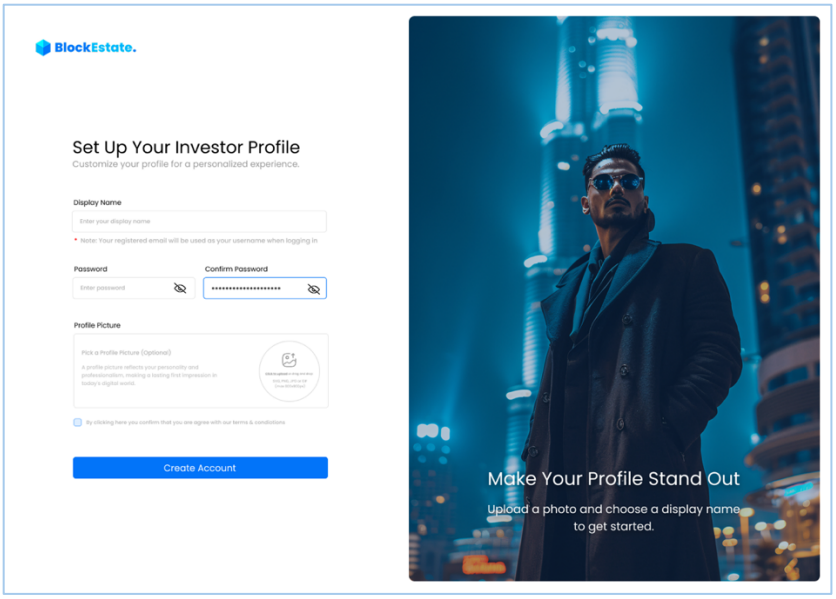

Figure 32 - Investor Account Setup Light

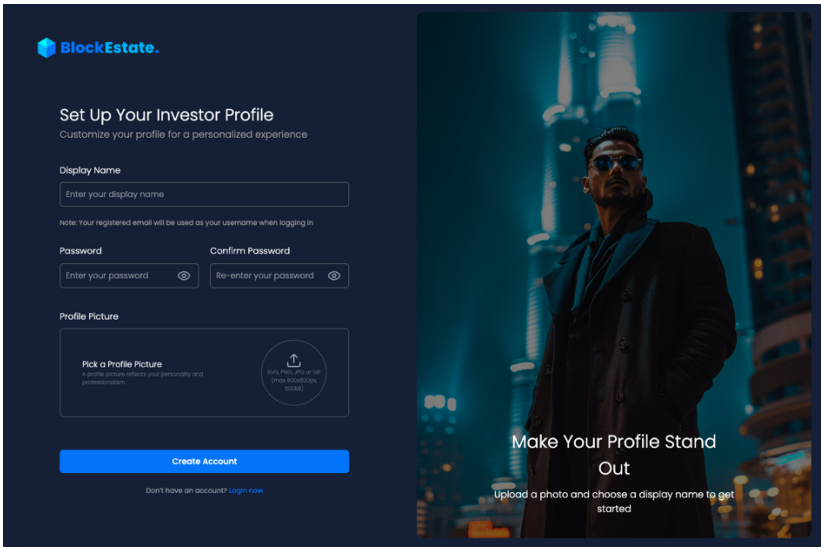

Figure 33 - Investor Account Setup Dark

If the user is registering as an agency, they will also be required to provide their TIN number and business registration documents to complete the setup process and validate their business identity.

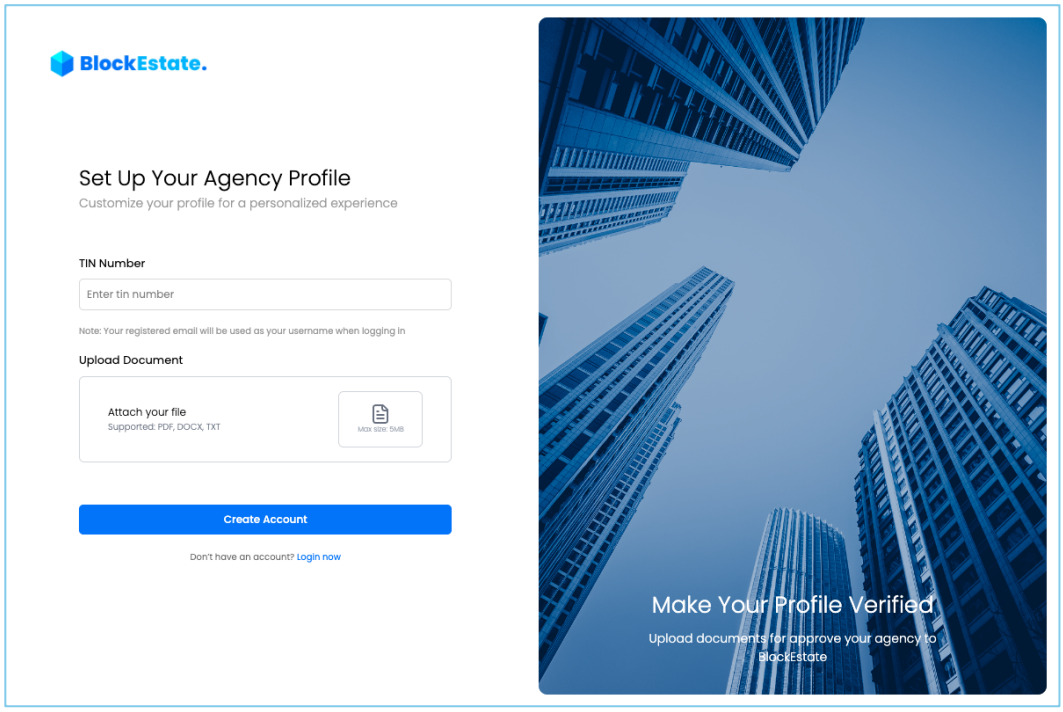

Figure 34 - Agency Approve Light

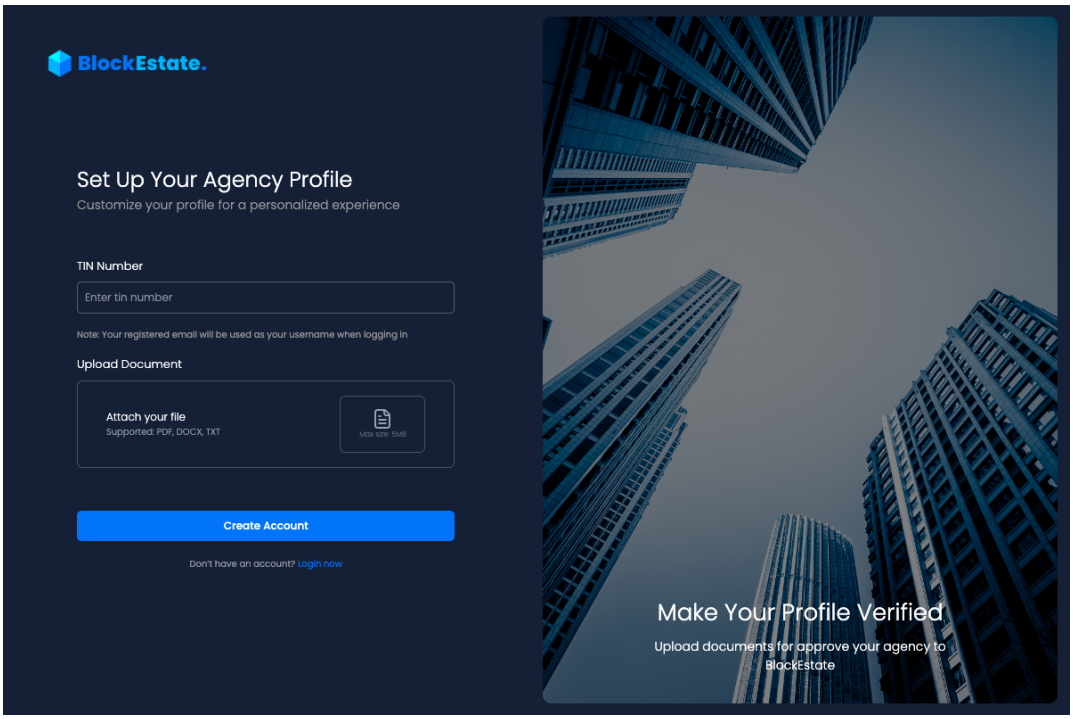

Figure 35 - Agency Approve Dark

Dashboards

The Admin Dashboard is the core console of the BlockEstate system, developed to allow the administrator to facilitate their platform management process. It contains well-structured sections that enable the admin to control user accounts, property listings, transactions, report creating, and agency verification. Using explicit interfaces and data insights, the admin can watch the events, uphold the platform integrity, and make sure the operation is going smoothly.

The Agency Dashboard was made especially for property agencies to arrange and control property listings and view business results in an online form. An agency can, namely, add new properties, change current ones, check the division of property ownership, cope with client requests and monitor the payments of rent. It gives agencies control over their listed properties and the way of interaction with the investors.

The Investor Dashboard is a place where an investor can have an overview of his investments in properties. It exhibits the investor's property blocks, forthcoming rental income, property growth, and finally the recent transactions. This dashboard is aimed to assist the investors in being aware, managing their portfolios, and making the right moves or decisions all in the platform, where they easily operate.

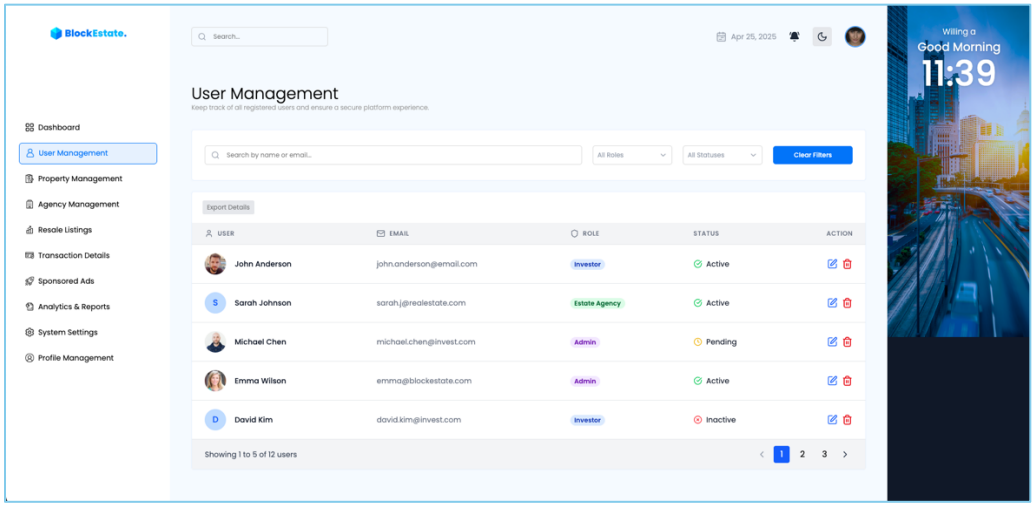

Figure 36 - Dashboard Light

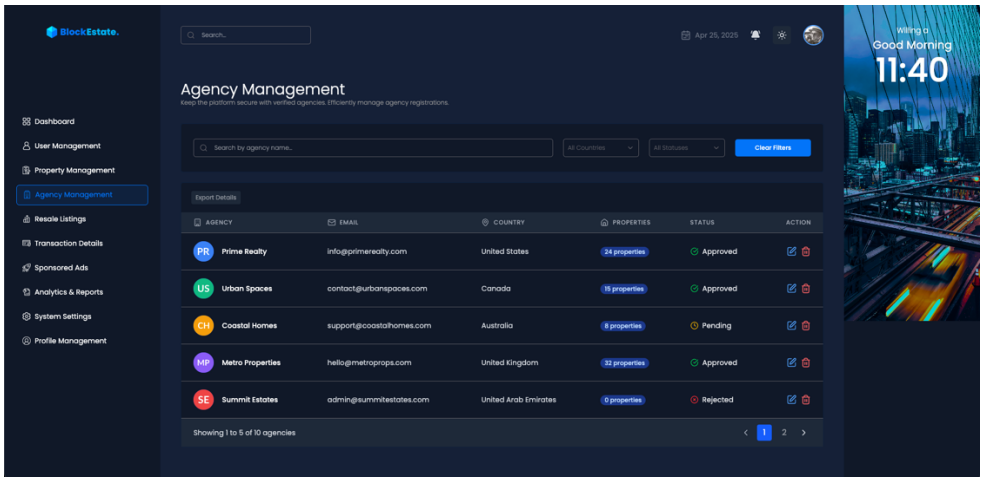

Figure 37 - Dashboard Dark

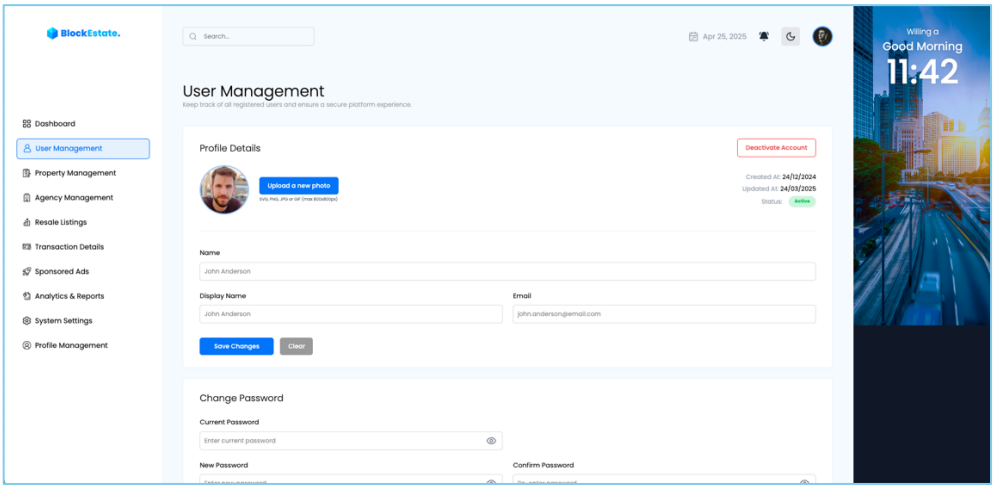

Figure 38 - Dashboard 2 Light

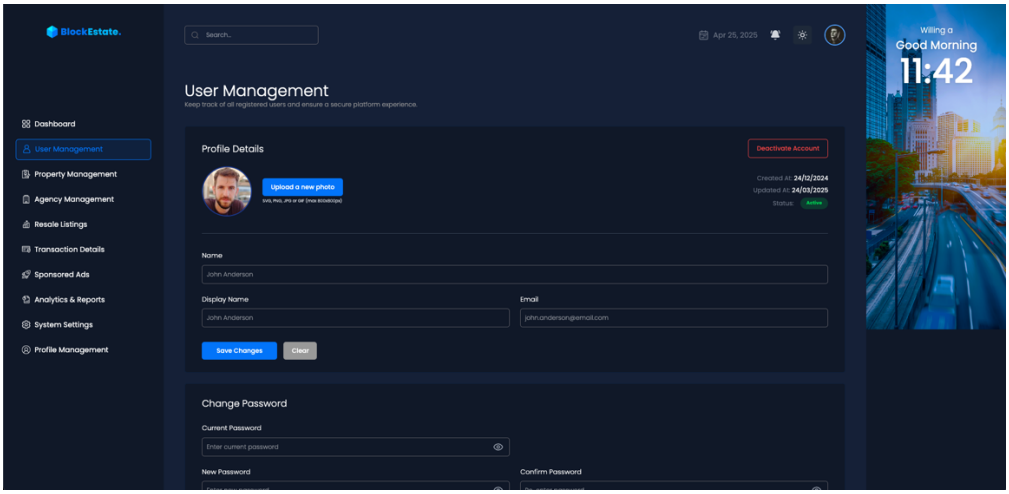

Figure 39 - Dashboard 2 Dark

6. Chapter 06 – Implementation

This chapter gives a detailed realisation of the BlockEstate project. Not only the front-end but also the back-end development processes have been addressed. The main content of it is the presentation of the MERN stack (MongoDB, Express, React, Node.js) based system, along with Firebase Authentication and other necessary tools that will provide built-in security, prove to be scalable, and future consistent application. The part introduces the different technological features of the system, e.g., folder structures, reusable component designs, routing methods, context usage, and backend logic through visual means. The author serves us not only with definitions and theory but also with a description of the actual process of the front-end and back-end exercises. This was done with the help of various real-life coding samples, excerpts, and snapshots, making this chapter a clear example of the practical choice made to breathe life into the system.

6.1 Front-end

6.1.1 Folder Structure

The front end of the BlockEstate project follows a pattern which is not only clear and capable of growth but also encourages reuse. The main folders that include this of components, layouts, pages, contexts, routes, and assets, are well-arranged, considering their corresponding responsibilities. Here you can find a wide collection of UI elements which are part of the components folder, for example, the button, the theme-toggler, and the image uploader are the basic UI elements that are used to maintain the same look app-wide. The use of separate components for each page layout is achieved by placing each of these components in the layouts folder, e.g., the website layout and the admin dashboard layout, separate components also allow the reuse of common layout patterns. Within the contexts folder lies the part of the operation related to the global state, therefore, the establishment of a custom context is important here. The Firebase Authentication system is a good example where one of the custom contexts is used to manage the functionalities like logging in or out the user, tracking even their id and so on. This is where the use of React Context technology makes the smooth and centralized operation of user authorization possible without the demand for prop-drilling. Also, this approach allowed me to keep the project well-structured and the author could easily add to it as it went to the next phase.

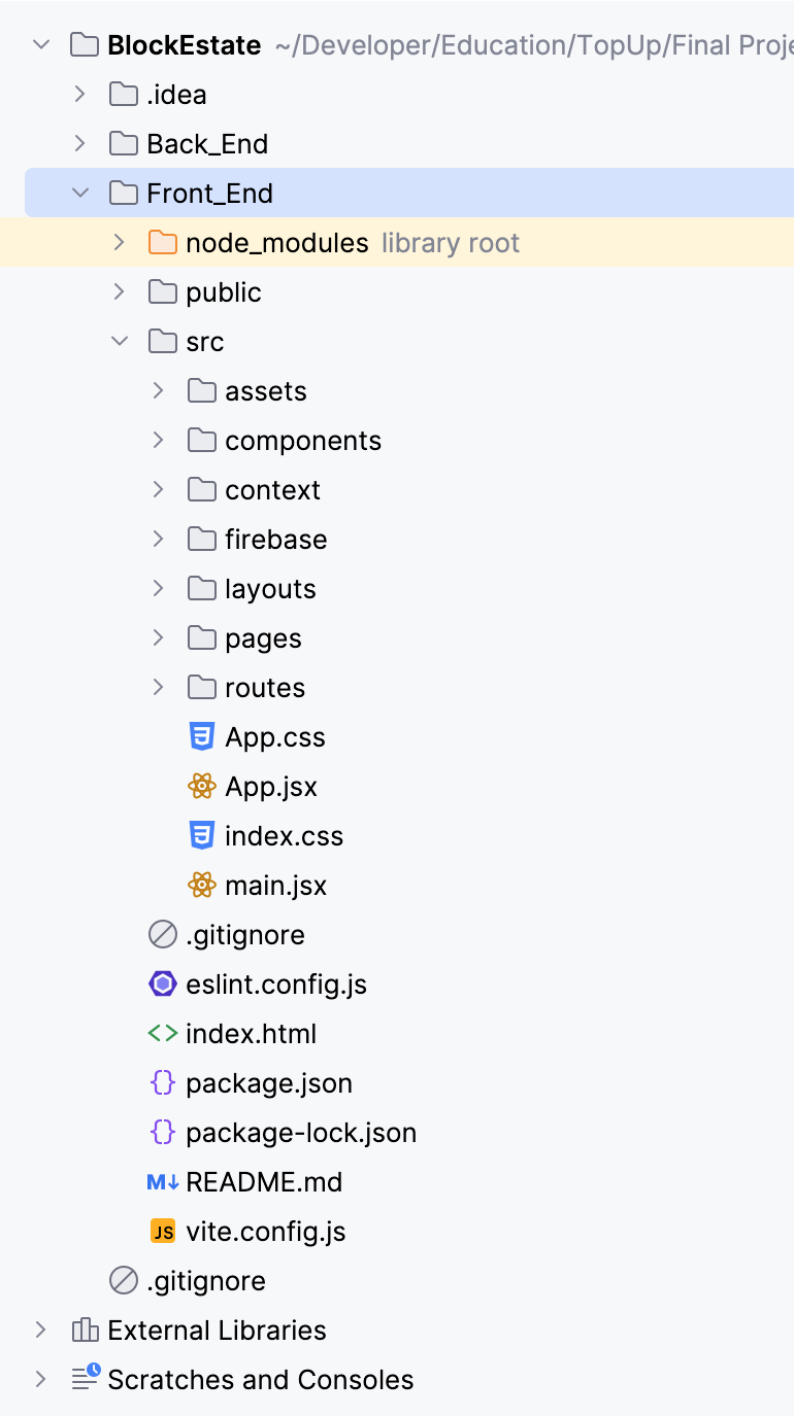

Figure 40 - Front-end Folder Structure

6.1.2 Common UI Components

For purposes of the application's visual identity and cutting down on recurring code the author built some regular components. These are a default standardized Button, a Theme Toggler that enables the light-dark mode switch, and an original Image Uploader with validation and preview functionalities. Such components are geared towards maximum adaptability and are not only the registration, dashboard and form submission pages but other pages as well. Through the confluence of the styling and logic in these components, the author achieved two major benefits – a coherent design system that is scalable and at the same time, the application became much easier to maintain and grow.

ThemeToggle.jsx

```jsx
import { useEffect, useState } from "react";
import { Moon, Sun } from "lucide-react";

const ThemeToggle = () => {  Show usages
    const [theme : string , setTheme] = useState( initialState: localStorage.getItem( key: "theme") || "light");

    useEffect( effect: () : void  => {
        document.documentElement.classList.toggle( token: "dark",  force: theme === "dark");
        localStorage.setItem("theme", theme);
    }, deps: [theme]);

    return (
        <button
            onClick={() : void  => setTheme( value: theme === "dark" ? "light" : "dark")}
            className="p-2 rounded-[5px] bg-gray-200 dark:bg-gray-800 text-gray-800 dark:text-gray-200
               transform transition-transform hover:scale-105 ease-out">
            {theme === "dark" ? <Sun size={20} /> : <Moon size={20} />}
        </button>
    );
};

export default ThemeToggle;  Show usages

```

Figure 41 - Theme Toggler Component

Button.jsx

```jsx
export const Button = ({ Show usages
                           children,
                           hoverColor,       // Custom hover color (optional)
                           border,           // Boolean for white border
                           large,            // Boolean for large size
                           onclick,          // Click handler
                           red,              // Boolean for red outline style
                           green,            // Boolean for green outline style
                           bgColor,          // Custom background color (optional)
                           textColor : string  = "white" // Default text color
                       }) => {
    // Default colors
    const defaultBg : string  = "var(--color-primary)";
    const defaultHover : string  = "#0060D0";
    const redColor : string  = 'rgb(239 68 68)';
    const greenColor : string  = 'rgb(34 197 94)';

    // Determine button style based on props
    const getButtonStyle = () : {...} | {...}  => { Show usages
        if (red) {
            return {
                backgroundColor: 'transparent',
                color: redColor,
                border: `1px solid ${redColor}`
            };
        }
        if (green) {
            return {
                backgroundColor: 'transparent',
                color: greenColor,
                border: `1px solid ${greenColor}`
            };
        }
        return {
            backgroundColor: bgColor || defaultBg,
            color: textColor,
            border: border ? "1px solid white" : 'none'
        };
    };

    // Handle hover effects
    const handleMouseEnter = (e) : void  => { Show usages
        if (red) {
            e.target.style.backgroundColor = 'rgba(239, 68, 68, 0.1)';
        } else if (green) {
            e.target.style.backgroundColor = 'rgba(34, 197, 94, 0.1)';
        } else {
            e.target.style.backgroundColor = hoverColor || (bgColor ? darkenColor(bgColor) : defaultHover);
        }
    };

    // Handle mouse leave effects
    const handleMouseLeave = (e) : void  => { Show usages
        if (red || green) {
            e.target.style.backgroundColor = 'transparent';
        } else {
            e.target.style.backgroundColor = bgColor || defaultBg;
        }
    };

    return (
        <button
            onClick={onclick}
            className={`font-medium w-full px-4 py-2 text-[12px] rounded-[5px] transition duration-300 ease-out ${
                border ? "border-1  border-white" : ""
            } ${
                large ? "text-[14px] px-5 py-3" : ""
            }`}
            style={getButtonStyle()}
            onMouseEnter={handleMouseEnter}
            onMouseLeave={handleMouseLeave}
        >
            {children}
        </button>
    );
};

// Helper function to darken colors for hover effect
function darkenColor(color, amount : number  = 0.2) : any  { Show usages
    if (color.startsWith('#')) {
        return color; // Simplified for example - implement proper hex darkening if needed
    }
    if (color.startsWith('rgb')) {
        // RGB color darkening
        return color.replace(/\d+/g, match => Math.max( values: 0, parseInt(match) * (1 - amount)));
    }
    return color;
}
```

Figure 42 - Button Component

6.1.3 Layout Components

To ensure the reusability of the code and its clean structure, I've built different layout components for distinct parts of the system. The Website Layout centralizes the general user-facing pages and shares components such as the navbar, footer, and theme toggler. Conversely, the Admin Dashboard Layout is where the author put all the admin functionalities into a structured container that houses a side navigation menu and a top bar that can be used to access management tools very quickly. The use of layout components makes it possible for me not to create the same structure for every page file which is also a visual representation of the DRY (Don't Repeat Yourself) model. Additionally, the code becomes easy to handle, and it is consistent throughout the whole project. Therefore, each page of relevance only needs to encase its content with the right layout further enabling better modularity and scalability within the application.

WebsiteLayout.jsx

```jsx
import {Header} from "../components/website/Header.jsx";
import {Footer} from "../components/website/Footer.jsx";
import {Outlet} from "react-router-dom";

export const WebsiteLayout = () => {  Show usages
    return (
        <div className="flex flex-col min-h-screen">
            <Header />

            <main className="flex-grow">
                <Outlet />
            </main>

            <Footer />
        </div>
    )
}
```

Figure 43 - Website Layout

6.1.4 Theme Management

For the usability and pleasure of the eye, the author have added a switch for the Dark/Light theme in the BlockEstate platform. the author accomplished it by creating a central Theme Context, with which the author enveloped the entire application. The Theme Context shares the current theme state across all the components. the author made use of React Context API for this, which makes easy for the individual elements by which it is called to get access or switch the theme. Besides, the theme preference selected by the user is saved in the browser using localStorage, so one can keep their settings even after re-entering or refreshing the page. Such universal theme settings management not only ensures the constant clear look for the whole app, but it also empowers users to opt for their choice of mode of viewing.

ThemeContext.jsx

```jsx
import { createContext, useEffect, useState, useContext } from "react";

const ThemeContext : Context<unknown> = createContext();

export const ThemeProvider = ({ children }) => {  Show usages
    const [theme : string , setTheme] = useState( initialState: localStorage.getItem( key: "theme") || "light");

    useEffect( effect: () : void  => {
        // Apply the theme to <html> tag
        document.documentElement.classList.toggle( token: "dark",  force: theme === "dark");
        localStorage.setItem("theme", theme);
    }, deps: [theme]);

    return (
        <ThemeContext.Provider value={{ theme, setTheme }}>
            {children}
        </ThemeContext.Provider>
    );
};

// Custom hook for easy access
export const useTheme = () => useContext(ThemeContext);  no usages
```

Figure 44 - Theme Context

6.1.5 Routing

Routing in the BlockEstate project was implemented with the help of React Router, a tool designed to enable the smooth transition between pages in an application, without the need to reload the app itself. the author organized the routing into various layout groups to be able to easily maintain and reuse the code. For instance, the author made use of a separate layout for general website pages, another one for the admin dashboard, and one for agency or investor dashboards. This was a way to keep the structure of each section clear and, at the same time, ensure role-based navigation. The main parts of the routing structure include the following types of routes: public (such as Home, About, and Contact), authentication, and private (available only for authorized users) ones.

```
{/* Auth Routes */}
<Route path="/auth" element={<AuthLayout />}>
    <Route index element={<LoginPage />} />

    <Route path="type" element={<TypeSelectPage />} />
    <Route path="otp" element={<OTPVerifyPage />} />
    <Route path="forget" element={<ForgetPasswordPage />} />
    <Route path="new-password" element={<NewPasswordPage />} />

    <Route path="investor-register" element={<InvestorRegisterPage />} />
    <Route path="investor-setup" element={<InvestorAccountSetupPage />} />

    <Route path="agency-register" element={<AgencyRegisterPage />} />
    <Route path="agency-setup" element={<AgencyAccountSetup />} />
    <Route path="agency-approve" element={<AgencyApprovePage />} />
</Route>

{/* Admin Routes */}
<Route path="/admin" element={<AdminDashboardLayout />}>
    <Route index element={<AdminDashboard />} />

    <Route path="user" element={<UserManagement />} />
    <Route path="user/:userId" element={<SingleUserManagement />} />

    <Route path="property" element={<UserManagement />} />

    <Route path="agency" element={<AgenciesManagement />} />
    <Route path="agency/:agencyId" element={<SingleAgencyManagement />} />


    <Route path="resale" element={<UserManagement />} />
    <Route path="transaction" element={<UserManagement />} />
    <Route path="sponsored" element={<UserManagement />} />
    <Route path="analytic" element={<UserManagement />} />
    <Route path="setting" element={<UserManagement />} />
    <Route path="profile" element={<UserManagement />} />
</Route>
```

Figure 45 - Front-end Routes

6.1.6 Firebase Authentication Setup

For user authentication, the author decided to implement Firebase Authentication in the client side of the application. the author created a Firebase config file where the author initialized all the Firebase services. After that, the author decided to use the React Context API to create a custom Firebase Auth Context which will hold the authentication logic and user after login, also, this information will be shared across the whole application. Among other things, this context verifies user's authentication and follows the auth state, it also helps the user to carry out the login, logout, and sign-up operations directly. It also ensures that the token is refreshed automatically, and that the user is still logged in until the expiry of the token or logout.

AuthContext.jsx

```
import { createContext, useContext, useEffect, useState } from "react";
import { onAuthStateChanged, signOut } from "firebase/auth";
import { auth } from "../firebase/firebase.config.js";

const AuthContext : Context<unknown> = createContext();

export const AuthProvider = ({ children }) => {  Show usages
    const [user, setUser] = useState( initialState: null);

    useEffect( effect: () => {
        const unsubscribe : Unsubscribe = onAuthStateChanged(auth, nextOrObserver: (currentUser : User ) : void => {
            console.log("onAuthStateChanged >>", currentUser);
            setUser(currentUser);
        });

        return () : void => unsubscribe();
    }, deps: []);

    const signOutUser = async () : Promise<void> => {  Show usages
        try {
            await signOut(auth);
            console.log("User signed out successfully.");
        } catch (error) {
            console.error("Sign out error:", error);
        }
    };

    return (
        <AuthContext.Provider value={{ user, signOutUser }}>
            {children}
        </AuthContext.Provider>
    );
};

export const useAuth = () => useContext(AuthContext);  Show usages
```

Figure 46 - Auth Context

FirebaseAuthContext simplifies global user management and improves security across protected routes.

6.2 Back-end

6.2.1 Folder Structure

For the proper keeping and scaling of the code base, the author fashioned the backend for BlockEstate following a monolithic architecture. This concept is based on the integration of all the backend components within one codebase, where each module (e.g., routes, controllers, models) fits into the same application, but the structure assures their separation for the purposes of clarity and being able to maintain them in the future. This pattern facilitates not only the management of your app but also its troubleshooting because all the logic and the components are in one place, therefore, the development process becomes very clear.

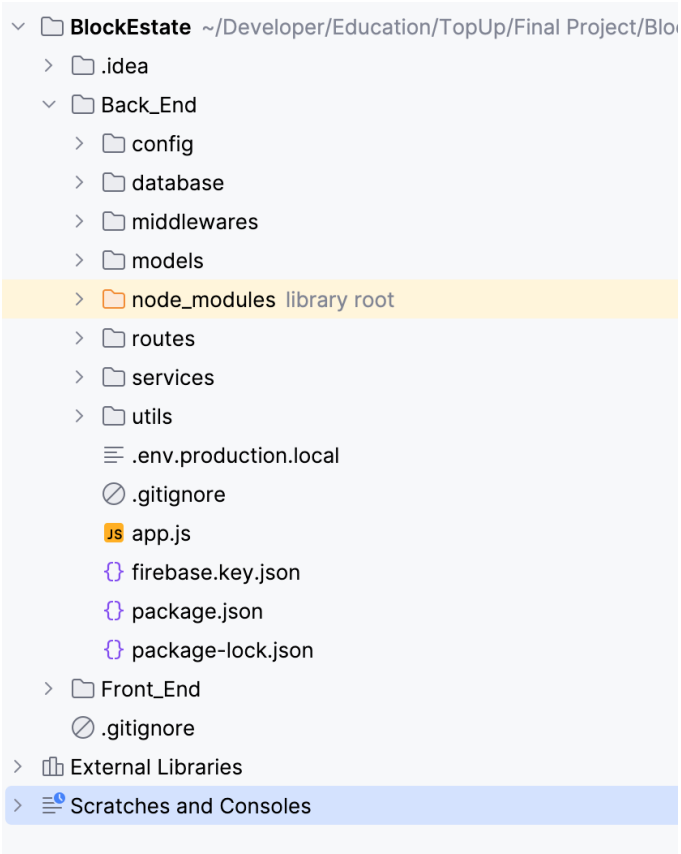

Figure 47 - Back-end Folder Structure

Here's a screenshot of the folders arranged systematically. The routes/ folder oversees managing API endpoints, controllers/ supervise business logic, models/ are used to describe MongoDB schemas, middleware/ consists of custom middleware (like auth and error handling), and config/ had the settings for databases and mail services made externally.

6.2.2 Database Connectivity

The project is taking advantage of MongoDB as the main database to hold all the centrally formatted data which includes user details, property listings, and investment records. The access is controlled via a single centralized configuration file and the official MongoDB Node.js driver (through Mongoose). Vital data such as the connection string is safely stored in a .env file making it possible to keep the application safe and secure.

mongodb.js

```
1  import mongoose from 'mongoose';
2  import { DB_URI } from "../config/env.config.js";
3
4  if(!DB_URI){
5      throw new Error("MongoDB URI doesn't exist");
6  }
7
8  const connectToDatabase = async () : Promise<void> => {  Show usages
9      try {
10         await mongoose.connect(DB_URI);
11         console.log("MongoDB Connected successfully");
12     } catch(err){
13         console.error("Error connecting database");
14         process.exit( code: 1);
15     }
16 }
17
18 export default connectToDatabase;  Show usages
```

Figure 48 - Database Connectivity

The code above initializes the MongoDB connection using mongoose.connect() and logs the connection status. The DB_URI is read from environment variables for better security.

6.2.3 Error Handling Middleware

The author have developed a custom error handling middleware to handle backend errors more efficiently. The middleware ensures that every type of error is properly intercepted and replied to in a coherent format, regardless of whether it is a database operation failure or an invalid route. It is beneficial for both debugging and giving explicit frontend-related answers.

error.middleware.js

```
const errorMiddleware = (err, req, res, next) : void => {  Show usages
    console.error("Error:", err);

    let statusCode : number  = err.statusCode || 500;
    let message : string  = err.message || 'Internal Server Error';

    // Handle MongoDB ObjectId errors
    if (err.name === 'CastError') {
        message = 'Resource not found. Invalid ID.';
        statusCode = 404;
    }

    // Handle Mongoose duplicate key errors
    if (err.code === 11000) {
        const field : string  = Object.keys(err.keyValue)[0];
        message = `Duplicate value entered for field: "${field}"`;
        statusCode = 400;
    }

    // Handle Mongoose validation errors
    if (err.name === 'ValidationError') {
        message = Object.values(err.errors).map(val => val.message).join(', ');
        statusCode = 400;
    }

    // Handle Firebase auth errors (if any)
    if (err.code?.startsWith('auth/')) {
        message = `Firebase Auth Error: ${err.message}`;
        statusCode = 401;
    }

    res.status(statusCode).json({
        success: false,
        error: message
    });
};

export default errorMiddleware;  Show usages
```

Figure 49 - Error Handling Middleware

This code captures any thrown errors and sends a proper status code and message to the client. It also handles custom error messages if thrown intentionally from the controller.

6.2.4 Models

The backend is composed of many models according to the schema requirements. Specifically, the User model interestingly the basic structure for both investors and agencies, while the Property model stores the details of the listed properties. Furthermore, these models come with the validation rules and schema relationships used for data consistency guaranteeing.

admin.model.js

```
import mongoose from 'mongoose';

const adminSchema : Schema<any, Model<...>, {...}, {...}, {...}, {...}, DefaultSchemaOptions, {...}> = new mongoose.Schema( definition: {
    firstName: {
        type: String,
        minlength: 2,
        maxlength: 60,
        trim: true,
    },
    lastName: {
        type: String,
        minlength: 2,
        maxlength: 60,
        trim: true,
    },
    nic: {
        type: String,
        minlength: 2,
        maxlength: 20,
        default: "NOTSET",
        trim: true,
    },
});

const Admin : Model<...> = mongoose.model( name: 'Admin', adminSchema);
export default Admin; no usages
```

Figure 50 - Admin Model

user.model.js

```
import mongoose from 'mongoose';

const userSchema : Schema<any, Model<...>, {...}, {...}, {...}, {...}, {...}, {...}> = new mongoose.Schema( definition: {
    firebaseId: {
        type: String,
        required: true,
        unique: true,
    },
    role: {
        type: String,
        enum: ['ADMIN', 'AGENCY', 'INVESTOR'],
        default: 'INVESTOR',
        trim: true,
    },
    displayName: {
        type: String,
        required: true,
        trim: true,
    },
    profileImageUrl: {
        type: String,
        required: true,
    },
    isActive: {
        type: String,
        enum: ['ACTIVE', 'INACTIVE'],
        default: 'ACTIVE',
        trim: true,
    },

}, options: { timestamps: true });

const User : Model<...> = mongoose.model( name: 'User', userSchema);
export default User;  Show usages
```

Figure 51 - User Model

The User schema defines fields like firebaseUid, role, and personal info, while Property includes title, description, blocks, and more. Both use Mongoose validators and timestamps.

6.2.5 API Endpoints

Through the frontend-to-backward communication, the author have drawn up RESTful API endpoints employing Express routes and controllers. Each route method (GET, POST, etc.) corresponds to a controller function that processes the data logic. This division of routes and controllers is helpful for managing and testing each API separately since we can focus use-case logic in a single place.

```
545  authRouter.get( path: '/check-firebase-user/:firebaseId', handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> ,
546      const { firebaseId } = req.params;
547
548      console.log(firebaseId)
549
550      let user : Query<...> = await User.findOne( filter: { firebaseId });
551
552      if (user) {
553          res.json( body: { exists: true, user });
554      } else {
555          res.json( body: { exists: false });
556      }
557  });
```

Figure 52 - API Endpoint 1

```
20  authRouter.get( path: '/register', handlers: (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<ResBody, LocalsObj:
21      res.send( body: {message: "Auth sign up!"});
22  });
23
24
25  authRouter.post( path: '/register-otp', handlers: async(req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<ResBo
26      const { email } = req.body;
27      const otp : number = Math.floor( x: 100000 + Math.random() * 900000);
28
29      const expiresAt : number = Date.now() + 3 * 60 * 1000;
30
31      try {
32          await sendOtpEmail(email, otp);
33          res.json( body: { message: "OTP sent successfully", otp, expiresAt });
34      } catch (error) {
35          console.error("Email sending error:", error);
36          res.status( code: 500).json( body: { error: "Failed to send email" });
37      }
38  });
39
```

Figure 53 - API Endpoint 2

6.2.6 Route Integration

All the API routes used in the application are consolidated into a single file, app.js. Each category of routes is isolated from its own file and linked up with a base path like /api/users or /api/properties. This is also the spot where middleware such as auth and error handlers is introduced.

app.js

```
1  import express from 'express';
2  import connectToDatabase from "./database/mongodb.js";
3  import errorMiddleware from "./middlewares/error.middleware.js";
4  import { PORT } from './config/env.config.js';
5  import authRouter from './routes/auth.routes.js'
6  import cookieParser from "cookie-parser";
7  import cors from "cors";
8
9
10 const app : Express = express();
11
12 app.use(express.json());
13 app.use(express.urlencoded( options: { extended: true }));
14 app.use(cookieParser());
15 app.use(cors());
16
17 // Routes
18 app.use('/api/auth/', authRouter);
19
20 app.use(errorMiddleware);
21
22 app.get('/', (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<ResBody, LocalsObj> ) : void => {
23     res.send( body: 'Welcome to BlockEstate api');
24 });
25
26 app.listen(PORT, hostname: async () : Promise<void> => {
27    console.log(`BlockEstate listening on port ${PORT}!`);
28  await connectToDatabase();
29 });
```

Figure 54 - App js

he app.js file imports and mounts all route files. It also applies middleware like JSON parsing, CORS, and error handling to ensure consistent behavior across the app.

6.2.7 Email Service Management

To empower features like email-based OTP or the delivery of notifications, the author created a mail service using nodemailer. The mail configuration (host, port, auth) is kept in a reusable config/mail.js file. The emails' contents (HTML content) are maintained at different locations and are added on the fly as they are sent.

Mail Configuration

mailer.config.js

```
1  import nodemailer from 'nodemailer';
2  import { SMTP_USER, SMTP_PASS } from "./env.config.js";
3
4  const transporter = nodemailer.createTransport( transporter: {
5      service: 'gmail',
6      auth: {
7          user: SMTP_USER,
8          pass: SMTP_PASS,
9      },
10 });
11
12 export default transporter;  Show usages
```

Figure 55 - Mail Configuration

Mail Service

mailer.config.js email.service.js

```
// services/email.service.js
import { otpTemplate } from '../utils/otpEmailTemplate.utils.js';
import { SMTP_USER } from "../config/env.config.js";
import transporter from "../config/mailer.config.js";
import {welcomeTemplate} from "../utils/welcomeEmailTemplate.js";
import {welcomeAgencyTemplate} from "../utils/welcomeAgencyEmailTemplate.js";


export const sendOtpEmail = async (to, otp) : Promise<void>  => {  Show usages
    const mailOptions :{...}  = {
        from: `"BlockEstate" <${SMTP_USER}>`,
        to,
        subject: "Your OTP Code",
        html: otpTemplate(otp),
    };

    await transporter.sendMail(mailOptions);
};

export const sendWelcomeEmail = async (to, name) : Promise<void>  => {  Show usages
    const mailOptions :{...}  = {
        from: `"BlockEstate" <${SMTP_USER}>`,
        to,
        subject: "Welcome to BlockEstate! 🎉",
        html: welcomeTemplate(name),
    };
    await transporter.sendMail(mailOptions);
};

export const sendAgencyWelcomeEmail = async (to, name) : Promise<void>  => {  Show usages
    const mailOptions :{...}  = {
        from: `"BlockEstate" <${SMTP_USER}>`,
        to,
        subject: "Welcome to BlockEstate! 🎉",
        html: welcomeAgencyTemplate(name),
    };
    await transporter.sendMail(mailOptions);
};
```

Figure 56 - Mail Service

Mail Template

otpEmailTemplate.utils.js

```
export const otpTemplate = (otp) : string => ` Show usages
  <div style="font-family: Arial, sans-serif; max-width: 600px; margin: auto; padding: 20px; border: 1px solid #e0e0e0; border-radius: 8p

    <h1 style="color: #0274F9; text-align: center;">BlockEstate</h1>

    <p style="font-size: 16px; color: #333;">
      Hi there 👋 ,
    </p>

    <p style="font-size: 16px; color: #333;">
      Thanks for signing up with <strong>BlockEstate</strong>. To complete your registration, please verify your email address by enterin
    </p>

    <div style="text-align: center; margin: 20px 0;">
      <span style="font-size: 28px; font-weight: bold; letter-spacing: 4px; background-color: #f3f4f6; padding: 10px 20px; border-radius:
        ${otp}
      </span>
    </div>

    <p style="font-size: 14px; color: #555;">
      This code will expire in <strong>10 minutes</strong>. If you didn't request this, you can safely ignore this email.
    </p>

    <hr style="margin: 30px 0;">

    <p style="font-size: 12px; color: #888; text-align: center;">
      © ${new Date().getFullYear()} BlockEstate Inc. All rights reserved.
    </p>
  </div>
`;
```

Figure 57 - Mail Template

6.2.8 Firebase Auth and Token Management

Firebase Authentication is responsible for login and verified users. When there is an authentication status on the front-end, the token for Firebase is sent to the back end. After the token is verified with Firebase Admin SDK by the custom middleware, they also check for user roles shch as an Investor or Agency. The token is expired according to the Firebase 1-hour expiration time, and then the users will receive a notification to authenticate again.

firebaseAdmin.config.js verifyToken.middleware.js

```
1  const admin :{…} = require("firebase-admin");
2
3  // initialize admin only once
4  if (!admin.apps.length) {
5      admin.initializeApp( options: {
6          credential: admin.credential.cert(require("../config/firebaseAdmin.config.js")),
7      });
8  }
9
10 const verifyFirebaseToken = async (req, res, next) :Promise<…> => {  Show usages
11     const authHeader :string = req.headers.authorization;
12
13     if (!authHeader || !authHeader.startsWith("Bearer ")) {
14         return res.status(401).json({ message: "Unauthorized" });
15     }
16
17     const token :string = authHeader.split( separator: " ")[1];
18
19     try {
20         const decodedToken :DecodedIdToken = await admin.auth().verifyIdToken(token);
21         req.user = decodedToken; // you can access uid, email, etc. from here
22         next();
23     } catch (error) {
24         console.error("Token verification failed:", error);
25         return res.status(403).json({ message: "Invalid token" });
26     }
27 };
28
29 module.exports = verifyFirebaseToken;
```

Figure 58 - Token Middleware

6.2.9 Firebase User Creation

For me to securely handle user identities, the author resort to Firebase Authentication in conjunction with the backend and MongoDB. After a user finish setting up by, for instance, typing a display name, a password, and uploading a profile picture, their Firebase account is registered. Following that, the backend API manages the user's basic profile information—e.g., the database is kept as Firebase UID, display name, email, and role. Thus, the system can bind each Firebase user to the database-stored extensive profile data through this method.

```
// Step 1: Save user in firebase
const userRecord : UserRecord  = await admin.auth().createUser( properties: {
    email,
    password,
    displayName,
    emailVerified: true,
});

uid = userRecord.uid;
```

Figure 59 - Save User Firebase

```
// Step 3: Save data in database
const imageUrl : string  = `https://storage.googleapis.com/${bucket.name}/${fileName}`;

newUser = new User( doc: {
    firebaseId: uid,
    role: role || 'INVESTOR',
    displayName,
    profileImageUrl: imageUrl,
});
await newUser.save();

newInvestor = new Investor( doc: {
    firebaseId: uid,
    firstName,
    lastName,
    nic,
    address,
});
await newInvestor.save();

await sendWelcomeEmail(email, displayName);
```

Figure 60 - Database Saving

6.2.10 Firebase Storage

Firebase Storage is the place where assets are kept, which is the PDF documents and profile pictures. Also, Defour verification files. Using Firebase SDK, the files are uploaded from the frontend and then they are kept securely in the configured storage bucket. When the files are uploaded, the URL can be fetched, and it can be saved in the MongoDB database as one of the user or property records. The separation results in well-managed resources and data without being overwrought by large files.

```
// Step 2: Upload profile picture with UID as the file name
const bucket : Bucket  = admin.storage().bucket();

const extension = profilePicture.originalname.split('.').pop();
fileName = `Profile_Pictures/${uid}.${extension}`;

const file : File  = bucket.file(fileName);

await file.save(profilePicture.buffer,  options: {
    contentType: profilePicture.mimetype,
    public: true,
});
```

Figure 61 - Firebase Storage

The uploaded file is resized and compressed before uploading (e.g., profile pictures are 200x200 pixels and under 500KB). After upload, the file's download URL is saved to the backend as part of the user's or agency's profile data. This helps in serving images efficiently and managing storage separately from database logic.