A Project Report on

BLOCKCHAIN BASED LAND REGISTRY SYSTEM WITH AADHAR AUTHENTICATION

Submitted in partial fulfilment of the academic requirement for the award of the degree of Bachelor of Technology in Information Technology

By

Achintya Prathikantam - 19251A1261

Ayushi Verma - 19251A1264

Geetha Krishna Guruju - 19251A1272

Priyanka Myaragalla - 19251A1292

Project Batch Number – 19B03

Under the guidance of

Mrs. Ramya Ponnada (Assistant Professor)

Department Of Information Technology

G. Narayanamma Institute of Technology and Science (for Women)

(AUTONOMOUS)

Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad

Accredited by NBA & NAAC Shaikpet, Hyderabad – 500104, TS.

August 2022 - April 2023

CERTIFICATE

This is to certify that the project report entitled **Blockchain based Land Registry System with Aadhar Authentication** is bonafide work done by:

Achintya Prathikantam	-	19251A1261
Ayushi Verma	-	19251A1264
Geetha Krishna Guruju	-	19251A1272
Priyanka Myaragalla	-	19251A1292

under the guidance of Mrs. P.N.Ramya, during **August 2022 to April 2023**, in partial fulfillment for the award of degree in B.Tech in Information Technology, from G.Narayanamma Institute of Technology and Science.

Internal Guide
Mrs. P.N.Ramya
Assistant Professor
Department of IT

Head of the Department
Dr. I. Ravi Prakash Reddy

ACKNOWLEDGEMENT

We would like to express our sincere thanks to **Dr. K. Ramesh Reddy, Principal**, GNITS, for providing the working facilities in the college. Our sincere thanks and gratitude to **Dr. I. Ravi Prakash Reddy, Professor and HOD, Dept. of IT**, GNITS for all the timely support and valuable suggestions during the period of our Major project.

We are extremely thankful and indebted to our internal guide, **Mrs.P.N.Ramya**, **Assistant Professor, Department of IT**, GNITS for her constant guidance, continuous advice, encouragement and moral support throughout the Major project.

Finally, we would also like to thank all the faculty and staff of the IT Department who helped us directly or indirectly and for their cooperation in completing the Major project work.

ABSTRACT

Property Registry is one of the use cases that involve a lot of intermediaries to put trust in the system. In the existing system, tracking who owns which piece of land is challenging when you have thousands of property records to maintain and mostly all these records are not digitized or take a long time to get digitized.

Blockchain in the land registry is utilized for secure transfer of property. The transparent nature of blockchain makes it possible to track the transfer of ownership from one individual to another reliably. Blockchain's immutable, auditable, and traceable features are enticing governments around the world to implement the decentralized technology in the land registry process. In our project we try to make a system that is trustworthy and transparent in the dealings of property records. This is achieved by adding multiple layers of security including aadhar authentication. One-time-password based verification improves the overall personal security and prevents unauthorized access to the system. A systematic approach is used, right from the registration of the land inspector/buyer/seller to the registration of lands, making it available to sell, etc.

A solution of decentralized application or DAPP is proposed through this project, which will be a one stop platform for buying, selling, or registering land.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 GENERAL DESCRIPTION

Land is one of India's most contentious topics. It lacks a correct scheme for maintaining property documents and providing the outcomes of rare and long drawn legal conflict to an individual with conclusive titles. Today in India land title does not ensure its full rights to an owner. In addition, property transactions are carried out on paper and not very frequently updated, resulting in countless conflicts over property. The integrity and proper track of ownership/transfer records of land is a highly challenging task.

As the ownership of land can constantly change over time and that too sometimes very frequently, it poses a daunting task of keeping elaborate and long ownership transfer records. The problem further escalates due to presence of fraudulent or incomplete registries which are very difficult to trace back through time. Thus, ownership disputes in the system, lead to litigation running for years, leading to wastage of valuable time, energy, and resource for solving these disputes. Most of the issues root from the problem of the current land registration systems being either having legacy paper document trails or from poorly kept non-transparent centralized systems. Fraudulent users may try to forge paper documents or modify electronic records to change the land ownership record.

Blockchain technology is a combination of blockchain data-structure and blockchain system or network. Data in blockchain is stored in a chronologically ordered list of interconnected units called blocks, which form blockchain data-structure. These blocks are made of a block header and transaction data. The block header is made of three main elements:

- Block metadata, which stores information about a block itself (e.g., index and creation timestamp.
- Hash reference to a previous block, which is created by application of a hash function to contents of the previous block.
- Hash reference to stored transaction data, which are generated by application of a hash function to the transaction data.

The transaction data is a list of transactions and their respective data saved within the

block. A hash function maps data of an arbitrary size to a unique bit string of a fixed size called hash value or hash reference. Due to the functions' properties, hash values are easy to calculate but very difficult to invert from the point of view of computational theory.

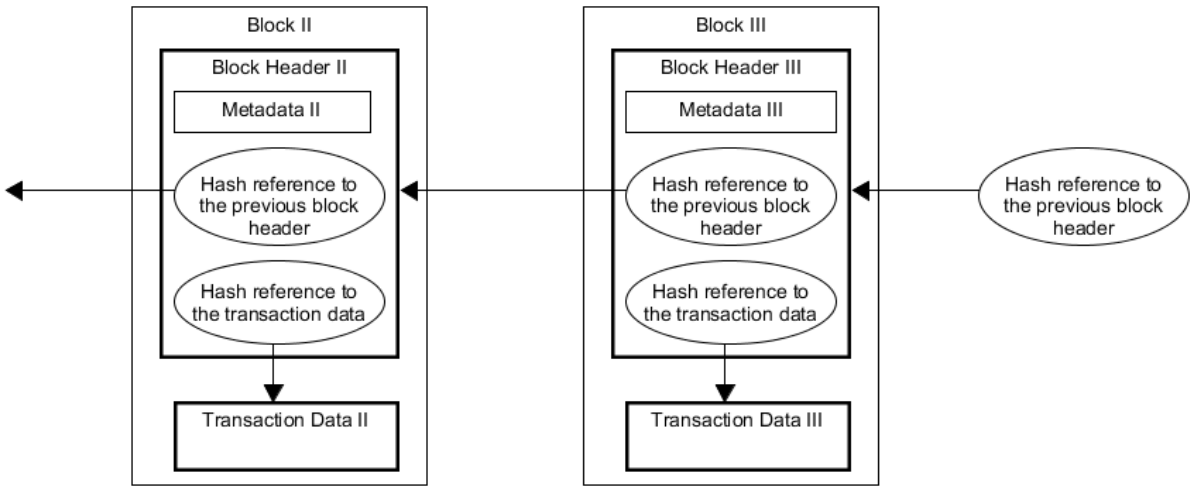

Fig: 1.1 Structure of a blockchain data structure

1.2 OBJECIVE & SCOPE OF THE PROJECT

The main objective of our system is to create a Decentralized application (DAPP) which will be a one stop platform for buying, selling, and registering land securely and conveniently.

- To implement the land registry process using blockchain technology for enhanced security.
- To make a decentralized system which is immutable and tamper proof.
- To add Aadhar authentication for personal security and authorized access.
- To accelerate the process of removing the middle men that hold the information.
- To bring transparency in the system as buyers and sellers can directly communicate with each other and users can access their records very easily.

The primary scope of this project would be to create a one stop decentralized application for buying, selling, and registering land. The Aadhar authentication would ensure security and prevent any unauthorized access. Maintaining immutable and tamper-proof records of transactions between buyers and sellers by enabling direct communication between them will ensure transparency in the system.

1.3 PROBLEM STATEMENT

In India, property transactions are carried out on paper and not very frequently updated, resulting in countless conflicts over property. Land documents are centralized and preserved in the sub-registrar's office in India. It is also possible that the record may be altered or manipulated.

It is a tedious process and most often people are not aware of the entire rules to be followed during registration process. Also, more documents need to be verified and thus it takes delay in completing registration. In addition to this, the middlemen collect bribes to complete this process. Mistakes also may occur while processing land records.

1.4 ORGANIZATION OF PROJECT REPORT

The report has been divided into 5 major chapters.

Chapter 1, *Introduction*, is the introduction to the project, giving a brief overview about the entire project. Also describing the objectives and scope of the project.

Chapter 2, *Literature Survey* consists of the information about the existing systems.This chapter gives the detailed description of the existing systems advantages and disadvantages. Also, the motivation for the proposed system.

Chapter 3, *Requirements Specification* is devoted to the proposed system, Blockchain based Land Registry system with Aadhar Authentication.This chapter describes the overall description of the project. It consists of functional and non-functional requirements; followed by the system requirements of the project.

Chapter 4, *Design Specification* consists of the design related information along with the UML (Unified Modelling Language) diagrams.

Chapter 5, *Implementation*, this chapter depicts the Implementation of proposed system which are reviewed in detail about Methodology and System Architecture and the Modules.

Chapter 6, *Testing*, this chapter includes the test plan and test cases incorporated in our project.

Chapter 7, *Results & Conclusion*, this chapter includes the result and performance analysis along with the conclusion and future scope.

The *Appendix* includes the screenshots of the application, the code used and the references.

All the parts and subparts of the report, including figures and tables, have been numbered sequentially for easy identification.

2. LITERATURE SURVEY

2.1 EXISTING SYSTEM

Blockchain enabled Digitization of Land Registration

Published in May 2021, by *IEEE*

Authors: By Suganthe, R. C., N. Shanthi, R. S. Latha, K. Gowtham, S. Deepakkumar, and R. Elango

The land registration refers to a system whereby ownership and land-related rights are recorded by a government entity. The land records provide evidence of title, facilitate transactions, and prevent fraud. Blockchain network is transparent and data once entered in the block cannot be tampered. When the whole process and person involved in land registry are present in the blockchain network that will nearly make it impossible for anyone to show miscalculated and alerted value, because all the transactions made are transparent and in case any fault happens it is easier to identify.

The system proposed in this paper aims at providing the exact details of land records and ownership. These details are already provided in normal paper-based registration system, but those data could be altered easily but with the use of blockchain and storing such details in a block will ensure no fraud takes place and data will remain unaltered.

The modules identified and implemented in the proposed system are: 1. Setting up ether wallet for executing smart contracts in Ethereum blockchain network, 2. Collecting details from user through web app and 3. Creating blocks using smart contracts. Solidity language is used for implementing smart contracts. These contracts are compiled and deployed using Remix IDE. Injected Web3 Environment are selected while compiling to get the Application Binary Interface (ABI) codes which is required for connecting the web app with Ethereum network using web3.js.

The major drawbacks of this system are that it doesn't enable users to purchase the land or change the ownership of the land details. The proposed system can only get the land details from owners and these details are stored within the block by using the blockchain technology. More features like purchasing, selling, gifting, ownership transfer, land splitting and verification can improve the traceability, transparency and eliminate the third-party involvement in the system.

Land Registration using Blockchain Technology

Published in June 2021, by *International Journal of Emerging Technologies and Innovative Research*

Authors: By Mohammed Moazzam Zahuruddin, Dr. Sangeeta Gupta, Shaik Waseem Akram

This work dealt with the basic smart contract creation and deployment of the Land Registration process. Block chain has been used for faster execution of land transactions. This system typically considers a land sale transaction and the process is carried in 5 steps. The proposed work is implemented on Ethereum blockchain platform with solidity as a notion to code the contract between various users. All the functionalities thought necessary in the land registration process have been implemented and tested on the Remix IDE. The work is initiated onto a personal device as an initial phase with a minimum of 4 GB memory requirement independent of the operating system.

In the proposed system, the user's data is obtained from a central citizenship data that is maintained by the government. The land owner has to initiate the transaction which is completed by the buyer, this is called as dual consensus regarding a transaction. This system handles the case where the owner of land is absent and thus allocating its owner as the government.

The major drawbacks of the system proposed are offline land details verification, no provision for land splitting process. The Land Registration process can be further enhanced by automating the Land Verification Process and Land Updating Process. Further versions could be developed to include Land Splitting Case. Also, this system mainly depends on the citizenship data that is maintained by the government, failure of which might lead to a halt in the process. It also has some assumptions that the land owner details are correct and true, and hence are not verified.

Blockchain and Smart Contract for Land Registration using Ethereum Network

Published in July 2022, by *International Journal of Engineering Research & Technology*

Authors: By Rakesh Kumar K V, Rithick Gokul A, V. Nirmal Kumar

This paper proposes a land document registration system based on Ethereum and IPFS. This method ensures that user papers kept in the IPFS garage are secure. They expanded an information garage software to illustrate the process. The garage utility records consumer actions such as add,

delete, and edit for consumer-specific files. These log files are saved on the IPFS network, which also provides the Hash. These hashes of provenance statistics are subsequently preserved as a transaction in Ethereum blockchain networks.

If an attempt is made to change a provenance statistics record, we also validate the mechanism. There could be an admin section on the land registry device. The Property Id will be entered by the administrator, and it will be searched in the database. All of the database's matched entries might be shown as URLs inside the browser. As a result, they may obtain a list of all of the belongings possessed by utilizing the owner. This system will solve the problems, such as single point of failure and centralized access. As a result, fraudsters may be prevented from using this.

The major drawback of the system is that it simply secures the documents stored on IPFS and doesn't provide any provision for communication between the land buyers and sellers. This can lead to issues like double selling. Without direct communication, there might be disputes about the price and location of the land.

Land Registry Management using Blockchain

Published in 2020, by *IEEE*

Authors: By Ameya Thosar, Mayur Harne, Ashutosh Sarode, Dr. Parminder Kaur

Land Registration is a use case which involves lot of middlemen and central authorities in the process which then puts trust in the system. Keeping traces of who owns which part of land is challenging when there are hundreds or thousands of land records to maintain. Using Blockchain will remove the middlemen in the system and also will reduce corruption and increase speed of the process. Land Registration is a simple decentralized application which is build using the Ethereum Blockchain principals. We can use this registration procedure as a substitute to bypass the existing system flaws.

In the proposed system, the main objective to build the platform is to Accelerate the Process removing the middle men that hold the information. Blockchain land registration is a platform that serves you a distributed database where anyone has a right to record and access information without participation of any centralized authority. By maintaining an immutable records of transactions, Blockchain is able to prove that you are the owner of the land and prevent use of false documents to prove the ownership of the land.

The major drawbacks of the system proposed are that the use of a simple database might put the system at risk. Here we can access the database anytime to store land related data and verify

transactions between users. Threats to the database might lead to loss of a huge amount of important land and user data.

Blockchain based land registry system using Ethereum Blockchain

Published in April 2020, by *Journal of Xi'an University of Architecture & Technology*

Authors: By Rizwan Khan, Saksham Sachdeva, Shadab Ansari, Sneha Jain

Land registry in India as well as in many parts of the world is very slow and cumbersome process. There are also many intermediaries involved in the process of land registration. Developing a system that not only accelerate the process of land registration, but also make it easier for Buyers, Sellers and Government registrars to transfer the land ownership from seller to a new buyer, is only possible by creating a distributed system that store all the transactions made during the process of land buying. In this paper we'll try to explore the possibilities and problems solved by using a blockchain based system for land ownership transfer.

The current system that they are trying to implement is based on Ethereum Blockchain that will store all the transactions made during the process of land ownership transfer. Using the concept of smart contracts of blockchain technology we can triggers various events like access of land documents to a land inspector and fund transfer event from buyer to seller after successful verification of the land ownership transfer. This system will solve the problems faced by all the three parties during the land registration and will also remove the intermediaries like property dealers. This system makes the process of land registration resilient and decrease the cases of fraud in the process. Using the system, validation of the lands is also possible as immutable transactions are being stored in the public ledger.

2.2 DRAWBACKS IN EXISTING SYSTEM

In the paper "Blockchain enabled Digitization of Land Registration", there is a significant scope to develop this project further by designing a suitable web application and integrating it with the smart contract and Ethereum Meta Mask application to make it more user friendly and easy to use. The major drawbacks of this project are offline land details verification, no provision for land splitting process. The Land Registration process can be further enhanced by automating the Land Verification Process and Land Updating Process.

The paper "Blockchain and Smart Contract for Land Registration using Ethereum Network" The system simply secures the documents stored on IPFS and doesn't provide any provision for communication between the land buyers and sellers. This can lead to issues like double selling. Without direct communication, there might be disputes about the price and location of the land.

As mentioned in this project "Land Registration using Blockchain Technology" could be developed further and used for other land related use case such as land registration, transfer of properties. The existing system just stores the land details on blockchain but doesn't support the purchase of land and the transfer of ownership. There is no proper verification and authentication for users and lands. The system has been implemented and tested only on an IDE and is not integrated with a frontend for the users to access it. The details of the land owner are not verified hence could be fraudulent. The system simply secures the documents stored on IPFS and doesn't provide any provision for communication between the land buyers and sellers. The system lacks the provision for land splitting, gifting or inheritance.

2.3 MOTIVATION OF PROPOSED SYSTEM

The system we proposed provides a user-friendly interface for direct communication between the buyers and sellers without any middle man. The system verifies the user's (buyer/seller) identity with Aadhar authentication and the land inspector verifies the identity document produced at the time of registration. Once the user is authenticated, the option to add lands is enabled. The land inspector verifies the land documents before approving the addition of land to the user's profile. Once the land is verified, it gets added to the land gallery of all the users. The user can make it available for sale or can buy an already available land. If the seller accepts the request, payment is made and transaction begins. The land inspector verifies the transaction and transfers the ownership of the land in the presence of a witness. A transfer of ownership document is generated which gets stored in IPFS.

3. REQUIREMENT SPECIFICATION

3.1 INTRODUCTION

The following lists the requirements that are to be imposed on the design and verification of the project. It contains functional and non-functional requirements of the Blockchain based land registry system with aadhar authentication. The software and hardware requirements for the completion of this project are listed as well.

3.2 SYSTEM ENVIRONMENT

The proposed system uses Ganache tool, which is part of the Truffle Suite ecosystem, to deploy a local Ethereum blockchain, where the transactions are stored. The system environment mentions the hardware requirements and software requirements which are listed below.

3.3 FUNCTIONAL REQUIREMENTS SPECIFICATION

3.3.1 Use Case Descriptions

A use case diagram, at its simplest, is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. Use cases are used during the analysis phase of a project to identify system functionality. They separate the system into actors and use cases. The 3 actors in the use case diagram, as shown in Fig 3.1, are:

- User – This actor uses the web application.
- Contract Owner – This actor handles the backend.
- Land Inspector – This actor is the one verifying the users, lands and transact

The use cases in the system are:

1. Use case: Register

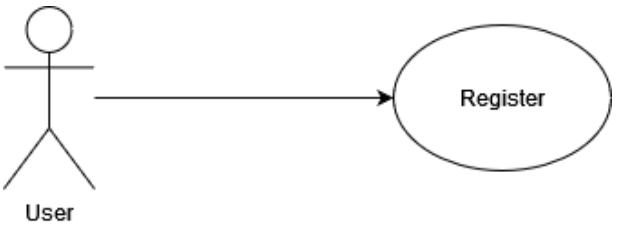

Fig: 3.1 Register Use Case

Definition: The user must register in order to login

Use-Case Name	Register
Trigger	Clicking on the “Login” option present in the home page
Pre-condition	When the user logs in for the first time
Basic Path	1. User enters details such as Name, Age, Address, Email ID, Aadhar Number, Pan Number, Aadhar document. 2. An image of the user will be captured 3. User clicks on the “Register” button
Post-condition	A record of the new user is saved

2. Use case: Login

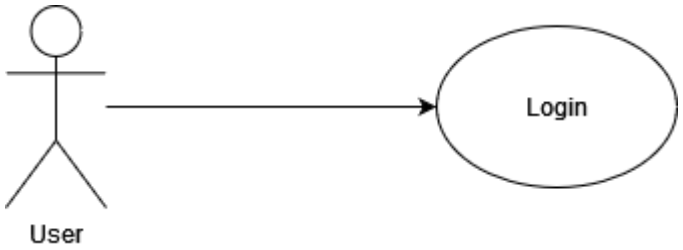

Fig: 3.2 Login Use Case

Definition: The user has to login in order to use the application

Use-Case Name	Login
Trigger	Clicking on the “Login” option present in the home page
Pre-condition	Only a registered user can login
Basic Path	1. User enters their unique private key or logins with metamask 2. The user can only proceed after Aadhar authentication
Post-condition	The user can explore the options

3. Use case: Verify user

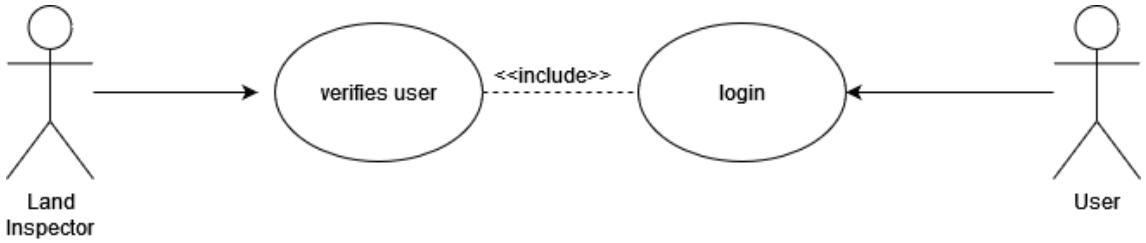

Fig: 3.3 Verify Use Case

Definition: The land inspector verifies the user's identity

Use-Case Name	Login
Trigger	Clicking on the "Verify user" option present on the land inspector dashboard
Pre-condition	A user should be registered in order to be verified
Basic Path	1. The land inspector logs in with the private key 2. Land Inspector now checks for the list of users yet to be verified
Post-condition	The user can add/buy/sell lands

4. Use case: Add land

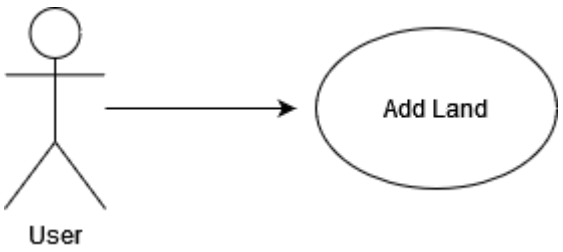

Fig: 3.4 Add land Use Case

Definition: The user can add lands to his profile

Use-Case Name	Login
Trigger	Clicking on the "Add Lands" option present on the user dashboard
Pre-condition	Only verified users can add lands

Basic Path	1. User selects the “Add Lands” option on the dashboard 2. User enters the land details and uploads the land document
Post-condition	The land will be added to the user’s profile

5. Use case: Verify land

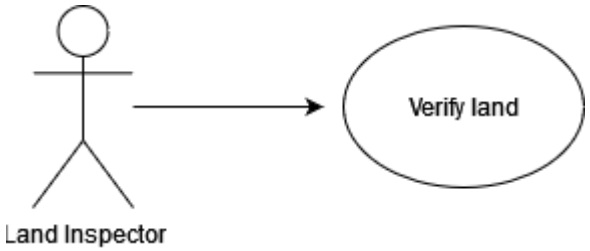

Fig: 3.5 Verify land Use Case

Definition: The Land Inspector verifies the authenticity of the land details

Use-Case Name	Login
Trigger	Clicking on the “Verify Land” option present on the Land Inspector dashboard
Pre-condition	A land should be added by the user
Basic Path	1. Land Inspector views the lands under the “verify lands” option on the dashboard 2. The land details and document are checked
Post-condition	The user can make the land available for sale

6. Use case: Buy land

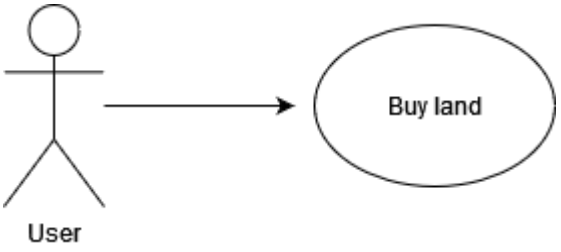

Fig: 3.6 Buy land Use Case

Definition: The user can send a request to buy a land which is available in the land gallery

Use-Case Name	Login

Trigger	Clicking on the “Send request to buy” option present in the land gallery
Pre-condition	The user should select a land from the land gallery
Basic Path	1. User views the land gallery and selects a land 2. User sends a request to buy the land
Post-condition	The request to buy the land is visible under “My sent land request”

7. Use case: Make payment

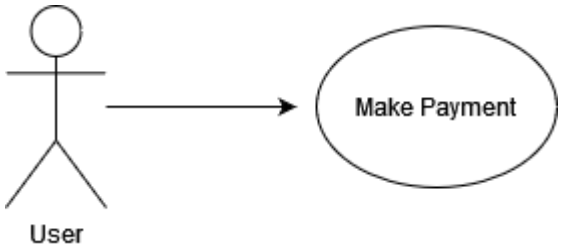

Fig: 3.7 Make payment Use Case

Definition: The buyer makes the payment of the land to the seller

Use-Case Name	Login
Trigger	Clicking on the “Make Payment” option visible under the sent land request dashboard
Pre-condition	The seller should approve the buyer’s request
Basic Path	1. User selects the “My Sent Land Request” option on his dashboard 2. The user selects the make payment option for that land 3. The amount is deducted from the metamask wallet
Post-condition	The Land Inspector can now proceed to transfer the ownership

8. Use case: Transfer ownership

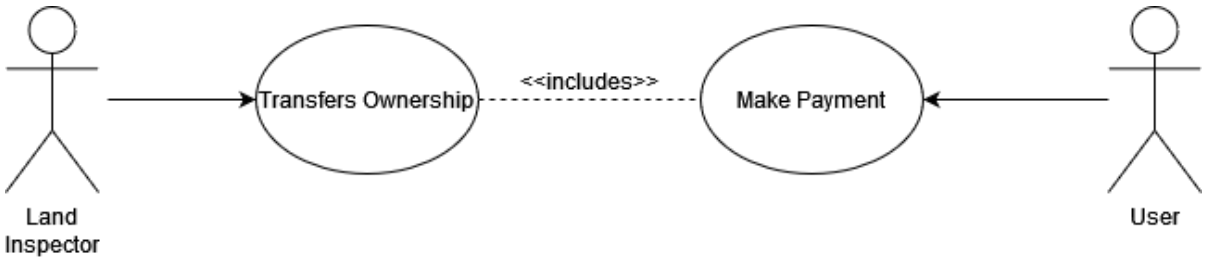

Fig: 3.8 Transfer ownership Use Case

Definition: The Land Inspector transfers the ownership of the land from the seller to the buyer

Use-Case Name	Login
Trigger	Clicking on the “Transfer” option under the Transfer Ownership dashboard
Pre-condition	The buyer should make the payment to the seller
Basic Path	1. Land Inspector views the lands under the “Transfer Ownership” option on the dashboard and clicks on transfer 2. The ownership is transferred in the presence of a witness
Post-condition	A digital document of the transfer of ownership is generated

3.3.2 List of Functional Requirements

In Software Engineering, a functional requirement defines a system or its component. It describes the functions a software must perform. The functional requirements are described in Table 3.1.

The functional requirements of the system are:

R1	The system must enable the contract owner to add the land inspector.
R2	The system must require a new user to register.
R3	The system must enable a registered user to login.
R4	The system must enable the land inspector to verify the user.
R5	The system must enable the user to add lands.
R6	The system must require the land inspector to verify lands.
R7	The system must enable the user to put land for sale.
R8	The system should enable the user to view land gallery.
R9	The system must enable the user to send a request to buy lands.
R10	The system must enable the user to accept the land purchase request.
R11	The system must enable the user to make payment.
R12	The system must enable the land inspector to verify transactions.
R13	The system must update the land gallery after transactions.
R14	The system must enable the land inspector to transfer ownership.

Table: 3.1 Functional Requirements of the system

3.4 NON-FUNCTIONAL REQUIREMENTS

NF_01	The system shall be highly secure
NF_02	The system shall be user-friendly
NF_03	The system shall be immutable and tamper-proof
NF_04	The system shall have high accessibility

NF_05	The system shall provide high privacy to the users
NF_06	The user shall be highly transparent
NF_07	The system shall not have unexpected downtime
NF_08	The system shall have high availability

Table: 3.2 Non-Functional Requirements of the system

3.5 SOFTWARE AND HARWARE REQUIREMENTS

The hardware and software components of a computer system that are required to install and use software efficiently. The software manufacturer will list the system requirements on the software package.

System requirements for operating systems will be hardware components, while other application software will list both hardware and operating system requirements. System requirements are most seen listed as minimum and recommended requirements. The minimum system requirements need to be met for the software to run at all on the system, and the recommended system requirements, if met, will offer better software usability.

Software Requirements:

The appropriation of requirements and implementation constraints gives the general overview of the project in regards to what the areas of strength and deficit are and how to tackle them. The tools used in implementation of this project are-

- Flutter
- Metamask
- Ganache
- Truffle framework
- Solidity
- Web3Dart

Hardware Requirements:

The following hardware is required to run this project at user end-

- 64-bit Windows OS
- 8 GB RAM
- 1 TB Hard Disk.

4. DESIGN SPECIFICATION

4.1 OVERALL USE CASE DIAGRAM

A use case diagram, at its simplest, is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. Use cases are used during the analysis phase of a project to identify system functionality. They separate the system into actors and use cases. The 3 actors in the use case diagram, as shown are:

- User – This actor uses the web application.
- Contract Owner – This actor deploys the contract and adds land inspectors.
- Land Inspector – This actor validates the users and lands.

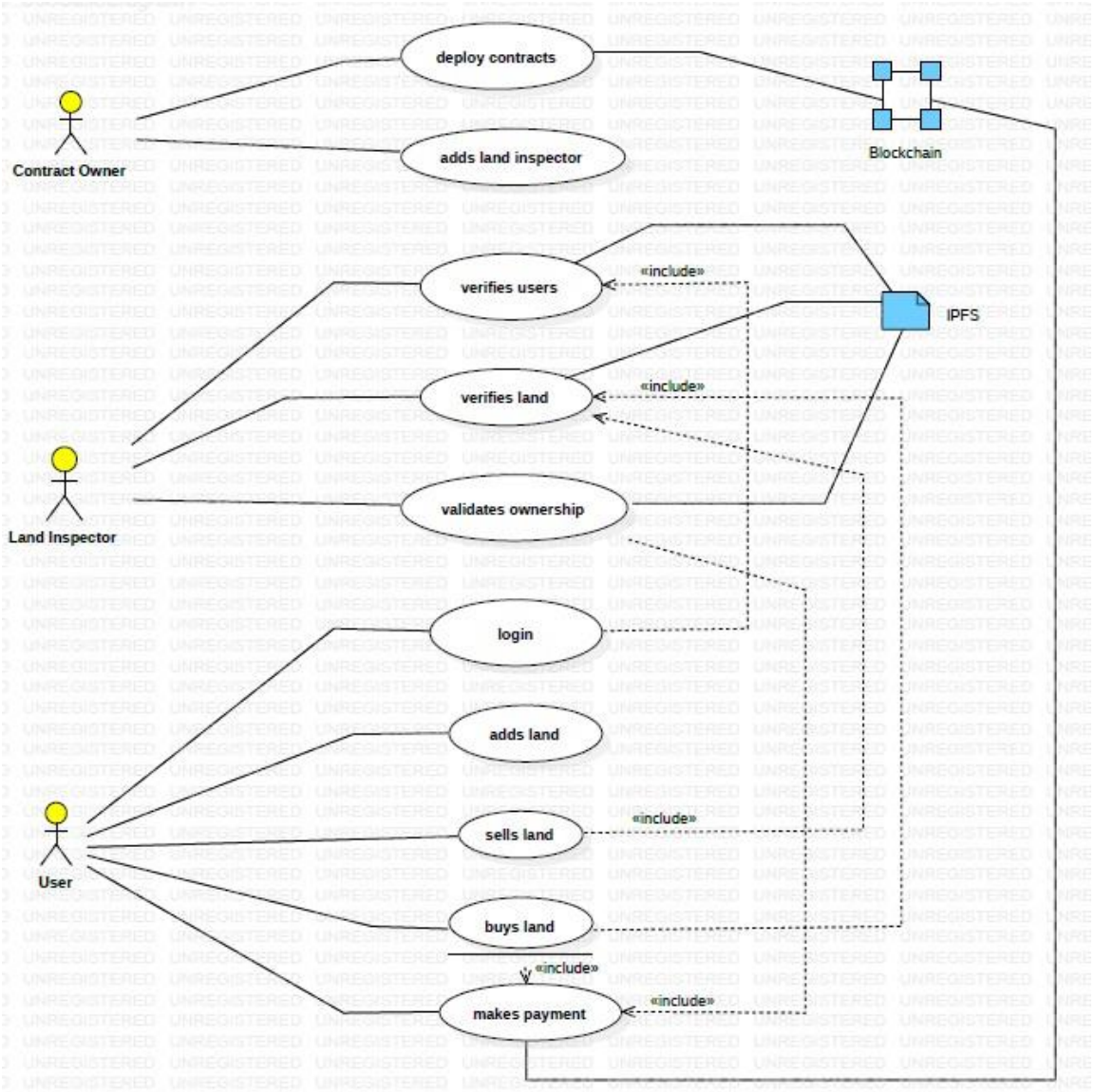

Fig: 4.1 Use case Diagram

4.2 CLASS DIAGRAM

A class diagram is a static structure diagram describing the structure of the system.This comprises the classes included in the project along with the attributes and functions present in each class.

The class diagram, as shown in Fig 3.2, has the following classes:

- User

  Attributes: Private_key, User_Name, User_Age, User_Address, Govt_ID_Proof, Email

  Methods: Upload_Docs(), Register(), Login()

- Login

  Attributes: Private key,Aadhar_Auth

  Methods: Authentication() , Login()

- Land

  Attributes: LID, Land_Area, Land_Price , Owner_Address , Survey_No , Land_Address , Land_Docs

  Methods: Add_Land() , Buy_land() , Sell_Land() , Locate_Land()

- Verification

  Attributes: Land_Inspector_ID , User_Name , User_Address , Govt_ID_Proof , LID , Land_Address , Land_Docs , Owner_Address

  Methods: Verify_User() , Verify_Land()

- Land Inspector

  Attributes: Land_Inspector_ID , Private_Key

  Methods: Login() , Verify_User() , Verify_Land() ,

Transfer_Ownership()

- Transaction

  Attributes: TID, Amount, Sender_ID, Receiver_ID, Transaction_Fee

  Methods: Payment()

- Ownership

  Attributes: Buyer_ID , Seller_ID , Land_Inspector_ID , LID , Land_Docs , Land_Address , Survey_No , Land_Price

  Methods:Transfer_Ownership() , Generate_Document()

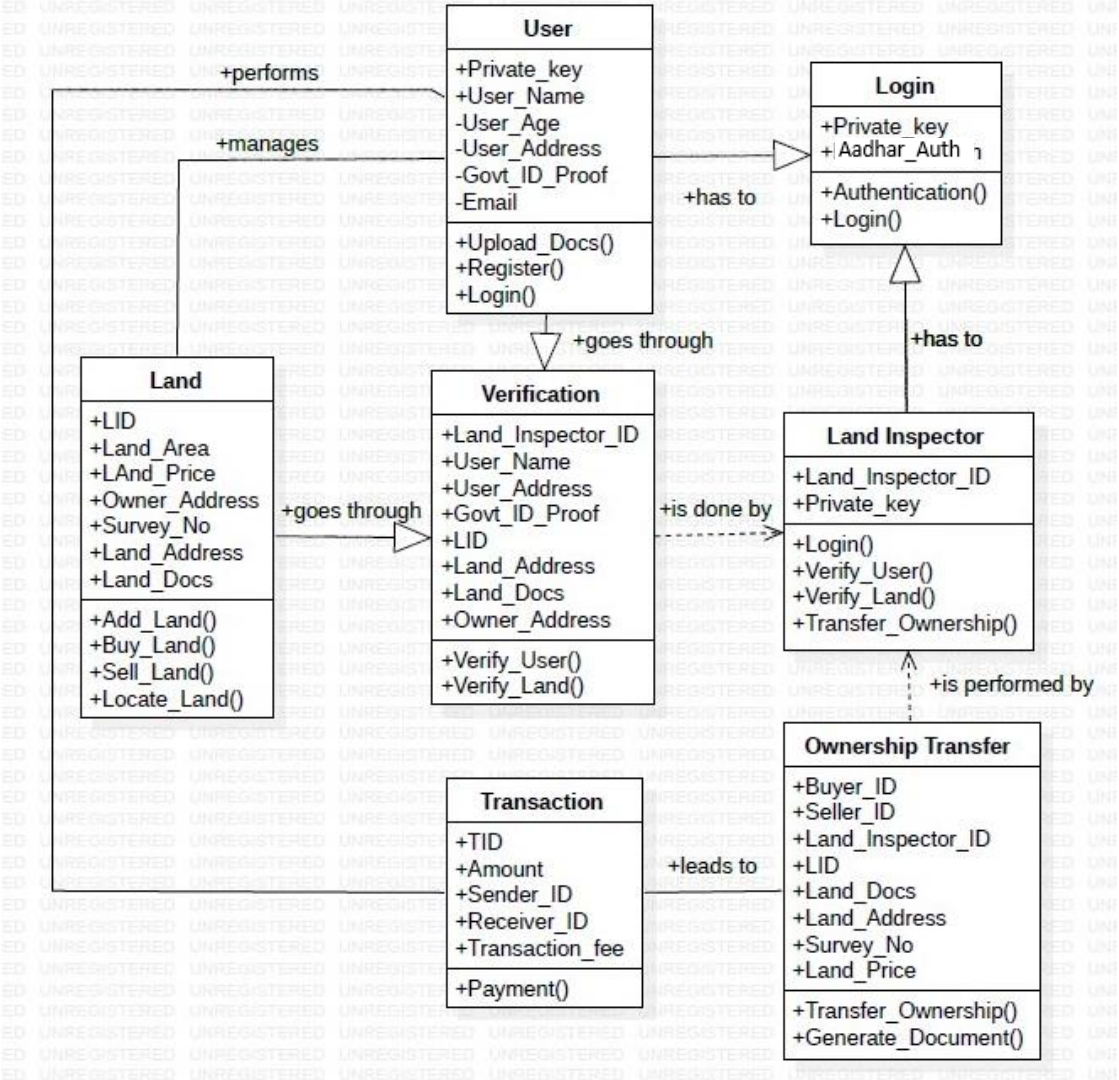

Fig: 4.2 Class Diagram

4.3 ACTIVITY DIAGRAM

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration, and concurrency. The activity diagram, as shown in Fig 3.4, shows the data flow of the application.

- The contract owner logs in to the system and adds the land inspector.
- The system verifies the user's (buyer/seller) identity with Aadhar authentication and the land inspector verifies the identity document produced at the time of registration.
- Once the user is authenticated, the option to add lands is enabled.
- The land inspector verifies the land documents before approving the addition of land to the user's profile.
- Once the land is verified, it gets added to the land gallery of all the users. The user can make it available for sale or can buy an already available land.
- If the seller accepts the request, payment is made and transaction begins.
- The land inspector verifies the transaction and transfers the ownership of the land in the presence of a witness.
- A transfer of ownership document is generated which gets stored in IPFS.

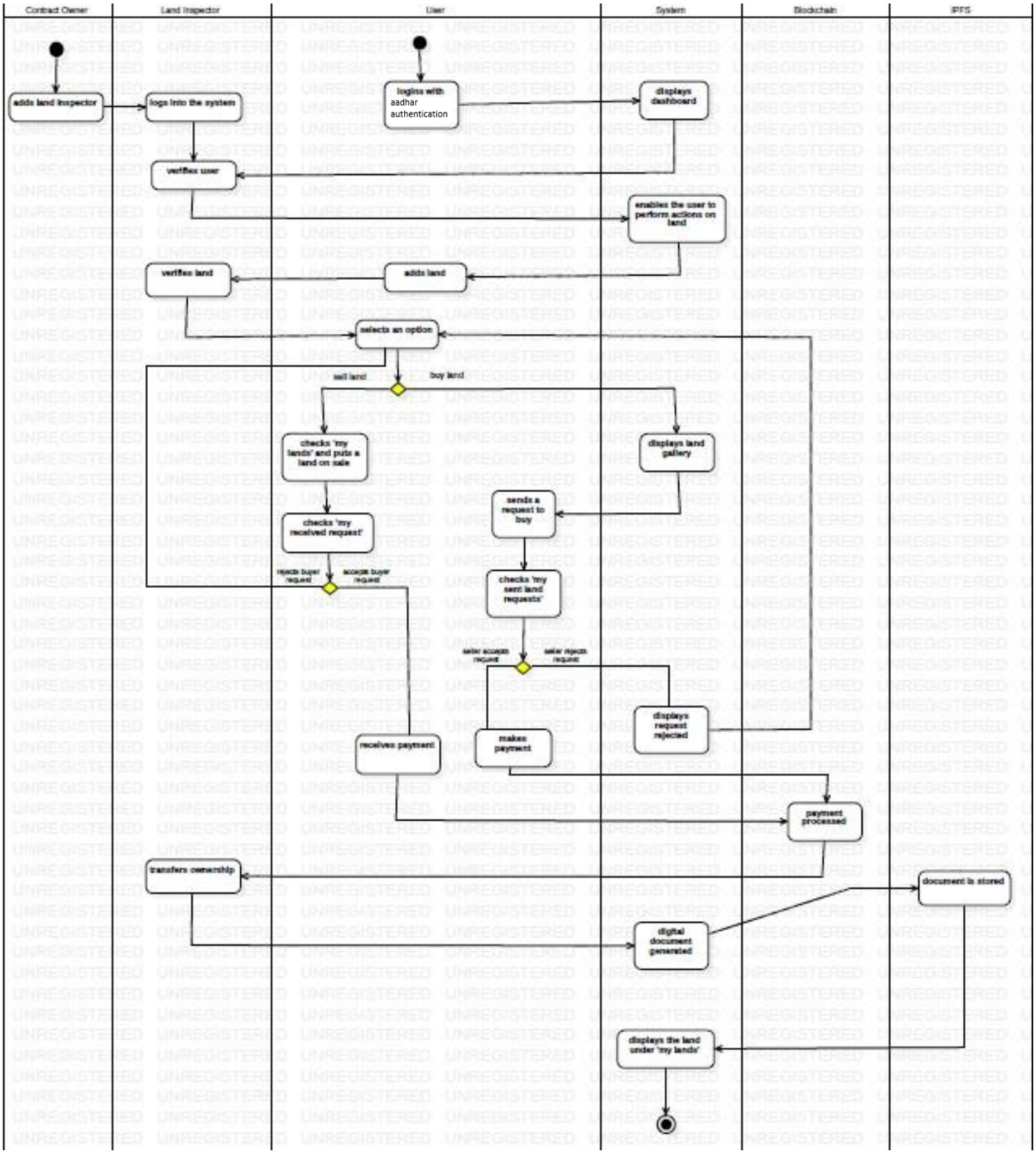

Fig: 4.3 Activity Diagram

4.4 SEQUENCE DIAGRAM

A sequence diagram is a UML diagram that illustrates the sequence of messages between objects in an interaction. A sequence diagram consists of a group of objects that are represented by lifelines, and the messages that they exchange over time during the interaction. The sequence diagrams, as shown in Fig 4.4 and Fig 4.5, show the sequence of messages between objects in an interaction of the application from buyer's and seller's perspective.

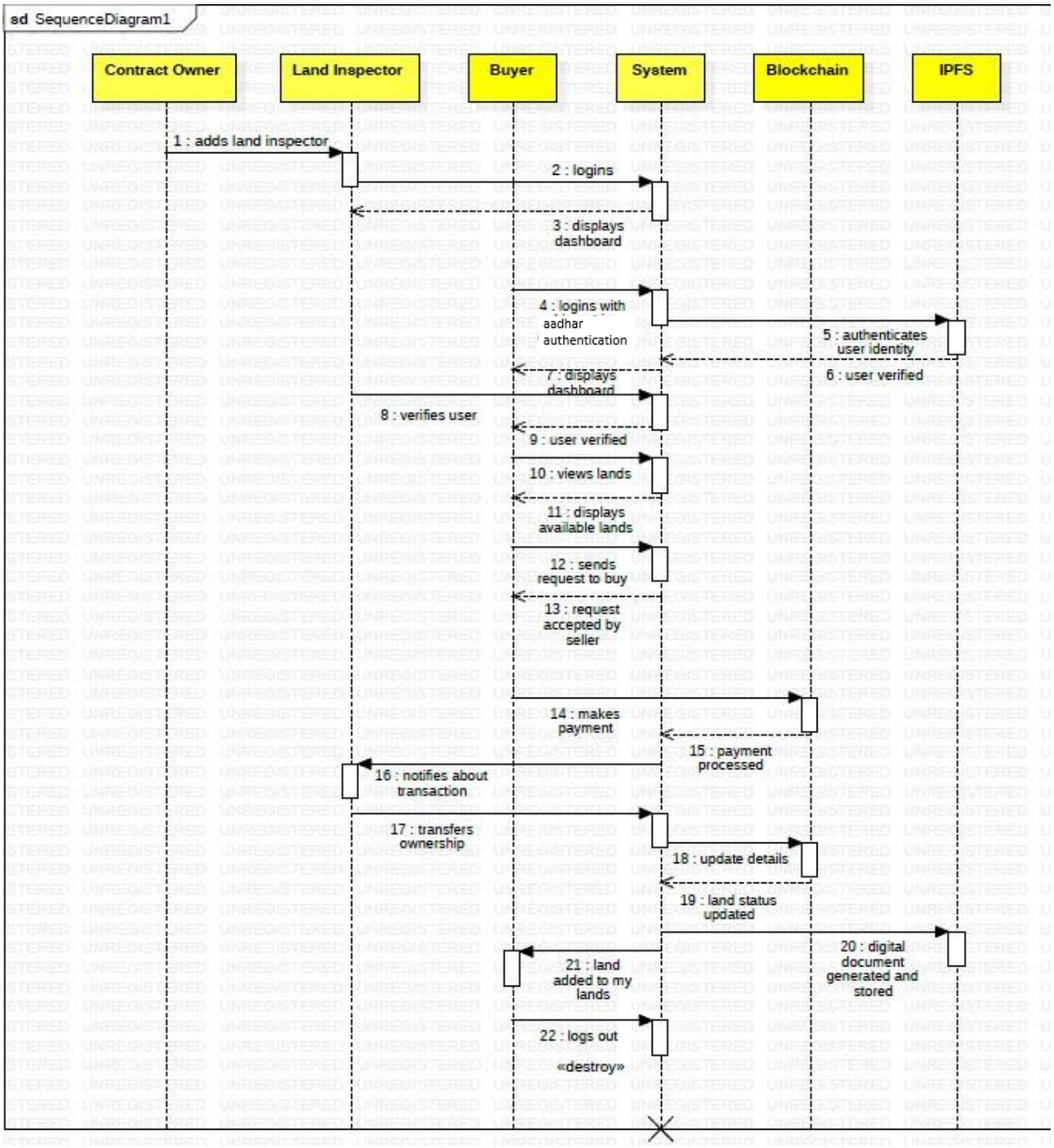

Fig: 4.4 Sequence Diagram for Buyer

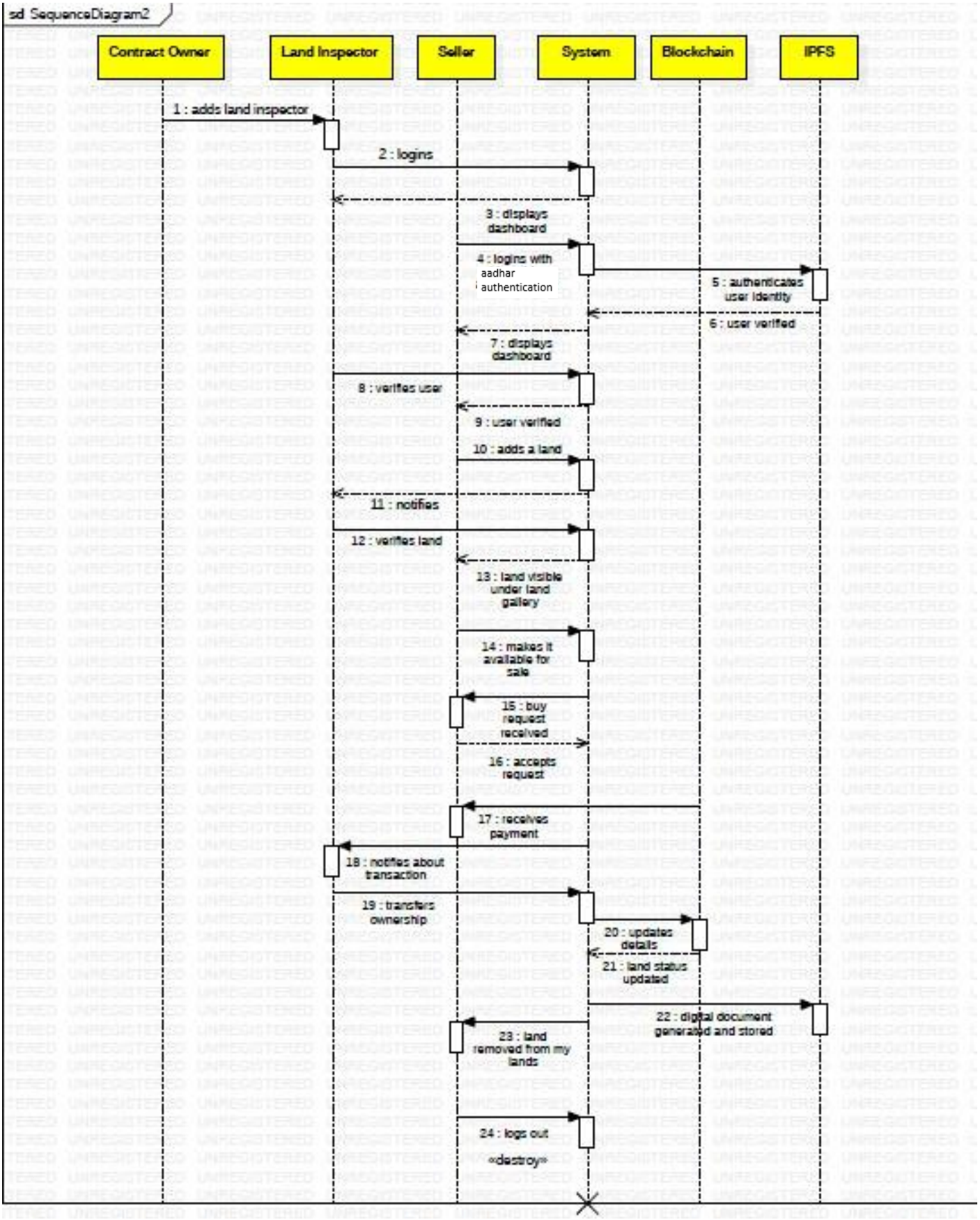

Fig: 4.5 Sequence Diagram for Seller

4.5 COLLABORATION DIAGRAM

A collaboration diagram, also known as a communication diagram, is an illustration of the relationships and interactions among software objects in the UML. It can be used to portray the dynamic behavior of a particular use case and define the role of each object.

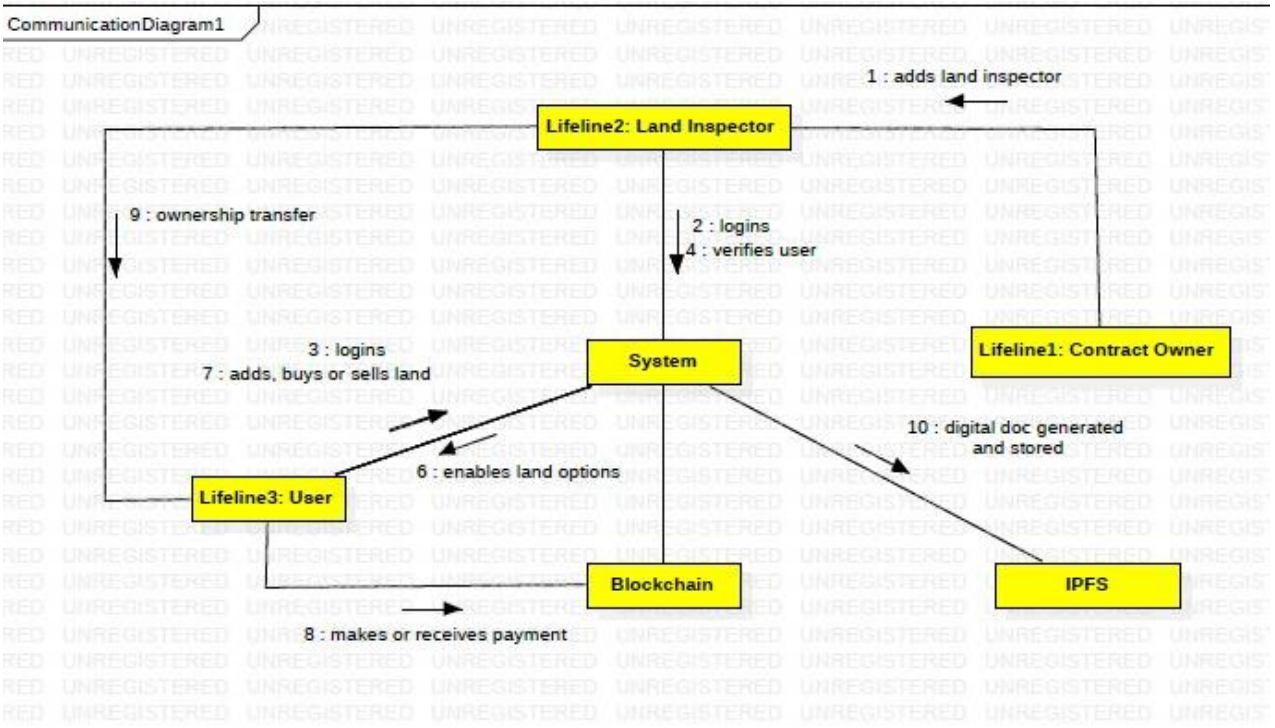

Fig: 4.6 Collaboration Diagram

5. IMPLEMENTATION

5.1 METHODOLOGY

TECHNOLOGY, TECHNIQUES AND METHODS

Ethereum is a decentralized blockchain-based software that has smart contract functionality. It is open source and used primarily to support the second-largest cryptocurrency in the world known as Ether. Ethereum enables the smart contracts and applications built on its blockchain to run smoothly without fraud, downtime, control, or any third-party interference.

Blockchain is decentralized because its public ledger is not stored in a single place. The public ledger is stored on thousands of volunteer's computers around the globe, each of which is called a node. The verification of the data stored on blockchain involves more than half of the nodes before being certified as correct. Cryptography is used to keep transactions on the blockchain network secure and to verify them also.
Computers are used to solve complex mathematical equations that help to confirm transactions on the network and input new blocks to the chain.

Ethereum is not controlled by any third party or entity. Instead, it is controlled by codes. Several pieces come together to ensure that Ethereum is functioning accordingly:

- Smart Contracts: The whole point of Ethereum having a system not controlled by a third party but by codes is induced by smart contracts. Smart contracts are automatically executed when certain stated conditions are met without the help of any external body. Smart contracts are involved in any cryptocurrency. They are not restricted to and can be used outside Ethereum, but they are popularly known for their Ethereum usage.

- Ethereum Blockchain: This is where the history of all the smart contracts executed are stored. Hundreds of nodes from all over the world store a copy of the entire blockchain. Thousands of computers process a smart contract whenever it is executed to ensure that all the stated rules were adhered to. The nodes store transaction details, accounts, smart

contract code, and smart contract state. All the nodes follow the same rule set for verifying a transaction, and they are all connected.

- Ethereum Virtual Machine (EVM): The Ethereum virtual machine executes the smart contracts. It helps translate the smart contract written in a language the computer that can't read into a language (bytecode) that they can read. The EVM can execute at least 140 different codes with specific tasks.

- Ether: Ether is Ethereum's native cryptocurrency. Ether is stored in accounts, and there are two types of accounts. Externally owned accounts are used to hold and send Ether by users, and Contract accounts are the accounts that hold smart contracts.

- Proof-of-Work: When a block of a transaction is created, miners, to get the correct value of the block, generate values until they get it. A hash value is then sent across the network for the nodes to verify when the miner finds it. If it is validated, the miner receives the Ether it unlocked when it discovered the hash. There is, however, a plan to switch to a new algorithm called proof-of-stake, which is tipped to consume less computing power and electricity than proof-of-work.

Ether

Ether is a cryptocurrency that can be used in financial transactions as a digital currency. Ether also serves as a medium through which users can carry out any task on Ethereum. To store Ether, a user requires an Ethereum wallet. Most of these wallets are digital and can be accessed via a laptop or smartphone. The Ethereum wallet stores the private key of the user. If a user loses their private key, they have lost their Ether, and there is nothing that can help them recover their private key.

Truffle

Truffle is a development environment utilizing the EVM (Ethereum Virtual Machine) as a basis. This environment specializes in smart contract development and features a number of great functionalities that help DApp developers tremendously, such as:

- Smart Contract Management: Truffle helps manage the artifacts of all smart contracts utilized in your dApps. As Truffle takes care of this, we can focus on other parts of the development process. This further means that Truffle supports library linking, custom deployments, and more complex Ethereum dApps.
- Automated Contract Testing: The main benefit of automated contract testing is that we can shorten the development process of our smart contracts.
- Scriptable Migration and Deployment: With Truffle, we can write deployment scripts that allow us to account for the fact that our dApps will change over time. This means that we can maintain our smart contracts far into the future and in the long term.
- Network Management: Truffle helps us take care of your network by managing our artifacts, allowing us to focus our attention on other tasks.
- Interactive Console: Truffle features an interactive console that allows us to access all Truffle commands and contracts that we have built.

Ganache

Ganache is a tool used to set up our own local Ethereum blockchain that we can use to deploy and test our smart contracts/dApps before launching them onto an authentic chain. There are two reasons why we should utilize a local blockchain when developing our applications. The first reason is that we can save money, and the second one is that we can save time.

Uploading contracts to an authentic chain such as the Ethereum main-net costs money in the form of gas fees. This can be a problem since the fees can be extraordinarily high and unpredictable. As a result, it can become very costly to upload contracts that are still not working correctly. Furthermore, making transactions on the authentic chains takes time, which we want to avoid when developing dApps. So, this can be solved by spinning up a local blockchain with Ganache as we can deploy smart contracts instantly. This means that Ganache can help us save both time and money, something that is highly appreciated when developing a business.

Ganache comes in two flavors: CLI and UI. It sets up 10 default Ethereum addresses, complete with private keys and all, and pre-loads them with 100 simulated Ether each. local blockchain with Ganache as we can deploy smart contracts instantly. This means that Ganache can help us save both time and money, something that is highly appreciated when developing a business.

Ganache comes in two flavors: CLI and UI. It sets up 10 default Ethereum addresses, complete with private keys and all, and pre-loads them with 100 simulated Ether each.

There is no "mining" per-se with Ganache - instead, it immediately confirms any transaction coming its way. This makes iterative development possible - you can write unit tests for your code which execute on this simulated blockchain, deploy smart contracts, play around, call functions, and then tear it all down for further simulation or new tests, returning all addresses to their initial state of 100 Ether.

When we start Ganache, it will default to a certain port and IP address. Usually that's localhost:7545 or 127.0.0.1:7545, while the CLI version tends to go with port 8545.

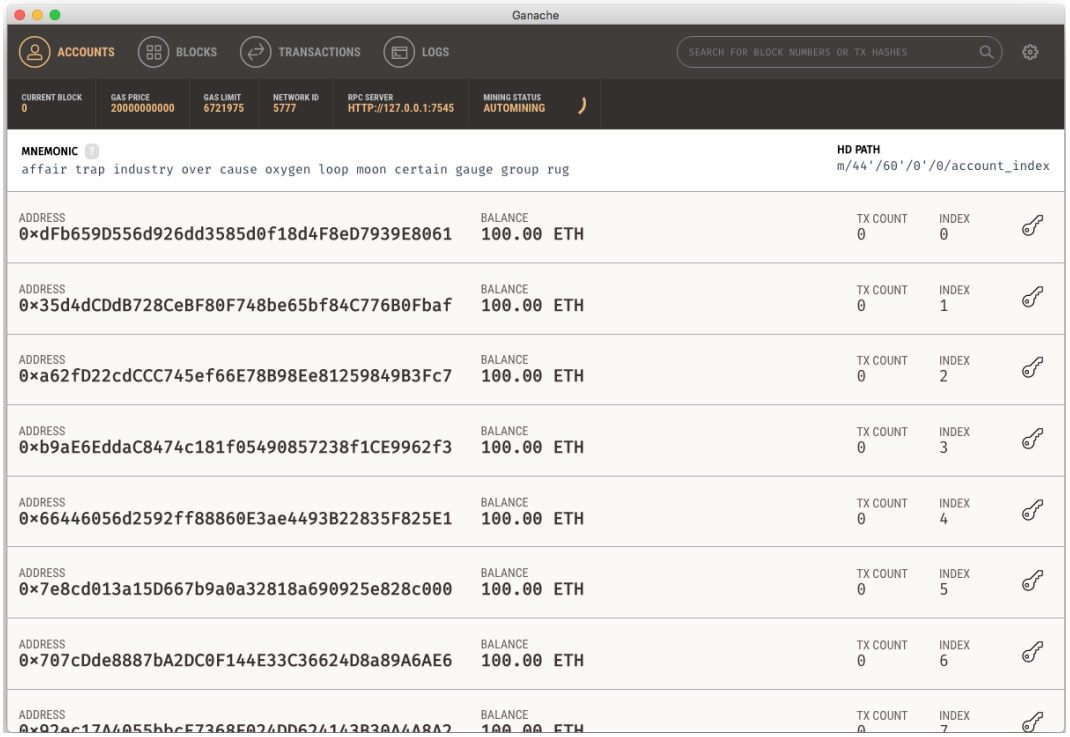

Fig: 5.1 Ganache interface

Metamask

MetaMask provides the simplest yet most secure way to connect to blockchain-based applications. Metamask is a browser plugin that serves as an Ethereum wallet, and is installed like any other browser plugin. Once it's installed, it allows users to store Ether and other ERC-20 tokens, enabling them to transact with any Ethereum address. Metamask is available as a browser extension and as a mobile app, and equips users with a key vault, secure login, token wallet, and token exchange.

When we first install MetaMask, we will need to accept some terms and conditions, then input a password. Next, we will be given some seed words, which should be stored in a safe place. These words are derived from a private key and their combination always produces that key when pulled through a specific algorithm. The only way to access a wallet directly without a password is by accessing its private key.

Once we have gone through this process, MetaMask will open our first auto-generated wallet, called Account 1. To use our Ganache wallets instead of the auto-generated one, we need to link our Ganache network to Metamask by adding a New RPC URL input of http://127.0.0.1:7545. Then we must import a Ganache account into Metamask.

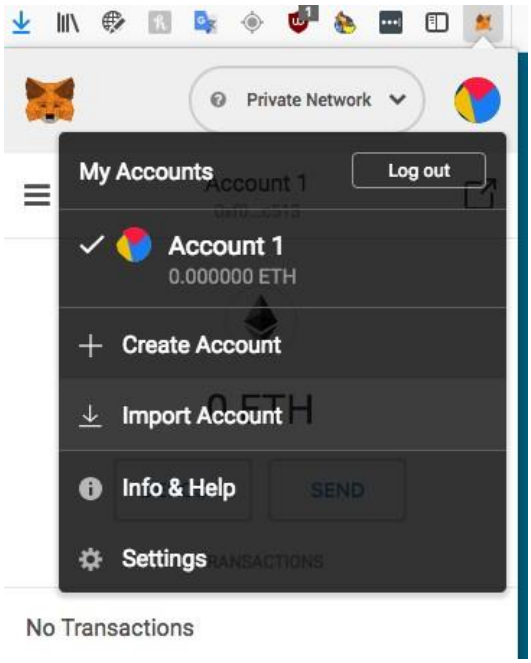

Fig: 5.2 Metamask interface

In Ganache, we must select the private key of the first wallet in the list by clicking the key icon to the right of the wallet. We must then copy this private key and paste it into MetaMask's import.

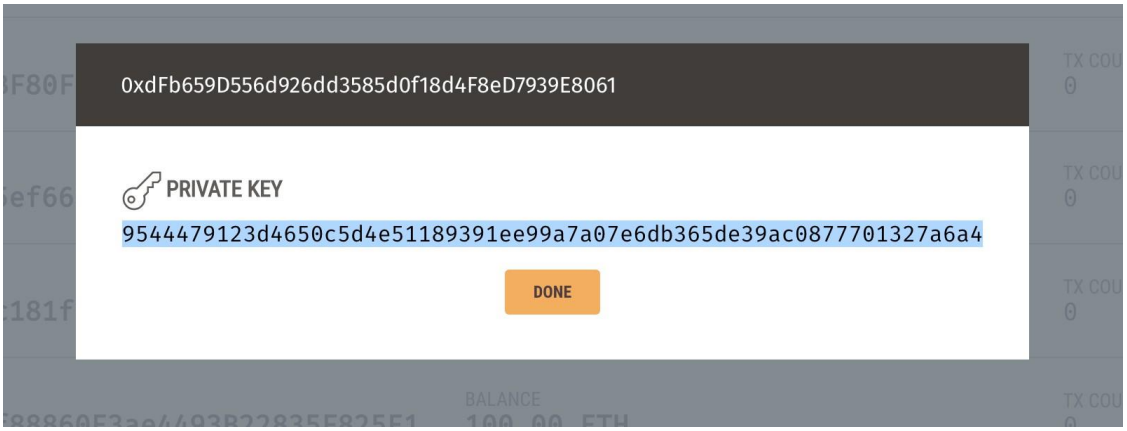

Fig: 5.3 Private Key for a user account

Now we will have 100 Ether ready for sending on our private Ganache blockchain.

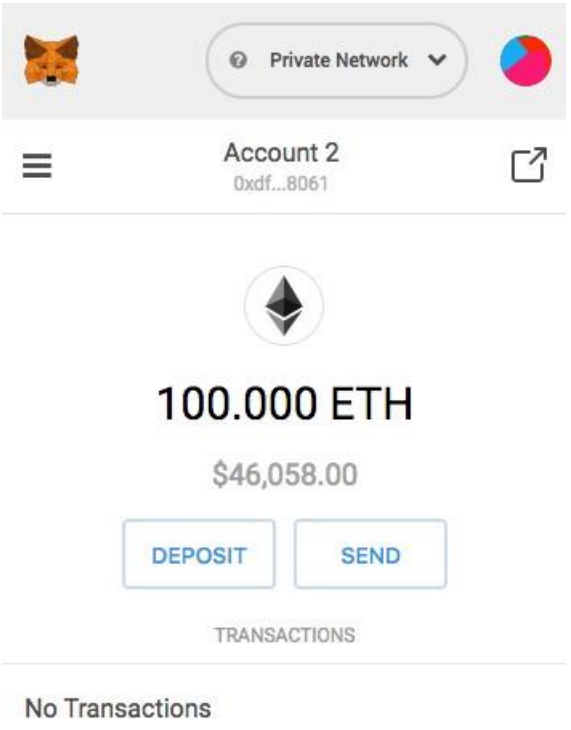

Fig: 5.4 Amount associated with an account

Web3.js

Web3.js enables us to fulfill the responsibility of developing clients that interact with the Ethereum blockchain. It is a collection of libraries that allows us to perform actions like send Ether from one account to another, read and write data from smart contracts, create smart contracts, etc. It allows us to interact with a local or remote Ethereum node using HTTP, IPC or WebSocket.

Django

In this project, we have built our website and user authentication components using Django. Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. It is a powerful structure written with and using all thefunctionality of the Python programming language to create commercial and home websites of varying complexity. Django is an open-source project that supports the implementation of the most popular packages and Python tools.

We can create a full-fledged end-user web sites from scratch using Django with extensive administration and visualization. Many of the most modern and popular frameworks and libraries can be implemented using Djangosuch asBootstrap, Angular, Vuejs, Backbone or others. Django has a default databaseof SQLite (lightweight DB), and has a graphical administrator interface by default, which makes it a convenient tool for creating and managing websites with minimal programming experience.

Django apps are structured so that there is a separation of logic. It supports the Model- View-Controller Pattern, which is the architecture on which most web frameworks are built. The basic principle is that in each application there are three separate files that handle the three main pieces of logic separately:

- Model defines the data structure. This is usually a database and is the base layerto an application.
- View displays some or all the data to the user with HTML and CSS.
- Controller handles how the database and the view interact.

Fig 5.5 Flowchart

5.2 SYSTEM ARCHITECTURE

The system architecture of the project is shown below, the three stakeholders namely, contract owner, land inspector and the user interact with the decentralized web application. It is built using flutter and solidity. Web3.js is used as API support for communication between the DAPP and blockchain. An Ethereum wallet is required for performing all the blockchain transactions. The documents are stored on IPFS which is a decentralized file system.

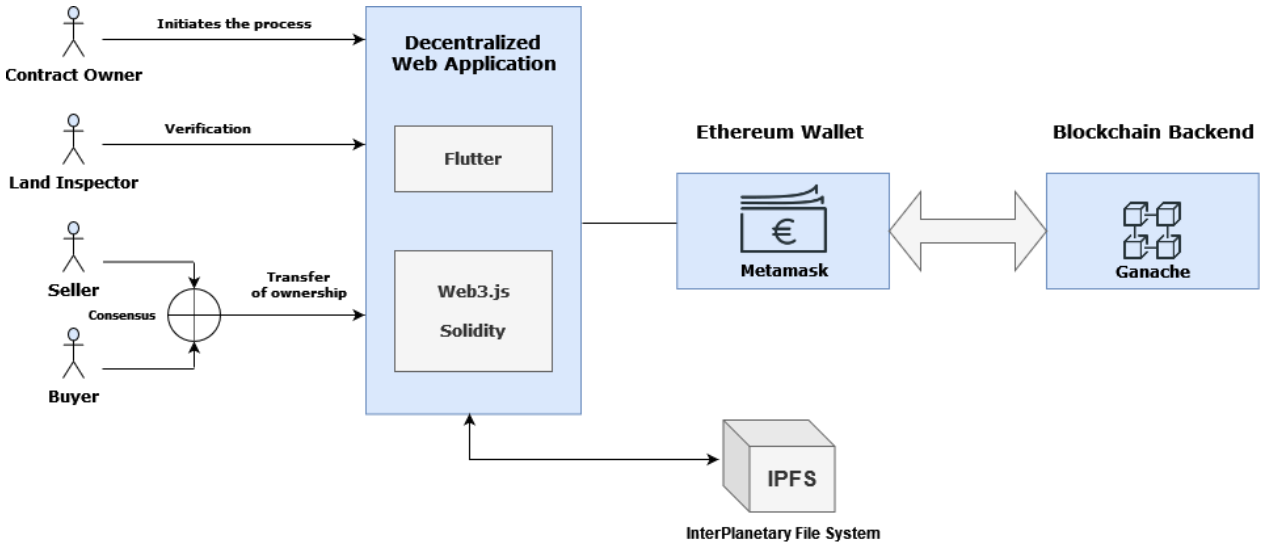

Fig: 5.6 System Architecture

5.3 MODULE DESCRIPTION

To implement this project, the following modules have been used -

- Authentication Module – In this module, the verification of the user and the lands is done. The user first registers with the help of his private key and biometrics. After the user has registered, the Land Inspector verifies his identity documents and authenticates him. If the user adds a land to his profile, the Land Inspector should verify the land documents in order to enable the user to take further actions.
- User Processes Module – In this module, the user can add lands to his dashboard, make lands available for sale or buy the available lands from the land gallery. Each of these actions are followed by verification in every step. Finally, the transfer of ownership takes place and a digital document is generated and stored.
- Transactions Module – Metamask is used for making transactions in our system. Right from the buyer's payment to recording the transfer of ownership, all the transactions are tamper-proof and unchangeable.

6. TESTING

6.1 TEST PLAN

Truffle uses the Mocha testing framework and Chai for assertions to provide us with a solid framework from which to write our JavaScript tests.

We verified the functionality of blockchain application by writing tests to check if:

1. The contract initializes with the right public key
2. The land inspector initializes with the right name, designation, age, and city
3. The user enters the right details including name, age, email, aadhar, PAN number and the identity document in the correct format
4. The land inspector can verify the users
5. The land inspector can verify the land
6. Only a verified user can add lands
7. Only verified users can send a request to buy a land
8. The land inspector can transfer the ownership of a land

To test user authentication and verification using Aadhar OTP, we processed each flow of the website and debugged it to ensure that it is working correctly, including test cases having incorrect input data.

6.2 TEST CASES

Test Case ID	**Test Case Name / Objective**	**Prerequisites/ pre-condition**	**Stepwise Description of Testing Process**	**Expected Result**	**Actual Result**	**Pass/Fail / Not executed/ suspended**	**Action/ Notes**
6.1	Contract Owner login	Should be assigned with the correct address	Incorrect address entered in the login page of contract owner	“You are not authorized” pop-up should be displayed	“You are not authorized” pop-up is displayed	Pass	Contract owner should login with the correct address

6.2	User login	Should enter the registered private key	User enters unregistered private key	User should be directed to registration page	User is directed to registration page	Pass	User should login with the registered private key
6.3	User should add lands	User should be verified by the land inspector	Unverified user trying to add lands	The option to add lands should be disabled	The option to add land is disabled	Pass	The land inspector should verify the user
6.4	User can put a land for sale	Land should be verified by the land inspector	User trying to make an unverified land for sale	The option to make a land for sale should be disabled	The option to make a land for sale is disabled	Pass	The land inspector should verify the land
6.5	Buyer can make the payment for land	The request sent to purchase land should be accepted by the seller	Buyer trying to make the payment for a request not accepted by the seller	The option to make payment should be disabled	The option to make payment is disabled	Pass	The seller should accept the request
6.7	User login	Should enter the correct private key	User enters incorrect private key	Incorrect Private Key pop-up should be displayed	Incorrect Private Key pop-up is displayed	Pass	User should login with the correct private key
6.8	User login	User enters the OTP	User enters incorrect OTP	“Incorrect OTP Please try again” pop-up should be displayed	“Incorrect OTP Please try again” pop-up is displayed	Pass	User should enter the correct OTP
6.9	User login	User enters the OTP	User enters correct OTP	User should be directed to User Dashboard	User is directed to User Dashboard	Pass	User should enter the correct OTP

Table: 6.1 Test Cases

7. RESULTS AND CONCLUSION

7.1 RESULT ANALYSIS

Our project has successfully accomplished the set goals of creating a Decentralized application (DAPP) which is a one stop platform for buying, selling, and registering land securely and conveniently. We were able to receive OTP to Aadhar linked number and use the private key to authenticate a user. We were able to eliminate the middle men involved in the process of land transfer.

Our objective of implementing this application using blockchain technology, was accelerated by maintaining an immutable and tamper proof transactions between the users. This resulted in transparency throughout the process of land transfer. The users were able to conveniently navigate through the application and find details specific to lands. A land document was produced at the end of the transfer of ownership which served as a tamper-proof document for the land transfer.

7.2 CONCLUSION & FUTURE SCOPE

The proposed system aims at providing a single platform to the users to buy/sell/register a land. By providing the exact details of the user, identity documents and land documents which are verified by a Land Inspector ensures the credibility of the data entered into our application. On the contrary, if these were provided in normal paper-based registration system, the data could be altered easily. With the use of blockchain and by storing the data on a decentralized file system, we have ensured that no fraud takes place and the data remains unaltered.

As we know that there's always a scope for improvement, there are certain aspects that could be added to our application to increase its overall efficacy. The process can be further improved by automating the user and land verification process. We can also predict the approximate price of land and suggest the users about the current land price trends. We can also add a messaging facility so that the users can communicate in the application directly.

8. APPENDIX

I. SCREENSHOTS

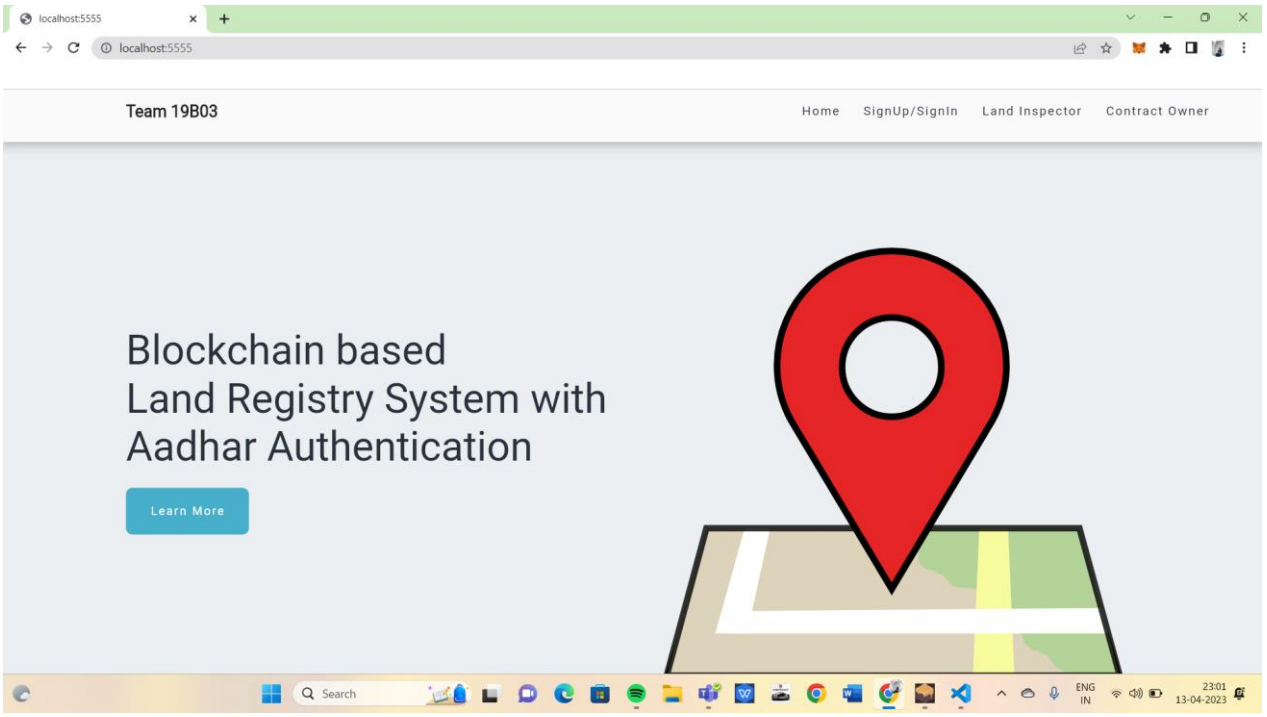

Fig: 8.1 Home Page

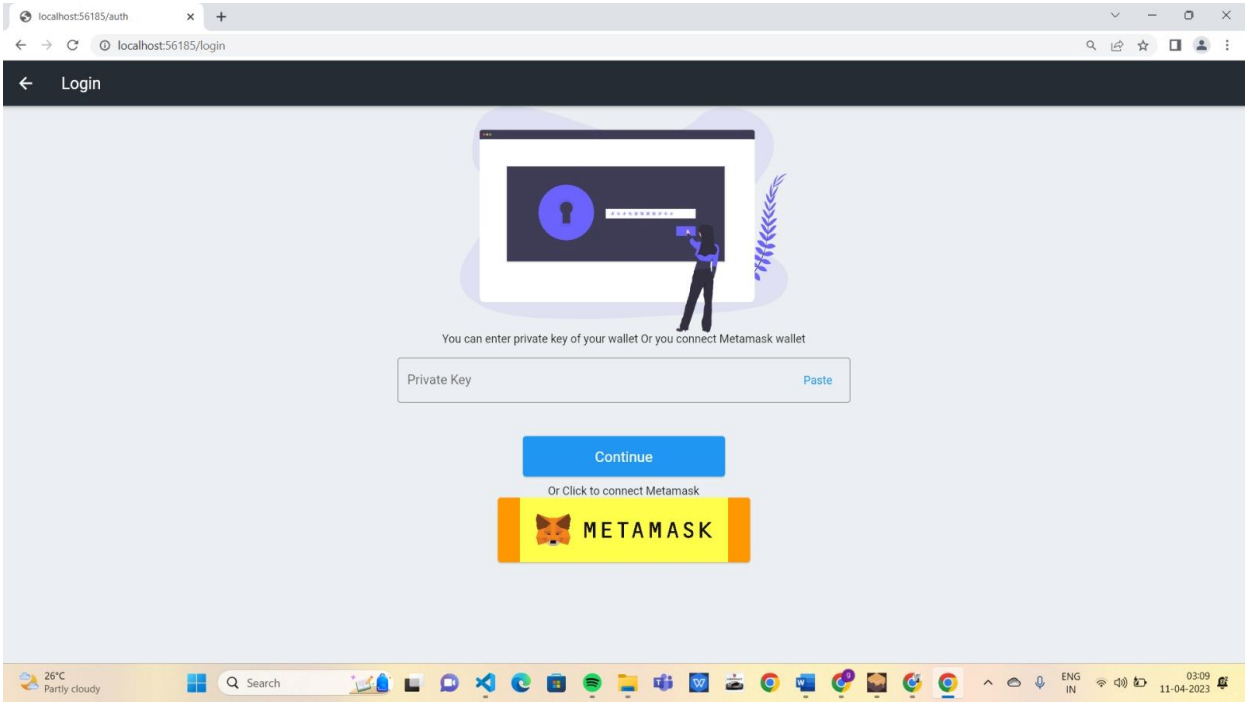

Fig: 8.2 Login Page

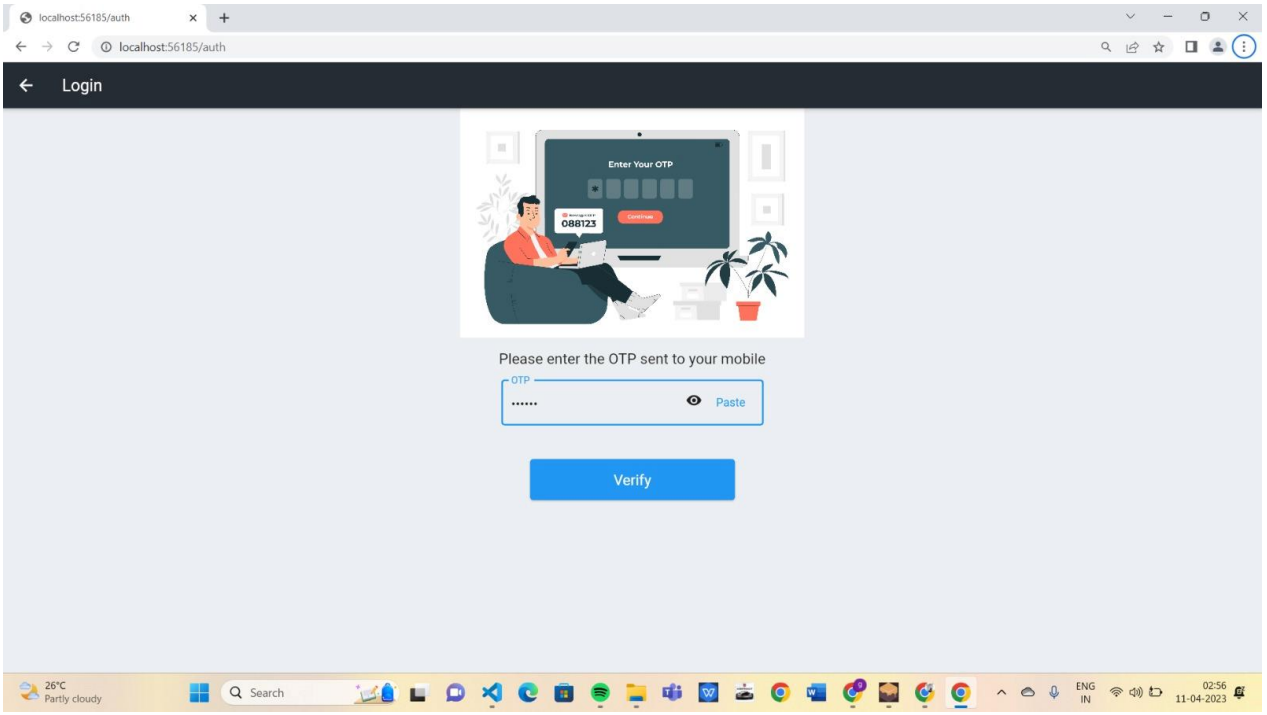

Fig: 8.3 OTP Page

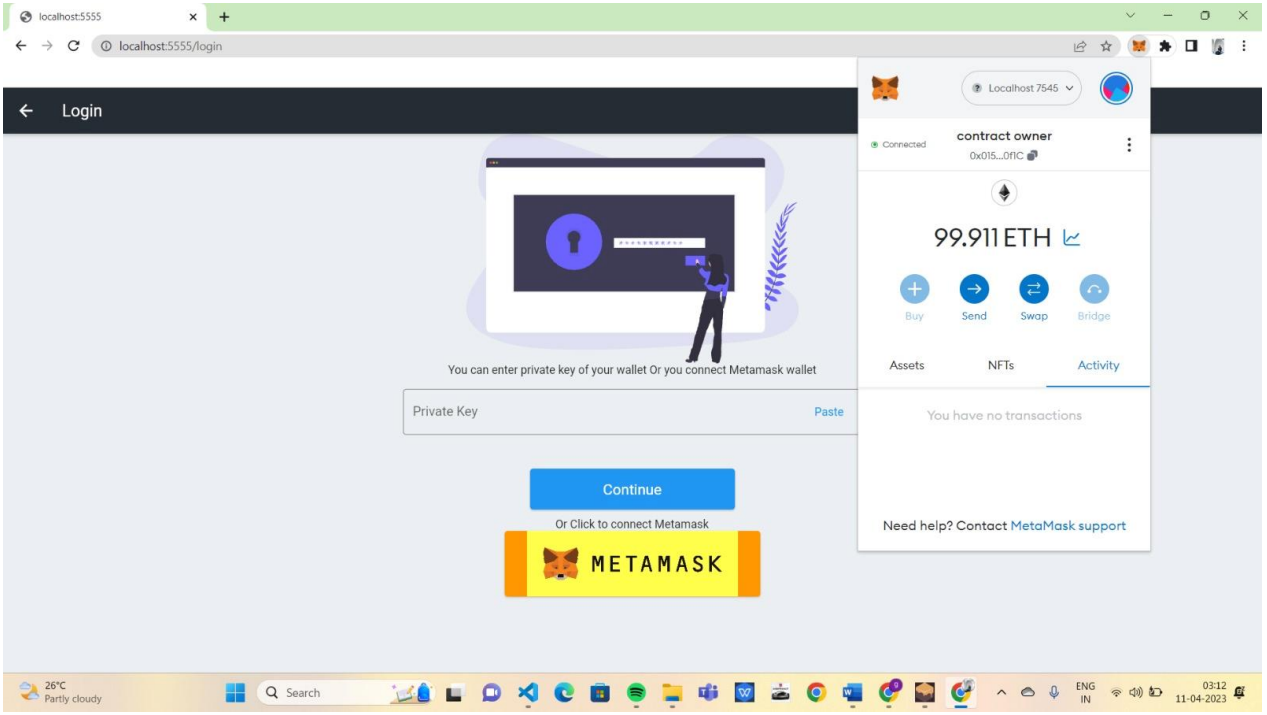

Fig: 8.4 Contract Owner Login Page

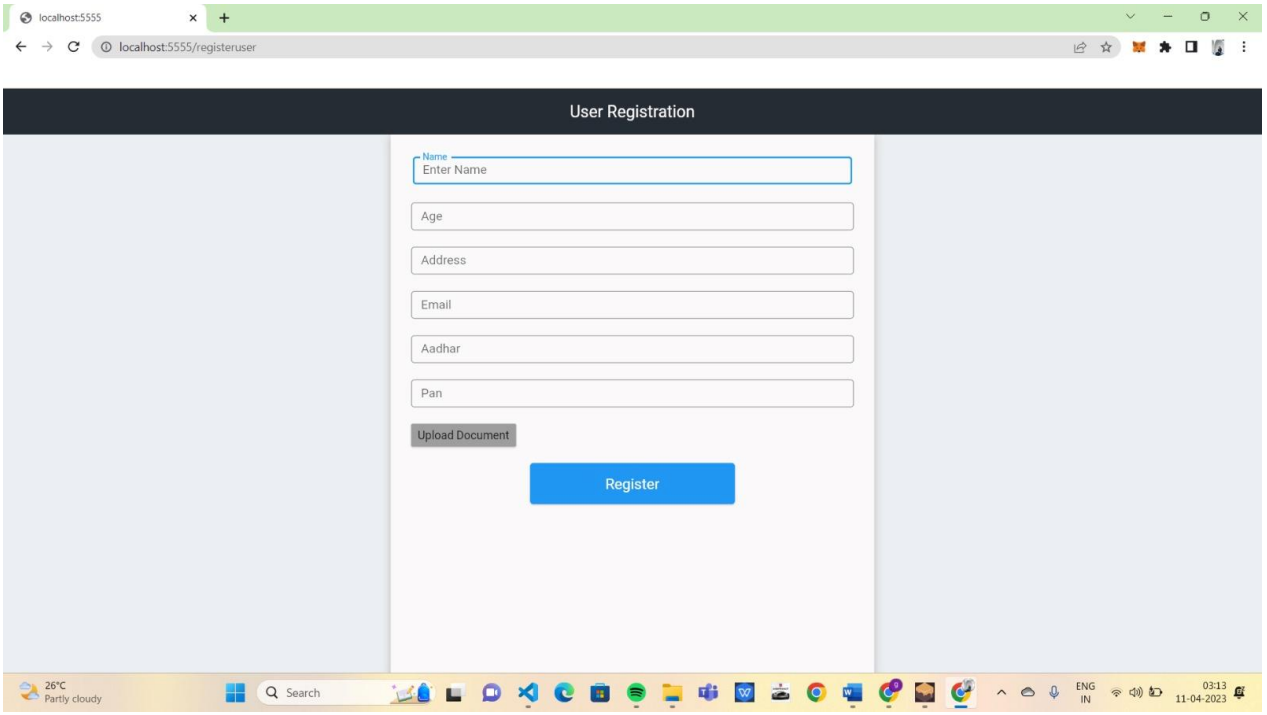

Fig: 8.5 User Registration Page

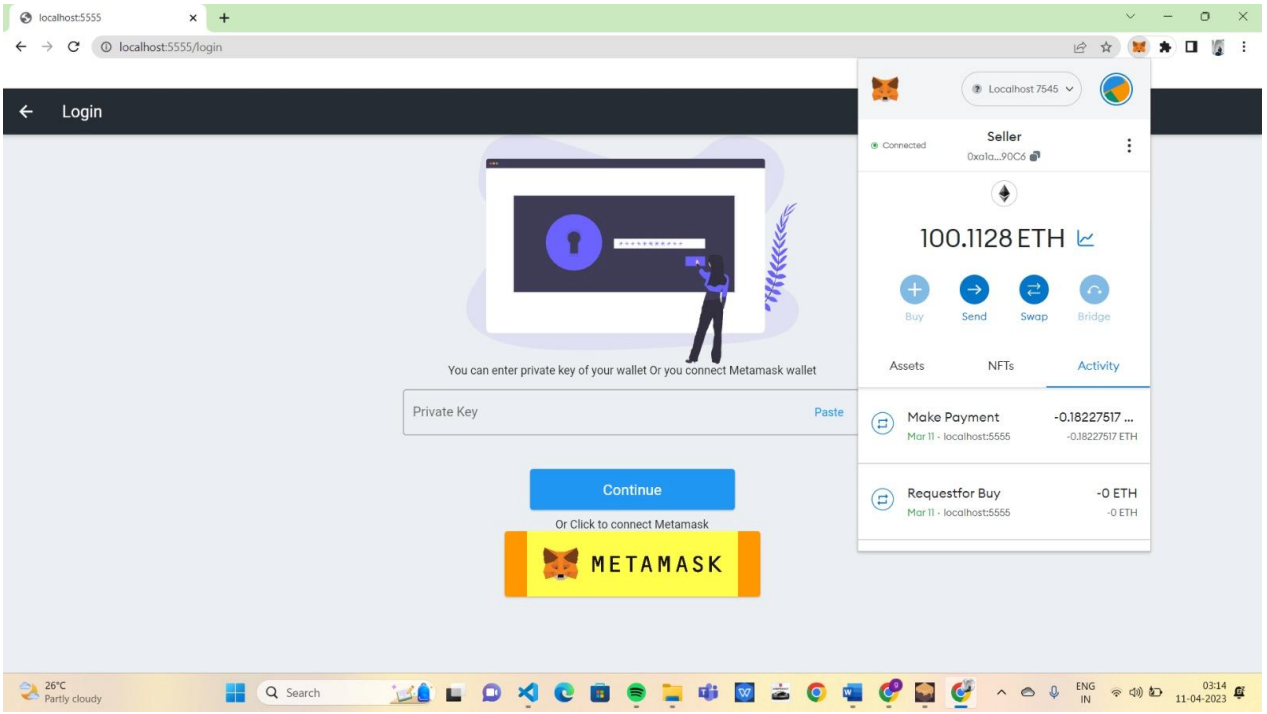

Fig: 8.6 Seller Login Page

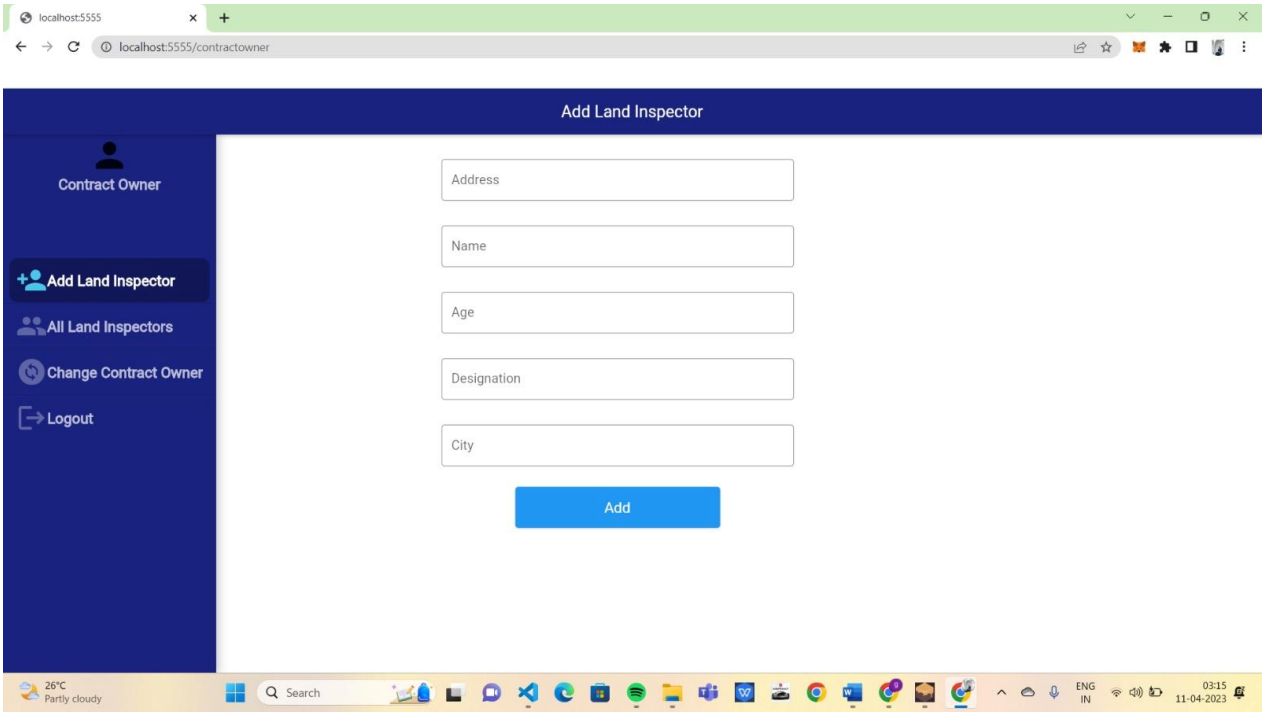

Fig: 8.7 Add Land inspector Page

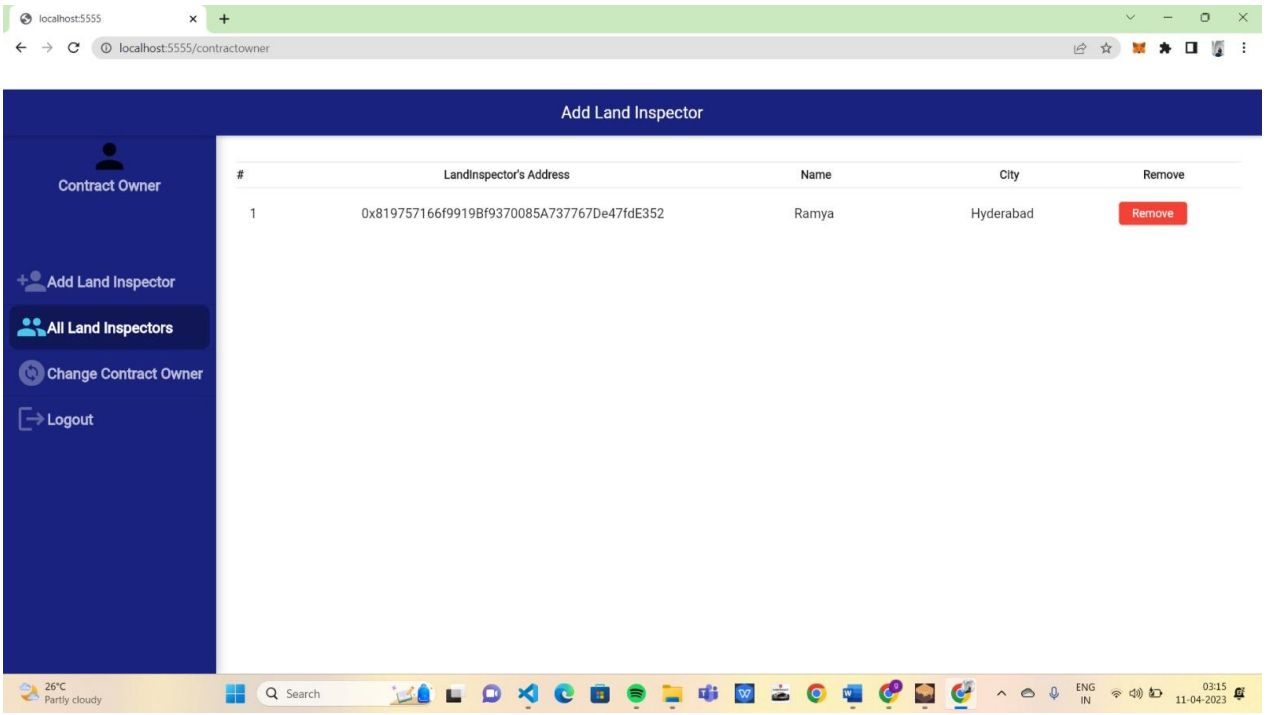

Fig: 8.8 View Land Inspectors Page

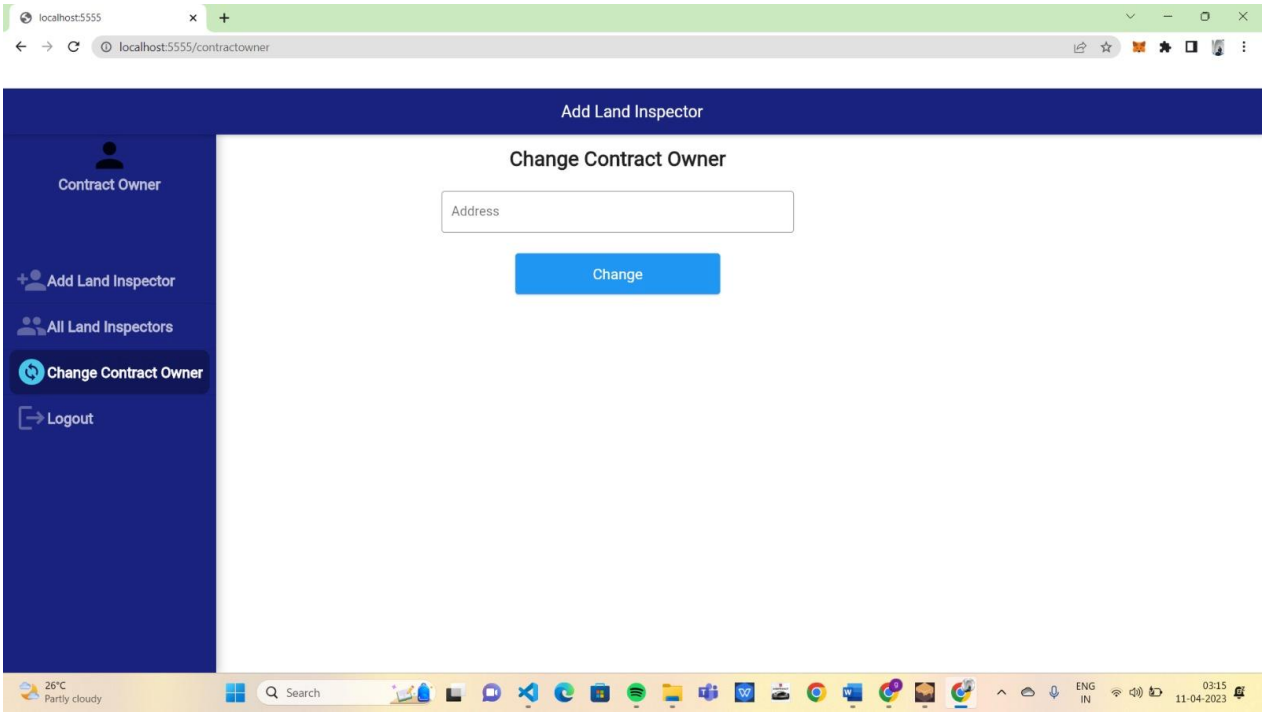

Fig: 8.9 Change Contract Owner Page

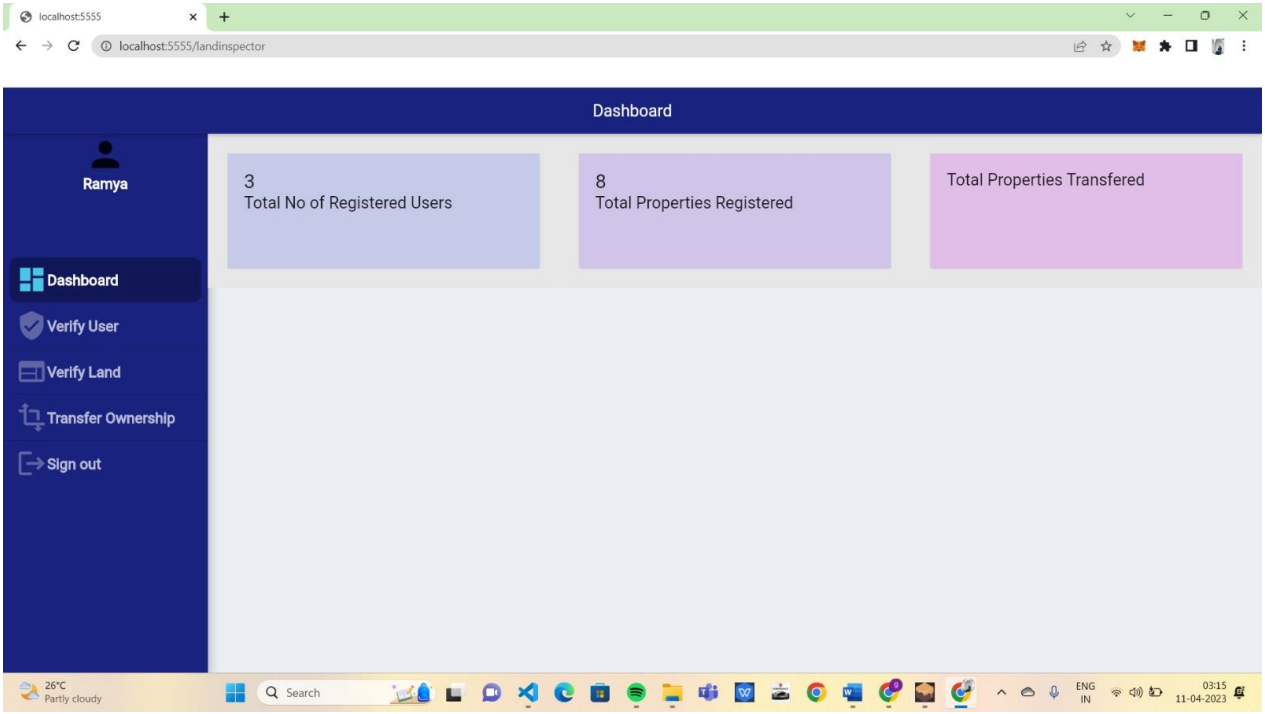

Fig: 8.10 Land Inspector Dashboard

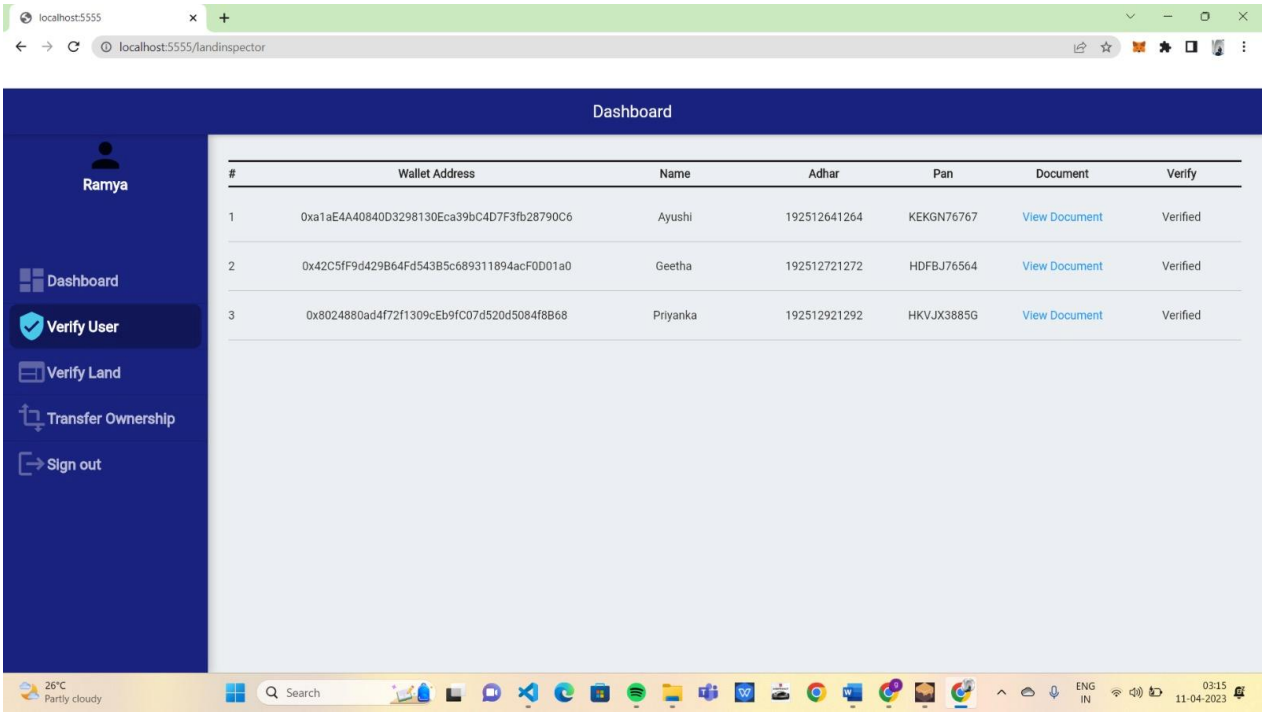

Fig: 8.11 Verify User Page

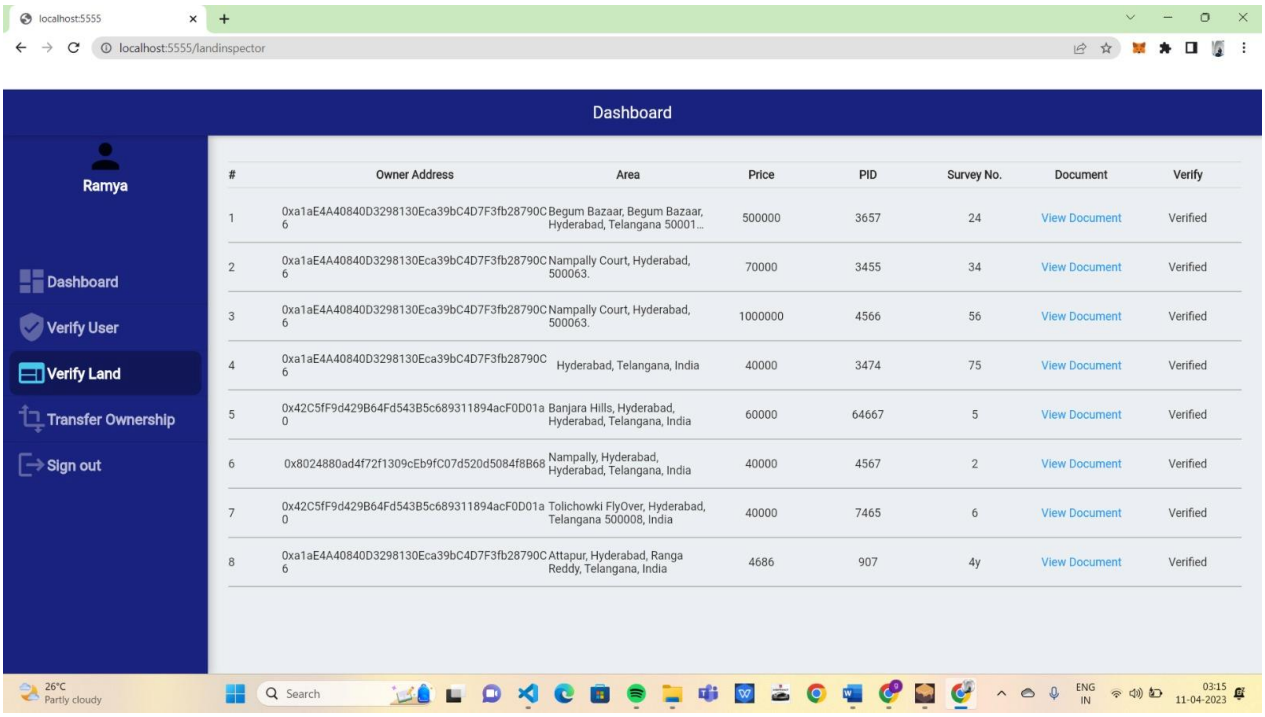

Fig: 8.12 Verify Land Page

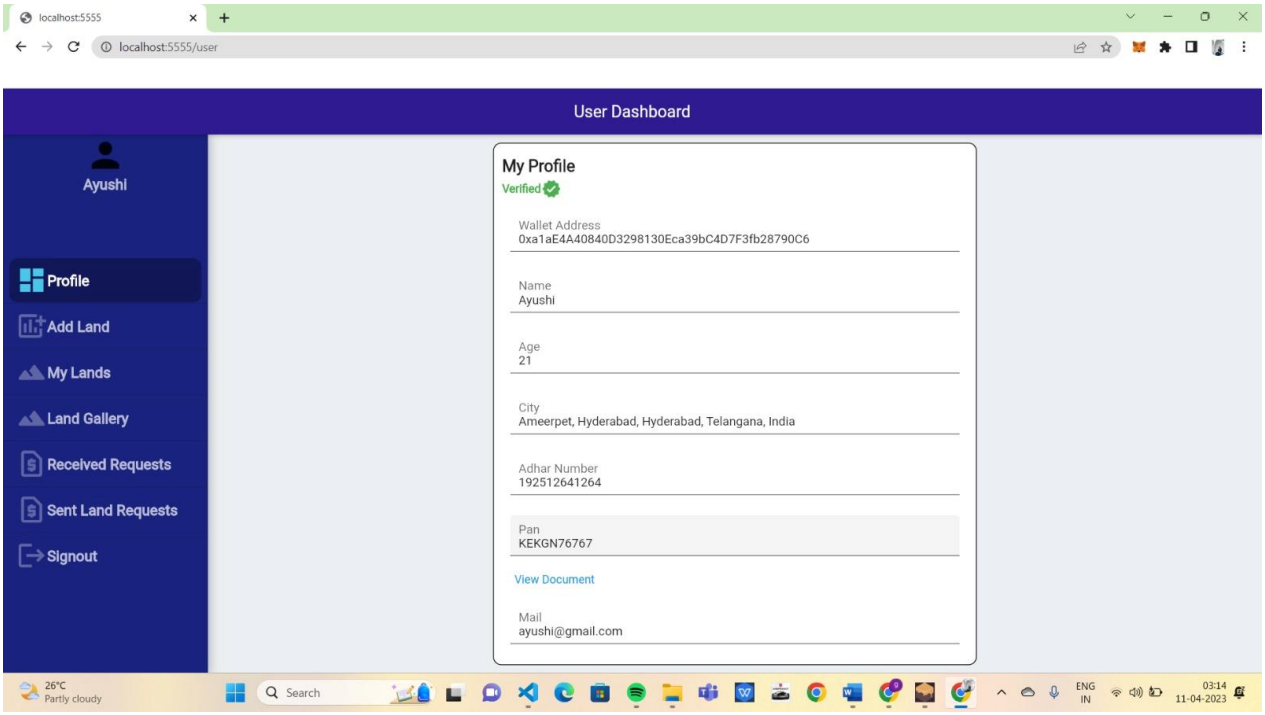

Fig: 8.13 User Profile Page

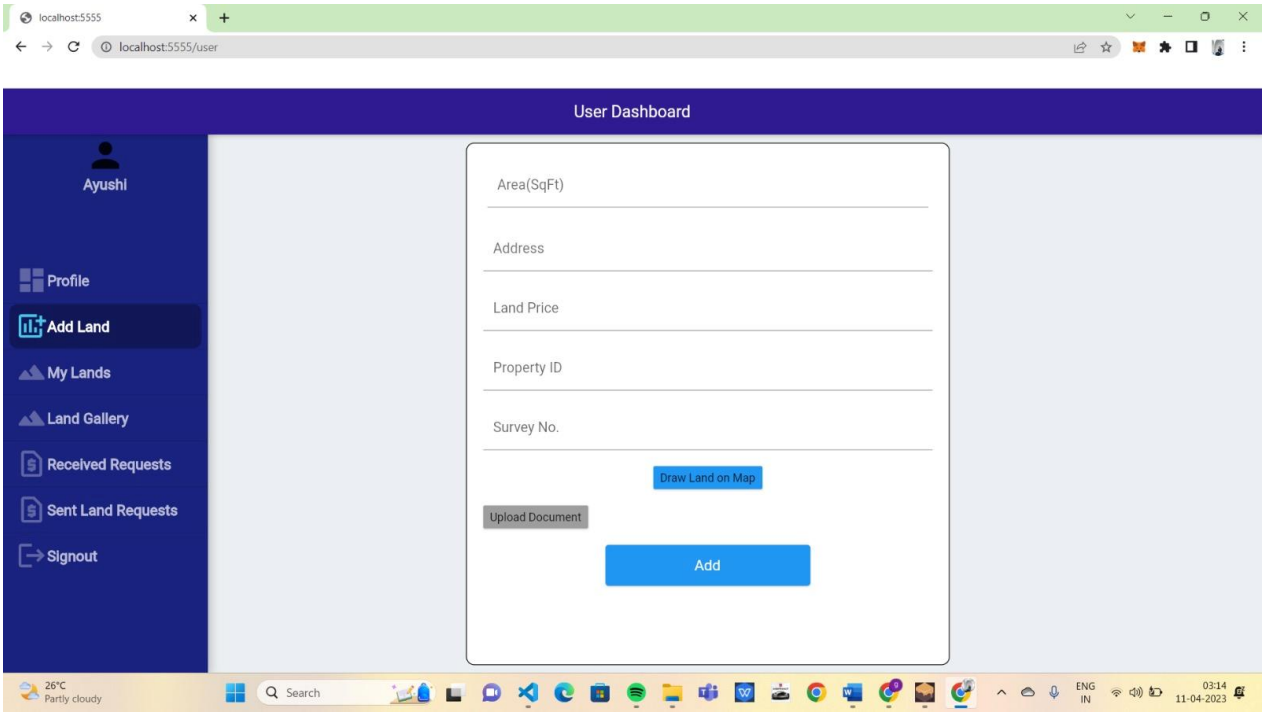

Fig: 8.14 Add Lands Page

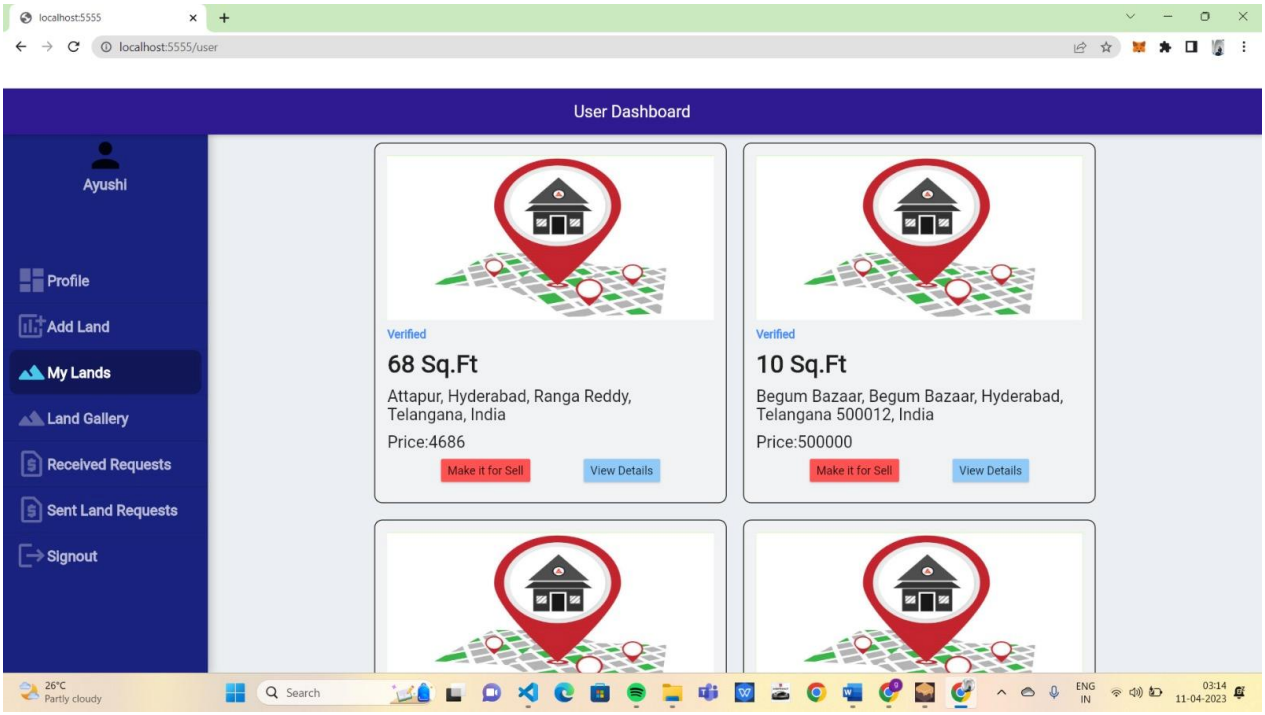

Fig: 8.15 My Lands Page

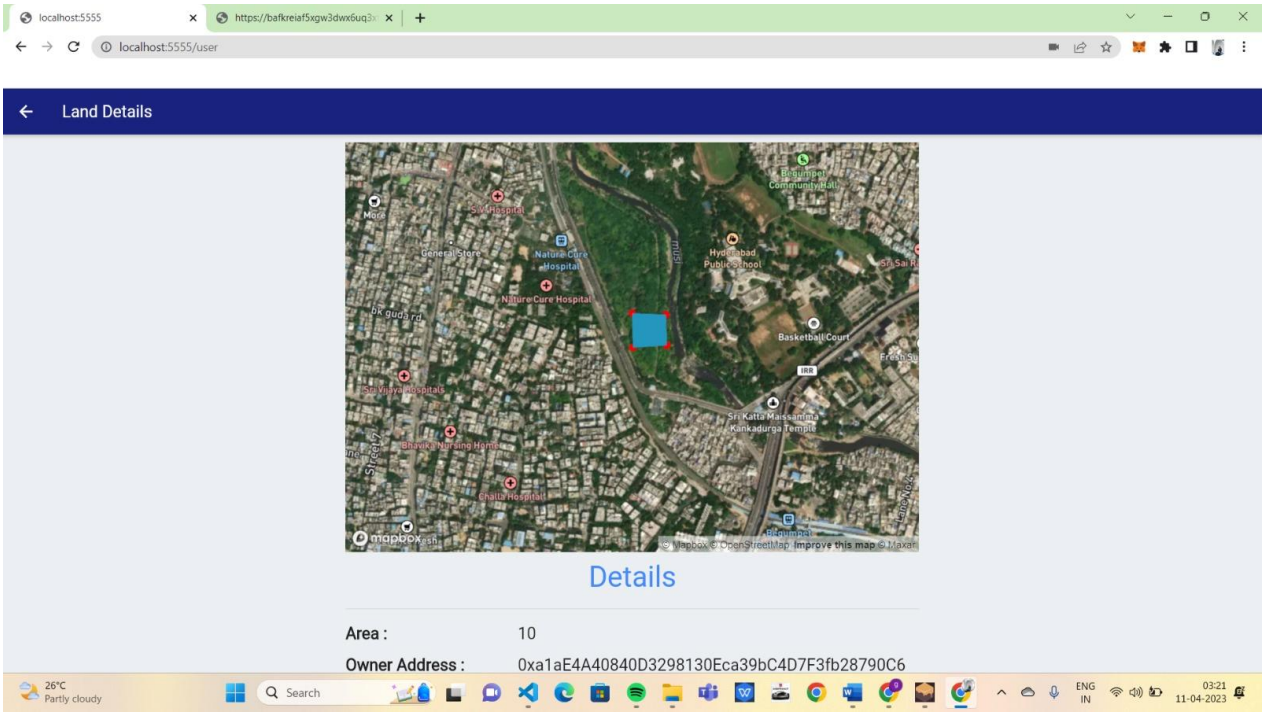

Fig: 8.16 Land Details Map

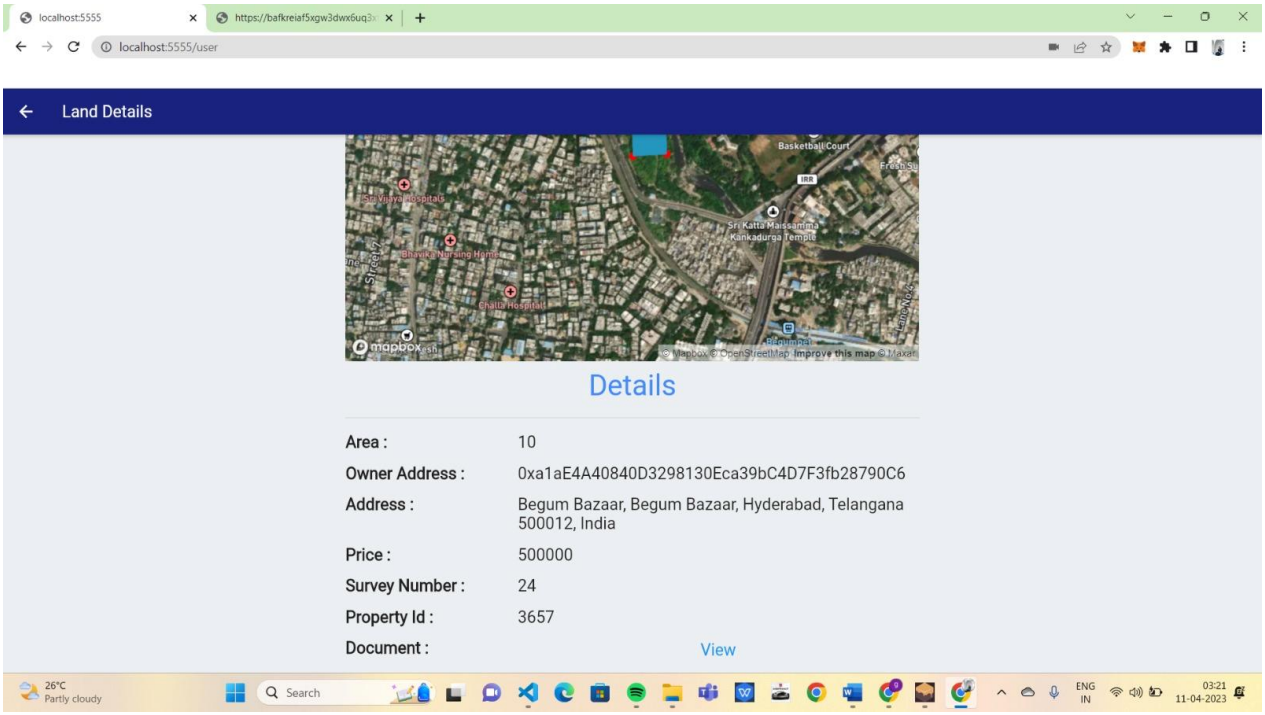

Fig: 8.17 Land Details Page

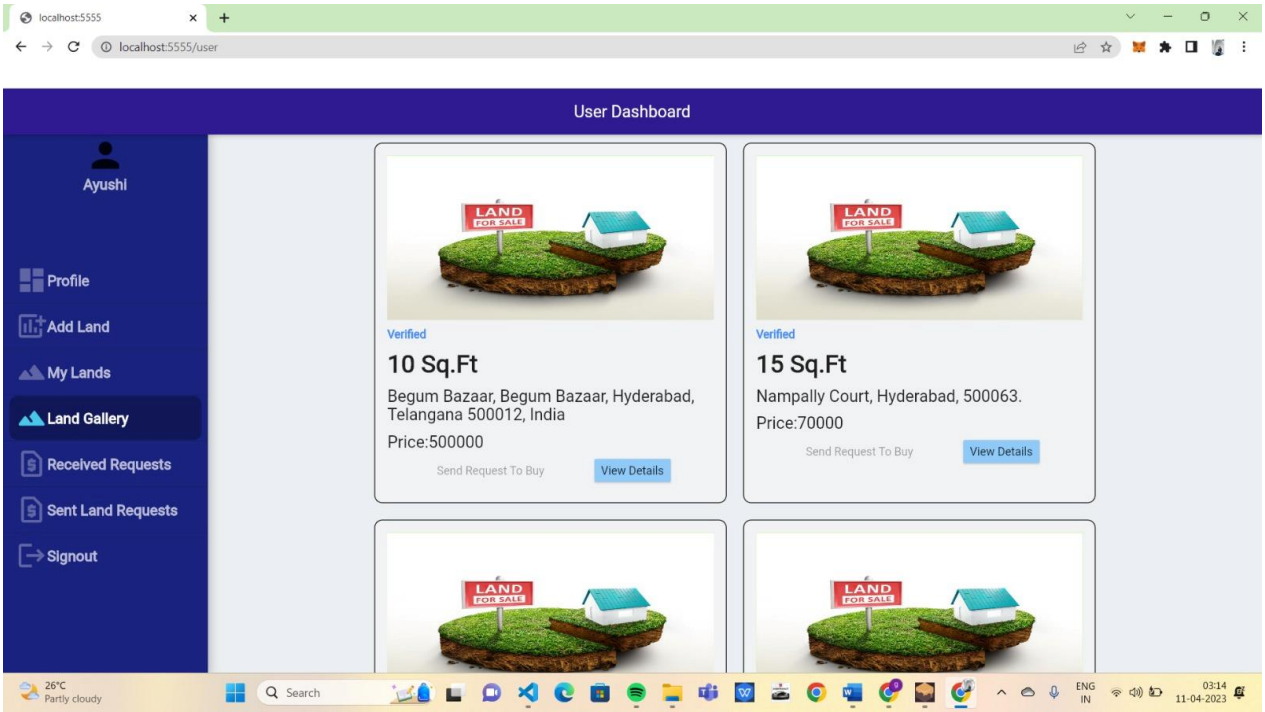

Fig: 8.18 User Land Gallery Page

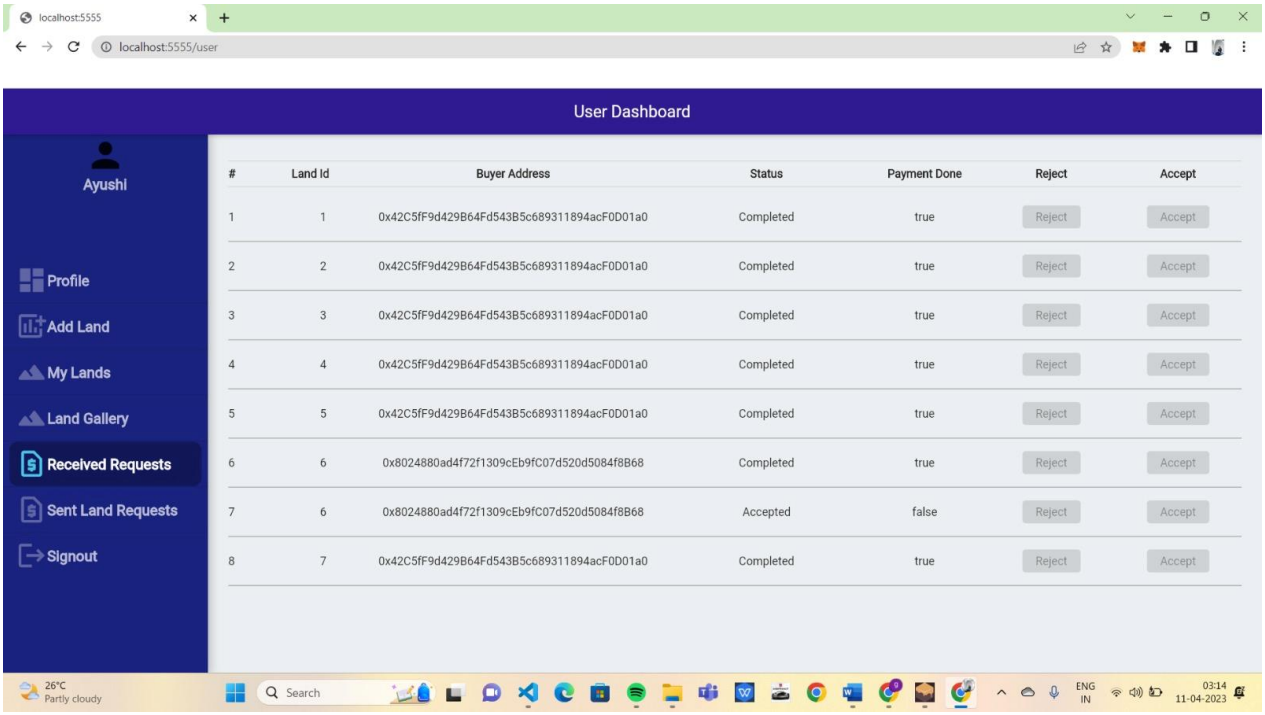

Fig: 8.19 Received Requests Page

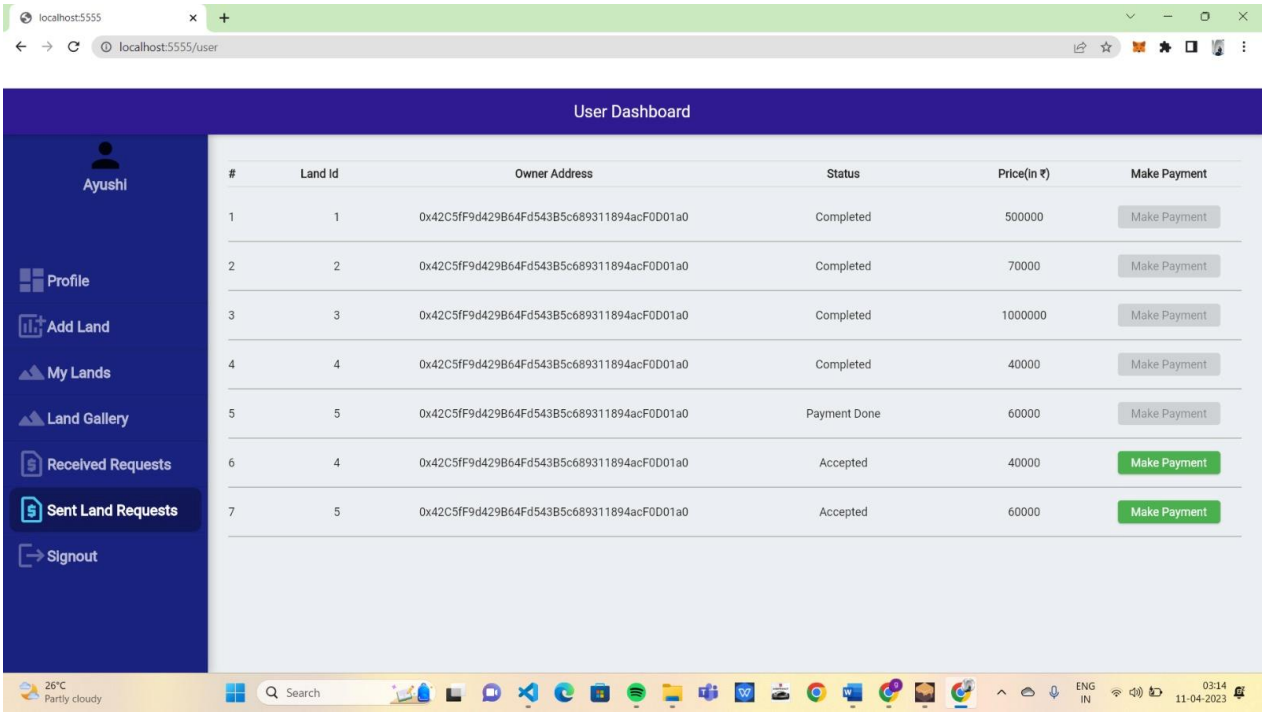

Fig: 8.20 Sent Land Requests Page

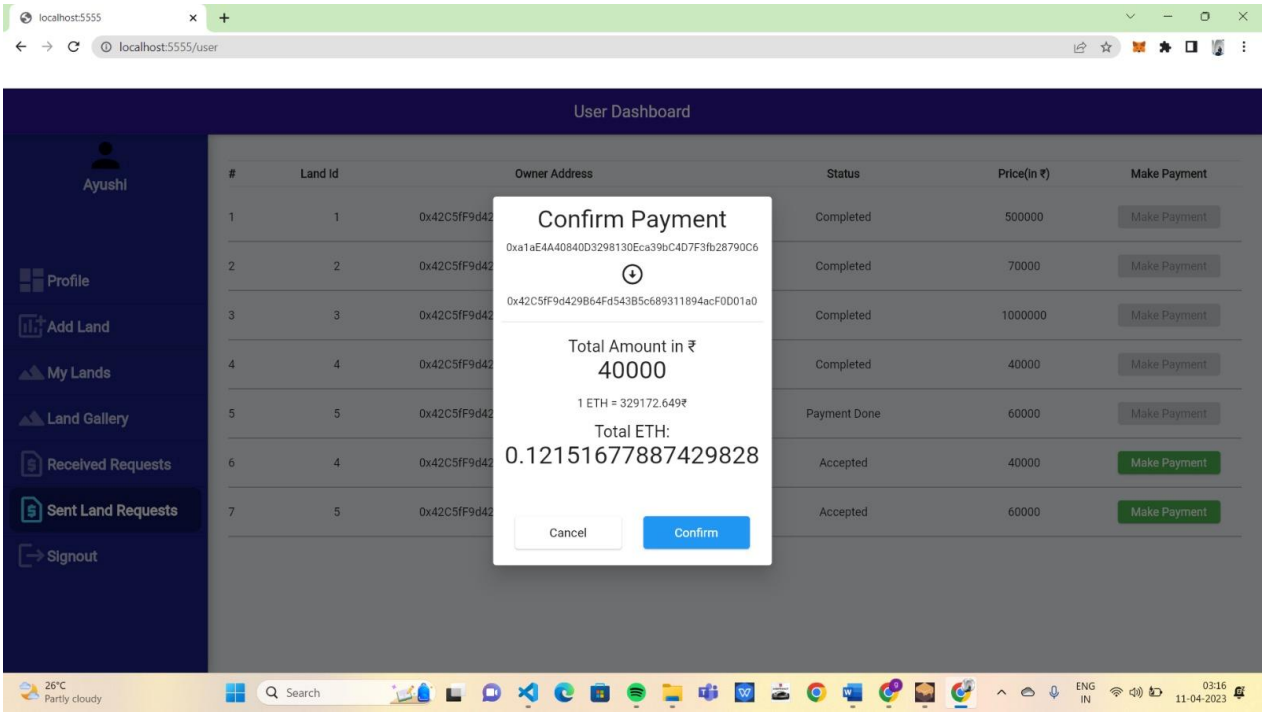

Fig: 8.21 Payment Confirmation Page

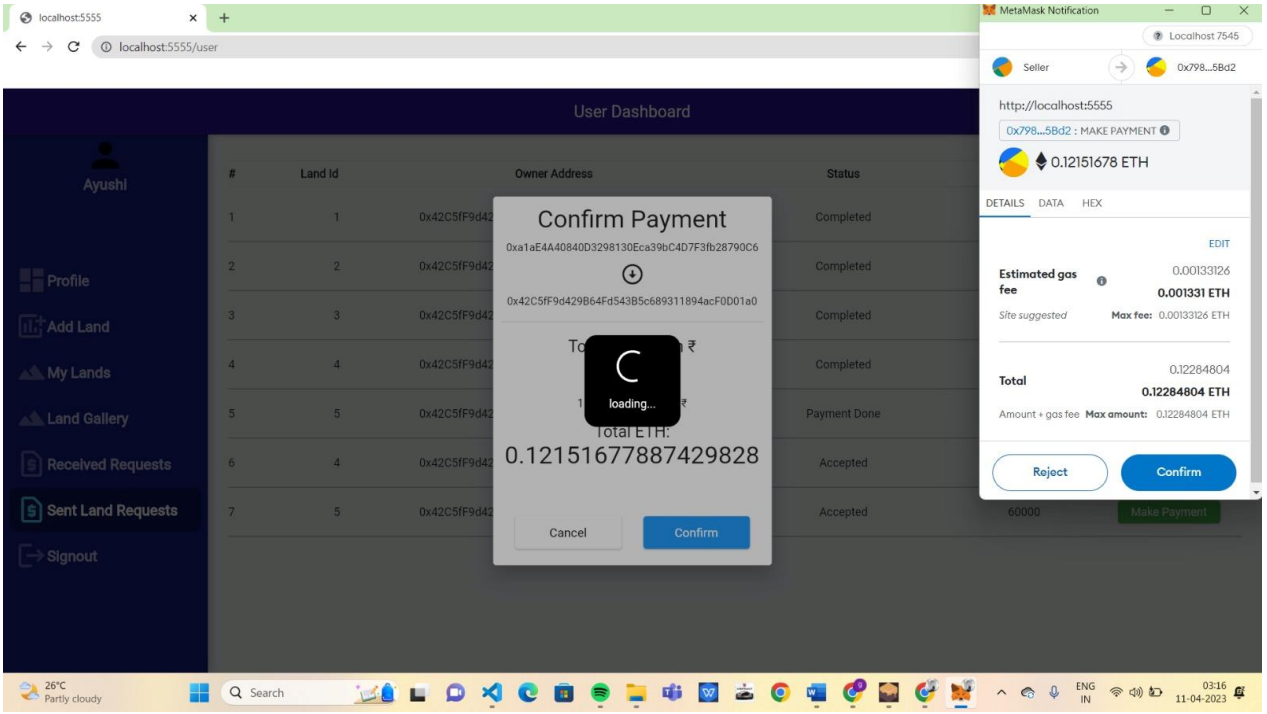

Fig: 8.22 Metamask Payment Page

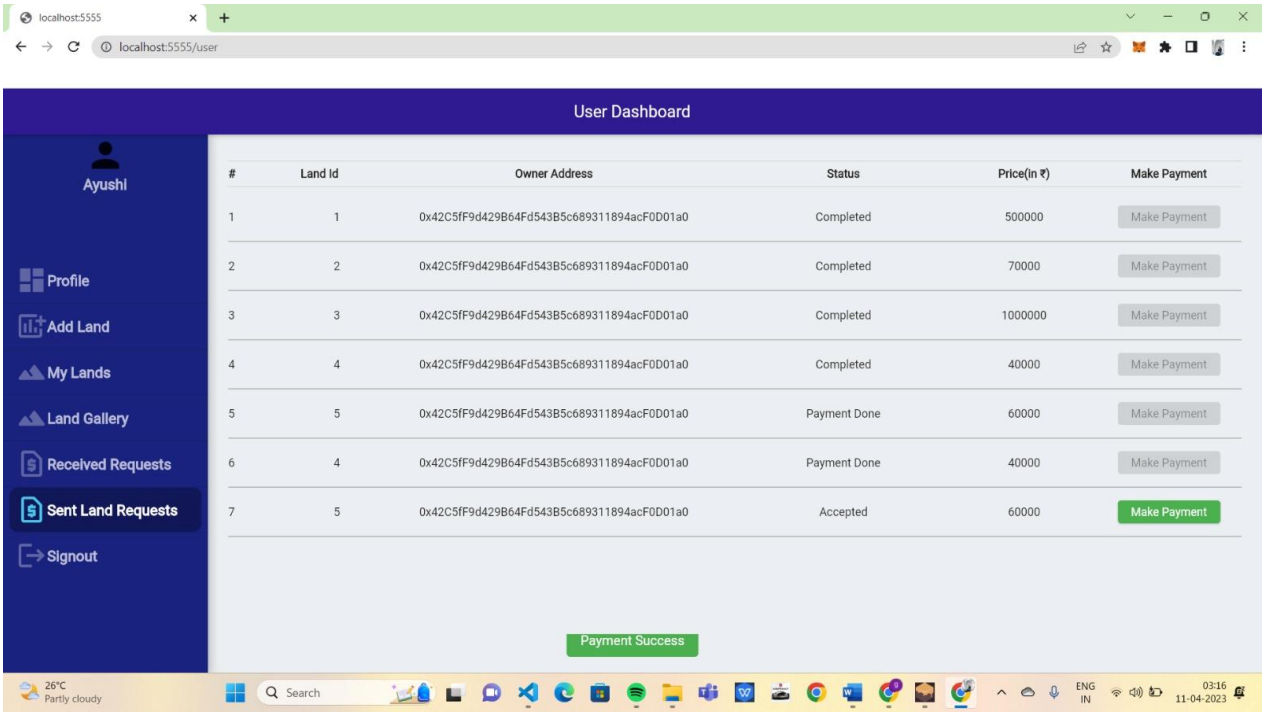

Fig: 8.23 Payment Success Page

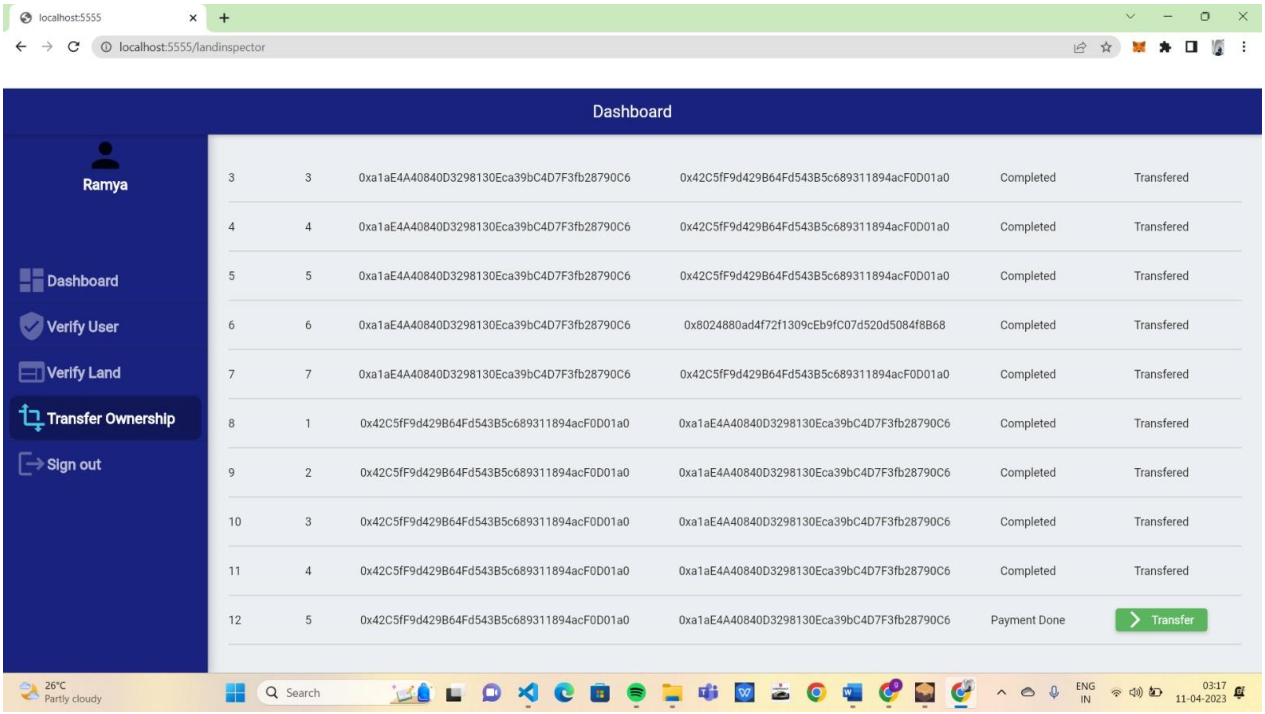

Fig: 8.24 Transfer Ownership Dashboard

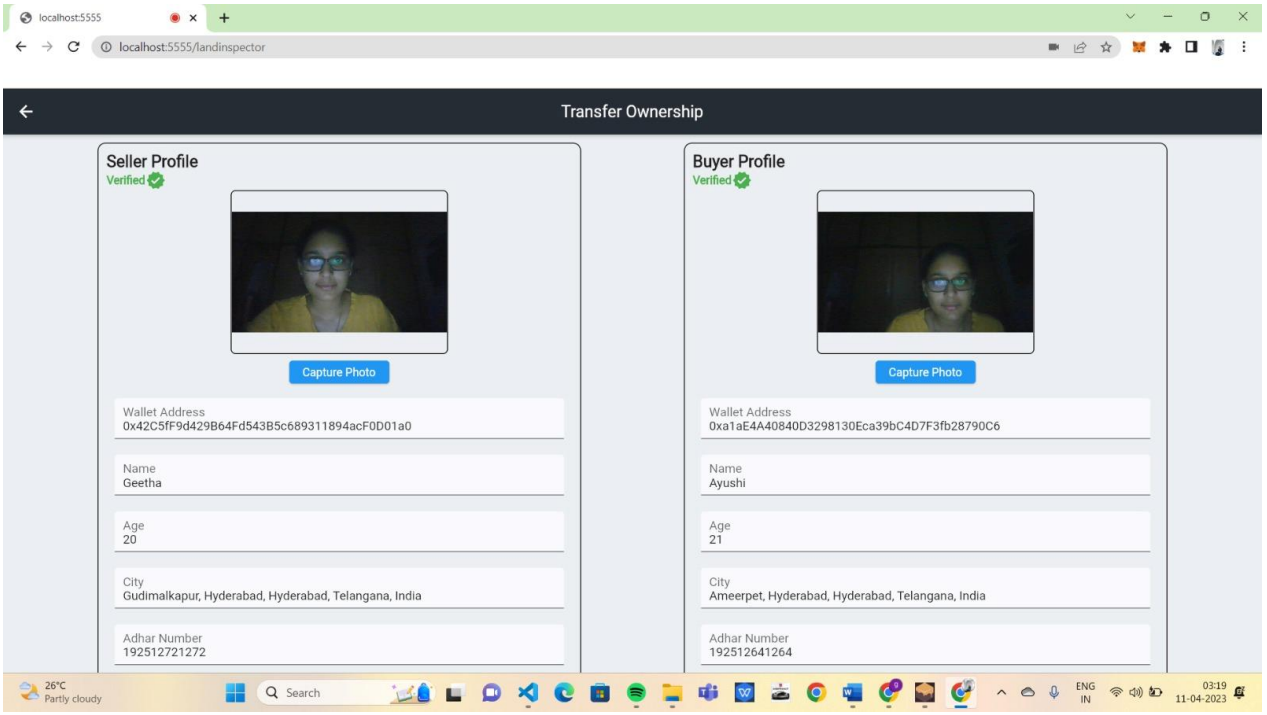

Fig: 8.25 Transfer Ownership Page

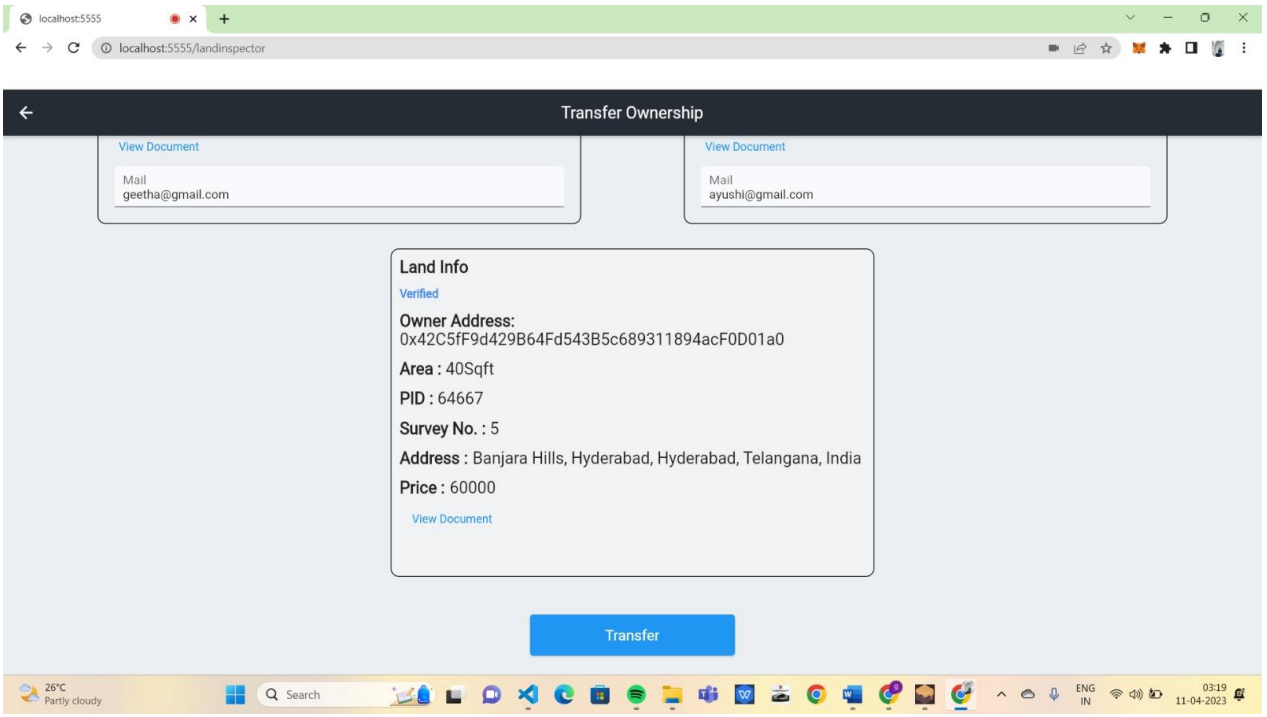

Fig: 8.26 Transfer Ownership Land Page

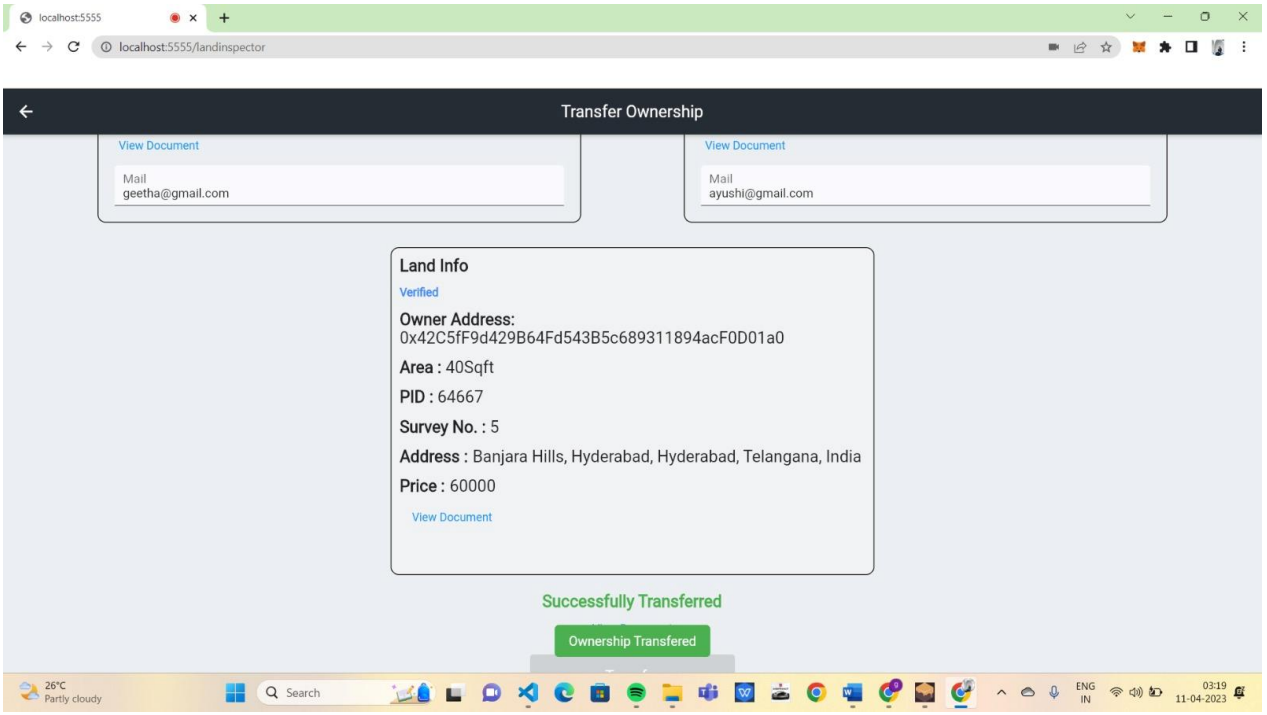

Fig: 8.27 Successful Transfer Page

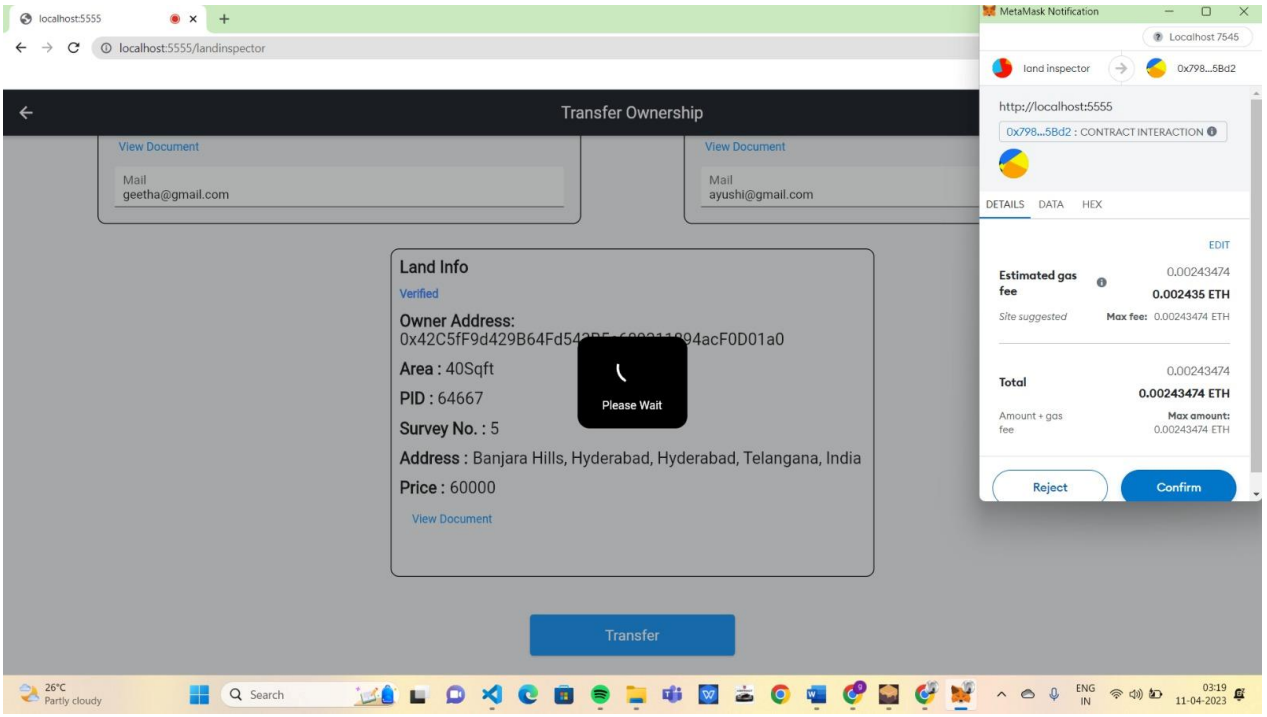

Fig: 8.28 Metamask Transfer Page

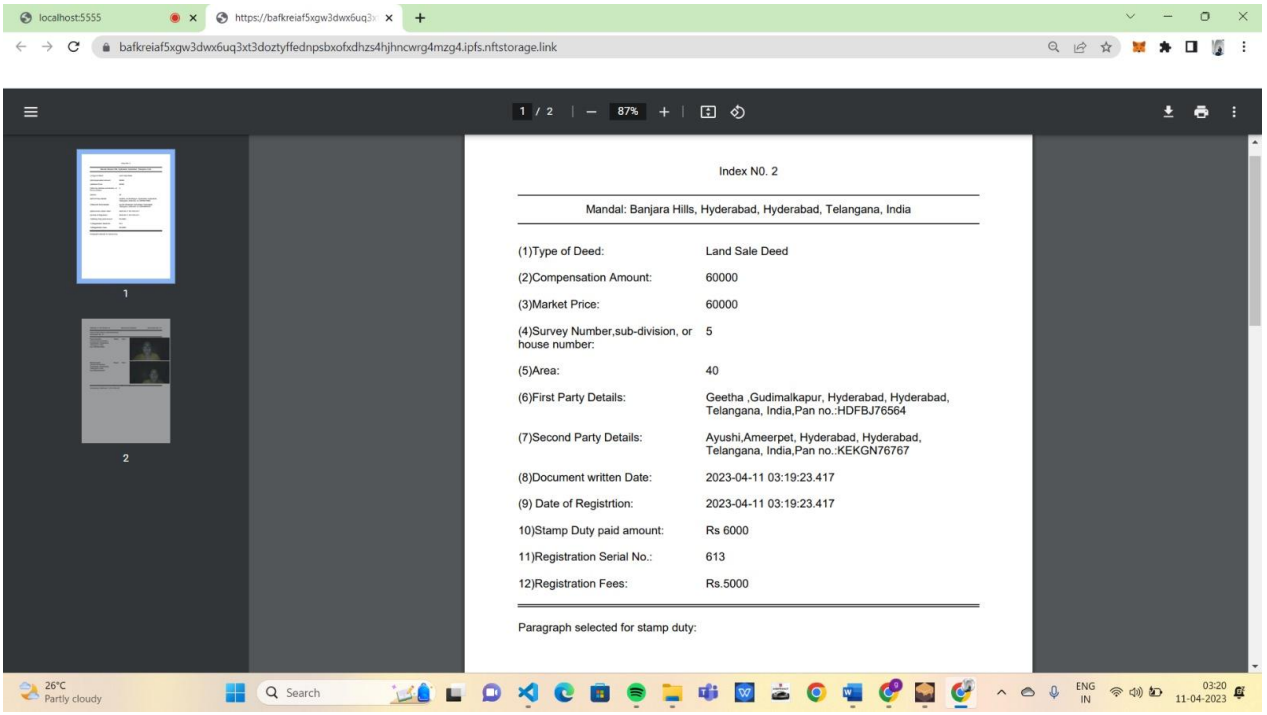

Fig: 8.29 Ownership Document

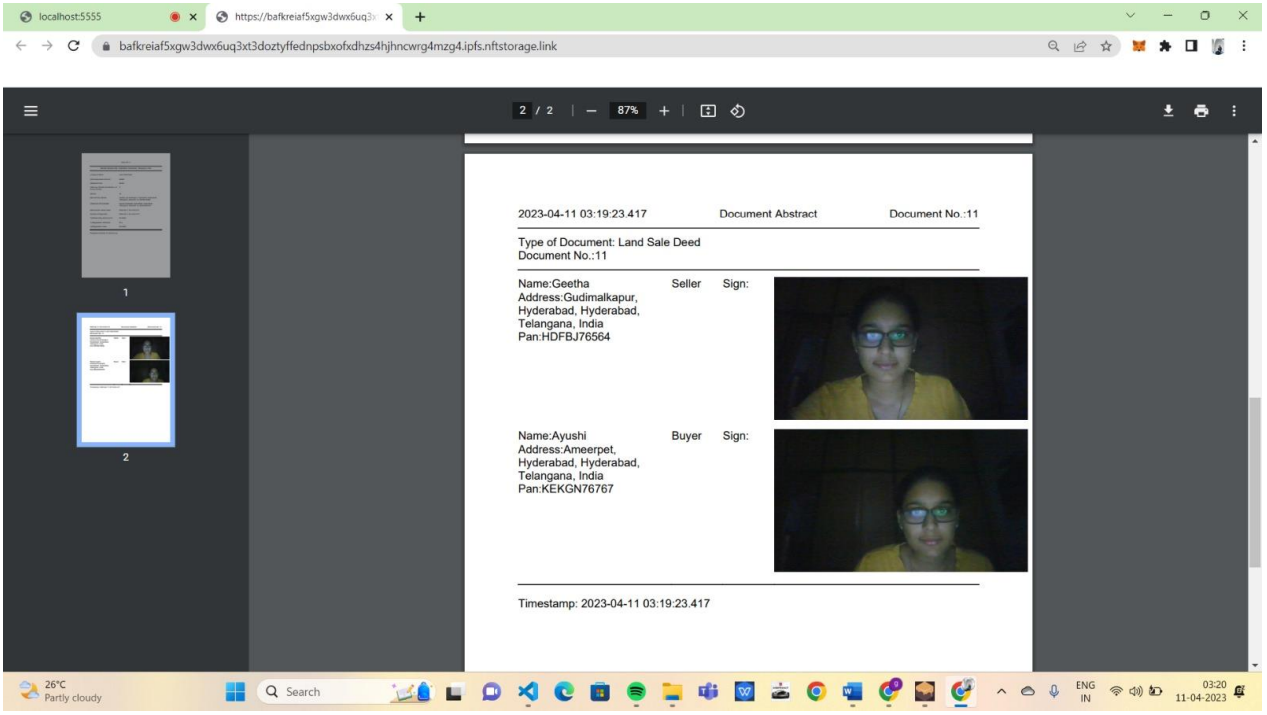

Fig: 8.30 Ownership Document cont.

II. CODE

```
pragma solidity ^0.6.0;

contract Land {
    address contractOwner;

    constructor() public{
        contractOwner = msg.sender;
    }

    struct Landreg {
        uint id;
        uint area;
        string landAddress;
        uint landPrice;
        string allLatitudeLongitude;
        //string allLongitude;
        uint propertyPID;
        string physicalSurveyNumber;
        string document;
        bool isforSell;
        address payable ownerAddress;
        bool isLandVerified;
    }

    struct User{
        address id;
        string name;
        uint age;
        string city;
        string aadharNumber;
        string panNumber;
        string document;
        string email;
        bool isUserVerified;
    }

    struct LandInspector {
        uint id;
        address _addr;
        string name;
        uint age;
        string designation;
        string city;
    }

    struct LandRequest{
        uint reqId;
        address payable sellerId;
        address payable buyerId;
        uint landId;
        reqStatus requestStatus;
        bool isPaymentDone;
    }
    enum reqStatus {requested,accepted,rejected,paymentdone,commpleted}


    uint inspectorsCount;
    uint public userCount;
    uint public landsCount;
    uint public documentId;
    uint requestCount;
```

```
    mapping(address => LandInspector) public InspectorMapping;
    mapping(uint => address[]) allLandInspectorList;
    mapping(address => bool)  RegisteredInspectorMapping;
    mapping(address => User) public UserMapping;
    mapping(uint => address)  AllUsers;
    mapping(uint => address[])  allUsersList;
    mapping(address => bool)  RegisteredUserMapping;
    mapping(address => uint[])  MyLands;
    mapping(uint => Landreg) public lands;
    mapping(uint => LandRequest) public LandRequestMapping;
    mapping(address => uint[])  MyReceivedLandRequest;
    mapping(address => uint[])  MySentLandRequest;
    mapping(uint => uint[])  allLandList;
    mapping(uint => uint[])  paymentDoneList;


    function isContractOwner(address _addr) public view returns(bool){
        if(_addr==contractOwner)
            return true;
        else
            return false;
    }

    function changeContractOwner(address _addr)public {
        require(msg.sender==contractOwner,"you are not contractOwner");

        contractOwner=_addr;
    }

    //-------------------------------------------------LandInspector----------------------------------------------

    function addLandInspector(address _addr,string memory _name, uint _age, string memory _designation,string memory _city)
public returns(bool){
        if(contractOwner!=msg.sender)
            return false;
        require(contractOwner==msg.sender);
        RegisteredInspectorMapping[_addr]=true;
        allLandInspectorList[1].push(_addr);
        InspectorMapping[_addr] = LandInspector(inspectorsCount,_addr,_name, _age, _designation,_city);
        return true;
    }

    function ReturnAllLandIncpectorList() public view returns(address[] memory)
    {
        return allLandInspectorList[1];
    }

    function removeLandInspector(address _addr) public{
        require(msg.sender==contractOwner,"You are not contractOwner");
        require(RegisteredInspectorMapping[_addr],"Land Inspector not found");
        RegisteredInspectorMapping[_addr]=false;


        uint len=allLandInspectorList[1].length;
        for(uint i=0;i<len;i++)
        {
            if(allLandInspectorList[1][i]==_addr)
            {
                allLandInspectorList[1][i]=allLandInspectorList[1][len-1];
                allLandInspectorList[1].pop();
                break;
            }
        }
    }

    function isLandInspector(address _id) public view returns (bool) {
```

```
        if(RegisteredInspectorMapping[_id]){
            return true;
        }else{
            return false;
        }
    }



    //-------------------------------------------------User-------------------------------------------------

    function isUserRegistered(address _addr) public view returns(bool)
    {
        if(RegisteredUserMapping[_addr]){
            return true;
        }else{
            return false;
        }
    }

    function registerUser(string memory _name, uint _age, string memory _city,string memory _aadharNumber, string memory _panNumber, string memory _document, string memory _email
    ) public {

        require(!RegisteredUserMapping[msg.sender]);

        RegisteredUserMapping[msg.sender] = true;
        userCount++;
        allUsersList[1].push(msg.sender);
        AllUsers[userCount]=msg.sender;
        UserMapping[msg.sender] = User(msg.sender, _name, _age, _city,_aadharNumber,_panNumber, _document,_email,false);
        //emit Registration(msg.sender);
    }

    function verifyUser(address _userId) public{
        require(isLandInspector(msg.sender));
        UserMapping[_userId].isUserVerified=true;
    }
    function isUserVerified(address id) public view returns(bool){
        return UserMapping[id].isUserVerified;
    }
    function ReturnAllUserList() public view returns(address[] memory)
    {
        return allUsersList[1];
    }


    //-------------------------------------------------Land-------------------------------------------------
    function addLand(uint _area, string memory _address, uint landPrice,string memory _allLatiLongi, uint _propertyPID,string memory _surveyNum, string memory _document) public {
        require(isUserVerified(msg.sender));
        landsCount++;
        lands[landsCount] = Landreg(landsCount, _area, _address, landPrice,_allLatiLongi,_propertyPID, _surveyNum , _document,false,msg.sender,false);
        MyLands[msg.sender].push(landsCount);
        allLandList[1].push(landsCount);
        // emit AddingLand(landsCount);
    }

    function ReturnAllLandList() public view returns(uint[] memory)
    {
        return allLandList[1];
    }

    function verifyLand(uint _id) public{
        require(isLandInspector(msg.sender));
        lands[_id].isLandVerified=true;
    }
```

```
    function isLandVerified(uint id) public view returns(bool){
        return lands[id].isLandVerified;
    }

    function myAllLands(address id) public view returns( uint[] memory){
        return MyLands[id];
    }


    function makeItforSell(uint id) public{
        require(lands[id].ownerAddress==msg.sender);
        lands[id].isforSell=true;
    }

    function requestforBuy(uint _landId) public
    {
        require(isUserVerified(msg.sender) && isLandVerified(_landId));
        requestCount++;

LandRequestMapping[requestCount]=LandRequest(requestCount,lands[_landId].ownerAddress,msg.sender,_landId,reqStatus.req
uested,false);
        MyReceivedLandRequest[lands[_landId].ownerAddress].push(requestCount);
        MySentLandRequest[msg.sender].push(requestCount);
    }

    function myReceivedLandRequests() public view returns(uint[] memory)
    {
        return MyReceivedLandRequest[msg.sender];
    }
    function mySentLandRequests() public view returns(uint[] memory)
    {
        return MySentLandRequest[msg.sender];
    }
    function acceptRequest(uint _requestId) public
    {
        require(LandRequestMapping[_requestId].sellerId==msg.sender);
        LandRequestMapping[_requestId].requestStatus=reqStatus.accepted;
    }
    function rejectRequest(uint _requestId) public
    {
        require(LandRequestMapping[_requestId].sellerId==msg.sender);
        LandRequestMapping[_requestId].requestStatus=reqStatus.rejected;
    }

    function requesteStatus(uint id) public view returns(bool)
    {
        return LandRequestMapping[id].isPaymentDone;
    }

    function landPrice(uint id) public view returns(uint)
    {
        return lands[id].landPrice;
    }
    function makePayment(uint _requestId) public payable
    {
        require(LandRequestMapping[_requestId].buyerId==msg.sender                                                    &&
LandRequestMapping[_requestId].requestStatus==reqStatus.accepted);

        LandRequestMapping[_requestId].requestStatus=reqStatus.paymentdone;
        //LandRequestMapping[_requestId].sellerId.transfer(lands[LandRequestMapping[_requestId].landId].landPrice);

//lands[LandRequestMapping[_requestId].landId].ownerAddress.transfer(lands[LandRequestMapping[_requestId].landId].landPri
ce);
        lands[LandRequestMapping[_requestId].landId].ownerAddress.transfer(msg.value);
        LandRequestMapping[_requestId].isPaymentDone=true;
        paymentDoneList[1].push(_requestId);
    }
```

```
    function returnPaymentDoneList() public view returns(uint[] memory)
    {
        return paymentDoneList[1];
    }

    function transferOwnership(uint _requestId,string memory documentUrl) public returns(bool)
    {
        require(isLandInspector(msg.sender));
        if(LandRequestMapping[_requestId].isPaymentDone==false)
            return false;
        documentId++;
        LandRequestMapping[_requestId].requestStatus=reqStatus.commpleted;
        MyLands[LandRequestMapping[_requestId].buyerId].push(LandRequestMapping[_requestId].landId);

        uint len=MyLands[LandRequestMapping[_requestId].sellerId].length;
        for(uint i=0;i<len;i++)
        {
            if(MyLands[LandRequestMapping[_requestId].sellerId][i]==LandRequestMapping[_requestId].landId)
            {
                MyLands[LandRequestMapping[_requestId].sellerId][i]=MyLands[LandRequestMapping[_requestId].sellerId][len-1];
                //MyLands[LandRequestMapping[_requestId].sellerId].length--;
                MyLands[LandRequestMapping[_requestId].sellerId].pop();
                break;
            }
        }
        lands[LandRequestMapping[_requestId].landId].document=documentUrl;
        lands[LandRequestMapping[_requestId].landId].isforSell=false;
        lands[LandRequestMapping[_requestId].landId].ownerAddress=LandRequestMapping[_requestId].buyerId;
        return true;
    }
    function makePaymentTestFun(address payable _reveiver) public payable
    {
        _reveiver.transfer(msg.value);
    }
}

import 'dart:convert';

import 'package:file_picker/file_picker.dart';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:flutter_styled_toast/flutter_styled_toast.dart';
import 'package:land_registration/providers/LandRegisterModel.dart';
import 'package:land_registration/constant/loadingScreen.dart';
import 'package:land_registration/screens/ChooseLandMap.dart';
import 'package:land_registration/screens/viewLandDetails.dart';
import 'package:land_registration/widget/land_container.dart';
import 'package:land_registration/widget/menu_item_tile.dart';
import 'package:mapbox_search/mapbox_search.dart';
import 'package:provider/provider.dart';
import '../providers/MetamaskProvider.dart';
import '../constant/constants.dart';
import 'package:flutter_smart_dialog/flutter_smart_dialog.dart';
import 'package:http/http.dart' as http;
import '../constant/utils.dart';

class UserDashBoard extends StatefulWidget {
  const UserDashBoard({Key? key}) : super(key: key);

  @override
  _UserDashBoardState createState() => _UserDashBoardState();
}

class _UserDashBoardState extends State<UserDashBoard> {
  var model, model2;
  final GlobalKey<ScaffoldState> _scaffoldKey = GlobalKey<ScaffoldState>();
  int screen = 0;
  late List<dynamic> userInfo;
```

```
bool isLoading = true, isUserVerified = false;
bool isUpdated = true;
List<List<dynamic>> LandGall = [];
String name = "";

final _formKey = GlobalKey<FormState>();
late String area,
    landAddress,
    landPrice,
    propertyID,
    surveyNo,
    document,
    allLatiLongi;
List<List<dynamic>> landInfo = [];
List<List<dynamic>> receivedRequestInfo = [];
List<List<dynamic>> sentRequestInfo = [];
List<dynamic> prices = [];
List<Menu> menuItems = [
  Menu(title: 'Profile', icon: Icons.dashboard),
  Menu(title: 'Add Land', icon: Icons.add_chart),
  Menu(title: 'My Lands', icon: Icons.landscape_rounded),
  Menu(title: 'Land Gallery', icon: Icons.landscape_rounded),
  Menu(title: 'Received Requests', icon: Icons.request_page_outlined),
  Menu(title: 'Sent Land Requests', icon: Icons.request_page_outlined),
  Menu(title: 'Signout', icon: Icons.logout),
];
Map<String, String> requestStatus = {
  '0': 'Pending',
  '1': 'Accepted',
  '2': 'Rejected',
  '3': 'Payment Done',
  '4': 'Completed'
};

List<MapBoxPlace> predictions = [];
late PlacesSearch placesSearch;
final FocusNode _focusNode = FocusNode();
late OverlayEntry _overlayEntry;
final LayerLink _layerLink = LayerLink();
TextEditingController addressController = TextEditingController();

OverlayEntry _createOverlayEntry() {
  return OverlayEntry(
      builder: (context) => Positioned(
          width: 540,
          child: CompositedTransformFollower(
            link: _layerLink,
            showWhenUnlinked: false,
            offset: const Offset(0.0, 40 + 5.0),
            child: Material(
              elevation: 4.0,
              child: ListView(
                padding: EdgeInsets.zero,
                shrinkWrap: true,
                children: List.generate(
                    predictions.length,
                    (index) => ListTile(
                          title:
                              Text(predictions[index].placeName.toString()),
                          onTap: () {
                            addressController.text =
                                predictions[index].placeName.toString();

                            setState(() {});
                            _overlayEntry.remove();
                            _overlayEntry.dispose();
                          },
                        )),
```

```
              ),
            ),
          ),
        ));
}

Future<void> autocomplete(value) async {
  List<MapBoxPlace>? res = await placesSearch.getPlaces(value);
  if (res != null) predictions = res;
  setState(() {});
  // print(res);
  // print(res![0].placeName);
  // print(res![0].geometry!.coordinates);
  // print(res![0]);
}

@override
void initState() {
  placesSearch = PlacesSearch(
    apiKey: mapBoxApiKey,
    limit: 10,
  );

  _focusNode.addListener(() {
    if (_focusNode.hasFocus) {
      _overlayEntry = _createOverlayEntry();
      Overlay.of(context)!.insert(_overlayEntry);
    } else {
      _overlayEntry.remove();
    }
  });
  super.initState();
}

getLandInfo() async {
  setState(() {
    landInfo = [];
    isLoading = true;
  });
  List<dynamic> landList;
  if (connectedWithMetamask) {
    landList = await model2.myAllLands();
  } else {
    landList = await model.myAllLands();
  }

  List<List<dynamic>> info = [];
  List<dynamic> temp;
  for (int i = 0; i < landList.length; i++) {
    if (connectedWithMetamask) {
      temp = await model2.landInfo(landList[i]);
    } else {
      temp = await model.landInfo(landList[i]);
    }
    landInfo.add(temp);
    setState(() {
      isLoading = false;
    });
  }
  setState(() {
    isLoading = false;
  });
}

getLandGallery() async {
  setState(() {
    isLoading = true;
    LandGall = [];
```

```
  });
  List<dynamic> landList;
  if (connectedWithMetamask) {
    landList = await model2.allLandList();
  } else {
    landList = await model.allLandList();
  }

  // List<List<dynamic>> allInfo = [];
  List<dynamic> temp;
  for (int i = 0; i < landList.length; i++) {
    if (connectedWithMetamask) {
      temp = await model2.landInfo(landList[i]);
    } else {
      temp = await model.landInfo(landList[i]);
    }
    LandGall.add(temp);
    setState(() {
      isLoading = false;
    });
  }
 // screen = 3;
  isLoading = false;
  setState(() {});
}

getMySentRequest() async {
  //SmartDialog.showLoading();
  sentRequestInfo = [];
  setState(() {
    isLoading = true;
  });
  await getEthToInr();
  List<dynamic> requestList;
  if (connectedWithMetamask) {
    requestList = await model2.mySentRequest();
  } else {
    requestList = await model.mySentRequest();
  }

  List<dynamic> temp;
  var pri;
  for (int i = 0; i < requestList.length; i++) {
    if (connectedWithMetamask) {
      temp = await model2.requestInfo(requestList[i]);
      pri = await model2.landPrice(temp[3]);
    } else {
      temp = await model.requestInfo(requestList[i]);
      pri = await model.landPrice(temp[3]);
    }
    prices.add(pri);
    sentRequestInfo.add(temp);
    isLoading = false;

    // SmartDialog.dismiss();
    setState(() {});
  }

 // screen = 5;
  isLoading = false;

  // SmartDialog.dismiss();
  setState(() {});
}

getMyReceivedRequest() async {
  receivedRequestInfo = [];
  setState(() {
```

```
    isLoading = true;
  });
  List<dynamic> requestList;
  if (connectedWithMetamask) {
    requestList = await model2.myReceivedRequest();
  } else {
    requestList = await model.myReceivedRequest();
  }

  List<dynamic> temp;
  for (int i = 0; i < requestList.length; i++) {
    if (connectedWithMetamask) {
      temp = await model2.requestInfo(requestList[i]);
    } else {
      temp = await model.requestInfo(requestList[i]);
    }
    receivedRequestInfo.add(temp);
    isLoading = false;
    setState(() {});
  }
  isLoading = false;
//  screen = 4;
  setState(() {});
}

Future<void> getProfileInfo() async {
  // setState(() {
  //   isLoading = true;
  // });
  if (connectedWithMetamask) {
    userInfo = await model2.myProfileInfo();
  } else {
    userInfo = await model.myProfileInfo();
  }
  name = userInfo[1];
  setState(() {
    isLoading = false;
  });
}

String docuName = "";
late PlatformFile documentFile;
String cid = "", docUrl = "";
bool isFilePicked = false;

pickDocument() async {
  FilePickerResult? result = await FilePicker.platform.pickFiles(
    type: FileType.custom,
    allowedExtensions: ['jpg', 'pdf'],
  );

  if (result != null) {
    isFilePicked = true;
    docuName = result.files.single.name;
    documentFile = result.files.first;
  }
  setState(() {});
}

Future<bool> uploadDocument() async {
  String url = "https://api.nft.storage/upload";
  var header = {"Authorization": "Bearer $nftStorageApiKey"};

  if (isFilePicked) {
    try {
      final response = await http.post(Uri.parse(url),
          headers: header, body: documentFile.bytes);
      var data = jsonDecode(response.body);
```

```
      //print(data);
      if (data['ok']) {
        cid = data["value"]["cid"];
        docUrl = "https://" + cid + ".ipfs.dweb.link";

        return true;
      }
    } catch (e) {
      print(e);
      showToast("Something went wrong,while document uploading",
          context: context, backgroundColor: Colors.red);
    }
  } else {
    showToast("Choose Document",
        context: context, backgroundColor: Colors.red);
    return false;
  }
  return false;
}

@override
Widget build(BuildContext context) {
  model = Provider.of<LandRegisterModel>(context);
  model2 = Provider.of<MetaMaskProvider>(context);
  if (isUpdated) {
    getProfileInfo();
    isUpdated = false;
  }

  return Scaffold(
    key: _scaffoldKey,
    appBar: AppBar(
      centerTitle: true,
      backgroundColor: const Color(0xFF311B92),
      leading: isDesktop
          ? Container()
          : GestureDetector(
              child: const Padding(
                padding: EdgeInsets.all(8.0),
                child: Icon(
                  Icons.menu,
                  color: Colors.white,
                ), //AnimatedIcon(icon: AnimatedIcons.menu_arrow,progress: _animationController,),
              ),
              onTap: () {
                _scaffoldKey.currentState!.openDrawer();
              },
            ),
      title: const Text('User Dashboard'),
    ),
    drawer: drawer2(),
    drawerScrimColor: Colors.transparent,
    body: Row(
      children: [
        isDesktop ? drawer2() : Container(),
        if (screen == 0)
          userProfile()
        else if (screen == 1)
          addLand()
        else if (screen == 2)
          myLands()
        else if (screen == 3)
          landGallery()
        else if (screen == 4)
          Expanded(
            child: Container(
              padding: const EdgeInsets.all(25),
              child: receivedRequest(),
```

```
            ),
          )
        else if (screen == 5)
          Expanded(
            child: Container(
              padding: const EdgeInsets.all(25),
              child: sentRequest(),
            ),
          )
      ],
    ),
  );
 }

Widget sentRequest() {

   return ListView.builder(
    itemCount: sentRequestInfo == null ? 1 : sentRequestInfo.length + 1,
    itemBuilder: (BuildContext context, int index) {
     if (index == 0) {
      return Column(
       children: [
        const Divider(
         height: 15,
        ),
        Row(
         children: const [
          Expanded(
           child: Text(
            '#',
            style: TextStyle(fontWeight: FontWeight.bold),
           ),
           flex: 1,
          ),
          Expanded(
           child: Text(
            'Land Id',
            style: TextStyle(fontWeight: FontWeight.bold),
           ),
           flex: 1,
          ),
          Expanded(
             child: Center(
              child: Text('Owner Address',
                 style: TextStyle(fontWeight: FontWeight.bold)),
             ),
             flex: 5),
          Expanded(
           child: Center(
             child: Text('Status',
                style: TextStyle(fontWeight: FontWeight.bold)),
           ),
           flex: 3,
          ),
          Expanded(
           child: Center(
             child: Text('Price(in ₹)',
                style: TextStyle(fontWeight: FontWeight.bold)),
           ),
           flex: 2,
          ),
          Expanded(
           child: Center(
             child: Text('Make Payment',
                style: TextStyle(fontWeight: FontWeight.bold)),
           ),
           flex: 2,
          )
```

```
        ],
      ),
      const Divider(
        height: 15,
      )
    ],
  );
}
index -= 1;
List<dynamic> data = sentRequestInfo[index];
return Container(
  height: 60,
  decoration: const BoxDecoration(
    border: Border(bottom: BorderSide(color: Colors.grey, width: 1)),
  ),
  child: Row(
    children: [
      Expanded(
        child: Text((index + 1).toString()),
        flex: 1,
      ),
      Expanded(child: Center(child: Text(data[3].toString())), flex: 1),
      Expanded(
          child: Center(
            child: Text(data[1].toString()),
          ),
          flex: 5),
      Expanded(
          child: Center(
            child: Text(requestStatus[data[4].toString()].toString()),
          ),
          flex: 3),
      Expanded(
          child: Center(
            child: Text(prices[index].toString()),
          ),
          flex: 2),
      Expanded(
          child: Center(
            child: ElevatedButton(
                style: ElevatedButton.styleFrom(primary: Colors.green),
                onPressed: data[4].toString() != '1'
                    ? null
                    : () async {
                        _paymentDialog(
                            data[2],
                            data[1],
                            prices[index].toString(),
                            double.parse(prices[index].toString()) /
                                ethToInr,
                            ethToInr,
                            data[0]);
                        // SmartDialog.showLoading();
                        // try {
                        //   //await model.rejectRequest(data[0]);
                        //   //await getMyReceivedRequest();
                        // } catch (e) {
                        //   print(e);
                        // }
                        //
                        // //await Future.delayed(Duration(seconds: 2));
                        // SmartDialog.dismiss();
                      },
                child: const Text('Make Payment')),
          ),
          flex: 2),
    ],
  ),
```

```
      );
    },
  );
}

Widget receivedRequest() {

  return ListView.builder(
    itemCount:
        receivedRequestInfo == null ? 1 : receivedRequestInfo.length + 1,
    itemBuilder: (BuildContext context, int index) {
      if (index == 0) {
        return Column(
          children: [
            const Divider(
              height: 15,
            ),
            Row(
              children: const [
                Expanded(
                  child: Text(
                    '#',
                    style: TextStyle(fontWeight: FontWeight.bold),
                  ),
                  flex: 1,
                ),
                Expanded(
                  child: Text(
                    'Land Id',
                    style: TextStyle(fontWeight: FontWeight.bold),
                  ),
                  flex: 1,
                ),
                Expanded(
                    child: Center(
                      child: Text('Buyer Address',
                          style: TextStyle(fontWeight: FontWeight.bold)),
                    ),
                    flex: 5),
                Expanded(
                  child: Center(
                    child: Text('Status',
                        style: TextStyle(fontWeight: FontWeight.bold)),
                  ),
                  flex: 3,
                ),
                Expanded(
                  child: Center(
                    child: Text('Payment Done',
                        style: TextStyle(fontWeight: FontWeight.bold)),
                  ),
                  flex: 2,
                ),
                Expanded(
                  child: Center(
                    child: Text('Reject',
                        style: TextStyle(fontWeight: FontWeight.bold)),
                  ),
                  flex: 2,
                ),
                Expanded(
                  child: Center(
                    child: Text('Accept',
                        style: TextStyle(fontWeight: FontWeight.bold)),
                  ),
                  flex: 2,
                )
              ],
```

```
      ),
      const Divider(
        height: 15,
      )
    ],
  );
}
index -= 1;
List<dynamic> data = receivedRequestInfo[index];
return Container(
  height: 60,
  decoration: const BoxDecoration(
    border: Border(bottom: BorderSide(color: Colors.grey, width: 1)),
  ),
  child: Row(
    children: [
      Expanded(
        child: Text((index + 1).toString()),
        flex: 1,
      ),
      Expanded(child: Center(child: Text(data[3].toString())), flex: 1),
      Expanded(
          child: Center(
            child: Text(data[2].toString()),
          ),
          flex: 5),
      Expanded(
          child: Center(
            child: Text(requestStatus[data[4].toString()].toString()),
          ),
          flex: 3),
      Expanded(child: Center(child: Text(data[5].toString())), flex: 2),
      Expanded(
          child: Center(
            child: ElevatedButton(
                style:
                    ElevatedButton.styleFrom(primary: Colors.redAccent),
                onPressed: data[4].toString() != '0'
                    ? null
                    : () async {
                        SmartDialog.showLoading();
                        try {
                          if (connectedWithMetamask) {
                            await model2.rejectRequest(data[0]);
                          } else {
                            await model.rejectRequest(data[0]);
                          }
                          await getMyReceivedRequest();
                        } catch (e) {
                          print(e);
                        }

                        //await Future.delayed(Duration(seconds: 2));
                        SmartDialog.dismiss();
                      },
                child: const Text('Reject')),
          ),
          flex: 2),
      Expanded(
          child: Center(
            child: ElevatedButton(
                style: ElevatedButton.styleFrom(
                    primary: Colors.greenAccent),
                onPressed: data[4].toString() != '0'
                    ? null
                    : () async {
                        SmartDialog.showLoading();
                        try {
```

```
                        if (connectedWithMetamask) {
                          await model2.acceptRequest(data[0]);
                        } else {
                          await model.acceptRequest(data[0]);
                        }
                        await getMyReceivedRequest();
                      } catch (e) {
                        print(e);
                      }

                      //await Future.delayed(Duration(seconds: 2));
                      SmartDialog.dismiss();
                    },
                child: const Text('Accept')),
            ),
            flex: 2),
        ],
      ),
    );
  },
 );
}


Widget landGallery() {
 if (isLoading) {
  return const Expanded(child: Center(child: CircularProgressIndicator()));
 }

 if (LandGall.isEmpty) {
  return const Expanded(
      child: Center(
          child: Text(
    'No Lands Added yet',
    style: TextStyle(fontSize: 22, fontWeight: FontWeight.w500),
  )));
 }
 return Expanded(
  child: Center(
    child: SizedBox(
      width: isDesktop ? 900 : width,
      child: GridView.builder(
        padding: const EdgeInsets.all(10),
        scrollDirection: Axis.vertical,
        gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
            mainAxisExtent: 440,
            crossAxisCount: isDesktop ? 2 : 1,
            crossAxisSpacing: 20,
            mainAxisSpacing: 20),
        itemCount: LandGall.length,
        itemBuilder: (context, index) {
          return landWid2(
              LandGall[index][10],
              LandGall[index][1].toString(),
              LandGall[index][2].toString(),
              LandGall[index][3].toString(),
              LandGall[index][9] == userInfo[0],
              LandGall[index][8], () async {
            if (isUserVerified) {
              SmartDialog.showLoading();
              try {
                if (connectedWithMetamask) {
                  await model2.sendRequestToBuy(LandGall[index][0]);
                } else {
                  await model.sendRequestToBuy(LandGall[index][0]);
                }
                showToast("Request sent",
                    context: context, backgroundColor: Colors.green);
```

```
            } catch (e) {
              print(e);
              showToast("Something Went Wrong",
                  context: context, backgroundColor: Colors.red);
            }
            SmartDialog.dismiss();
          } else {
            showToast("You are not verified",
                context: context, backgroundColor: Colors.red);
          }
        }, () {
          List<String> allLatiLongi =
              LandGall[index][4].toString().split('|');

          LandInfo landinfo = LandInfo(
              LandGall[index][1].toString(),
              LandGall[index][2].toString(),
              LandGall[index][3].toString(),
              LandGall[index][5].toString(),
              LandGall[index][6].toString(),
              LandGall[index][7].toString(),
              LandGall[index][8],
              LandGall[index][9].toString(),
              LandGall[index][10]);
          Navigator.push(
              context,
              MaterialPageRoute(
                  builder: (context) => viewLandDetails(
                        allLatitude: allLatiLongi[0],
                        allLongitude: allLatiLongi[1],
                        landinfo: landinfo,
                      )));
        });
      },
    ),
  ),
 ),
);
}

Widget myLands() {
 if (isLoading) {
  return const Expanded(child: Center(child: CircularProgressIndicator()));
 }
 if (landInfo.isEmpty) {
  return const Expanded(
      child: Center(
          child: Text(
   'No Lands Added yet',
   style: TextStyle(fontSize: 22, fontWeight: FontWeight.w500),
  )));
 }
 return Expanded(
  child: Center(
   child: SizedBox(
    width: isDesktop ? 900 : width,
    child: GridView.builder(
     padding: const EdgeInsets.all(10),
     scrollDirection: Axis.vertical,
     gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
         mainAxisExtent: 440,
         crossAxisCount: isDesktop ? 2 : 1,
         crossAxisSpacing: 20,
         mainAxisSpacing: 20),
     itemCount: landInfo.length,
     itemBuilder: (context, index) {
      return landWid(
          landInfo[index][10],
```

```
            landInfo[index][1].toString(),
            landInfo[index][2].toString(),
            landInfo[index][3].toString(),
            landInfo[index][8],
            () =>
                confirmDialog('Are you sure to make it on sell?', context,
                    () async {
                  SmartDialog.showLoading();
                  if (connectedWithMetamask) {
                    await model2.makeForSell(landInfo[index][0]);
                  } else {
                    await model.makeForSell(landInfo[index][0]);
                  }
                  Navigator.pop(context);
                  await getLandInfo();
                  SmartDialog.dismiss();
                }));
        },
      ),
    ),
  ),
 );
}

Widget addLand() {
  return Center(
    widthFactor: isDesktop ? 2 : 1,
    child: Container(
      margin: const EdgeInsets.all(10),
      padding: const EdgeInsets.all(10),
      decoration: BoxDecoration(
          color: Colors.white,
          borderRadius: const BorderRadius.all(Radius.circular(10)),
          border: Border.all()),
      width: width,
      child: Form(
        key: _formKey,
        child: Column(
          // scrollDirection: Axis.vertical,
          // shrinkWrap: true,
          children: <Widget>[
            Padding(
              padding: const EdgeInsets.all(15),
              child: TextFormField(
                style: const TextStyle(
                  fontSize: 17,
                ),
                validator: (value) {
                  if (value == null || value.isEmpty) {
                    return 'Please enter some text';
                  }
                  return null;
                },
                onChanged: (val) {
                  area = val;
                },
                keyboardType: TextInputType.number,
                inputFormatters: <TextInputFormatter>[
                  FilteringTextInputFormatter.allow(RegExp(r'[0-9]'))
                ],
                decoration: const InputDecoration(
                 // isDense: true, // Added this
                  contentPadding: EdgeInsets.all(12),
                  //border: OutlineInputBorder(),
                  labelText: 'Area(SqFt)',
                  hintText: 'Enter Area in SqFt',
                ),
              ),
```

```
),
Padding(
  padding: const EdgeInsets.all(10),
  child: CompositedTransformTarget(
    link: _layerLink,
    child: TextFormField(
      validator: (value) {
        if (value == null || value.isEmpty) {
          return 'Please enter Land Address';
        }
        return null;
      },
      style: const TextStyle(
        fontSize: 17,
      ),
      controller: addressController,
      onChanged: (value) {
        if (value.isNotEmpty) {
          autocomplete(value);
          _overlayEntry.remove();
          _overlayEntry = _createOverlayEntry();
          Overlay.of(context)!.insert(_overlayEntry);
        } else {
          if (predictions.isNotEmpty && mounted) {
            setState(() {
              predictions = [];
            });
          }
        }
      },
      focusNode: _focusNode,
      //obscureText: true,
      decoration: const InputDecoration(
        //isDense: true, // Added this
        contentPadding: EdgeInsets.all(12),
        //border: OutlineInputBorder(),
        labelText: 'Address',
        hintText: 'Enter Land Address',
      ),
    ),
  ),
),
Padding(
  padding: const EdgeInsets.all(10),
  child: TextFormField(
    validator: (value) {
      if (value == null || value.isEmpty) {
        return 'Please enter Land Price';
      }
      return null;
    },
    //maxLength: 12,
    style: const TextStyle(
      fontSize: 17,
    ),
    keyboardType: TextInputType.number,
    inputFormatters: <TextInputFormatter>[
      FilteringTextInputFormatter.allow(RegExp(r'[0-9]'))
    ],
    onChanged: (val) {
      landPrice = val;
    },
    //obscureText: true,
    decoration: const InputDecoration(
      //isDense: true, // Added this
      contentPadding: EdgeInsets.all(12),
      //border: OutlineInputBorder(),
      labelText: 'Land Price',
```

```
      hintText: 'Enter Land Price',
    ),
  ),
),
Padding(
  padding: const EdgeInsets.all(10),
  child: TextFormField(
    validator: (value) {
      if (value == null || value.isEmpty) {
        return 'Please enter Property ID';
      }
      return null;
    },
    style: const TextStyle(
      fontSize: 17,
    ),
    //maxLength: 10,
    keyboardType: TextInputType.number,
    inputFormatters: <TextInputFormatter>[
      FilteringTextInputFormatter.allow(RegExp(r'[0-9]'))
    ],
    onChanged: (val) {
      propertyID = val;
    },
    //obscureText: true,
    decoration: const InputDecoration(
      //isDense: true, // Added this
      contentPadding: EdgeInsets.all(12),
      //border: OutlineInputBorder(),
      labelText: 'Property ID',
      hintText: 'Enter Property ID',
    ),
  ),
),
Padding(
  padding: const EdgeInsets.all(10),
  child: TextFormField(
    validator: (value) {
      if (value == null || value.isEmpty) {
        return 'Please enter some text';
      }
      return null;
    },
    onChanged: (val) {
      surveyNo = val;
    },
    style: const TextStyle(
      fontSize: 17,
    ),
    //obscureText: true,
    decoration: const InputDecoration(
      //isDense: true, // Added this
      contentPadding: EdgeInsets.all(12),
      //border: OutlineInputBorder(),
      labelText: 'Survey No.',
      hintText: 'Survey No.',
    ),
  ),
),
Padding(
  padding: const EdgeInsets.all(10),
  child: MaterialButton(
    color: Colors.blue,
    onPressed: () async {
      allLatiLongi = await Navigator.push(
          context,
          MaterialPageRoute(
              builder: (context) => const landOnMap()));
```

```
      if (allLatiLongi.isEmpty || allLatiLongi == "") {
        showToast("Please select area on map",
            context: context, backgroundColor: Colors.red);
      }
      //print(res);
    },
    child: const Text('Draw Land on Map'),
  ),
),
Padding(
  padding: const EdgeInsets.all(10),
  child: Row(
    children: [
      MaterialButton(
        color: Colors.grey,
        onPressed: pickDocument,
        child: const Text('Upload Document'),
      ),
      Text(docuName)
    ],
  ),
),
CustomButton(
    'Add',
    isLoading || !isUserVerified
        ? null
        : () async {
            if (_formKey.currentState!.validate() &&
                allLatiLongi.isNotEmpty &&
                allLatiLongi != "") {
              setState(() {
                isLoading = true;
              });
              try {
                SmartDialog.showLoading(
                    msg: "Uploading Document");
                bool isFileupload = await uploadDocument();
                SmartDialog.dismiss();
                if (isFileupload) {
                  if (connectedWithMetamask) {
                    await model2.addLand(
                        area,
                        addressController.text,
                        allLatiLongi,
                        landPrice,
                        propertyID,
                        surveyNo,
                        docUrl);
                  } else {
                    await model.addLand(
                        area,
                        addressController.text,
                        allLatiLongi,
                        landPrice,
                        propertyID,
                        surveyNo,
                        docUrl);
                  }
                  showToast("Land Successfully Added",
                      context: context,
                      backgroundColor: Colors.green);
                  isFilePicked = false;
                }
              } catch (e) {
                print(e);
                showToast("Something Went Wrong",
                    context: context,
                    backgroundColor: Colors.red);
```

```
                    }

                    setState(() {
                      isLoading = false;
                    });
                  }

                  //model.makePaymentTestFun();
                }),
          if (!isUserVerified)
            const Text('You are not verified',
                style: TextStyle(color: Colors.redAccent)),
          isLoading ? spinkitLoader : Container()
        ],
      ),
    ),
  ),
 );
}

Widget userProfile() {
 if (isLoading) {
  return const Expanded(child: Center(child: CircularProgressIndicator()));
 }
 isUserVerified = userInfo[8];
 return Expanded(
  child: Center(
   child: Container(
    width: width,
    margin: const EdgeInsets.all(10),
    padding: const EdgeInsets.all(10),
    decoration: BoxDecoration(
       color: Colors.white,
       borderRadius: const BorderRadius.all(Radius.circular(10)),
       border: Border.all()),
    child: Column(
     mainAxisAlignment: MainAxisAlignment.spaceEvenly,
     crossAxisAlignment: CrossAxisAlignment.start,
     children: [
      const Text(
       'My Profile',
       style: TextStyle(fontWeight: FontWeight.bold, fontSize: 20),
      ),
      userInfo[8]
          ? Row(
              children: const [
                Text(
                  'Verified',
                  style: TextStyle(
                      fontWeight: FontWeight.bold, color: Colors.green),
                ),
                Icon(
                  Icons.verified,
                  color: Colors.green,
                )
              ],
            )
          : const Text(
              'Not Yet Verified',
              style: TextStyle(
                  fontWeight: FontWeight.bold,
                  color: Colors.blueAccent),
            ),
      CustomTextFiled(userInfo[0].toString(), 'Wallet Address'),
      CustomTextFiled(userInfo[1].toString(), 'Name'),
      CustomTextFiled(userInfo[2].toString(), 'Age'),
      CustomTextFiled(userInfo[3].toString(), 'City'),
      CustomTextFiled(userInfo[4].toString(), 'Adhar Number'),
```

```
          CustomTextFiled(userInfo[5].toString(), 'Pan'),
          TextButton(
            onPressed: () {
              launchUrl(userInfo[6].toString());
            },
            child: const Text(
              '  View Document',
              style: TextStyle(color: Colors.blue),
            ),
          ),
          CustomTextFiled(userInfo[7].toString(), 'Mail'),
        ],
      ),
    ),
  ),
 );
}

Widget drawer2() {
 return Container(
   decoration: const BoxDecoration(
    boxShadow: [
     BoxShadow(blurRadius: 10, color: Colors.black26, spreadRadius: 2)
    ],
    color: Color(0xFF1A237E),
   ),
   width: 250,
   child: Column(
    mainAxisAlignment: MainAxisAlignment.center,
    crossAxisAlignment: CrossAxisAlignment.center,
    children: <Widget>[
     const SizedBox(
      width: 20,
     ),
     const Icon(
      Icons.person,
      size: 50,
     ),
     const SizedBox(
      width: 30,
     ),
     Text(name,
        style: const TextStyle(
          color: Colors.white70,
          fontSize: 18,
          fontWeight: FontWeight.bold)),
     const SizedBox(
      height: 80,
     ),
     Expanded(
      child: ListView.separated(
       separatorBuilder: (context, counter) {
        return const Divider(
         height: 2,
        );
       },
       itemCount: menuItems.length,
       itemBuilder: (BuildContext context, int index) {
        return MenuItemTile(
         title: menuItems[index].title,
         icon: menuItems[index].icon,
         isSelected: screen == index,
         onTap: () {
          if (index == 6) {
           Navigator.pop(context);
           // Navigator.push(
           //    context,
           //    MaterialPageRoute(
```

```
                //        builder: (context) => const home_page()));
                Navigator.of(context).pushNamed(
                  '/',
                );
              }
              if (index == 0) getProfileInfo();
              if (index == 2) getLandInfo();
              if (index == 3) getLandGallery();
              if (index == 4) getMyReceivedRequest();
              if (index == 5) getMySentRequest();
              setState(() {
                screen = index;
              });
            },
          );
        },
      ),
      ),
      const SizedBox(
        height: 20,
      )
    ],
  ),
 );
}

_paymentDialog(buyerAdd, sellAdd, amountINR, total, ethval, reqID) async {
 return showDialog<void>(
    context: context,
    barrierDismissible: false,
    builder: (BuildContext context) {
      return Dialog(
        backgroundColor: Colors.white,
        child: Container(
          margin: const EdgeInsets.all(10),
          height: 430.0,
          width: 320,
          child: Column(
            //mainAxisAlignment: MainAxisAlignment.spaceEvenly,
            children: [
              const Text(
                'Confirm Payment',
                style: TextStyle(fontSize: 30),
              ),
              const SizedBox(
                height: 10,
              ),
              Text(
                buyerAdd.toString(),
                style: const TextStyle(
                  color: Colors.black87,
                  fontSize: 13.0,
                ),
              ),
              const SizedBox(
                height: 10,
              ),
              const Icon(
                Icons.arrow_circle_down,
                size: 30,
              ),
              const SizedBox(
                height: 10,
              ),
              Text(
                sellAdd.toString(),
                style: const TextStyle(
                  color: Colors.black87,
```

```
            fontSize: 13.0,
          ),
        ),
        const SizedBox(
          height: 10,
        ),
        const Divider(),
        const SizedBox(
          height: 10,
        ),
        const Text(
          "Total Amount in ₹",
          style: TextStyle(fontSize: 20),
        ),
        Text(
          amountINR,
          style: const TextStyle(fontSize: 30),
        ),
        const SizedBox(
          height: 15,
        ),
        Text(
          '1 ETH = ' + ethval.toString() + '₹',
        ),
        const SizedBox(
          height: 15,
        ),
        const Text(
          "Total ETH:",
          style: TextStyle(fontSize: 20),
        ),
        Text(
          total.toString(),
          style: const TextStyle(fontSize: 30),
        ),
        const Spacer(),
        Row(
          mainAxisAlignment: MainAxisAlignment.spaceEvenly,
          children: [
            CustomButton3('Cancel', () {
              Navigator.of(context).pop();
            }, Colors.white),
            CustomButton3('Confirm', () async {
              SmartDialog.showLoading();
              try {
                if (connectedWithMetamask) {
                  await model2.makePayment(reqID, total);
                } else {
                  await model.makePayment(reqID, total);
                }
                await getMySentRequest();
                showToast("Payment Success",
                    context: context,
                    backgroundColor: Colors.green);
              } catch (e) {
                print(e);
                showToast("Something Went Wrong",
                    context: context, backgroundColor: Colors.red);
              }
              SmartDialog.dismiss();
              Navigator.of(context).pop();
            }, Colors.blue)
          ],
        )
      ],
    ),
  ));
});
```

III. BIBLIOGRAPHY

[1] Suganthe, R. C., N. Shanthi, R. S. Latha, K. Gowtham, S. Deepakkumar, and R. Elango. "Blockchain enabled Digitization of Land Registration" 2021 International Conference on Computer Communication and Informatics (ICCCI), pp. 1-5. IEEE, 2021

[2] Mohammed Moazzam Zahuruddin, Dr. Sangeeta Gupta, Shaik Waseem Akram, "Land Registration using Blockchain Technology", International Journal of Emerging Technologies and Innovative Research, ISSN:2349-5162, Vol.8, Issue 6, page no. b657-b667, June-2021

[3] Rakesh Kumar K V, Rithick Gokul A, V. Nirmal Kumar "Blockchain and Smart Contract for Land Registration using Ethereum Network" International Journal of Engineering Research & Technology (IJERT), paper ID - IJERTCONV10IS08005, (Volume 10 – Issue 08), July, 2022

[4] C.Roopa, Dr.R.C.Suganthe, Dr.N.Shanthi. "Blockchain Based Certificate Verification Using Ethereum And Smart Contract", "Journal of Critical Reviews",Vol 7, Issue 9, 2020, pp.330-336.

[5] P.Chinnasamy, P.Deepalakshmi, V. Praveena, K.Rajakumari, P.Hamsagayathri,(2019) "Blockchain Technology: A Step Towards Sustainable Development "International Journal of Innovative Technology and Exploring Engineering (IJITEE), Volume-9 Issue-2S2.

[6] A. Sahai and R. Pandey, "Smart Contract Definition for Land Registry in Blockchain," 2020 IEEE 9th International Conference on Communication Systems and Network Technologies (CSNT), Gwalior, India, 2020, pp. 230-235, doi: 10.1109/CSNT48778.2020.9115752.

[7] Vinay Thakur, M.N. Doja, Yogesh K. Dwivedi, Tanvir Ahmad, Ganesh Khadanga, Land records on Blockchain for implementation of Land Titling in India, International Journal of Information Management, Volume 52,2020. https://doi.org/10.1016/j.ijinfomgt.2019.04.013.

[8] N. Kshetri and J. Voas, "Blockchain in Developing Countries," in IT Professional, vol. 20, no. 2, pp. 11-14, Mar./Apr. 2018, doi: 10.1109/MITP.2018.021921645.

[9] Shuaib, Mohammed, et al. "Blockchain-based framework for secure and reliable land registry system." Telkomnika 18.5 (2020): 2560-2571.