Land registration system using Permissioned Blockchain

This Dissertation is Submitted in Fulfillment
of the Requirements for the Degree of

Bachelor of Science (B.Sc.)

in

Computer Science and Engineering (CSE)

by

Md Rokib Khan(C181053)
Ali Khatami (C191068)
Md Rafidul Islam(C191065)

TO
FACULTY OF SCIENCE AND ENGINEERING
INTERNATIONAL ISLAMIC UNIVERSITY CHITTAGONG
Autumn 22

DECLARATION

We confirm the following assertions in relation to our project:

1. We have accomplished the project successfully as a component of our undergraduate degree programme at International Islamic University Chittagong.
2. The project work does not contain any previously published or third-party content without proper citation.
3. The project has not been previously presented for any other degree or diploma at any other university or institution.
4. We have duly recognised all significant sources of contribution in the project.

Student's Full Name and Metric ID:

Md Rokib Khan **(C181053)**
Ali Khatami **(C191068)**
Md Rafidul Islam **(C191065)**

SUPERVISOR'S DECLARATION

I formally declare that I have reviewed this project and affirm its satisfactory quality and scope, warranting the award of the Bachelor of Science degree in Computer Science and Engineering.

ABDULLAHIL KAFI
Assistant Professor
Department of Computer Science and Engineering
International Islamic University Chittagong

DEDICATION

This project report is dedicated to ourselves, our supervisor, and our family. The teamwork was satisfactory, and the unwavering support from our family was truly remarkable. We express gratitude to our dedicated and exceptionally hardworking supervisor, who has been a consistent pillar of support throughout this period. The document also recognises the contributions made.

ACKNOWLEDGMENT

First and foremost, we express gratitude to the Almighty Allah for His mercy, which enabled us to complete our thesis despite numerous obstacles. Secondly, we want to convey our appreciation to our supervisor, Abdullahil Kafi, for providing continuous guidance and support since the inception of our research.

ETHICAL STATEMENT

We affirm that our project was completed without resorting to any unethical practices. The data utilised for research is original, and we meticulously verified all citations. The three authors of this work take full responsibility for any potential violations of project rules.

ABSTRACT

This project aims to create a decentralised land registration system for Bangladesh. The solution uses the Raft consensus algorithm, which helps to make the network distributed by allowing each organisation to have its own peers. Hyperledger Fabric, a well-known permissioned blockchain technology, is used to store and secure important registration data and land details. The Raft ordering service, along with the permissioned nature of Hyperledger Fabric, ensures that citizens' data cannot be changed. To address worries about transaction fees, we've taken steps to prevent regular people from being burdened, making sure everyone can use the system. We've also planned for the system's growth as the population increases. We're working to clear up legal uncertainties related to blockchain in countries like Bangladesh, where people might have concerns about using cryptocurrency. In tests, users were able to log in, submit new land registration applications, and securely store details on the ledger. Ministry of Land administrators can review and approve applications, and Land Revenue Office administrators can smoothly register approved lands. If someone owns land registered on the blockchain, they can transfer or sell ownership. We use Hyperledger Fabric, Express.js, Node.js, Go, and MongoDB to ensure data security, access control, and a strong framework. In short, this project makes land registration easier and tackles challenges in decentralised systems, providing a safe, transparent, and efficient way to manage land records in Bangladesh.

Keywords: Blockchain, Hyperledger Fabric, Land Registration, Consensus Algorithm, Membership Service Providers, Chaincode.

Table of Contents

DECLARATION ii

SUPERVISOR DECLARATION iii

Dedication iv

Acknowledgment v

Ethical Statement vi

Abstract vii

Table of Contents viii

List of Figures xi

List of Tables xiv

Abbreviation xv

1 Introduction 1
- 1.1 Overview . . . 1
- 1.2 Motivation . . . 1
- 1.3 Problem Statement . . . 2
- 1.4 Objective of the project . . . 2
- 1.5 Organization of the Thesis . . . 2

2 Literature Review 3
- 2.1 Introduction . . . 3
- 2.2 Related Work . . . 3
- 2.3 Definition . . . 5
- 2.4 Existing Land Registration System . . . 10
  - 2.4.1 Negotiation between the buyer and seller: . . . 10
  - 2.4.2 Seller provides buyer necessary documents: . . . 10
  - 2.4.3 Verification of Documents: . . . 12
  - 2.4.4 Applying for Mutation: . . . 12
  - 2.4.5 Pay tax and stamp duty: . . . 12
  - 2.4.6 Payment: . . . 13
  - 2.4.7 Apply for registration: . . . 13

2.4.8 Register the change in ownership: . . . 13

3 Methodology **14**
3.1 Introduction . . . 14
3.2 Tools used: . . . 14
3.2.1 Hashing Algorithm: . . . 15
3.2.2 Consensus Algorithm: . . . 15
3.2.3 Certificate: . . . 16
3.2.4 Certificate Authority Database: . . . 16
3.3 Network Deployment Steps: . . . 17
3.4 Land Registration Flow . . . 18
3.5 Step by step process of our Hyperledger Fabric based land registration system: . . . 18
3.5.1 Client side: . . . 18
3.5.2 Admin(Ministry of Land) side: . . . 20
3.5.3 Admin(Land Revenue Office) side: . . . 21
3.6 Necessary function's Chaincode . . . 22
3.6.1 CreateApplication() . . . 22
3.6.2 ResubmitApplication . . . 24
3.6.3 ReadApplicationByID() . . . 26
3.6.4 GetAllUserApplication() . . . 27
3.6.5 ReadDeedByID() . . . 28
3.6.6 GetAllCertificates() . . . 29
3.6.7 TransferOwnerShip() . . . 30
3.6.8 GetAllAdminApplication . . . 31
3.6.9 ApproveLandApplication() . . . 32
3.6.10 RegisterLandApplication() . . . 34
3.6.11 GenerateDeed() . . . 36
3.7 User Interface of our project . . . 37
3.8 Entity Relationship Diagram . . . 44
3.8.1 Logical Schema . . . 45
3.9 Use case Diagram . . . 46
3.10 Sequence Diagram . . . 49
3.10.1 Client . . . 49
3.10.2 Admin(Ministry of Land) . . . 50
3.10.3 Admin(Land Revenue Office) . . . 51

4 Results and Discussions **52**
4.1 Introduction . . . 52
4.2 API Test: . . . 52
4.3 Tamperproof test: . . . 57
4.4 Response time test: . . . 58
4.5 Discussion: . . . 59

5 Conclusion **60**
5.1 Summary of the findings . . . 60
5.2 Limitation and Future Work: . . . 60
5.3 Contribution . . . 61

References **62**

List of Figures

2.1 Existing Land Registration process of Bangladesh[9] 10

3.1 Hashing algorithm 15

3.2 Orderer Consensus 15

3.3 Channel Consensus 16

3.4 Network Architecture 17

3.5 Chaincode of CreateApplication function 22

3.6 Chaincode of ResubmitApplication function 24

3.7 Chaincode of ReadApplication function 26

3.8 Chaincode of GetAllUserApplication() function 27

3.9 Chaincode of ReadDeedByID function 28

3.10 Chaincode of GetAllCertificate function 29

3.11 Chaincode of TransferOwnership function 30

3.12 Chaincode of GetAllAdminApplication function 31

3.13 Chaincode of ApproveLandApplication function by Ministry of Land 32

3.14 Chaincode of RegisterLandApplication function by Land Revenue Office 34

3.15 Chaincode of GenerateDeed function 36

3.16 Login page 37

3.17 Land Registration page 37

3.18 Dashboard page 39

3.19 Details of single application page 39

3.20 Details of single application page 40

3.21 Generated Certificate 41

3.22 Transfer Ownership page 42

3.23 Interface page for Admin of Ministry of Land 42

3.24 Interface page of admin of Land Revenue Office 43

3.25 Entity Relationship Diagram 44

3.26 Use case diagram of Client(primary actor) and Admin of Ministry of Land(secondary actor) 47

3.27 Use case diagram of Client(primary actor) and Admin of Land Revenue Office(secondary actor) 48

3.28 Sequence Diagram[39] for Client side 49

3.29 Sequence Diagram for Admin(Ministry of Land) 50

3.30 Sequence Diagram for Admin(Land Revenue Office) 51

4.1 API test of Login 52

4.2 API test of Profile 53

4.3 API test of Update Profile 53

4.4 API test of RegistrationApplication 54

4.5 API test of User Dashboard 54

4.6 API test of ReadSingleApplication 55

4.7 API test of Land Application Verification 55

4.8 API test of Land Application after Verification 56

4.9 API test of DEED of Registered Land 56

4.10 API test of TransferLandOwnership 57

4.11 Test of Tamperproof 57

4.12 Response time of Land Registration by user 58

4.13 Response time of User Dashboard 59

List of Tables

2.1 Related Work 4

2.2 Permissioned vs Permissionless 9

3.1 Tools used for our project work 14

3.2 Input of CreateApplication function 23

3.3 Inputs of ResubmitApplication function 25

3.4 Inputs of ReadApplication function 26

3.5 Inputs of GetAllUserApplication function 27

3.6 Inputs of TransferOwnership function 30

3.7 Inputs of GetAllAdminApplication function 31

3.8 Inputs of ApproveLandApplication function 33

3.9 Inputs of RegisterLandApplication function 35

ABBREVIATION

The following list provides descriptions of various symbols and abbreviations that will be utilized in the subsequent sections of the document.

MSP : Membership Service Provider
CA : Certificate Authority
MOL : Ministry of Land
LRO : Land Revenue Office
TLS : Transport Layer Security
API : Application Programming Interface

Chapter 1

Introduction

1.1 Overview

In our effort to create a better land registration system in Bangladesh, we're setting up a decentralised solution using the Raft[1] algorithm and Hyperledger Fabric[2] technology for safety and efficiency. We want everyone using our system to be treated fairly. Despite some legal questions about using blockchain[3] in Bangladesh, our main goal is to build a safe and effective system that secures land records. To make it happen, we'll build the user application using the Express.js[4] framework of Node.js[5]. For logging in, we'll use MongoDB. This way, things will be easy for everyone using the system.

1.2 Motivation

Regular ways of registering land have problems like data being at risk, people paying fees for transactions, struggling to handle more people, not being sure about the law with blockchain, and everything being too centralized. Our project uses a special blockchain for land registration that needs permission. We want to fix these issues by making sure data can't be changed, reducing fees for people, handling more users, clearing up legal questions, and using a decentralised system. Our goal is to change how land registration works, making it more trustworthy, efficient, and fair for everyone.

1.3 Problem Statement

Some bad people work with helpers and dishonest officials to make fake papers and take control of land by force. This leaves the real owners in long fights in court to get their land back. These land sharks use bribes to change papers, selling the land to people they don't know, causing big legal problems between the true owner and the new buyer with fake documents. The original landowners might make many agreements, taking money from different buyers using a special paper called a pledge deed (Binah Dalil), leading to common court problems. Fixing these issues usually means making friendly agreements or going to the police. But dealing with the police in the legal system brings more money and time problems, lasting for months or even years.

1.4 Objective of the project

The main objective of this project are:

- To create a system that stops bad people from making fake papers to take over land, make a secure and easy way to check who owns land, and keep clear records of land deals to stop dishonest officials from cheating.
- To implement the Raft ordering service,which is better than Kafka and other ordering services
- To ensure that citizens' data is completely immutable
- To prevent common individuals from being burdened by transaction fees
- To prevent a gradual increase in the load on the blockchain system and transactions due to population growth
- Avoid the use of cryptocurrency to prevent violation of laws regarding cryptocurrency in many countries like Bangladesh

1.5 Organization of the Thesis

In Chapter 2 of our report we discussed the the literature review, and at chapter 3 we focused on methodology we used , and at chapter 4 we focused on Result and discussion and at last chapter 5 we have discussed the conclusion and our future work.

Chapter 2

Literature Review

2.1 Introduction

Blockchain[3] has changed many things in different areas, and now it could do the same for keeping track of who owns land. People in charge of land want things to be more clear and work better, and using blockchain seems like a good idea.This section discusses existing work on land registration systems using blockchain and the definition of some terms related to it

2.2 Related Work

We have shown here existing work on blockchain on land registration systems in a table where each column describes the names of authors, methodology, results, and limitations.

TABLE 2.1: Related Work

Author	Methodology	Result	Limitation
Thakur, V., Doja, M. N., Dwivedi, Y. K., Ahmad, T., Khadanga, G. (2020) A	The proposal suggests leveraging blockchain, particularly smart[6] contracts, to document property transactions encompassing sales, inheritance, court orders, and land acquisition.	Successfully addressed the issue of implementing both public and ., private blockchain on land registration systems in India and successfully solved it.	The load on the Blockchain system will gradually increase and the transactions will increase because of population growth
Fernando, D., Ranasinghe, N. (2019)	Implemented on Hyperledger Fabric v1.2[7], we assessed performance on AWS t2.large with 2 vCPUs and 8 GB of memory, considering transaction density and node failures.	AM2 unifies channels, employing a single chaincode and island-wide land ledger, outperforming AM1's separate channels.	Using Kafka for ordering service doesn't decentralise nodes when operated by different organisations, as all connect to a single-controlled Kafka cluster.
Lazuashvili, N., Norta, A., Draheim, D. (2019)	They have used Bitcoin[8] blockchain technology.	Project achieved heightened safety, security for citizen data, and improved transparency, ensuring traceability of information	Citizen data can be altered before being stored on immutable blockchain storage.
Alam, K. M., Rahman, J. A., Tasnim, A., Akther, A. (2022)	Ethereum deployed smart land title contracts[9]. using Solidity. Ganache-cli for local testing, Remix as an online IDE, and Metamask for browser blockchain interaction.	Efficiently manage land ownership, ensure traceability, minimize travel, time, costs	The transaction fee varies with the ETH value, which will not be acceptable for people.
Zhang, L., Ci, L., Wu, Y., Wiwatan apataphee, B. (2023).	They built their system on the Ethereum[] blockchain[10] network.	Cuts real estate authentication expenses, speeds up transactions, eliminates third-party verification	Uncertain blockchain legality impacts practical use for businesses and organizations.

2.3 Definition

Blockchain:A blockchain[2] is a collaborative and unchangeable record that simplifies the recording of transactions and monitoring of assets within a business network. Assets, whether tangible (such as real estate, vehicles, money, or land) or intangible (including intellectual property, patents, copyrights, and brand assets), can be documented and exchanged on a blockchain network. This process minimizes risks and lowers expenses for all participants involved.

Hyperledger: Hyperledger[2] is a set of open-source initiatives designed to facilitate the advancement of blockchain-driven distributed ledgers. Its objective is to establish essential frameworks, standards, tools, and libraries for constructing blockchains and associated applications. The Hyperledger Foundation, responsible for supporting, maintaining, and hosting these projects, was initiated by the Linux Foundation in 2016. Since its inception, Hyperledger has received contributions from notable organisations, including IBM, Intel, Samsung, Microsoft, Visa, American Express, and various blockchain startups.

Hyperledger Fabric: Hyperledger Fabric[2] stands out as a prominent project within the Hyperledger ecosystem. Functioning as a permissioned blockchain infrastructure, it is instrumental in constructing various blockchain-driven products, software, and applications. Developed collaboratively by IBM and Digital Asset, Hyperledger Fabric boasts a modular architecture that delineates roles among nodes, facilitates smart contract execution, and offers configurable consensus services. Noteworthy features include the incorporation of smart contracts and adaptable consensus protocols specific to Hyperledger Fabric. Notably, Fabric distinguishes itself by supporting smart contracts written in versatile programming languages such as Java, Go, and Node.js, departing from the conventional use of constrained domain-specific languages (DSL).

Ordering service: Within a Hyperledger Fabric network[11], a group of nodes constitutes an "ordering service," responsible for organizing transactions into blocks. Peers subsequently validate and commit these blocks to their ledgers. This distinguishes Fabric from other decentralized blockchains like Ethereum and Bitcoin, where any node can perform the ordering process.

Raft ordering service:

Raft serves as [1] a crash fault-tolerant (CFT) ordering service, adhering to a "leader and follower" model. In this setup, a leader node is elected per channel, and its decisions are copied by the follower nodes. Setting up and handling Raft[12] ordering services is anticipated to be more straightforward compared to those based on Kafka. Additionally, the design facilitates various organisations contributing nodes to a distributed ordering service.

Channel

A Hyperledger Fabric channel[13] functions as an exclusive communication "subnet" connecting two or more designated network members, facilitating private and confidential transactions. It is characterized by its members (organizations), anchor peers for each member, the shared ledger, chaincode application(s), and the ordering service node(s). Every transaction within the network occurs on a specific channel, necessitating authentication and authorization for each participating party on that channel.

Peer

A pivotal component within a Hyperledger Fabric blockchain network is the set of peer nodes, commonly referred to as peers[14]. Peers play a crucial role in managing ledgers, smart contracts, and, starting with Hyperledger Fabric v2.4, transaction proposals and endorsements through the Fabric Gateway service. It's essential to note that a ledger serves as an immutable record for all transactions initiated by smart contracts (chaincode in Hyperledger Fabric terminology) and endorsed by the necessary organizations. Peers, being responsible for hosting chaincodes and the ledger, actively engage in handling transaction proposals and responses, ensuring the ledger stays updated and consistent with validated and endorsed transactions. These attributes make peers a foundational element in comprehending the dynamics of a fabric network.

Organization:

An organisation[15], also referred to as "members," receives an invitation from a blockchain network provider to join the network. Joining the network involves incorporating its Membership Service Provider (MSP), which outlines how other network members can authenticate signatures, ensuring they originate from a valid identity issued by that specific organization. The access rights of identities within an MSP are determined by agreed-upon policies established during the organization's integration into the network. Organisations, ranging from multinational corporations to individuals, can participate. The transaction endpoint for an organisation is a peer, and a consortium is formed by a collection of organizations. It's important to note that while all network organisations are members, not every organisation is necessarily part of a consortium.

Certificate Authority

The Hyperledger Fabric CA[16] serves as a Certificate Authority (CA) specifically designed for Hyperledger Fabric. Its functionalities encompass the registration of identities, either independently or by connecting to LDAP for user registry purposes. Additionally, it facilitates the issuance of enrollment certificates (ECerts) and manages processes like certificate renewal and revocation. The Hyperledger Fabric CA[17] comprises both a server and a client component, working in tandem to provide robust and secure certificate-related services within the Hyperledger Fabric blockchain framework.

Postman

Postman[18] serves as an API platform designed for constructing and utilizing APIs. It simplifies[19] every stage of the API lifecycle, promoting efficient collaboration to facilitate the quicker creation of improved APIs.

Express.js: Express[4] is a versatile web[20] application framework for Node.js, offering a minimalistic yet powerful feature set for the development of web and mobile applications.

Node.js: Node.js[21] is a JavaScript runtime that operates asynchronously and focuses on creating scalable network applications. It shares design similarities with systems like Ruby's Event Machine and Python's Twisted but takes the event model a step further. Instead of treating the event loop as a library, Node.js[5] presents it as an intrinsic part of the runtime structure.

MongoDB: MongoDB[22] is a scalable and flexible document database that combines the desired scalability and flexibility with essential querying and indexing capabilities. The document model offered by MongoDB[23] is straightforward for developers to grasp and apply, ensuring simplicity while delivering all the necessary features to meet intricate requirements across various scales. With support for over 10 languages through provided drivers and numerous additional drivers built by the community, MongoDB caters to a wide range of programming needs.

Go Programming Language: Go[24] is a versatile programming language crafted with a focus on systems programming. It[25] features strong typing, garbage collection, and explicit support for concurrent programming. The construction of programmes involves assembling packages, which efficiently handle dependencies. The language boasts a concise syntax that is easily parsed, facilitating straightforward analysis through automated tools like integrated development environments.

React.js: React JS[26] is a JavaScript library used to construct user interfaces, offering a seamless way to develop interactive UIs. It simplifies the process of creating different views for each state in our application. With React[27], the system efficiently updates and displays the

appropriate components only when our data undergoes changes. The use of declarative views enhances code predictability and simplifies debugging, as it allows us to design straightforward views for various application states.

Ngrok: Ngrok[28] serves as a globally distributed reverse proxy, ensuring the security, protection, and acceleration of your applications and network services, regardless of their location. Consider Ngrok as the primary entry point for your applications. It is versatile and works seamlessly in any environment, delivering traffic to services without requiring adjustments to your network setup. Whether your app is hosted on AWS, Azure, Heroku, an on-premise Kubernetes cluster, a Raspberry Pi, or your laptop, ngrok provides a consistent experience across all platforms.

Firebase

Firebase,[29] a platform embraced by Google and relied upon by numerous global businesses, serves as a facilitator for the creation and enhancement of beloved apps and games. It[30] streamlines and expands app development by eliminating the need for infrastructure management, accelerating the process with minimal effort. Enhancing app quality is achieved efficiently, reducing the time and resources needed for testing, triaging, and troubleshooting. The platform allows for the careful rollout of features, monitoring adoption, and early identification and resolution of stability and performance issues. Firebase excels at boosting user engagement through comprehensive analytics, A/B testing, and messaging campaigns. By gaining insights into user behaviour, supporting them better, and conducting experiments, developers can tailor the app experience for different user segments.

Docker: Docker[31] serves as an open platform designed for the development, shipping, and execution of applications. It[32] empowers users to detach applications from infrastructure, facilitating swift software delivery. Docker enables the unified management of both applications and infrastructure, aligning their handling methodologies. Leveraging Docker's approaches for code shipping, testing, and deployment results in a substantial reduction in the time lag between code creation and its implementation in a production environment.

Docker Container

A container[33] serves as a standardised software unit, bundling code and dependencies to ensure the reliable and efficient execution of applications across diverse computing environments. In the Docker context, a container[34] image is a self-contained, lightweight package encompassing everything necessary for an application to run seamlessly: code, runtime, system tools, libraries, and configurations. When executed on the Docker Engine, these images transform into containers at runtime. Applicable to both Linux and Windows applications, containerised software exhibits consistent behaviour, offering uniformity irrespective of the underlying infrastructure.

Permissioned blockchain vs Permissionless blockchain[35]

TABLE 2.2: Permissioned vs Permissionless

Permission blockchain	Permissionless blockchain
Access is denied; approval is needed to connect to the network and engage in consensus.	Open; No authorization is needed to join the network and contribute to consensus.
Transitioning from partially decentralised to entirely centralised, with a controlling entity serving as the access regulator.	Completely decentralised, without any access control mechanisms.
Transactions remain confidential.	Transactions are visible and open for examination.
Swift and efficient performance.	Delayed transaction velocity
Transactions cost low	Transaction cost high
The governing body instills a measure of confidence in the system.	Trust is not required; the mathematics serves as evidence.

2.4 Existing Land Registration System

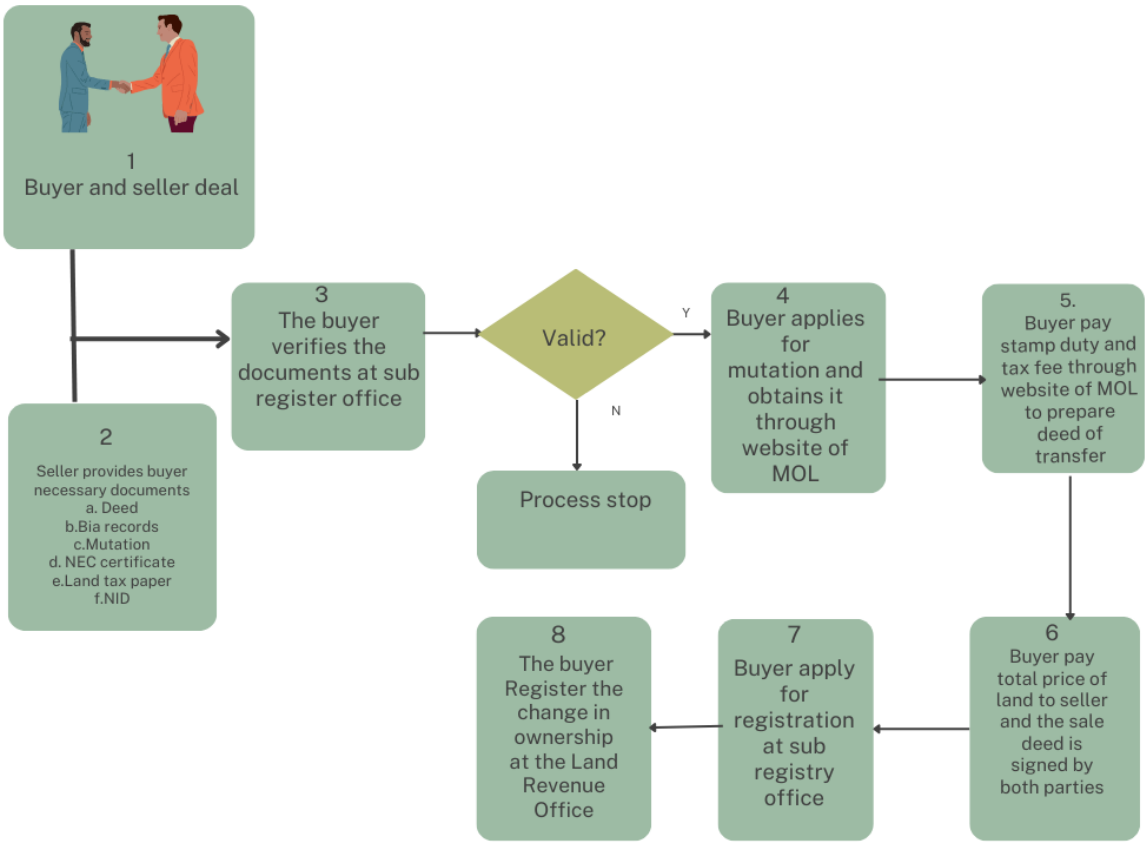

Fig. 2.1. Existing Land Registration process of Bangladesh[9]

2.4.1 Negotiation between the buyer and seller:

The initial stage involves the initiation of a transaction between the buyer(s) and seller(s)[9]. During this phase, agreements, including the price, are established. The seller is required to furnish original documents to the buyer for verification. Simultaneously, the buyer informs the land office and various government entities overseeing pertinent records associated with the specific land parcel.

2.4.2 Seller provides buyer necessary documents:

Seller provides buyer some necessary documents which include the Deed, Bia Records, Khatiyan and Non-Encumbrance Certificate. Additionally, buyers should acquire essential documents like the land tax paper and the National Identification Number (NID) of the landowner. If any of these documents are found to be unlawful, it significantly increases the risk of fraud or

harassment. Consequently, it is advisable to halt the process at that point, initiating further steps with the government to address and resolve the identified issues.

Deed

The fundamental legal document for transferring the Record of Rights (ROR) of any land in Bangladesh is the ownership document known as the deed. To be considered valid, a deed in the seller's name must explicitly state its nature, include a description of the property's surroundings, bear the valid signatures of a specified number of witnesses, and be transferred to the current owner. When purchasing, it is crucial for a new buyer to verify the document's authenticity with legal assistance. This involves a visit to the local sub-registry office, where an original copy of the existing deed is archived. Additionally, it is vital to ascertain if the property has been inherited. In such instances, the existence of a Bantan Nama (Partition Deed) is essential. This document outlines the distribution plan among the predecessors of the original owner and identifies any potential claims to the property.

Bia Record

The land administration distinguishes between ownership records and revenue records. In each upazila (sub-district), there are 11 administrative offices, and Bangladesh is geographically divided into 64 districts, with land registration services available in 61 of them, excluding the three hilly districts. The Upazila is further subdivided into smaller plots known as Mouza, each assigned a unique Mouza number, also referred to as a Jurisdiction List Number. Before proceeding with land registration, it is essential to examine a crucial set of documents known as Bia Records. These records encompass the complete transaction history of a property, generating a new record each time the land changes ownership.

Khatiyan Khatiyan[36], originating from the Persian language, is a document utilised for land identification. It is created through a survey to establish possession, ownership, and assess the land development tax. Commonly referred to as the Record of Rights, Sottolipi, or porcha, it serves as a comprehensive record of rights but does not constitute a deed of ownership.

Khatiyan can be classified into two types
A. Survey Khatiyan B. Mutation Khatiyan

A. Survey Khatiyan The creation of Khatiyan involves a land survey conducted by the Land Record and Survey Department located in Tejgoan[36]. They have their own printing press for Khatiyan documents, which are categorised based on the survey type, such as CS Khatiyan, RS Khatiyan, SA Khatiyan, and so on.

Types of Survey Khatiyan

A.1 CS Khatiyan: Formulated under the Bengal Tenancy Act 1885, recognised as Cadastral Survey, initiated in 1888 from Ramu in Cox's Bazar Upazila, concluding in 1940.

A.2 RS Khatiyan: Conducted 50 years after the CS survey, termed the Revisional Survey, resulting in RS Khatiyan. Its objective is to update land quantity, owner, and possessor details, and it holds greater authenticity compared to CS Khatiyan.

A.3 SA Khatiyan: Developed under the State Acquisition and Tenancy Act 1950, not a field-based survey. This khatiyan relies on information provided by zamindars or landlords. Also called PS Khatiyan or Pakistan Survey Khatiyan, it lacks authenticity.

A.4 BS Khatiyan: More reliable than other khatiyans, originating from the ongoing Bangladesh Survey that commenced in 1970. This survey is known as the Bangladesh Survey, and the resulting khatiyan is termed BS Khatiyan or Bangladesh Survey Khatiyan.

B. Mutation Khatiyan:

Typically, Khatyan creation involves the use of jareep. However, Jareep events are not constant. Property transfers may occur between two jareeps, necessitating an update in ownership details on the Khatiyan. The Khatiyan modification conducted through mutation proceedings is termed mutation Khatiyan, a process carried out by the Assistant Commissioner Land Office.

Non encumbrance certificate This certificate guarantees that the property is clear of legal obligations or financial issues. Often, individuals secure bank loans to acquire land, turning it into an asset backed by a mortgage. It is crucial to examine this certificate to prevent potential conflicts arising from mortgage-related issues in the future.

2.4.3 Verification of Documents:

The buyer verifies the documents at sub register office

2.4.4 Applying for Mutation:

The buyer applies for a mutation of property and obtains it through this website of Ministry of Land[36]

2.4.5 Pay tax and stamp duty:

The buyer pay stamp duty and necessary taxes, fees, and VAT through website[37] of Ministry of Land to prepare deed of transfer

2.4.6 Payment:

The purchaser has the option to either make the entire payment for the land before the registration period or enter into an agreement to partially pay and settle the remaining amount within the subsequent 6–12 months. During this period, the buyer is required to fulfil the total payment, after which the seller commences the registration process. In the event of either party violating the agreement, law enforcement intervention may be necessary, or disputes can be resolved through mutual agreement.

2.4.7 Apply for registration:

The buyer Apply for registration at the sub-registry office

2.4.8 Register the change in ownership:

The buyer Register the change in ownership at the Land Revenue Office

Chapter 3

Methodology

3.1 Introduction

In this section we have discussed the tools we used ,algorithms, Fabric deployment steps, the step by step details of our whole process, necessary chaincode of functions we used,user interface page and the necessary uml diagram.

3.2 Tools used:

The tools used are:

Table 3.1: Tools used for our project work

Technology	**Components**
Backend Server:	Node.js (Express)
Fabric API:	Node.js
Chaincode:	Go
Authentication Database:	MongoDB
Data storage Databse:	LevelDB
API Testing:	Postman
Testing with frontend:	ngrok
Consensus Algorithm:	Raft
Certificate format:	X.509
Hashing Algorithm:	SHA-256

```
################################################################################
# BCCSP (BlockChain Crypto Service Provider) section is used to select which
# crypto library implementation to use
################################################################################
bccsp:
    default: SW
    sw:
        hash: SHA2
        security: 256
        filekeystore:
            # The directory used for the software file-based keystore
            keystore: msp/keystore

################################################################################
```

Fig. 3.1. Hashing algorithm

3.2.1 Hashing Algorithm:

The algorithm used for computing the hash values encoded into the blocks of the blockchain. In particular, this affects the data hash, and the previous block hash fields of the block. The hashing algorithm used here is SHA-256.

3.2.2 Consensus Algorithm:

```
# SampleDevModeEtcdRaft defines a configuration that differs from the
# SampleDevModeSolo one only in that it uses the etcd/raft-based orderer.
SampleDevModeEtcdRaft:
    <<: *ChannelDefaults
    Orderer:
        <<: *OrdererDefaults
        OrdererType: etcdraft
        Organizations:
            - <<: *SampleOrg
              Policies:
                  <<: *SampleOrgPolicies
                  Admins:
                      Type: Signature
                      Rule: "OR('SampleOrg.member')"
```

Fig. 3.2. Orderer Consensus

Transactions must be written to the ledger in the order in which they occur, even though they might be between different sets of participants within the network. For this to happen, the

```
# SampleAppChannelEtcdRaft defines an application channel configuration
# that uses the etcd/raft-based orderer.
SampleAppChannelEtcdRaft:
    <<: *ChannelDefaults
    Orderer:
        <<: *OrdererDefaults
        OrdererType: etcdraft
        Organizations:
            - <<: *SampleOrg
              Policies:
                  <<: *SampleOrgPolicies
                  Admins:
                      Type: Signature
                      Rule: "OR('SampleOrg.member')"
```

Fig. 3.3. Channel Consensus

order of transactions must be established and a method for rejecting bad transactions that have been inserted into the ledger in error (or maliciously) must be put into place.

There are many ways to achieve it, each with different trade-offs. Hyperledger Fabric has been designed to allow network starters to choose a consensus mechanism that best represents the relationships that exist between participants. The default mechanisms previously used were SOLO, and KAFKA. But the latest version of Hyperledger Fabric uses RAFT as the default consensus mechanism which is more efficient than the previous ones.

3.2.3 Certificate:

A digital certificate is a document that holds a set of attributes relating to the holder of the certificate. The most common type of certificate is the one compliant with the X.509 standard, which allows the encoding of a party's identifying details in its structure.

When a certificate is signed by a trusted certificate authority, someone holding that certificate can use the public key it contains to establish secure communications with another party or validate documents digitally signed by the corresponding private key.

In Hyperledger Fabric Certificate Authorities uses X.509 standard certificate to give unique identity to clients.

3.2.4 Certificate Authority Database:

The default database for storing identity or X.509 certificates of clients is sqlite3

3.3 Network Deployment Steps:

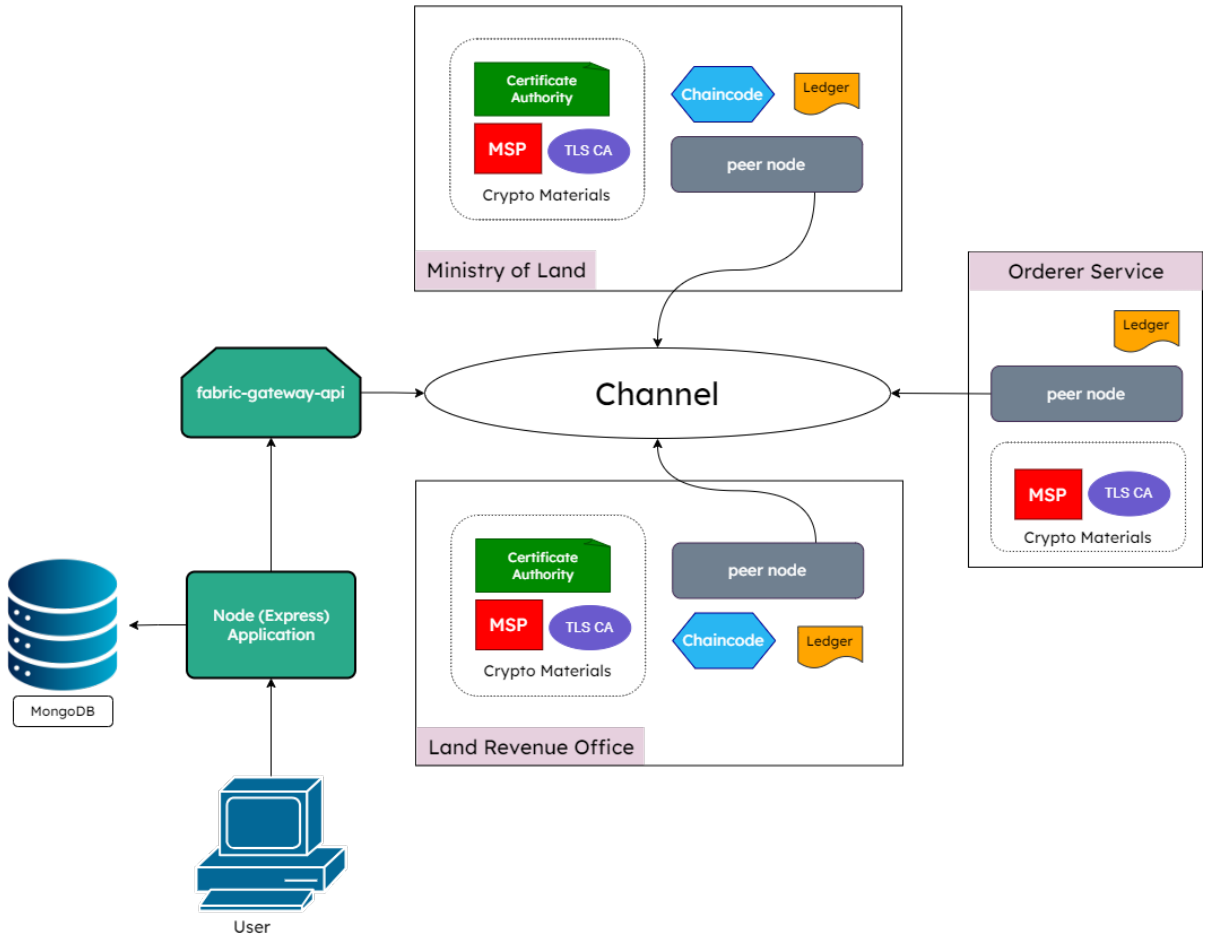

Fig. 3.4. Network Architecture

1. Fabric Certificate Authority Client (CA client)

The deployment sequence involves starting with the Fabric Certificate Authority Client (CA client), responsible for creating TLS certificates and Membership Service Providers (MSP) for every peer, orderer organisation, and any associated nodes. These credentials are organised into distinct folders for each entity.TLS ensures secure communication and facilitates the handshake process between different organizations.The MSP folder contains certificates, public-private keys, and signed certificates for each organization. When users from any organisation register and enrol via their CA server, they must interact through the CA

2. The Fabric Certificate Authority Server (CA server): It initiates by creating a self-signed root certificate, endorsed by TLS CA, serving as the shared public key for all interacting clients. It generates the private key for the CA server, storing it in the MSP folder, and establishes the CA server administrator for user registration.

3. Orderer: The Orderer's primary role is to sequence transactions into blocks, subsequently validated and committed by peers to their ledgers. Peers endorse client transactions, send them to the orderer, which organizes them into transaction blocks distributed to all peer nodes in the channel.

4. The Ministry of Land and Land Revenue Office representing peer organizations, hosts the ledger and client application.

Transaction requests from clients are sent to these organizations, where they undergo endorsement and processing.

3.4 Land Registration Flow

Every transaction request initiated by the user follows this path:

The user (client, Ministry of Land and Land Revenue Office) submits a transaction request, directed to the Node.js API, which verifies the user's login status. The transaction is then transmitted to Fabric-API, which forwards it to the relevant Ministry of Land (peer) (LRO peer for LRO admin). The peer node at the Ministry of Land endorses the transaction, appends a signed certificate, and includes the TLS CA from its local directory. This endorsed package is sent to the orderer, which validates the transaction using the provided certificate. After confirming validity, the orderer organises the transaction into a block and sends it back to the peer nodes. The peer nodes add the new block to the ledger, completing the transaction process.

3.5 Step by step process of our Hyperledger Fabric based land registration system:

3.5.1 Client side:

Registration:

1. User Input: Name, National ID (NID), Email, Phone, Address, Password

- **Validation:** Ensure (NID, Email) combination is unique, and all fields are filled.
- **Response:** OK if successful, Error if (NID, Email) is not unique or any field is empty

2. Login:

- **User Input:** Email, Password
- **Validation:** Check if Email and Password match records.
- **Response:** OK with a token if successful, Error if empty or didn't match.

3. Profile:

- **User Requirement:** Logged in.
- **Response:** OK to view the profile, Error for internal server issues.

4. Update Profile:

- **User Requirement:** Logged in, Provides new Phone and/or Address.
- **Response:** OK if successful, Error for internal server issues.

5. Apply for Registration:

- **User Requirement:** Logged in.
- **User Input:** Land information and National Electronic Code (NEC).
- **Response:** OK if successful, Error if land is already registered or already pending, or internal server issues.

6. Dashboard:

- **User Requirement:** Logged in.
- **Response:** OK to retrieve dashboard data, Error for internal server issues.

7. All Applications:

- **User Requirement:** Logged in.
- **Response:** OK to retrieve all applications, Error for internal server issues.

8. Single Application:

- User Requirement: Logged in.
- Response: OK to retrieve details, download NEC, and download Deed (if status is registered), Error for internal server issues.

9.Sell Land:

- **User Requirement:** Logged in.
- **User Input:** Transfer ownership to desired buyer.
- **User Input:** OK if ownership transferred, land status becomes pending; Error if land is not registered or internal server issues.

3.5.2 Admin(Ministry of Land) side:

1. Approve/Reject:

- **User Requirement:** User must be logged in, and the user must be an admin.
- **User Input:** Admin action to approve or reject a land registration application.
- **Response:** OK if the application status was pending, the admin is from the Ministry of Land; otherwise, an error is displayed. Internal Server Error for any server-related issues.

2. Approval Process:

- **User Requirement:** Admin privileges and logged in.
- **User Action:** Admin approves or rejects a land registration application.

3. Validation:

- Check if the application status is pending.
- . Verify that the admin is authorized (from the Ministry of Land).

4. Response:

- OK if the application is approved.
- Error if the application status is not pending, or the admin is not authorized, or internal server issues.

3.5.3 Admin(Land Revenue Office) side:

1. Register/Reject:

- **User Requirement:** User must be logged in, and the user must be an admin.
- **User Input:** Admin action to register or reject a land registration application
- **Response:** OK if the application status was approved, the admin is from the Land Revenue Office; otherwise, an error is displayed. Internal Server Error for any server-related issues.

2. Approval Process:

- **User Requirement:** Admin privileges and logged in.
- **User Action:** Admin registers or rejects a land registration application.

3. Validation:

- Check if the application status is pending.
- Verify that the admin is authorized (from the Land Revenue Office).

4. Response:

- OK if the application is registered.
- Error if the application status is not approved, or the admin is not authorized, or internal server issues.

3.6 Necessary function's Chaincode

3.6.1 CreateApplication()

```go
// Create Land Application
func (s *SmartContract) CreateApplication(ctx contractapi.TransactionContextInterface,
    dagNo int, dist string, div string, khatianNo int, mouza string, nec string, oName
    string, oNID int, payTx string, upazila string) (string, error) {

    var prefix1 string = strings.ToLower(dist)
    var prefix2 string = strings.ToLower(div)
    var prefix3 string = strings.ToLower(mouza)
    var prefix4 string = strings.ToLower(upazila)
    id := fmt.Sprintf("%d%s%s%d%s%s", dagNo, prefix1, prefix2, khatianNo, prefix3, prefix4)

    exists, err := s.AssetExists(ctx, id)
    if err != nil { return "", err }
    if exists { return "", fmt.Errorf("Someone already applied for %s", id) }

    land := Land{
        AssetID:        id,
        Authenticators: []int{},
        Comment:        "",
        DagNo:          dagNo,
        DeedID:         "",
        District:       dist,
        Division:       div,
        KhatianNo:      khatianNo,
        Mouza:          mouza,
        NEC:            nec,
        OwnerName:      oName,
        OwnerNID:       oNID,
        PayTx:          payTx,
        Status:         "pending",
        Upazila:        upazila,
    }

    landJSON, err := json.Marshal(land)
    if err != nil { return "", err }

    err = ctx.GetStub().PutState(id, landJSON)
    if err != nil {
        return "", fmt.Errorf("Failed to save application in ledger: %v", err)
    }

    return ctx.GetStub().GetTxID(), nil
}
```

Fig. 3.5. Chaincode of CreateApplication function

The inputs of the above functions are:

Table 3.2: Input of CreateApplication function

Variable	**Data type**
Dag Number	int
District	string
Division	string
Khatian Number	int
Mouza	string
NEC	string
Owner Name	string
Owner NID	int
Pay Transaction	string
Upazila	string

Output: This function will return the transaction ID if it is successful or an error if there is an error.

Description: This function takes the inputs as described above. Then make a unique ID [6-10] with dagNO, dist, div, khatianNo, mouza, and upazila. Then check in the ledger if any land has already been registered [12-14] with the same ID or not. If it is it will return an error, if it isn't then it will create a new asset [16-32] with the given inputs convert them into JSON [34-35], and put them in the ledger [37-40]. Lastly, it returns the transaction ID [42].Here number inside [ ] are code number.

3.6.2 ResubmitApplication

```go
// Resubmit Application
func (s *SmartContract) ResubmitApplication(ctx contractapi.TransactionContextInterface,
    id string, dagNo int, dist string, div string, khatianNo int, mouza string, nec string,
    oName string, oNID int, payTx string, upazila string) (*Land, error) {

    landJSON, err := ctx.GetStub().GetState(id)
    if err != nil { return nil, fmt.Errorf("Failed to read from world state: %v", err) }
    if landJSON == nil { return nil, fmt.Errorf("Land info %s doesn't exist", id) }

    var land Land
    err = json.Unmarshal(landJSON, &land)
    if err != nil { return nil, err }

    land.DagNo = dagNo
    land.District = dist
    land.Division = div
    land.KhatianNo = khatianNo
    land.Mouza = mouza
    land.NEC = nec
    land.OwnerName = oName
    land.OwnerNID = oNID
    land.PayTx = payTx
    land.Status = "pending"
    land.Upazila = upazila

    landJSON, err = json.Marshal(land)
    if err != nil { return nil, err }

    err = ctx.GetStub().PutState(id, landJSON)
    if err != nil {
        return nil, fmt.Errorf("Failed to save application in ledger: %v", err)
    }

    return &land, nil
}
```

Fig. 3.6. Chaincode of ResubmitApplication function

The inputs of the above function are :

Table 3.3: Inputs of ResubmitApplication function

Variable	**Data type**
Id	string
Dag Number	int
District	string
Division	string
Khatian Number	int
Mouza	string
NEC	string
Owner Name	string
Owner NID	int
Pay Transaction	string
Upazila	string

Output: This function will return a structure of type Land if it is successful or an error if there is an error.

Description: This function takes inputs as described above. Then checks if there is an existing land application with the given ID [6-8] if not returns an error if it is then fetches it from the ledger and converts it into an object from JSON [10-12] updates the existing info [14-24] converts it back into JSON [26-27] puts it back in the ledger [29-32] and then returns the land structure [34].

3.6.3 ReadApplicationByID()

```go
// Get Single Application from ledger
func (s *SmartContract) ReadApplicationByID(ctx contractapi.TransactionContextInterface,
    id string, oNID int, userType string) (*Land, error) {

    landJSON, err := ctx.GetStub().GetState(id)
    if err != nil { return nil, fmt.Errorf("Failed to read from world state: %v", err) }
    if landJSON == nil { return nil, fmt.Errorf("Land info %s doesn't exist", id) }

    var land Land
    err = json.Unmarshal(landJSON, &land)
    if err != nil { return nil, err }
    if land.PayTx == "" { return nil, fmt.Errorf("Not found") }

    clientMSPID, err := ctx.GetClientIdentity().GetMSPID()
    if err != nil { return nil, fmt.Errorf("Failed to get client's MSPID: %v", err) }

    if userType != "user" {
        if clientMSPID == molMSPID && land.Status != "pending" {
            return nil, fmt.Errorf("Not authorized MOL")
        }
        if clientMSPID == lroMSPID && land.Status != "approved" {
            return nil, fmt.Errorf("Not authorized LRO",)
        }
    } else {
        if land.OwnerNID != oNID {
            return nil, fmt.Errorf("Not authorized")
        }
    }
    return &land, nil
}
```

Fig. 3.7. Chaincode of ReadApplication function

The inputs of the above functions are:

TABLE 3.4: Inputs of ReadApplication function

Variable	Data type
Id	string
Owner NID	int
User Type	string

Output: This function will return a structure of type Land if it is successful or an error if there is an error.

Description: This function takes inputs as described above. Then checks if there is an existing land application with the given ID [5-7] if not returns an error if it is then fetches it from the ledger and converts it into an object from JSON [9-11] gets the MSP ID of the client [14-15]. After that, if the user is an admin it will send the asset to the Ministry of Land if it is pending [18=20] or send it to the Land Revenue Office if it is approved [21-23]. And if the client is a user

then it will send the asset if the client is the owner [24-27] and then returns the land structure [29].

3.6.4 GetAllUserApplication()

```go
// User Dashboard
func (s *SmartContract) GetAllUserApplications(ctx contractapi.TransactionContextInterface,
    oNID int, userType string) ([]*Dashboard, error) {

    resultsIterator, err := ctx.GetStub().GetStateByRange("", "")
    if err != nil { return nil, err }
    defer resultsIterator.Close()

    var lands []*Dashboard
    for resultsIterator.HasNext() {
        queryResponse, err := resultsIterator.Next()
        if err != nil { return nil, err }

        var land Land
        err = json.Unmarshal(queryResponse.Value, &land)
        if err != nil { return nil, err }
        if land.PayTx == "" { continue }

        // Checking if the land owned by the user
        if land.OwnerNID == oNID {
            dashboardLand := Dashboard{
                AssetID:  land.AssetID,
                District: land.District,
                Status:   land.Status,
                Upazila:  land.Upazila,
            }
            lands = append(lands, &dashboardLand)
        }
    }
    return lands, nil
}
```

Fig. 3.8. Chaincode of GetAllUserApplication() function

The inputs of the above function :

TABLE 3.5: Inputs of GetAllUserApplication function

Variable	Data type
Owner NID	int
User Type	string

Output: This function will return an array of structures of type Land if it is successful or an error if there is an error.

Description: This function takes inputs as described above. Then fetches all land applications [5-7] if not returns an error. Then it iterates through all the assets [11-12] and converts each

asset into an object [14-16]. If the client is the owner it will append the asset into an array of structure of type Dashboard [20-27]. Then returns the dashboard array or an error [30].

3.6.5 ReadDeedByID()

```
1  // Get a single Deed from ledger
2  func (s *SmartContract) ReadDeedByID(ctx contractapi.TransactionContextInterface,
3      id string) (*Deed, error) {
4
5      deedJSON, err := ctx.GetStub().GetState(id)
6      if err != nil {
7          return nil, fmt.Errorf("Failed to read from world state: %v", err)
8      }
9      if deedJSON == nil { return nil, fmt.Errorf("Deed info %s doesn't exist", id) }
10
11     var deed Deed
12     err = json.Unmarshal(deedJSON, &deed)
13     if err != nil { return nil, err }
14
15     return &deed, nil
16 }
```

Fig. 3.9. Chaincode of ReadDeedByID function

Here Input is only ID of string datatype

Output: This function will return a structure of type Deed if it is successful or an error if there is an error.

Description: This function takes inputs as described above. Then checks if there is an existing deed with the given ID [5-9] if not returns an error if it is then fetches it from the ledger and converts it into an object from JSON [11-13] then returns the deed structure [15].

3.6.6 GetAllCertificates()

It has no input

Output: This function will return an array of structures of type Deed if it is successful or an error if there is an error.

Description: This function takes inputs as described above. Then fetches all deeds [5-7]. Then convert the deeds into objects [14-16] one by one and append them [32] into an array of structures of type Deed. Then returns the deeds array or an error [34].

```
1  // Get all Registered Land Certificate
2  func (s *SmartContract) GetAllCertificates(ctx contractapi.TransactionContextInterface)
3  ([]*Deed, error) {
4
5      resultsIterator, err := ctx.GetStub().GetStateByRange("", "")
6      if err != nil { return nil, err }
7      defer resultsIterator.Close()
8
9      var deeds []*Deed
10     for resultsIterator.HasNext() {
11         queryResponse, err := resultsIterator.Next()
12         if err != nil { return nil, err }
13
14         var land Land
15         err = json.Unmarshal(queryResponse.Value, &land)
16         if err != nil { return nil, err }
17         if land.NEC != "" { continue }
18
19         deed := Deed{
20             AssetID:        land.AssetID,
21             Authenticators: land.Authenticators,
22             DagNo:          land.DagNo,
23             DeedID:         land.DeedID,
24             District:       land.District,
25             Division:       land.Division,
26             KhatianNo:      land.KhatianNo,
27             Mouza:          land.Mouza,
28             OwnerName:      land.OwnerName,
29             OwnerNID:       land.OwnerNID,
30             Upazila:        land.Upazila,
31         }
32         deeds = append(deeds, &deed)
33     }
34     return deeds, nil
35 }
```

Fig. 3.10. Chaincode of GetAllCertificate function

3.6.7 TransferOwnerShip()

```go
// Transfer ownership
func (s *SmartContract) TransferOwnerShip(ctx contractapi.TransactionContextInterface,
    id string, oNID int, newOName string, newONID int) error {

    landJSON, err := ctx.GetStub().GetState(id)
    if err != nil { return fmt.Errorf("Failed to read from world state: %v", err) }
    if landJSON == nil { return fmt.Errorf("Land info %s doesn't exist", id) }

    var land Land
    err = json.Unmarshal(landJSON, &land)
    if err != nil { return err }

    if land.OwnerNID != oNID {
        return fmt.Errorf("Not authorized\nOwner: %d\nRequesting: %d", land.OwnerNID, oNID)
    }
    err = s.DeleteAsset(ctx, land.DeedID)
    if err != nil { return err }

    land.OwnerName = newOName
    land.OwnerNID = newONID
    land.Status = "pending"
    landJSON, err = json.Marshal(land)
    if err != nil { return err }

    return ctx.GetStub().PutState(id, landJSON)
}
```

Fig. 3.11. Chaincode of TransferOwnership function

The inputs of the above functions are:

TABLE 3.6: Inputs of TransferOwnership function

Variable	Data type
Id	string
Owner NID	int
New Owner Name	string
New Owner NID	int

Output: This function will return nil if it is successful or an error if there is an error.

Description: This function takes inputs as described above. Then checks if there is an existing land application with the given ID [5-7] if not returns an error if it is then fetches it from the ledger and converts it into an object from JSON [9-11] and checks if the client is the owner [13-15] then deletes the existing deed of the land [16-17] update with new owner [19-23] and converts it back into JSON [22-23] puts it back in the ledger [25] and then returns nil or error.

3.6.8 GetAllAdminApplication

```go
// Admin Dashboard
func (s *SmartContract) GetAllAdminApplications(ctx contractapi.TransactionContextInterface,
    oNID int, userType string) ([]*Dashboard, error) {

    if userType == userString { return nil, fmt.Errorf("User not authorized") }

    resultsIterator, err := ctx.GetStub().GetStateByRange("", "")
    if err != nil { return nil, err }
    defer resultsIterator.Close()

    clientMSPID, err := ctx.GetClientIdentity().GetMSPID()
    if err != nil { return nil, fmt.Errorf("Failed to get client's MSPID: %v", err) }

    var lands []*Dashboard
    for resultsIterator.HasNext() {
        queryResponse, err := resultsIterator.Next()
        if err != nil { return nil, err }

        var land Land
        err = json.Unmarshal(queryResponse.Value, &land)
        if err != nil { return nil, err }

        if land.PayTx == "" { continue }

        if (clientMSPID == molMSPID && land.Status == "pending") ||
            (clientMSPID == lroMSPID && land.Status == "approved") {
            dashboardLand := Dashboard{
                AssetID:  land.AssetID,
                District: land.District,
                Status:   land.Status,
                Upazila:  land.Upazila,
            }
            lands = append(lands, &dashboardLand)
        }
    }
    return lands, nil
}
```

Fig. 3.12. Chaincode of GetAllAdminApplication function

The inputs of the above function are:

TABLE 3.7: Inputs of GetAllAdminApplication function

Variable	Data type
Owner NID	int
User Type	string

Output: This function will return an array of structures of type Land if it is successful or an error if there is an error.

Description: This function takes inputs as described above. First, it checks if the client is a user or an admin. Returns an error if it is a user [5]. Then fetches all land applications [7-9] if not returns an error. It collects the MSP ID of the client [11-12]. Then it iterates through all

the assets [16-17] and converts each asset into an object [19-21]. It will append the asset into an array of structures of type Dashboard [27-33] if the application status is pending and the client is from the Ministry Of Land [25] or the application status is approved and the client is from the Land Revenue Office [26]. Then returns the dashboard array or an error [30].

3.6.9 ApproveLandApplication()

```go
// Approve Land Application by Ministry of Land
func (s *SmartContract) ApproveApplication(ctx contractapi.TransactionContextInterface,
    id string, comment string, adminNID int, response bool, userType string) error {

    clientMSPID, err := ctx.GetClientIdentity().GetMSPID()
    if err != nil { return fmt.Errorf("Failed to get client's MSPID: %v", err) }

    if userType == userString || clientMSPID != molMSPID {
        return fmt.Errorf("Authorization limited to Ministry of Land: %v", clientMSPID)
    }

    landJSON, err := ctx.GetStub().GetState(id)
    if err != nil { return fmt.Errorf("Failed to read from world state: %v", err) }
    if landJSON == nil { return fmt.Errorf("Land info %s doesn't exist", id) }

    var land Land
    err = json.Unmarshal(landJSON, &land)
    if err != nil { return err }

    if land.Status != "pending" { return fmt.Errorf("Not authorized MOL") }

    var status string
    if response {
        status = "approved"
        comment = "Land registerd by Land Revenue Office"
    } else {
        status = "rejected"
    }

    land.Status = status
    land.Comment = comment
    appendUnique(&land, adminNID)
    landJSON, err = json.Marshal(land)
    if err != nil { return err }

    return ctx.GetStub().PutState(id, landJSON)
}
```

Fig. 3.13. Chaincode of ApproveLandApplication function by Ministry of Land

The inputs of the above functions are

Table 3.8: Inputs of ApproveLandApplication function

Variable	**Data type**
Id	string
Comment	string
Admin NID	int
Response	bool
User Type	string

Output: This function will return nil if it is successful or an error if there is an error.

Description: This function takes inputs as described above. First, it collects the MSP ID of the client [5-6]. Doesn't allow if the client is a user or not from the Ministry of Land [8-10]. Then checks if there is an existing land application with the given ID [12-14] if not returns an error if it is then fetches it from the ledger and converts it into an object from JSON [16-18]. After that, if the application status is not pending returns an error [20]. If not update the status to approved if the response is true [24-25] or to rejected if the response is false [27]. After that, it updates the status comment and appends the approver NID [30-32], and then converts the land application back into JSON [33-34] and puts it back into the ledger [36].

3.6.10 RegisterLandApplication()

```go
// Register Land Application by Land Revenue Office
func (s *SmartContract) RegisterApplication(ctx contractapi.TransactionContextInterface,
    id string, comment string, adminNID int, response bool, userType string) error {

    clientMSPID, err := ctx.GetClientIdentity().GetMSPID()
    if err != nil { return fmt.Errorf("Failed to get client's MSPID: %v", err) }

    if userType == userString || clientMSPID != lroMSPID {
        return fmt.Errorf("Authorization limited to Land Revenue Office: %v", clientMSPID)
    }

    landJSON, err := ctx.GetStub().GetState(id)
    if err != nil { return fmt.Errorf("Failed to read from world state: %v", err) }
    if landJSON == nil { return fmt.Errorf("Land info %s doesn't exist", id) }

    var land Land
    err = json.Unmarshal(landJSON, &land)
    if err != nil { return err }

    if land.Status != "approved" { return fmt.Errorf("Not authorized LRO") }

    var status string
    if response {
        status = "registered"
        comment = "Land registerd by Land Revenue Office"
    } else {
        status = "rejected"
    }

    land.Status = status
    land.Comment = comment
    appendUnique(&land, adminNID)
    if land.DeedID == "" {
        land.DeedID, err = s.GenerateDeed(ctx, land)
        if err != nil {
            return fmt.Errorf("Failed to generate Deed")
        }
    }
    landJSON, err = json.Marshal(land)
    if err != nil { return err }

    return ctx.GetStub().PutState(id, landJSON)
}
```

Fig. 3.14. Chaincode of RegisterLandApplication function by Land Revenue Office

The inputs of the above function :

Table 3.9: Inputs of RegisterLandApplication function

Variable	**Data type**
Id	string
Comment	string
Admin NID	int
Response	bool
User Type	string

Output: This function will return nil if it is successful or an error if there is an error.

Description: This function takes inputs as described above. First, it collects the MSP ID of the client [5-6]. Doesn't allow if the client is a user or not from the Land Revenue Office [8-10]. Then checks if there is an existing land application with the given ID [12-14] if not returns an error if it is then fetches it from the ledger and converts it into an object from JSON [16-18]. After that, if the application status is not approved returns an error [20]. If not update the status to registered if the response is true [24-25] or to rejected if the response is false [27]. After that, it updates the status comment and appends the approver NID [30-32], generates the deed of the land [33-37] and then converts the land application back into JSON [39-40] and puts it back into the ledger [42].

3.6.11 GenerateDeed()

```
1  // Generate Deed of Land
2  func (s *SmartContract) GenerateDeed(ctx contractapi.TransactionContextInterface,
3      land Land) (string, error) {
4
5      prfx := time.Now().Unix()
6      id := fmt.Sprintf("%d%s", prfx, land.AssetID)
7      exists, err := s.AssetExists(ctx, id)
8      if err != nil { return "", err }
9      if exists { return "", fmt.Errorf("Someone already applied for %s", id) }
10
11     deed := Deed{
12         AssetID:        land.AssetID,
13         Authenticators: land.Authenticators,
14         DagNo:          land.DagNo,
15         DeedID:         id,
16         District:       land.District,
17         Division:       land.Division,
18         KhatianNo:      land.KhatianNo,
19         Mouza:          land.Mouza,
20         OwnerName:      land.OwnerName,
21         OwnerNID:       land.OwnerNID,
22         Upazila:        land.Upazila,
23     }
24     deedJSON, err := json.Marshal(deed)
25     if err != nil { return "", err }
26
27     err = ctx.GetStub().PutState(id, deedJSON)
28     if err != nil { return "", err }
29
30     return id, nil
31 }
```

Fig. 3.15. Chaincode of GenerateDeed function

Data type is Land Information of data type Land.

Output: This function will return the deed ID if it is successful or an error if there is an error.

Description: This function takes inputs as described above. First, it creates the deed ID with the asset ID and a unique number [5-6]. Checks if any deed exists with the same ID [7-9] and returns an error if it is and if not creates a new deed [11-23]. Converts it into JSON [24-25], puts it in the ledger [27-28], and returns the deed ID [30].

3.7 User Interface of our project

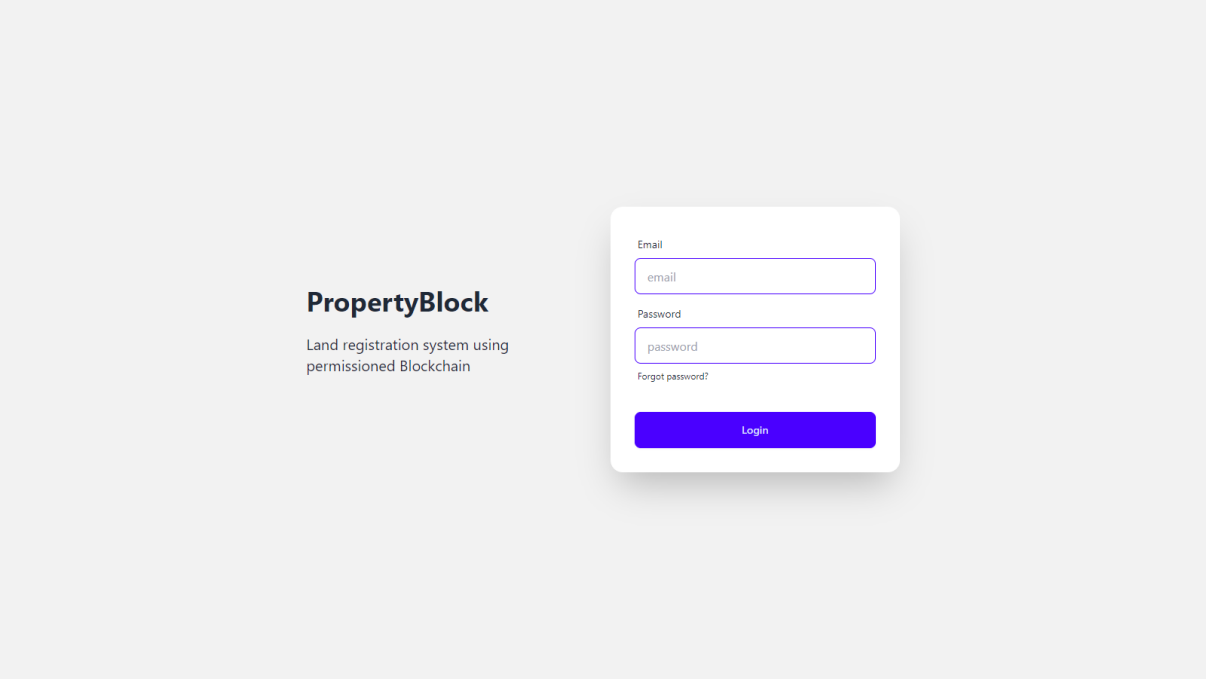

Fig. 3.16. Login page

On the login page, the client will be able to login if he or she provides the correct email and password.

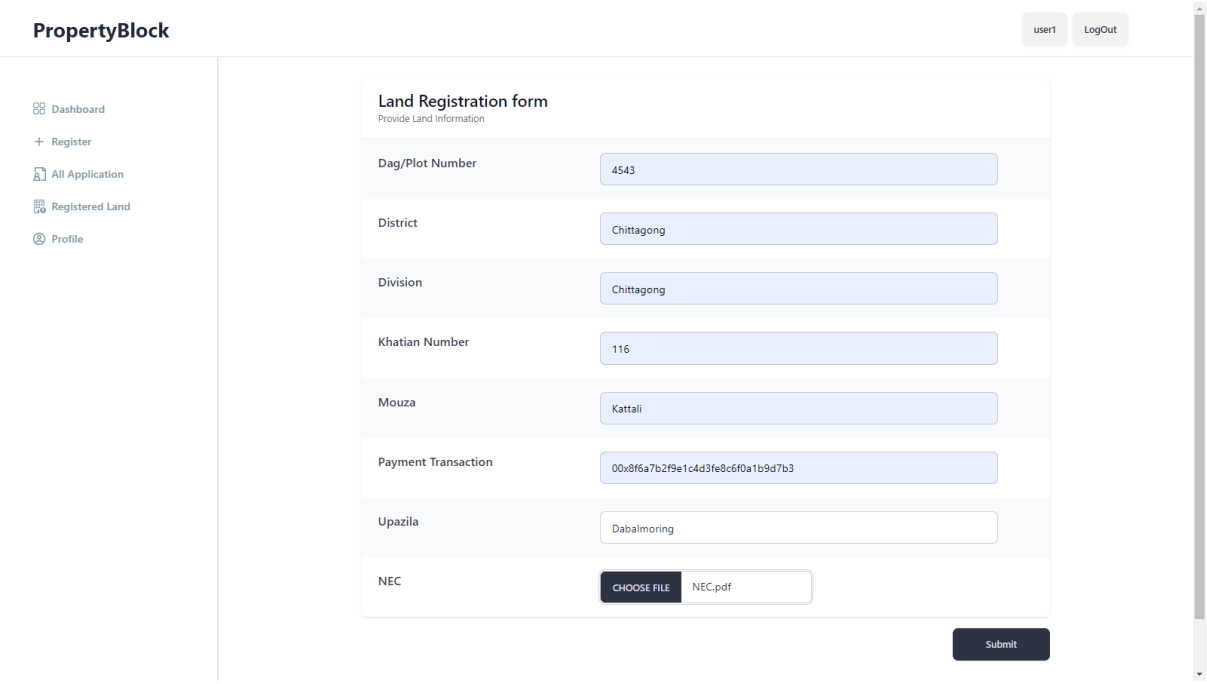

Fig. 3.17. Land Registration page

On the Land registration page, the client has to fill up the dag/plot number, district, division,KhatianNo,Mouza,Payment payment transaction, and upazila and upload the NEC certificate.

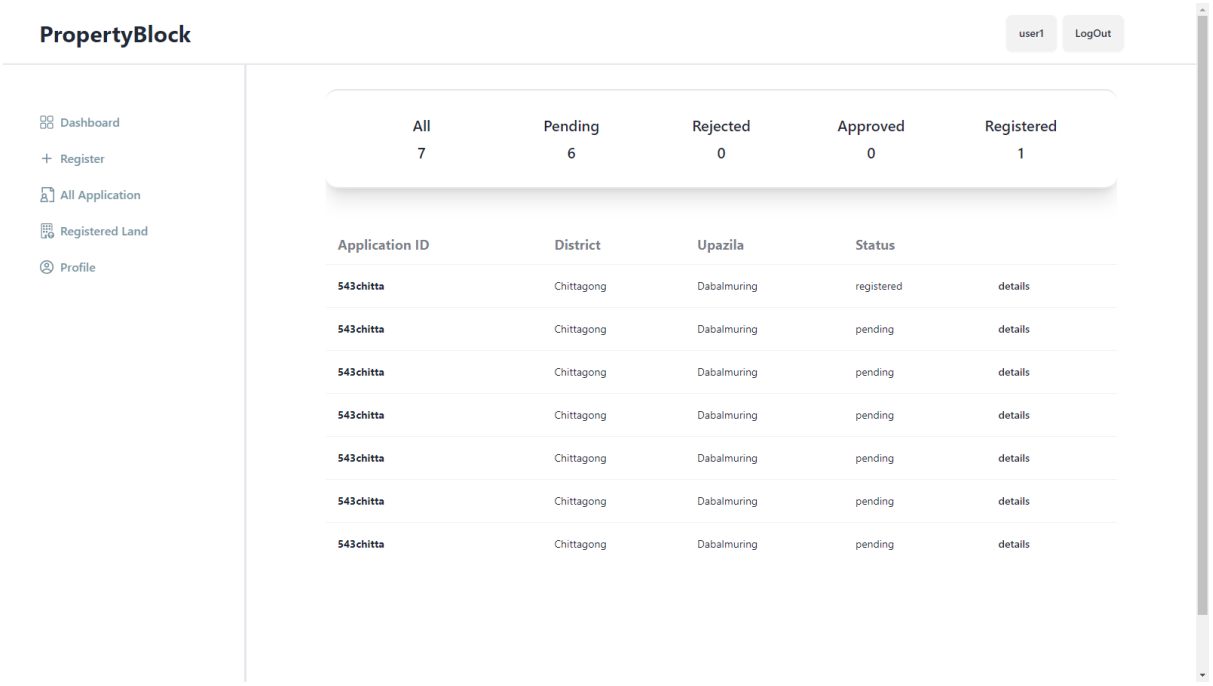

Fig. 3.18. Dashboard page

On the dashboard page, the user can see all his or her requested applications, pending applications, rejected applications, approved applications by the Ministry of Land, and registered applications by the Land Revenue Office.

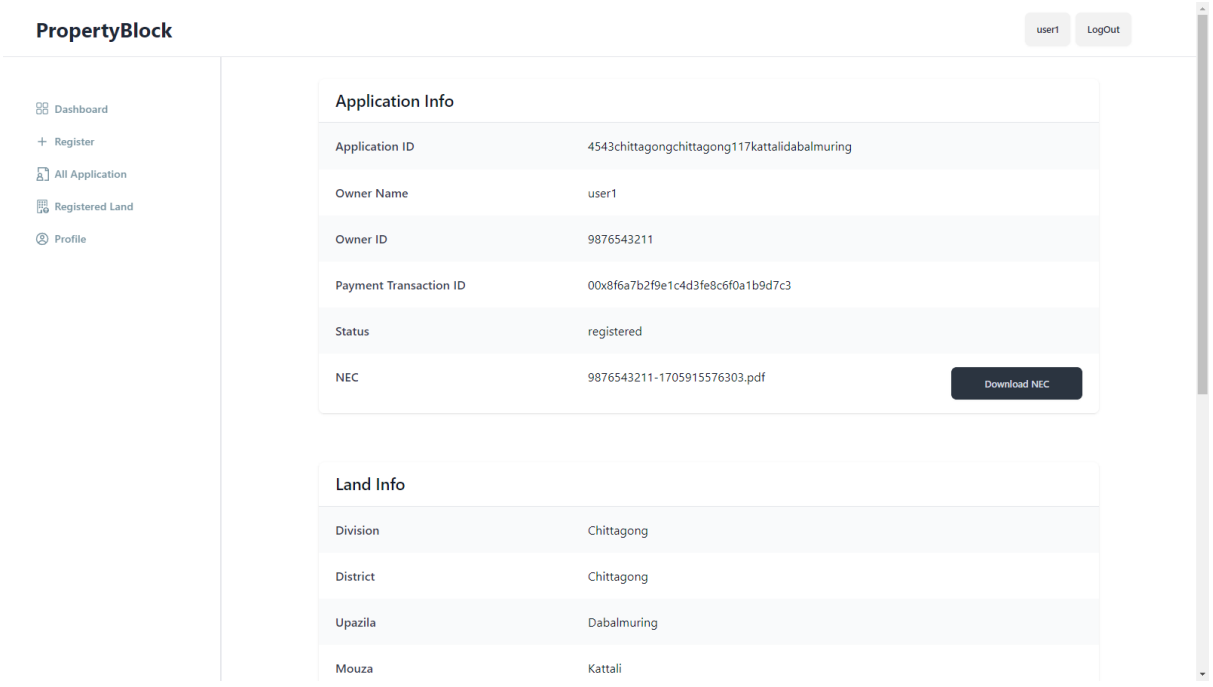

Fig. 3.19. Details of single application page

There is a Details button on the every Application at Dashboard page shown above. If the client clicks on it, he or she can see the first application information, which contains the application

ID, owner name, owner ID, payment transaction ID, status, and NEC certificate, which can be downloaded.

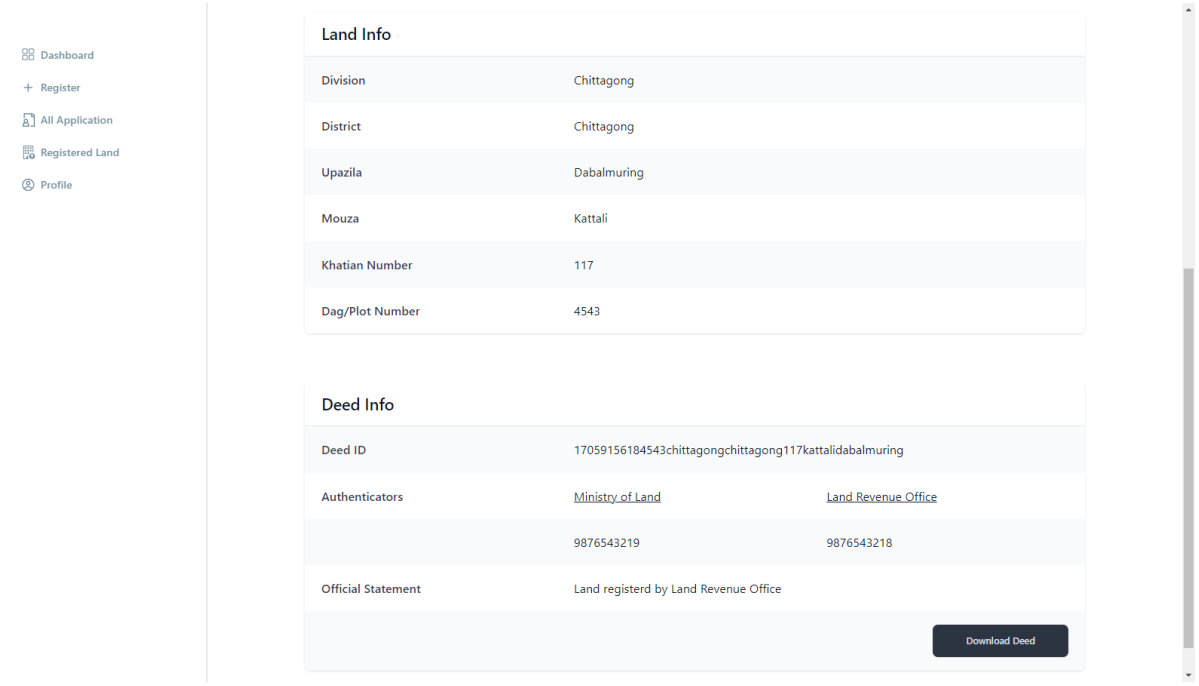

Fig. 3.20. Details of single application page

Here, the client can also see the land information, where they can see the division, district, upazila, mouza, Khatian no., and dag or plot number of the land. They can also see the deed information, which contains the deed ID, authenticators, and official statement.In the above deed information we can see that a number is generated below Ministry of Land and Land Revenue Office which is the NID number of admin of Ministry of Land and Land Revenue Office respectively , it only appears if the deed is approved by both admins , at first the admin of Ministry of Land will approve and then the admin of Land Revenue Office will be able to approve. The deed can be downloaded by clicking the Download Deed button

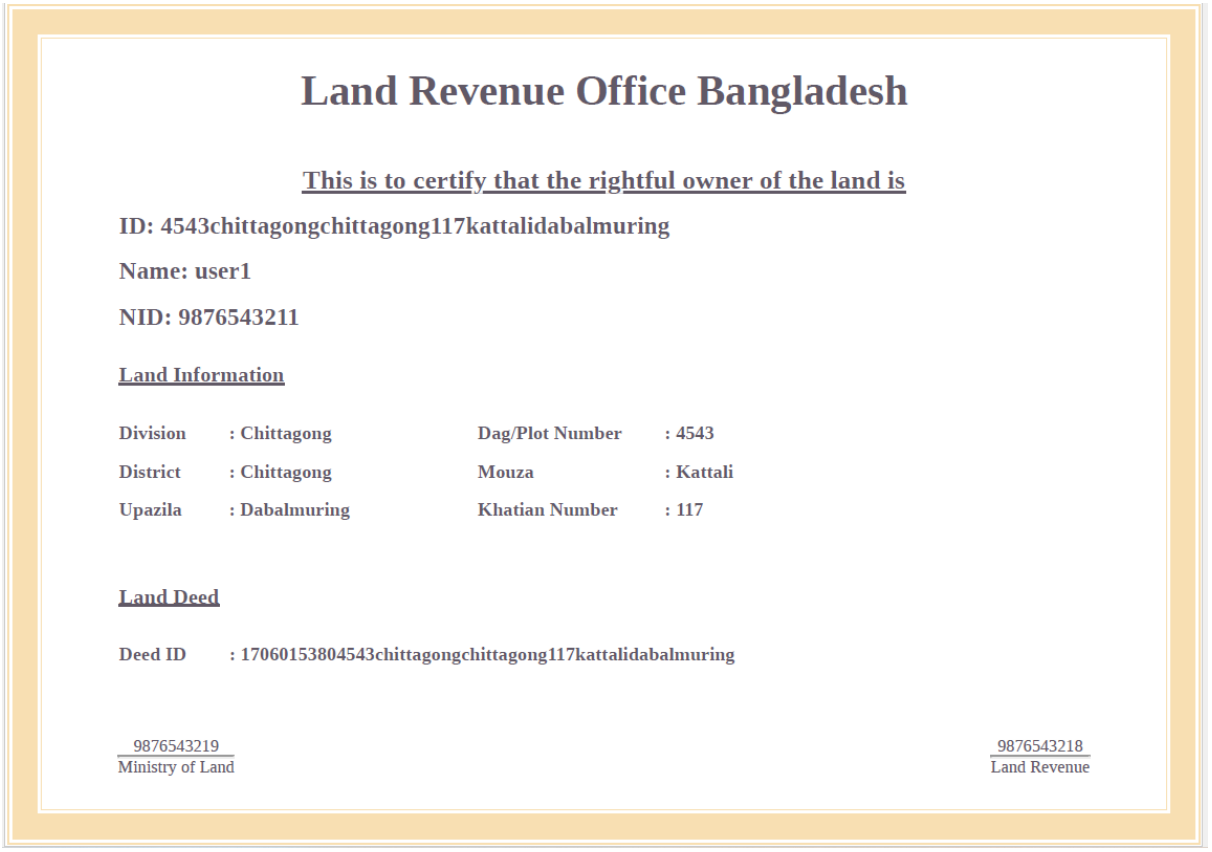

Fig. 3.21. Generated Certificate

If we click the Download Deed button shown in the previous image then the above certificate will be generated that will certify the rightful owner of the land

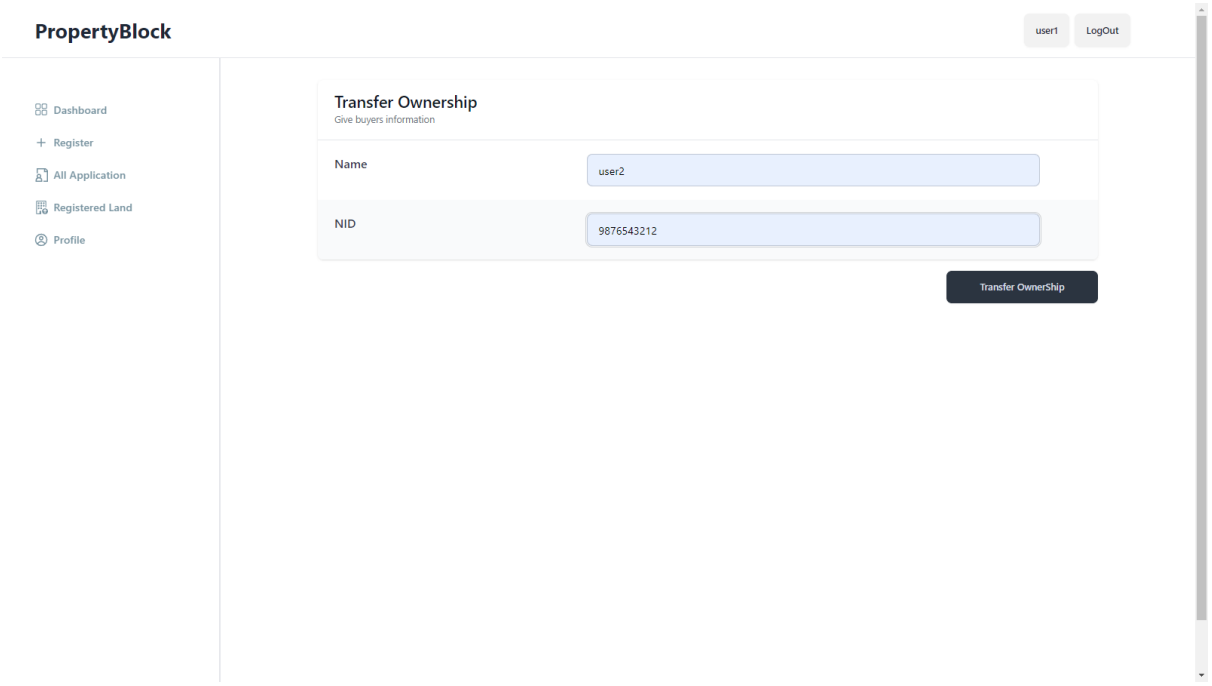

Fig. 3.22. Transfer Ownership page

On the transfer ownership page, the client has to give the buyer's information which is his or her name and NID.

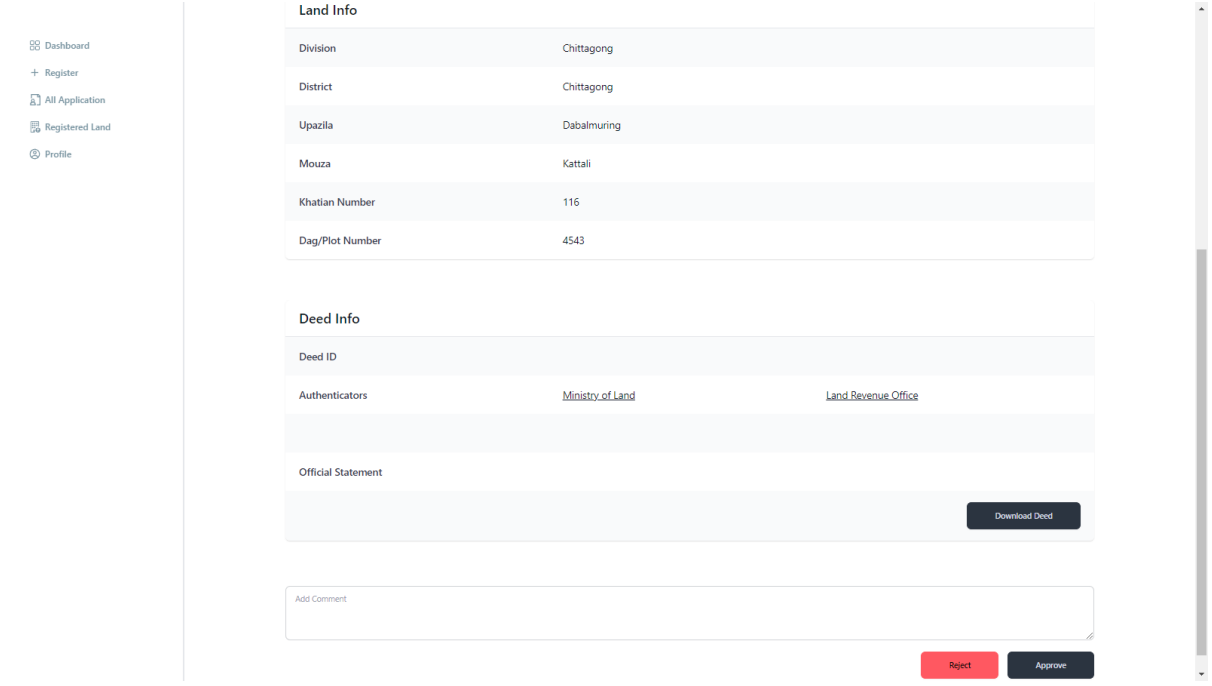

Fig. 3.23. Interface page for Admin of Ministry of Land

On this interface page, the admin of the Ministry of Land approves the application of the client by clicking the Approve button if it is found to be valid; otherwise, the admin rejects it by clicking the Reject button and by leaving a comment.

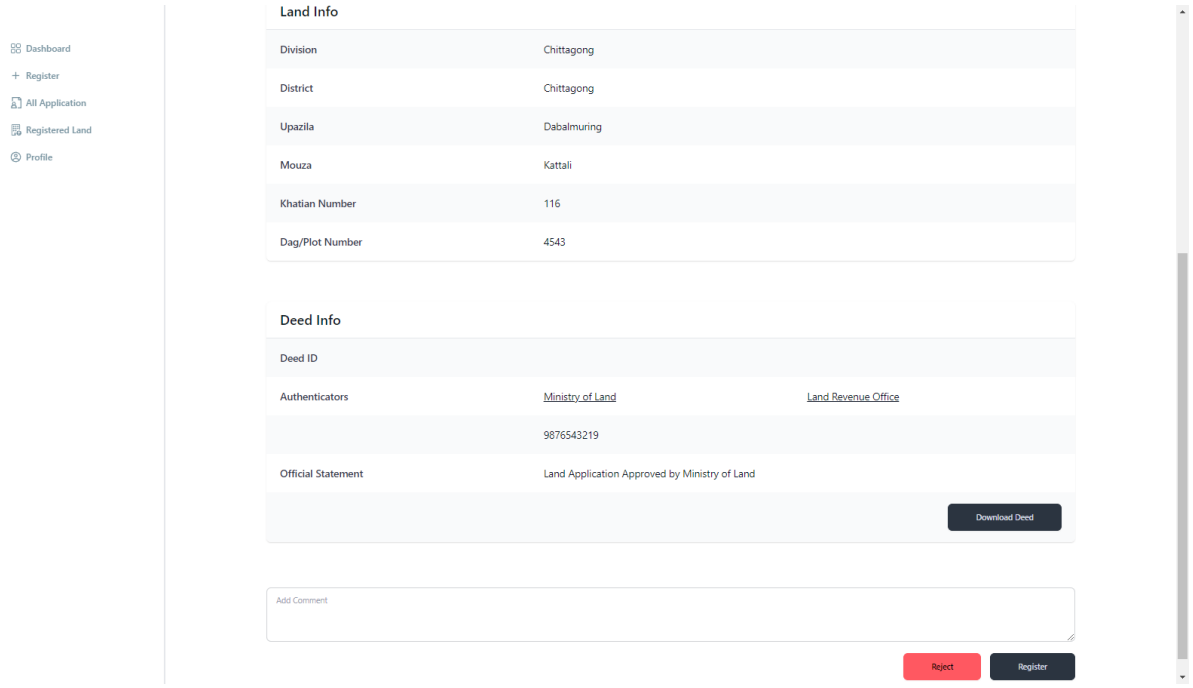

Fig. 3.24. Interface page of admin of Land Revenue Office

After approved by admin of Ministry of Land,on this interface page, the admin of the Land Revenue Office registers the application of the client by clicking the Register button if it is found to be valid; otherwise, the admin rejects it by clicking the Reject button and by leaving a comment.Here the NID of admin of Ministry of Land which proves that it was approved by admin of Ministry of Land

3.8 Entity Relationship Diagram

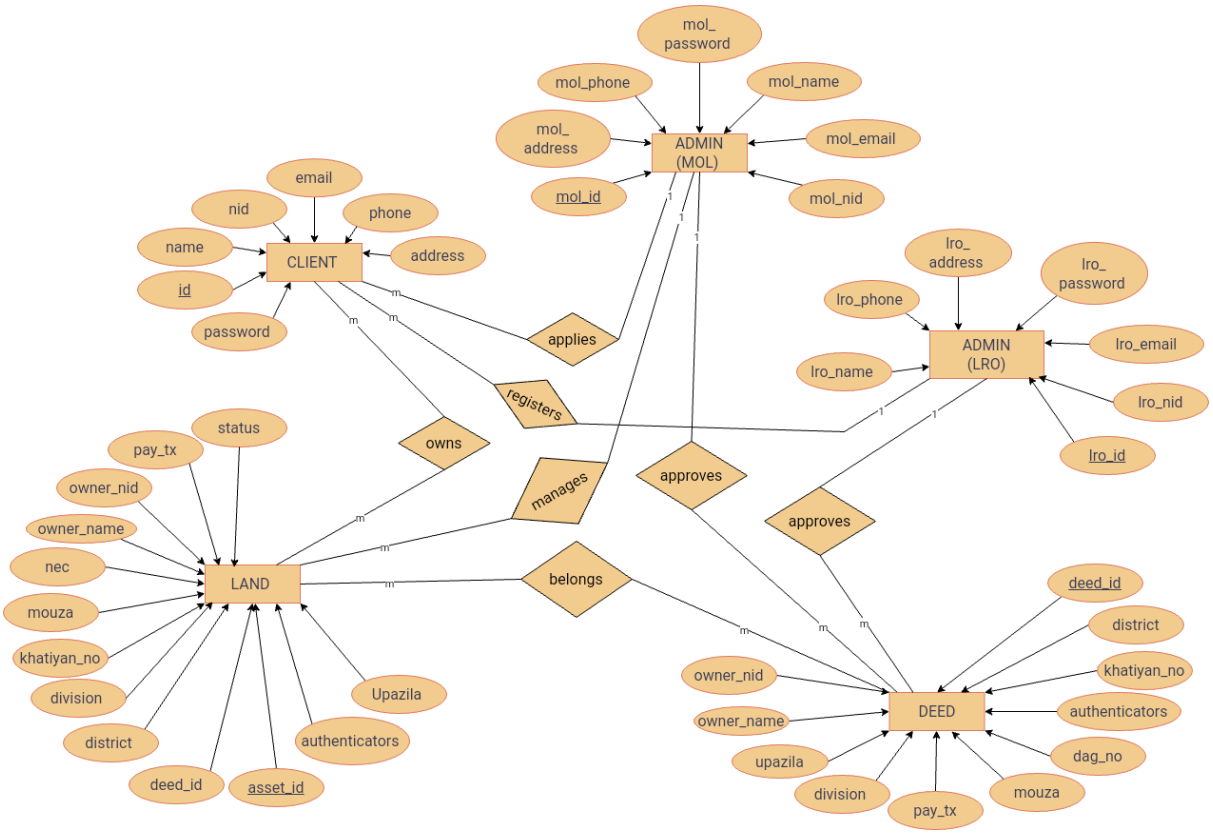

Fig. 3.25. Entity Relationship Diagram

An entity relationship diagram, often referred to as an ER diagram, is a visual representation illustrating the connections between entities within a database. In the realm of database management systems (DBMS), an ER[38] diagram holds significant importance in the database design process. Users articulate their requirements for business operations through an ER diagram, which is subsequently provided to database administrators for the actual database design.

3.8.1 Logical Schema

- CLIENT(id,name,nid,email,phone,address,password)
- LAND(asset_id,authenticators [ ]int,comment,deed_id,district,division, khatian_no,mouza,nec, owner_name,owner_nid,pay_tx,status,upazila)
- OWNS(client,land)
- ADMIN_MOL(mol_id,mol_name,mol_nid,mol_email,mol_phone,mol_address,mol_password)
- APPLIES(client,admin(mol))
- MANAGES(admin(mol),land)
- ADMIN_LRO(lro_id,lro_name,lro_nid,lro_email,lro_phone,lro_address,lro_password)
- DEED(deed_id,authenticators,district,division, khatian_no,mouza, owner_name,owner_nid,pay_tx,dag_no,upazila)
- REGISTERS(client,admin(lro))
- BELONGS(deed,land)
- APPROVES1(admin(mol),deed)
- APPROVES2(admin(lro),deed)

3.9 Use case Diagram

Use case diagrams show how different parts of a software system connect, like the jobs people do and the things they use. The lines between people and tasks show that they can do certain things in the system. It's important to know that these diagrams don't show exactly how things are done or the order they happen in. They give a general view of what people can do and what tasks are there, without getting into the detailed steps or when things happen. Basically, they give a picture of what people can do in the software system, but the exact steps are not shown in these diagrams.

In our work we have mentioned two use cases:

- Client(primary actor) and Ministry of Land (secondary actor)
- Client(primary actor) and Land Revenue Office(secondary actor)

Initially, all client requests, such as land registration applications, request to see all applications, requests to see single applications, and land sales, are routed to the Ministry of Land's admin. During this stage, the admin from the Land Revenue Office is unable to access these requests, creating a distinct separation in the process. The initial point of contact for these requests is the admin at the Ministry of Land, and access to these inquiries is restricted for the admin at the Land Revenue Office during this phase.

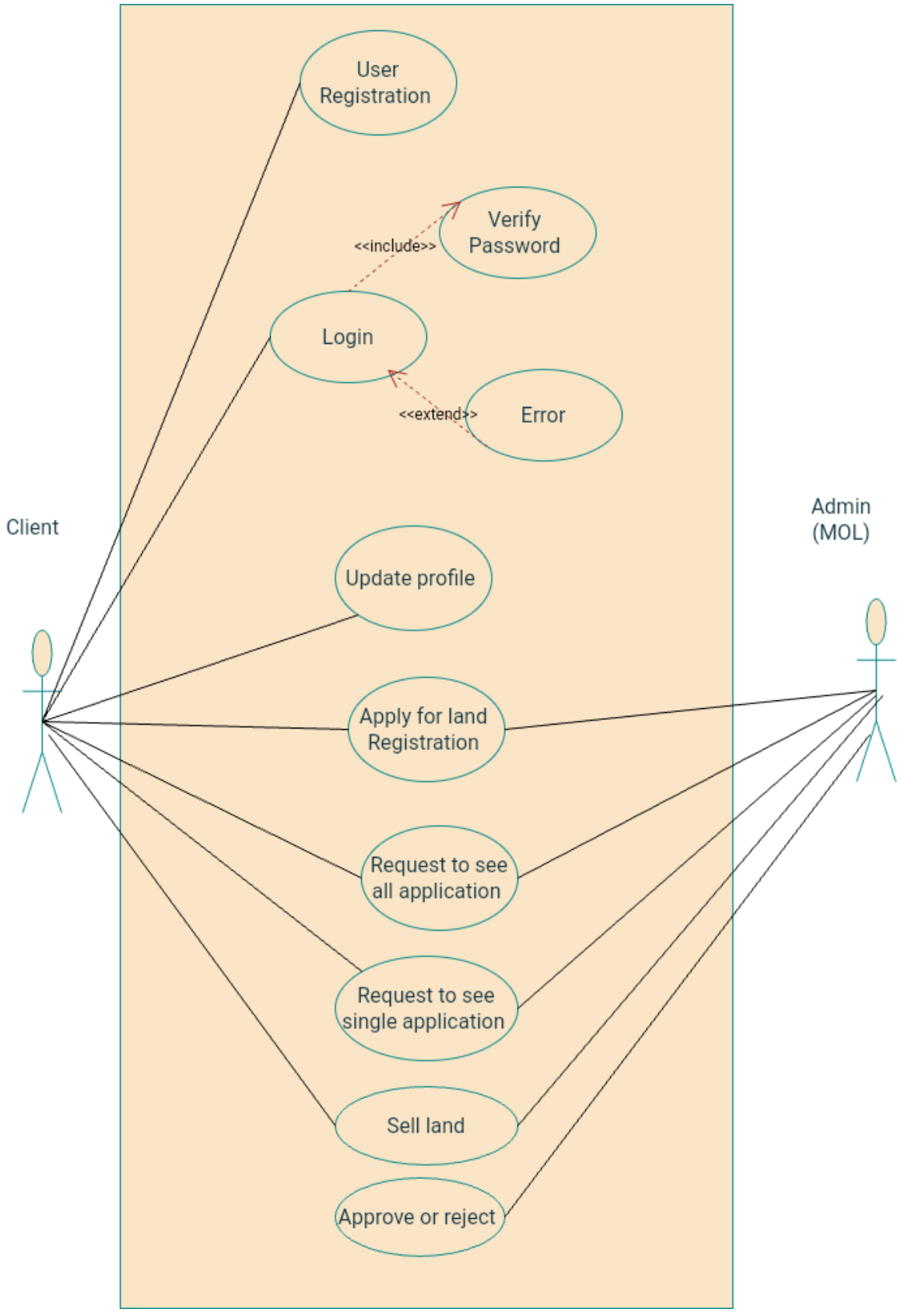

Fig. 3.26. Use case diagram of Client(primary actor) and Admin of Ministry of Land(secondary actor)

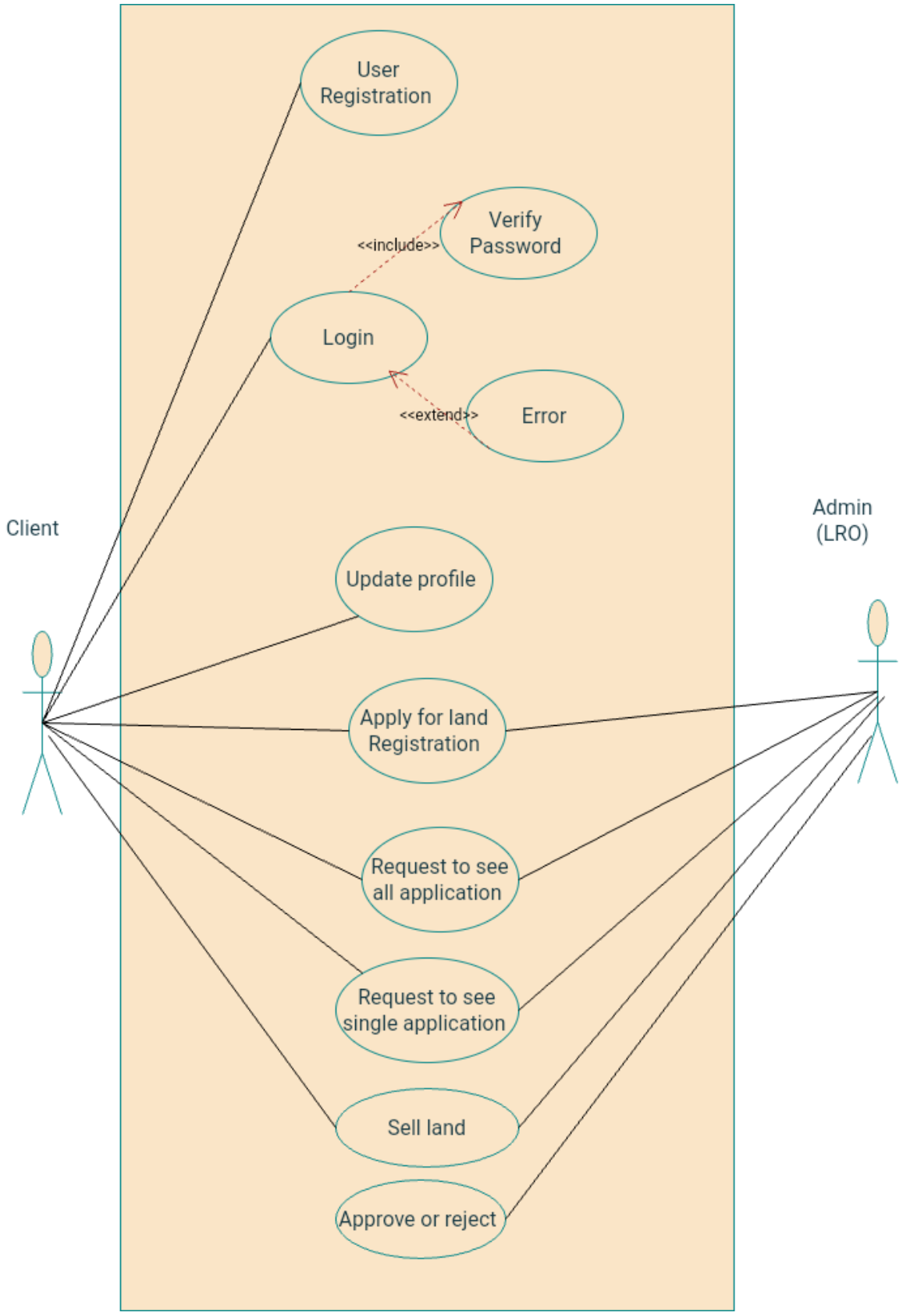

Fig. 3.27. Use case diagram of Client(primary actor) and Admin of Land Revenue Office(secondary actor)

3.10 Sequence Diagram

3.10.1 Client

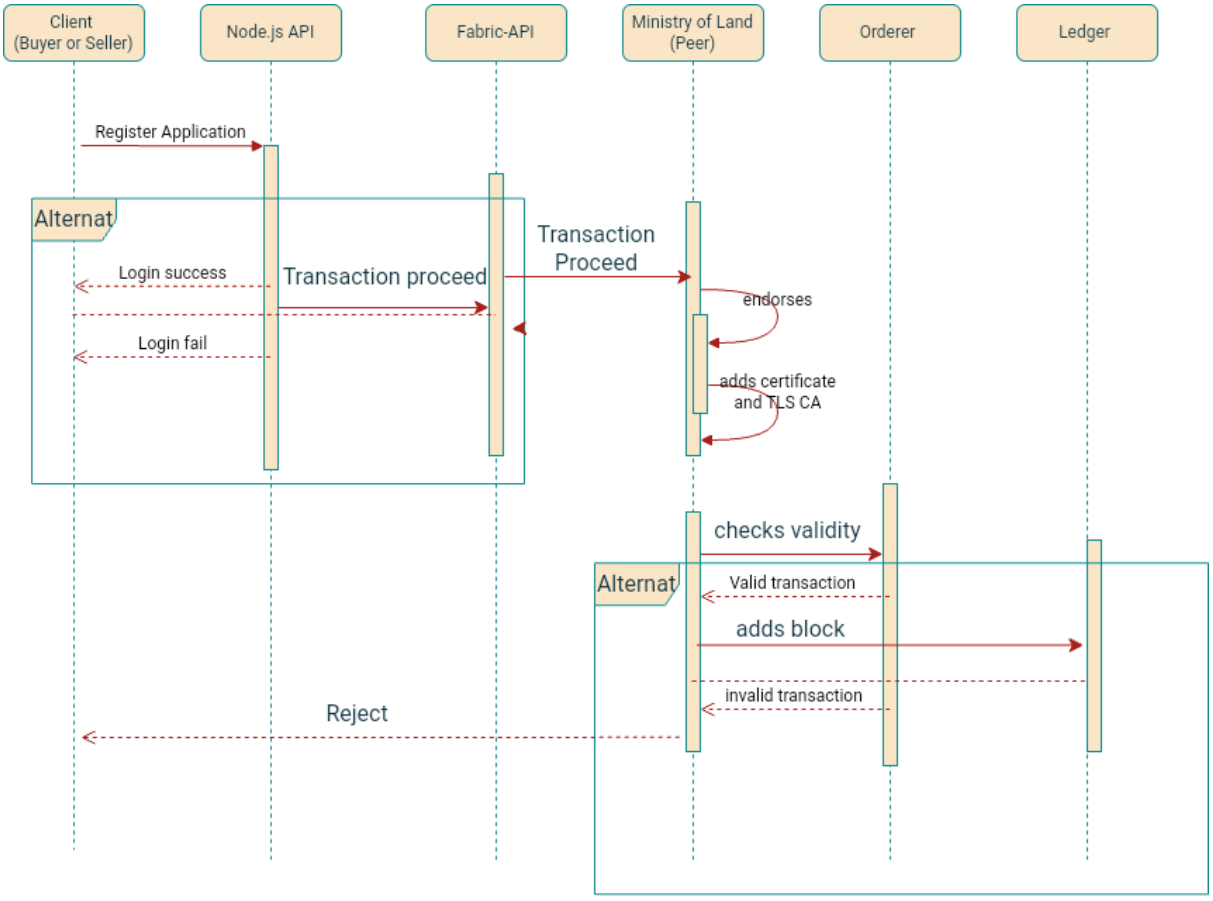

Fig. 3.28. Sequence Diagram[39] for Client side

When the client registers an application, it goes to the Node.js API. The Node API checks if the user is logged in or not. The transaction then goes to Fabric-API. The Fabric API sends the transaction to the Ministry of Land (peer). Their peer node endorses the transaction and then adds a signed certificate and the TLS CA from its local directory, and all of this is sent to the orderer. The orderer checks the validity of the transaction using the certificate sent with the transaction. Then it orders the transaction into blocks and then returns to the peer nodes. The peer then adds the new block to the ledger.

3.10.2 Admin(Ministry of Land)

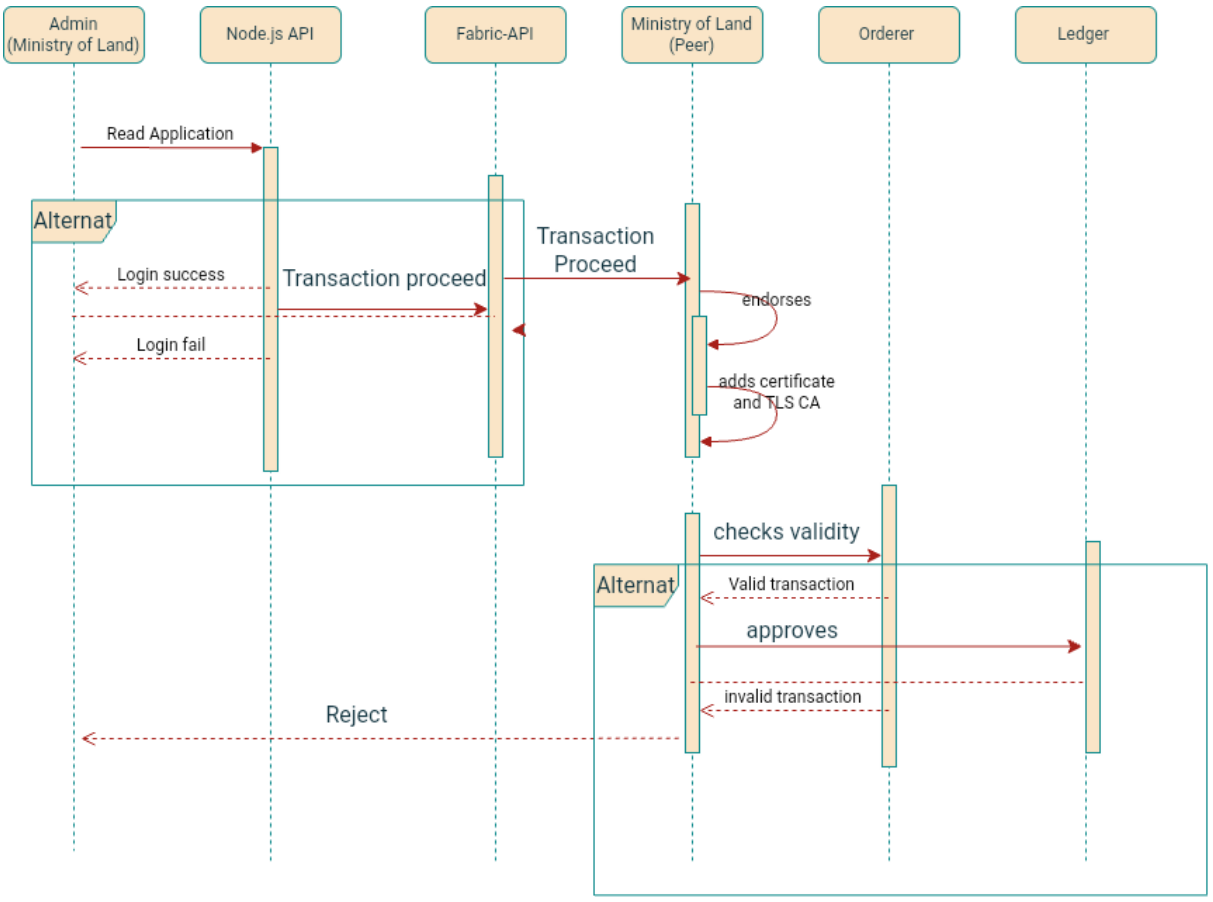

Fig. 3.29. Sequence Diagram for Admin(Ministry of Land)

The admin of the Ministry of Land reads the application of the client; it goes to the Node.js API. The Node API checks if the user is logged in or not. The transaction then goes to Fabric-API. The Fabric API sends the transaction to the Ministry of Land (peer). Their peer node endorses the transaction and then adds a signed certificate and the TLS CA from its local directory, and all of this is sent to the orderer. The orderer checks the validity of the transaction using the certificate sent with the transaction. Then it orders the transaction into blocks and then returns to the peer nodes. The peer then adds the new block to the ledger.

3.10.3 Admin(Land Revenue Office)

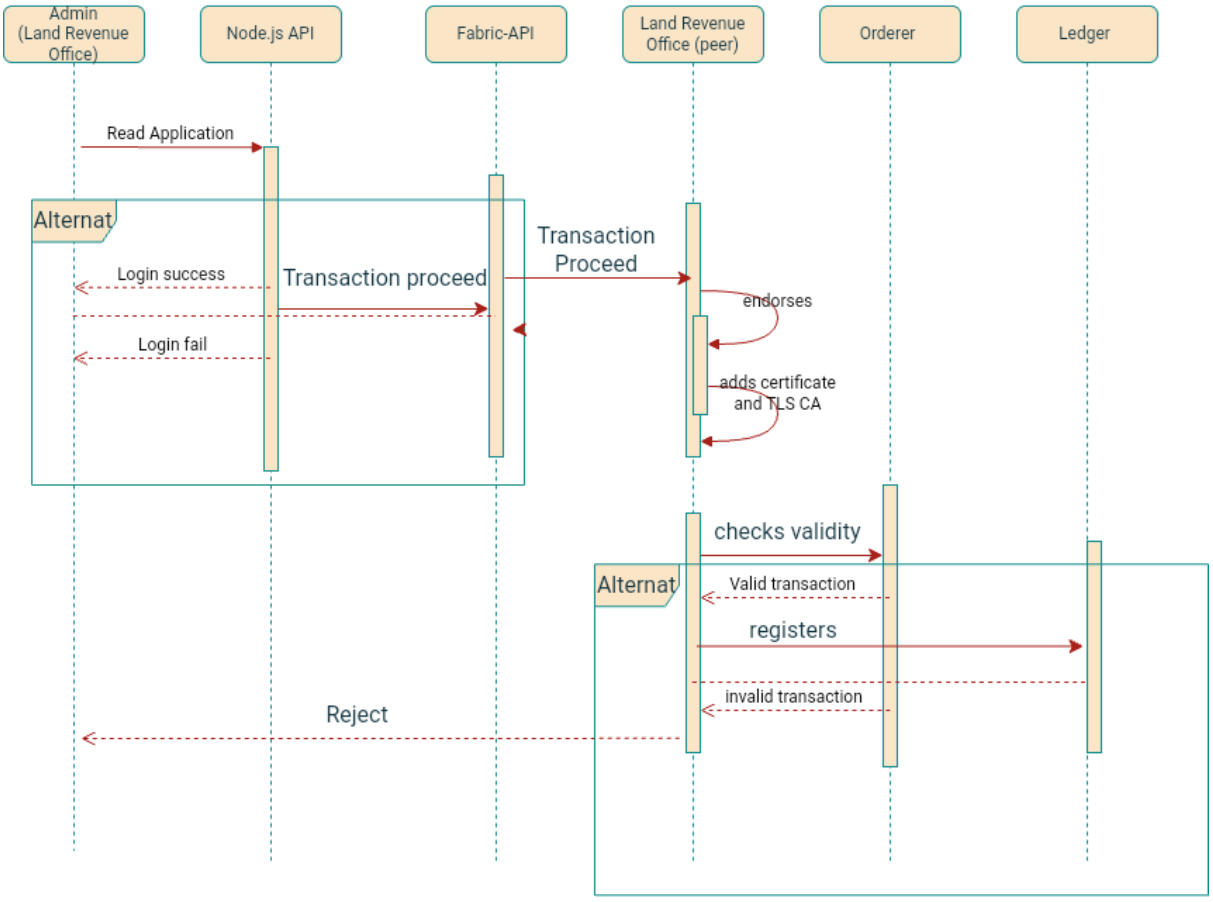

Fig. 3.30. Sequence Diagram for Admin(Land Revenue Office)

The Land Revenue Office read an application of the client, it goes to Node.js API. The Node API checks if the user is logged in or not. The transaction then goes to Fabric-API. Fabric API sends the transaction to the Land Revenue Office(peer) . Their peer node endorses the transaction and then adds a signed certificate and the TLS CA from its local directory and all of this is sent to the orderer.

Orderer checks the validity of the transaction using the certificate sent with the transaction. Then it orders the transaction into block and then returns to the peer nodes. The peer then adds the new block to the ledger.

Chapter 4

Results and Discussions

4.1 Introduction

Here we have shown the result obtained through API test,Tamperproof test and response time tests:

4.2 API Test:

The API test result we obtained are :

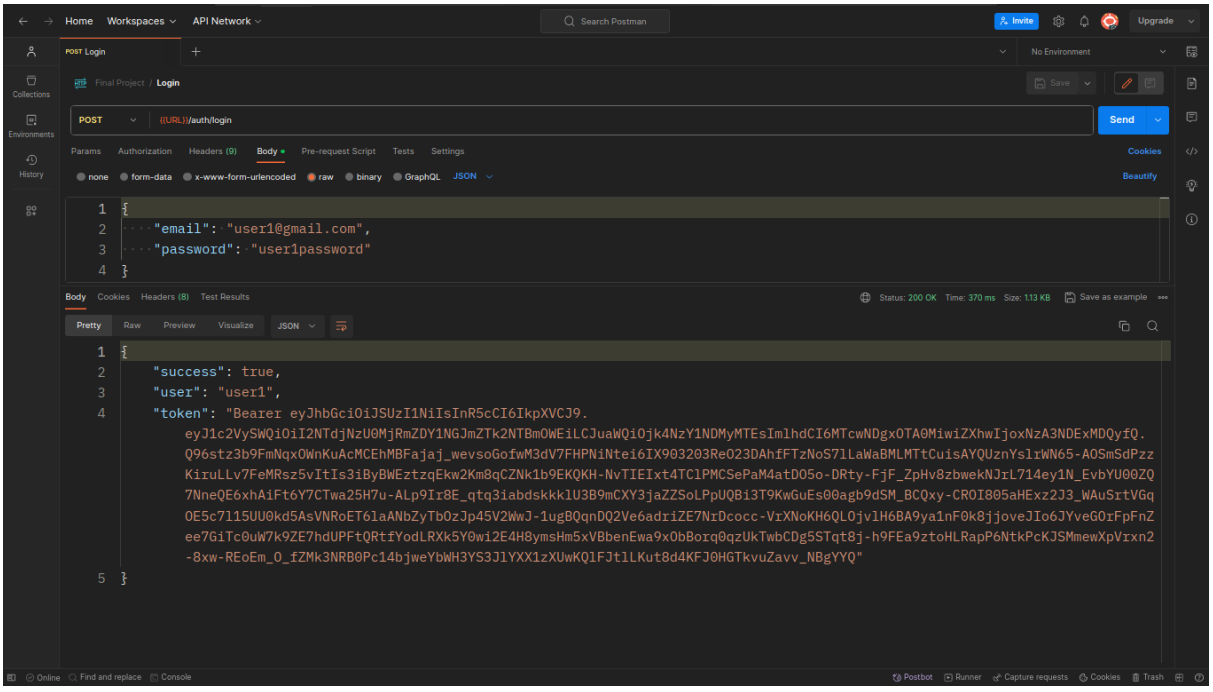

Fig. 4.1. API test of Login

Description:

User gives email and password if credentials are correct JSON Web Token will be returned.

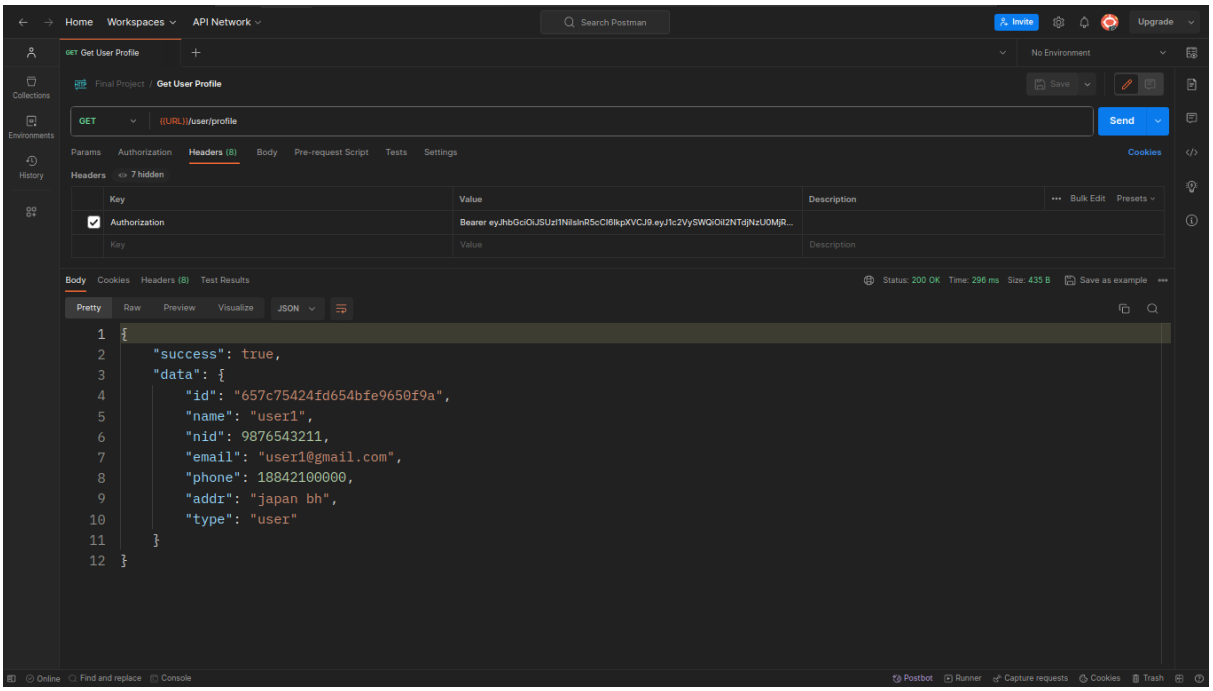

Fig. 4.2. API test of Profile

Description: Valid user after logging in they can access their profile info. Their profile info is returned here.

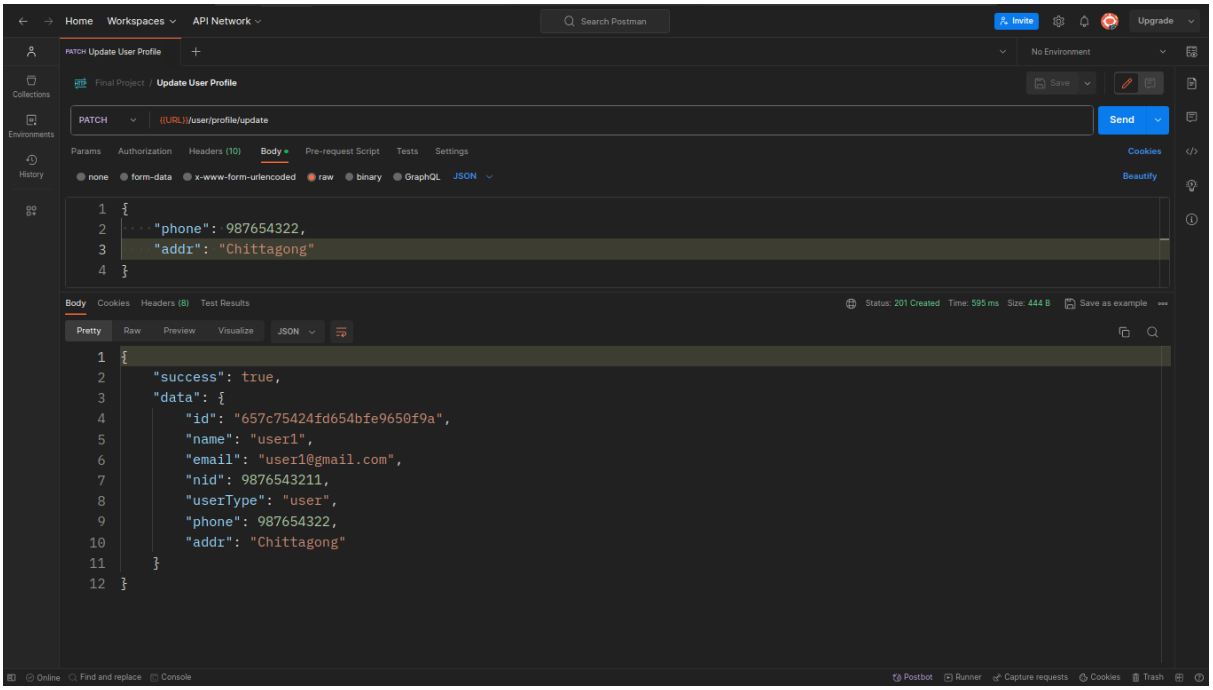

Fig. 4.3. API test of Update Profile

Description: Valid users can update their phone and address after logging in. Successful response after updating profile.

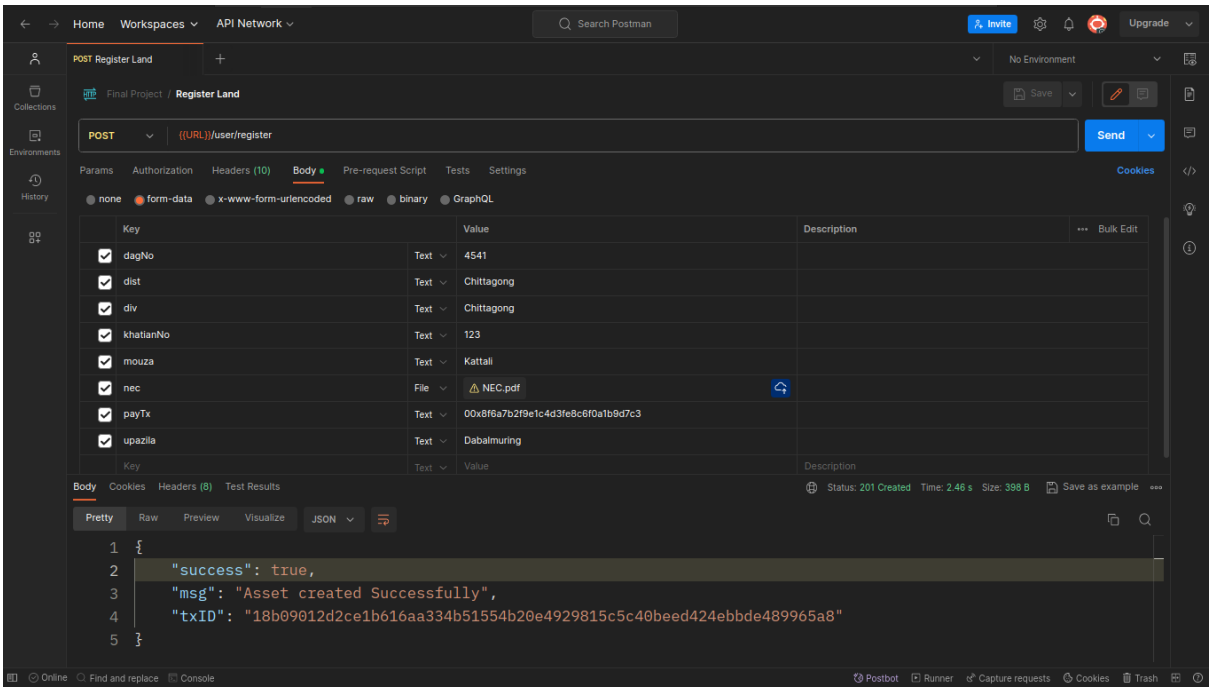

Fig. 4.4. API test of RegistrationApplication

Description: Valid users can apply for new land registration application after logging in. They will provide necessary details the application will be accepted for processing if no similar record is found. This is a successful application and the transaction id is returned.

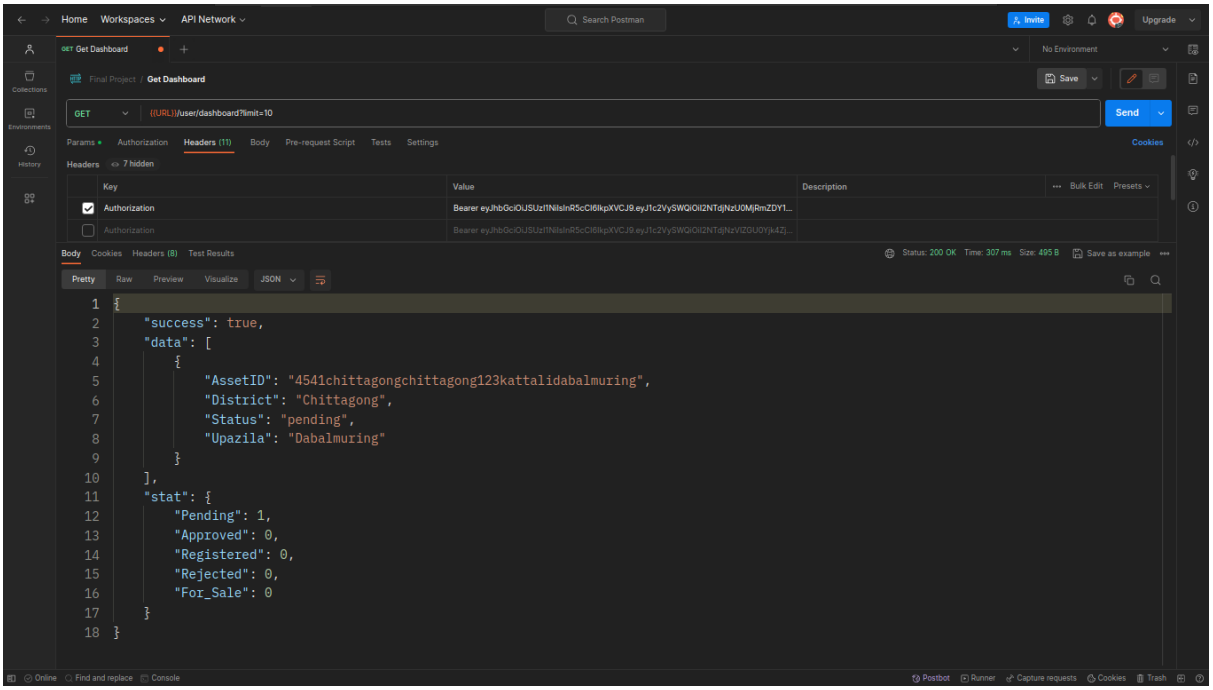

Fig. 4.5. API test of User Dashboard

Description: Valid users can access all the application they have applied for after logging in. This is the dashboard of a user.

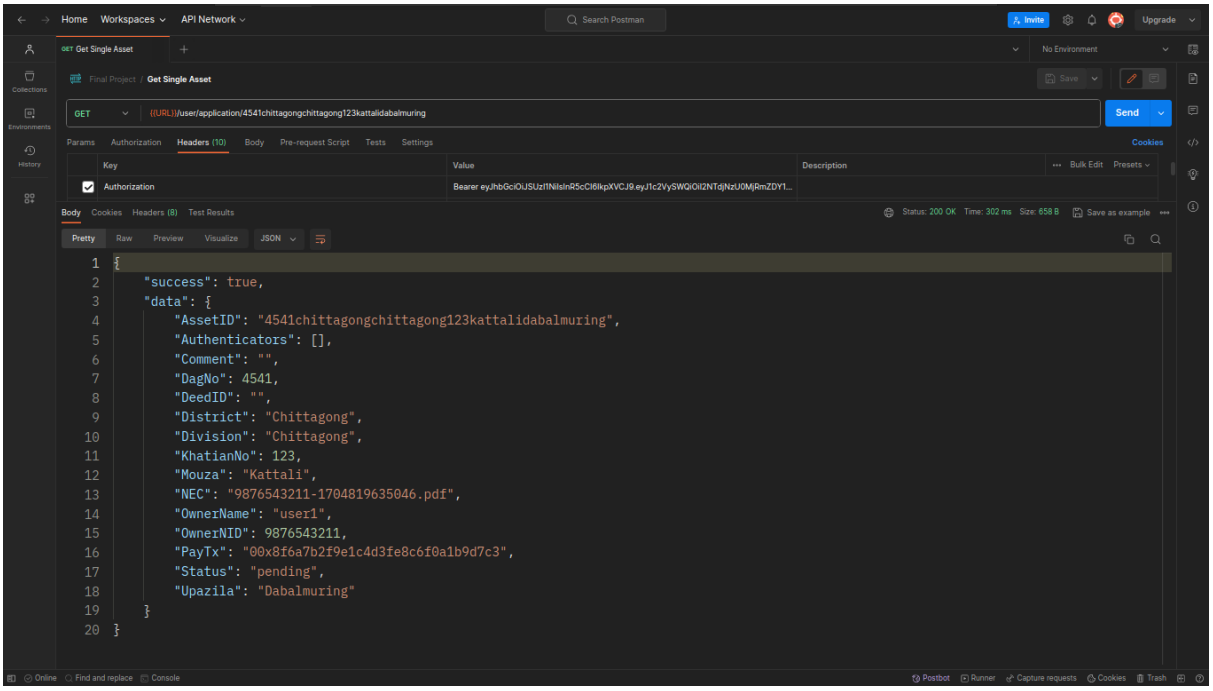

Fig. 4.6. API test of ReadSingleApplication

Description: Valid users can see each application from ledger after logging in if they are the owner or if they are from Ministry of Land and the application is in pending or they are from Land Revenue Office and the application is approved by Ministry of Land.

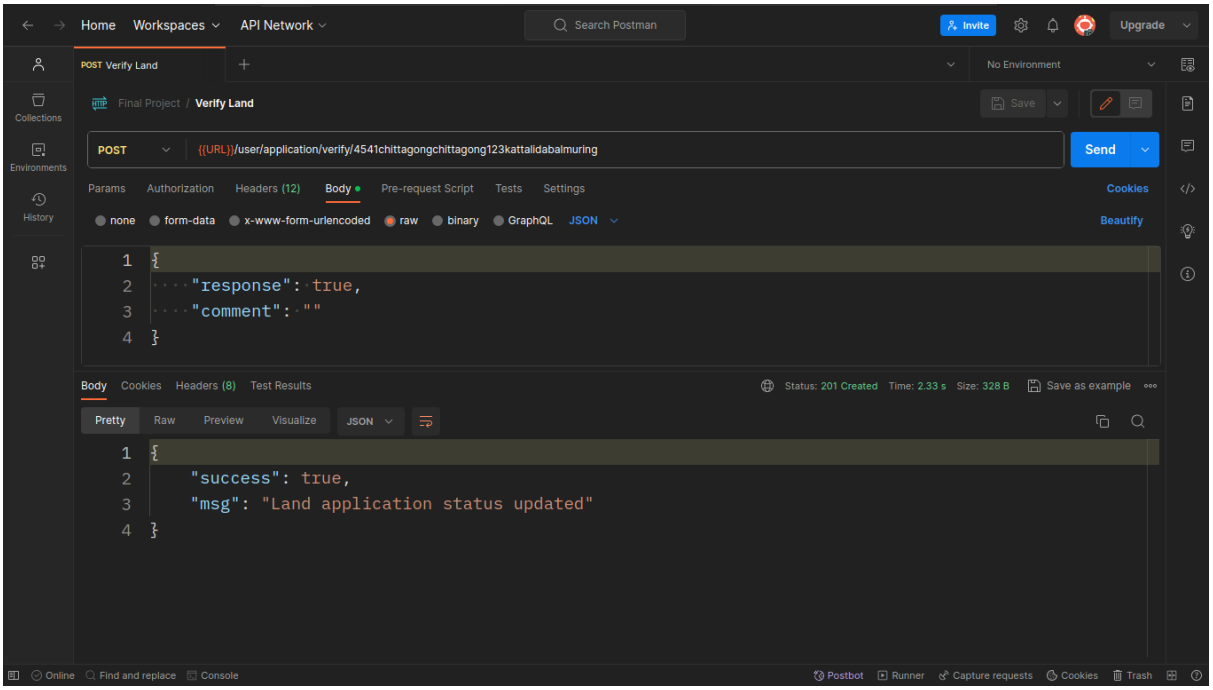

Fig. 4.7. API test of Land Application Verification

Description: After getting verification from Ministry of Land and Land Revenue Office the application will have the verifier NID and a Deed ID. A land application after getting verified.

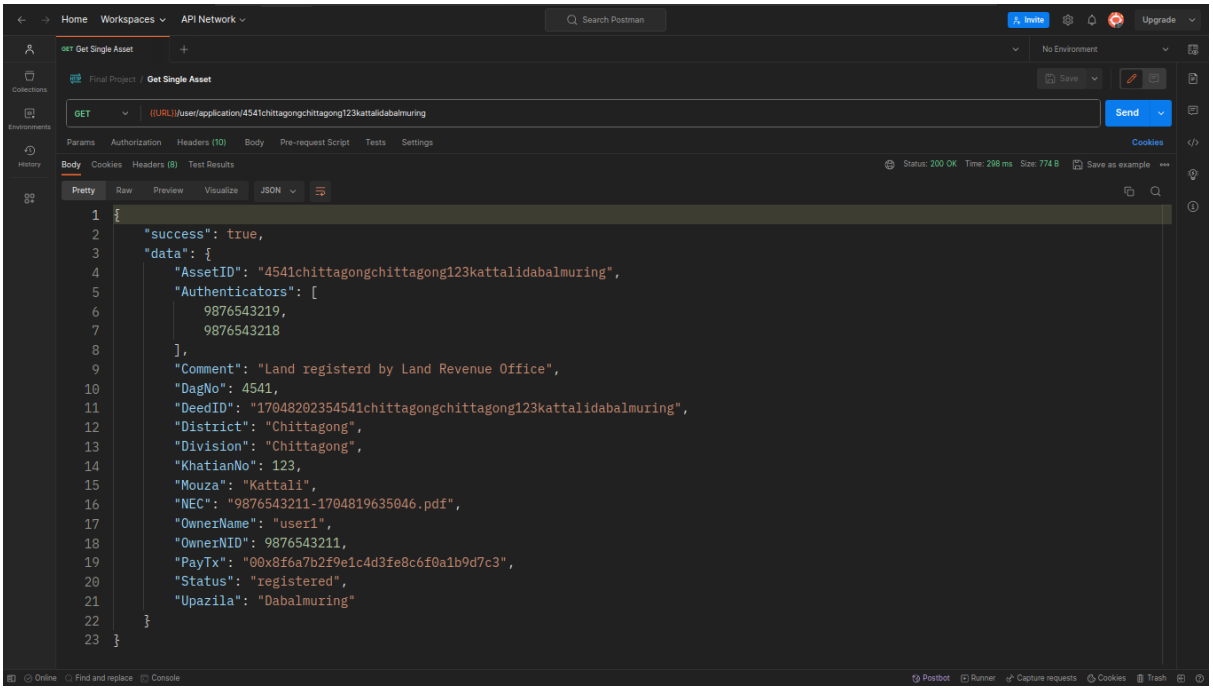

Fig. 4.8. API test of Land Application after Verification

Description: After getting verification from Ministry of Land and Land Revenue Office the application will have the verifier NID and a Deed ID. A land application after getting verified.

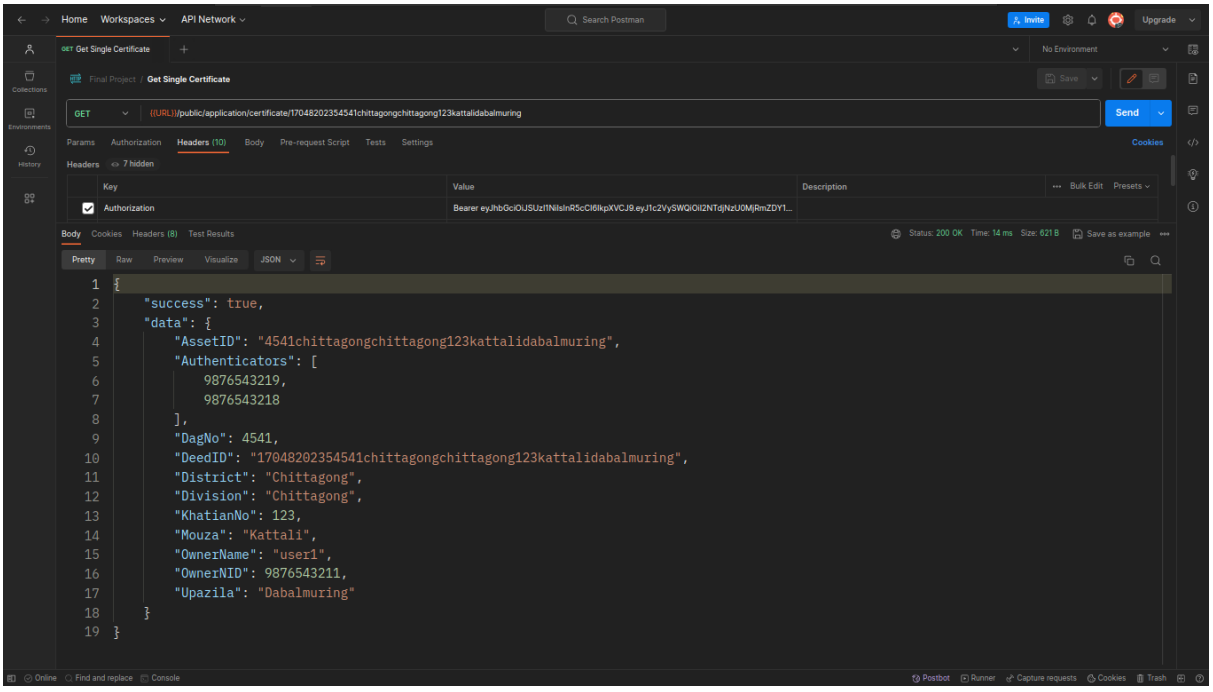

Fig. 4.9. API test of DEED of Registered Land

Description: Deed will be created of registered land. Deed of a registered land.

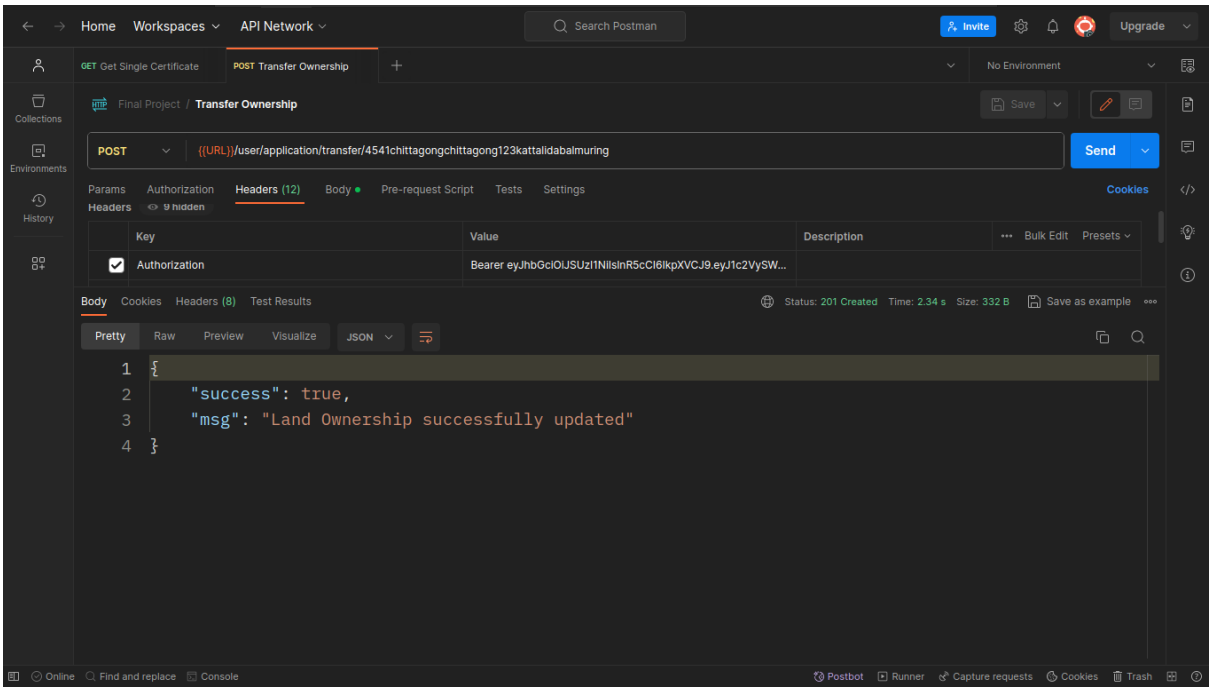

Fig. 4.10. API test of TransferLandOwnership

Description: Valid land owners can transfer or sell land to another valid user after logging in. Successful transfer of a land.

4.3 Tamperproof test:

Fig. 4.11. Test of Tamperproof

Description: Here we can see when we are going to change the data of the application we can see that the above error which is endorsement failure during invoke occurs and status shown 500 with message "Not authorized" shows

4.4 Response time test:

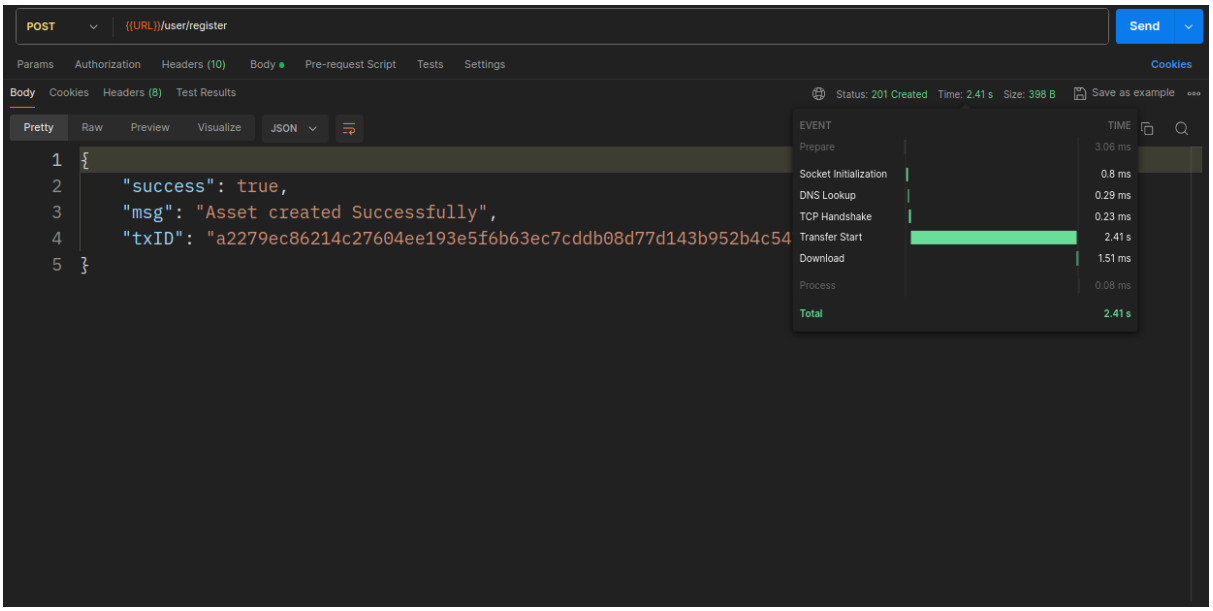

Fig. 4.12. Response time of Land Registration by user

Description: The above is the response time of Land registration taken by the user where Socket Initialization:0.8 ms,DNS Lookup: 0.29 ms,TCP Handshake:0.23 ms,Transfer Start:2.41 ms,Download:1.51 ms,prepare:3.06 ms,process:0.08 ms , total time taken is 2.41 seconds which is less time taken than existing land registration system

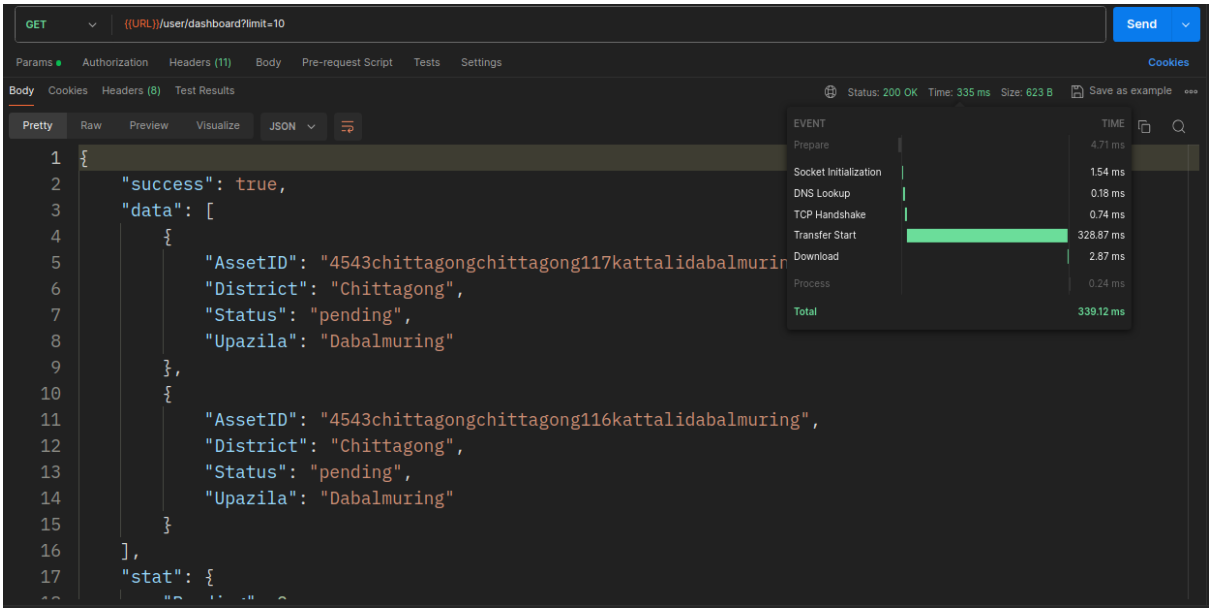

Fig. 4.13. Response time of User Dashboard

Description: The above is the response time of User Dashboard where Socket Initialization:1.54 ms,DNS Lookup: 0.18 ms,TCP Handshake:0.74 ms,Transfer Start:328.87 ms,Download:2.87 ms,prepare:4.71 ms,process:0.24 ms , total time taken is 339.12 ms which is less time taken than existing land registration system

4.5 Discussion:

The conducted tests indicate that users are able to log in to their accounts, submit new land registration applications, and have the application details securely stored on the ledger. Administrators from the Ministry of Land have the capability to retrieve pending applications, review the provided information, and approve applications that meet the necessary criteria. Subsequently, administrators from the Land Revenue Office can proceed to register the lands that have been approved by the Ministry of Land. Furthermore, if a client owns land registered on the blockchain, they have the option to transfer or sell ownership of the land.

Chapter 5

Conclusion

5.1 Summary of the findings

In conclusion, our project successfully implemented a decentralised land registration system for Bangladesh. By utilising the Raft consensus algorithm and Hyperledger Fabric, we've created a secure and transparent platform for citizens to register and manage land details. The system addresses concerns about transaction fees, ensures accessibility for all users, and plans for future population growth. Through testing, users were able to submit applications, and administrators could review and approve them seamlessly. The use of technologies like Hyperledger Fabric, Express.js, Node.js, Go, and MongoDB ensures data security and a robust framework. In essence, our project simplifies land registration, overcoming challenges in decentralised systems and providing a safe, transparent, and efficient solution for managing land records in Bangladesh.

5.2 Limitation and Future Work:

Presently, the organisation administrator utilises their CA certificate for ledger access, whereas users accessing the service through the client application use the organisation's CA certificate to access peers instead of their own CA certificate. In upcoming versions, we plan to shift to using the CA certificate for enrolling users in the service. This transition means that everyone will utilise their individual CA certificates for ledger access, effectively eliminating the necessity for MongoDB in user authentication.

5.3 Contribution

Md Rokib Khan played a crucial role in the project's success, leading the implementation of vital technologies such as Hyperledger Fabric, Go, and MongoDB. His expertise ensured robust data security, access control mechanisms, and the establishment of a strong framework. Integrating Hyperledger Fabric, Express.js, Node.js, Go, and MongoDB showcased his technical proficiency, creating a secure decentralized land registration system for Bangladesh. Ali Khatami's contribution is noteworthy for providing the project's main idea, conducting thorough research, and shaping the foundational framework. His comprehensive system design, including UML diagrams like ERD, use case, and sequence diagrams, added clarity and structure to the project. This design work laid out a well-defined roadmap for the development and implementation of the decentralized land registration system. Md Rafidul Islam significantly contributed to user experience, taking charge of front-end development using React.js, ensuring a seamless and user-friendly interface for engaging with the land registration system.

References

[1] Hyperledger Fabric, "The Ordering Service." https://hyperledger-fabric.readthedocs.io/en/latest/orderer/ordering_service.html. Accessed: 06/12/2023.

[2] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the thirteenth EuroSys conference*, pp. 1–15, 2018.

[3] M. Xu, X. Chen, and G. Kou, "A systematic review of blockchain," *Financial Innovation*, vol. 5, no. 1, pp. 1–14, 2019.

[4] A. Mardan and A. Mardan, "Using express. js to create node. js web apps," *Practical Node. js: Building Real-World Scalable Web Apps*, pp. 51–87, 2018.

[5] M. F. Karagoez and C. Turgut, "Design and implementation of restful wireless sensor network gateways using node. js framework," in *European Wireless 2014; 20th European Wireless Conference*, pp. 1–6, VDE, 2014.

[6] V. Thakur, M. Doja, Y. K. Dwivedi, T. Ahmad, and G. Khadanga, "Land records on blockchain for implementation of land titling in india," *International Journal of Information Management*, vol. 52, p. 101940, 2020.

[7] D. Fernando and N. Ranasinghe, "Permissioned distributed ledgers for land transactions; a case study," in *Business Process Management: Blockchain and Central and Eastern Europe Forum: BPM 2019 Blockchain and CEE Forum, Vienna, Austria, September 1–6, 2019, Proceedings 17*, pp. 136–150, Springer, 2019.

[8] N. Lazuashvili, A. Norta, and D. Draheim, "Integration of blockchain technology into a land registration system for immutable traceability: a casestudy of georgia," in *Business Process Management: Blockchain and Central and Eastern Europe Forum: BPM 2019 Blockchain and CEE Forum, Vienna, Austria, September 1–6, 2019, Proceedings 17*, pp. 219–233, Springer, 2019.

[9] K. M. Alam, J. A. Rahman, A. Tasnim, and A. Akther, "A blockchain-based land title management system for bangladesh," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 6, pp. 3096–3110, 2022.

[10] L. Zhang, L. Ci, Y. Wu, and B. Wiwatanapataphee, "The real estate time-stamping and registration system based on ethereum blockchain," *Blockchain: Research and Applications*, p. 100175, 2023.

[11] Hyperledger Fabric, "Planning for an ordering service." https://hyperledger-fabric.readthedocs.io/en/latest/deployorderer/ordererplan.html. Accessed: 06/12/2023.

[12] X. Piao, M. Li, F. Meng, and H. Song, "Latency analysis for raft consensus on hyperledger fabric," in *International Conference on Blockchain and Trustworthy Systems*, pp. 165–176, Springer, 2022.

[13] Hyperledger Fabric, "Channels." https://hyperledger-fabric.readthedocs.io/en/release-2.2/channels.html. Accessed: 07/09/2023.

[14] Hyperledger Fabric, "Peers." https://hyperledger-fabric.readthedocs.io/en/release-2.5/peers/peers.html. Accessed: 02/11/2023.

[15] Hyperledger Fabric, "Organization." https://hyperledger-fabric.readthedocs.io/en/latest/glossary.html#:~:text=to%20each%20Member.-,Organization,(MSP)%20to%20the%20network. Accessed: 06/11/2023.

[16] Hyperledger Fabric, "Fabric CA User's Guide." https://hyperledger-fabric-ca.readthedocs.io/en/release-1.4/users-guide.html#fabric-ca-server. Accessed: 06/11/2023.

[17] M. G. Santhosh and T. Reshmi, "Enhancing pki security in hyperledger fabric with an indigenous certificate authority," in *2023 IEEE International Conference on Public Key Infrastructure and its Applications (PKIA)*, pp. 1–5, IEEE, 2023.

[18] D. Westerveld, *API Testing and Development with Postman: A practical guide to creating, testing, and managing APIs for automated software testing.* Packt Publishing Ltd, 2021.

[19] Postman, Inc, "What is Postman?." https://www.postman.com/product/what-is-postman/. Accessed: 10/11/2023.

[20] OpenJS Foundation and expressjs.com contributors, "Fast, unopinionated, minimalist web framework for Node.js." https://expressjs.com/. Accessed: 10/11/2023.

[21] OpenJS Foundation and Node.js contributors, "About Node.js." https://nodejs.org/en/about. Accessed: 10/11/2023.

[22] C. Győrödi, R. Győrödi, G. Pecherle, and A. Olah, "A comparative study: Mongodb vs. mysql," in *2015 13th International Conference on Engineering of Modern Electric Systems (EMES)*, pp. 1–6, IEEE, 2015.

[23] MongoDB, Inc., "What Is MongoDB?." https://www.mongodb.com/what-is-mongodb. Accessed: 16/11/2023.

[24] Google, "The Go Programming Language Specification." https://go.dev/ref/spec. Accessed: 20/11/2023.

[25] J. Meyerson, "The go programming language," *IEEE software*, vol. 31, no. 5, pp. 104–104, 2014.

[26] Meta Platforms, Inc., "React." https://legacy.reactjs.org/. Accessed: 20/11/2023.

[27] A. Javeed, "Performance optimization techniques for reactjs," in *2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pp. 1–5, IEEE, 2019.

[28] ngrok, "What is ngrok?." https://ngrok.com/docs/what-is-ngrok/. Accessed: 02/11/2023.

[29] Google for Developers, "Make your app the best it can be." https://firebase.google.com/. Accessed: 10/12/2023.

[30] L. Moroney and L. Moroney, "An introduction to firebase," *The Definitive Guide to Firebase: Build Android Apps on Google's Mobile Platform*, pp. 1–24, 2017.

[31] Docker,Inc, "Docker overview." https://docs.docker.com/get-started/overview/. Accessed: 10/12/2023.

[32] T. Combe, A. Martin, and R. Di Pietro, "To docker or not to docker: A security perspective," *IEEE Cloud Computing*, vol. 3, no. 5, pp. 54–62, 2016.

[33] A. M. Potdar, D. Narayan, S. Kengond, and M. M. Mulla, "Performance evaluation of docker container and virtual machine," *Procedia Computer Science*, vol. 171, pp. 1419–1428, 2020.

[34] Docker,Inc, "Use containers to Build, Share and Run your applications." https://www.docker.com/resources/what-container/. Accessed: 22/12/2023.

[35] C. V. Helliar, L. Crawford, L. Rocca, C. Teodori, and M. Veneziani, "Permissionless and permissioned blockchain diffusion," *International Journal of Information Management*, vol. 54, p. 102136, 2020.

[36] Ministry of Land, "Record of Rights (Khatiyan) (RoR)." https://minland.gov.bd/site/page/c14f084e-8974-4255-8183-bae66d436ebb/. Accessed: 29/12/2023.

[37] Ministry of Land, "Land Development Tax: Civil." https://ldtax.gov.bd/. Accessed: 01/12/2023.

[38] Q. Li and Y.-L. Chen, "Entity-relationship diagram," in *Modeling and analysis of enterprise and information systems*, pp. 125–139, Springer, 2009.

[39] X. Li, Z. Liu, and H. Jifeng, "A formal semantics of uml sequence diagram," in *2004 Australian Software Engineering Conference. Proceedings.*, pp. 168–177, IEEE, 2004.