

DOCUMENT APPROVAL USING ETHEREUM

Submitted in fulfillment of the requirement of
the course Blockchain Technology by

Mounil Shah
(CE - 181070058)

Sarakshi Phate
(CE - 181071055)

Abhishek Kekane
(IT - 181080039)

Harsimran Virk
(IT - 181080031)

Shubhankar Gupta
(IT - 181080030)



**DEPARTMENT OF COMPUTER ENGINEERING AND
INFORMATION TECHNOLOGY**

VEERMATA JIJABAI TECHNOLOGICAL INSTITUTE

(An Autonomous Institute Affiliated to Mumbai University)

(Central Technological Institute, Maharashtra State)

Matunga, MUMBAI - 400019

TABLE OF CONTENTS

Sr. No.	Topic	Page No.
1	Introduction	3
2	Terminologies	4
3	Technology Stack	8
4	Architecture Diagram	9
5	Document Upload and Verify Smart Contract	11
6	Demonstration	17
7	Conclusion	26

1. INTRODUCTION

In an institution, document upload and verification is something that is always required. Often at times, the document may be submitted to a portal, and may have been forgotten to be accepted. We feel that there is a need to verify the authenticity of documents, and preserve the integrity of documents, so that it can be retrieved easily wherever and whenever needed.

In the world of blockchain technology, multisig wallets are no strangers. In fact, many classic cryptocurrencies like Bitcoin implement the functionality of a multisig wallet. With the advent of multisig wallets, it becomes very hard to obfuscate or worse, dupe people and steal their funds from a wallet. Ethereum does not natively support the concept of a multisig wallet, but where Ethereum does shine is in its capability to be extensible and flexible.

With the use of smart contracts and DApps, our project is designed to be an interface for both Students and Institutions, and provides a one stop destination for document upload and verification. Using the key concepts of a multisig wallet, our project ensures that no document is stored on the blockchain without consent of 2 parties - the Student, and the Institution.

2. TERMINOLOGIES

Blockchain

A blockchain is a distributed database that is shared among the nodes of a computer network. As a database, a blockchain stores information electronically in digital format. Blockchains are best known for their crucial role in cryptocurrency systems, such as Bitcoin, for maintaining a secure and decentralized record of transactions. The innovation with a blockchain is that it guarantees the fidelity and security of a record of data and generates trust without the need for a trusted third party.

One key difference between a typical database and a blockchain is how the data is structured. A blockchain collects information together in groups, known as blocks, that hold sets of information. Blocks have certain storage capacities and, when filled, are closed and linked to the previously filled block, forming a chain of data known as the blockchain. All new information that follows that freshly added block is compiled into a newly formed block that will then also be added to the chain once filled.

A database usually structures its data into tables, whereas a blockchain, like its name implies, structures its data into chunks (blocks) that are strung together. This data structure inherently makes an irreversible timeline of data when implemented in a decentralized nature. When a block is filled, it is set in stone and becomes a part of this timeline. Each block in the chain is given an exact time stamp when it is added to the chain.

Distributed Ledger

A distributed ledger is a database that is consensually shared and synchronized across multiple sites, institutions, or geographies, accessible by multiple people. It allows transactions to have public "witnesses." The participant at each node of the network can access the recordings shared across that network and can own an identical copy of it. Any changes or additions made to the ledger are reflected and copied to all participants in a matter of seconds or minutes.

A distributed ledger stands in contrast to a centralized ledger, which is the type of ledger that most companies use. A centralized ledger is more prone to cyber attacks and fraud, as it has a single point of failure.

Underlying distributed ledgers is the same technology that is used by blockchain, which is the technology that is used by bitcoin. Blockchain is a type of distributed ledger used by bitcoin.

Ethereum

Ethereum is a popular blockchain platform that is capable of executing Turing-complete programs called Smart Contracts. Ethereum also has its own cryptocurrency for monetary interactions called Ether. Ethereum and other blockchains store a massive amount of heterogeneous data

Ethereum platform only allows direct access to its first-class data elements, which includes blocks, transactions, accounts, and contracts

The information returned by the Ethereum platform when we access its blocks is encoded into a JSON-like structure. If we would like to search for a particular information inside a data element, we would need a unique identifier to access the block containing such information.

Ethereum platform does not provide a semantic way to search for information and neither an easy form to present such information

Smart Contracts

Smart contracts are the key element of Ethereum. In them any algorithm can be encoded. Smart contracts can carry arbitrary state and can perform any arbitrary computations

They are even able to call other smart contracts. This gives the scripting facilities of Ethereum tremendous flexibility. Smart contracts are run by each node as part of the block creation process

Just like Bitcoin, block creation is the moment where transactions actually take place, in the sense that once a transaction takes place inside a block, the

global blockchain state is changed. In contrast to Bitcoin, Ethereum follows a different pattern for selecting which blocks get added to the valid blockchain

Accounts

An Ethereum account is an entity with an ether (ETH) balance that can send transactions on the network. Ethereum has 2 types of accounts,

- Externally Owned Account - Controlled by any user for the purpose of transacting ETH cryptocurrency.
- Contract Account - A smart contract deployed on the network and controlled by the code.

Transactions

Transactions are cryptographically signed instructions from accounts. An account will initiate a transaction to update the state of the Ethereum network. The simplest transaction is transferring ETH from one account to another.

Transactions, which change the state of the EVM, need to be broadcast to the whole network. Any node can broadcast a request for a transaction to be executed on the EVM; after this happens, a miner will execute the transaction and propagate the resulting state change to the rest of the network. Transactions require a fee and must be mined to become valid.

Wallet

Wallets secure funds by guarding our private keys. These private keys act as the proof of ownership for our coins. A Bitcoin address is like an account number - the address denotes which wallet the coins should be sent to. Like a bank account number, where the difference lies in the wallets having multiple addresses.

However, one needs to double check the target address while sending. This is because Bitcoin transactions cannot be reversed, so one would lose his/her coins forever to a stranger!

MultiSig Wallet

A multi-signature wallet ("multisig" for short) is a cryptocurrency wallet that requires two or more private keys to sign and send a transaction. This type of digital signature makes it possible for two or more users to sign documents as a group. Co-owners and signatories to a shared multisig wallet are known as "copayers."

The number of signatures required to sign a transaction is dependent on the kind of wallet. It may be lower or equal to the number of copayers of the wallet.

Decentralised App (DApp)

A decentralized application (dapp) is an application built on a decentralized network that combines a smart contract and a frontend user interface. A dapp has its backend code running on a decentralized peer-to-peer network. A dapp can have frontend code and user interfaces written in any language (just like an app) to make calls to its backend.

Metamask

MetaMask is a bridge that allows us to visit the distributed web of tomorrow in our browser today. It allows us to run Ethereum dApps right in our browser without running a full Ethereum node.

3. TECHNOLOGY STACK

Frontend

React.js

React.js is a javascript library for building user interfaces. It allows us to develop UIs in a declarative manner and takes care of updating the application state whenever any data changes. It encourages component driven development which makes our codebase more modular.

Material UI

MUI provides a robust, customizable, and accessible library of foundational and advanced components, enabling us to build our own design system and develop React applications faster.

Blockchain

Web3.js

Web3.js is a JavaScript library that talks over Ethereum nodes. This could be a locally deployed Ethereum network or the live chain. It can be used to access information about tokens and Ether coins. One can also deploy their own application and access using Web3. This library enables you to connect your JavaScript-based frontend to the Ethereum network using HTTP, IPC, and WebSockets.

Truffle

Truffle is a world class development environment, testing framework, and asset pipeline for blockchains using the Ethereum Virtual Machine (EVM). Truffle also includes support for custom deployments, library linking, and complex Ethereum applications.

Metadata Storage

IPFS

The InterPlanetary File System, or simply IPFS, is a protocol and peer-to-peer network for storing and sharing data in a distributed file system. IPFS uses content-addressing to uniquely identify each file in a global namespace connecting all computing devices, making the web upgradeable, resilient, and more open.

4. ARCHITECTURE DIAGRAM

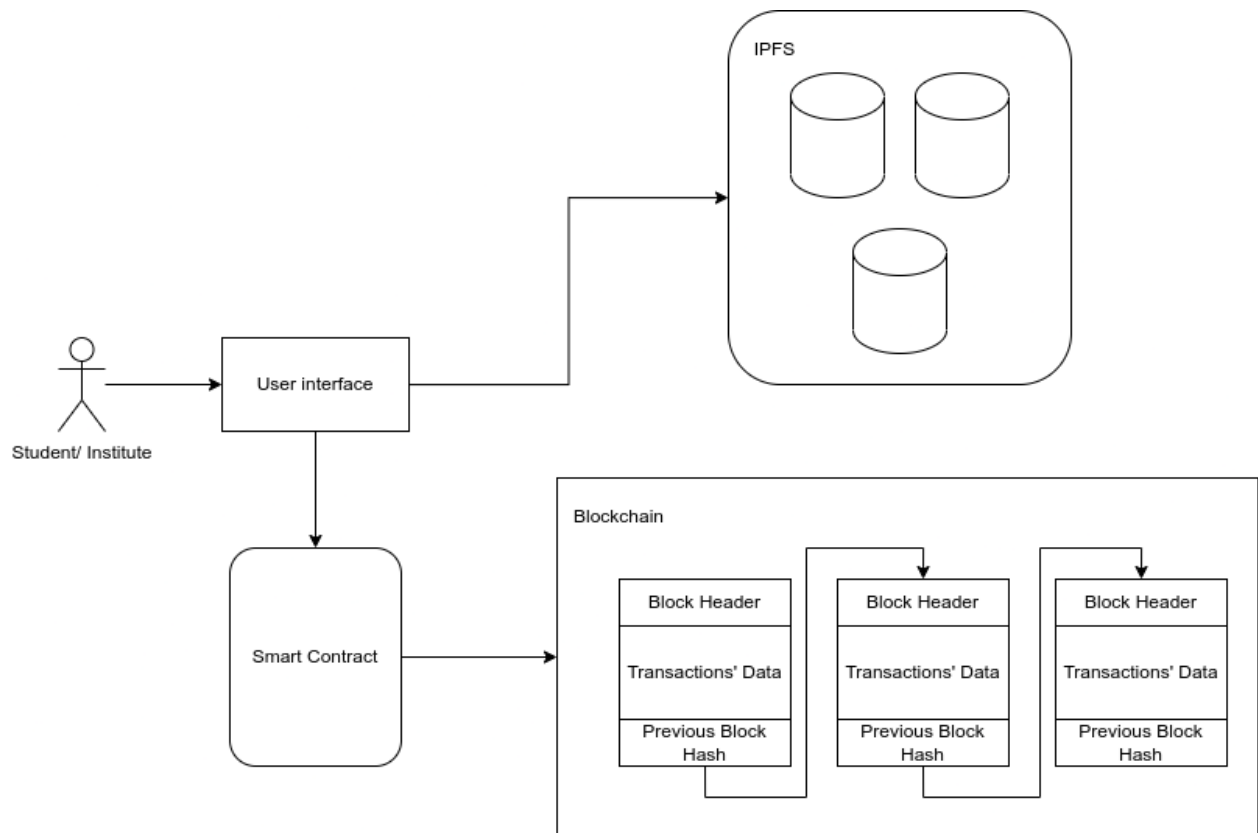


Fig 1. System Architecture of Document Verification and Upload using Ethereum

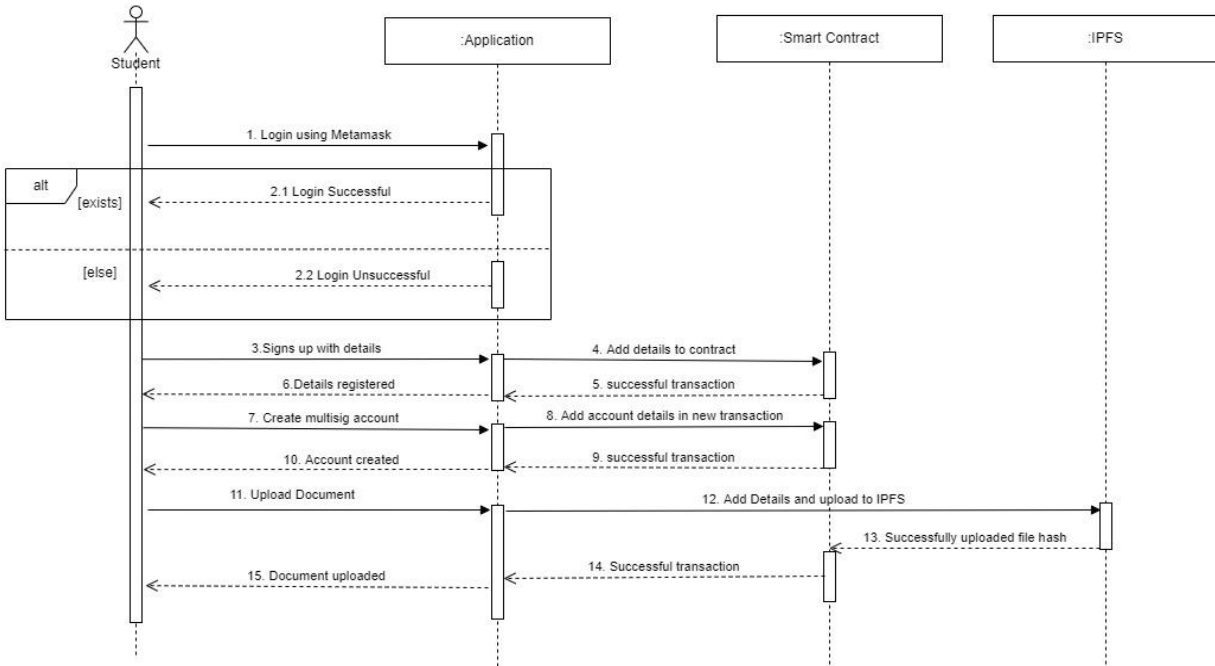


Fig 2. Sequence Diagram for uploading document by user

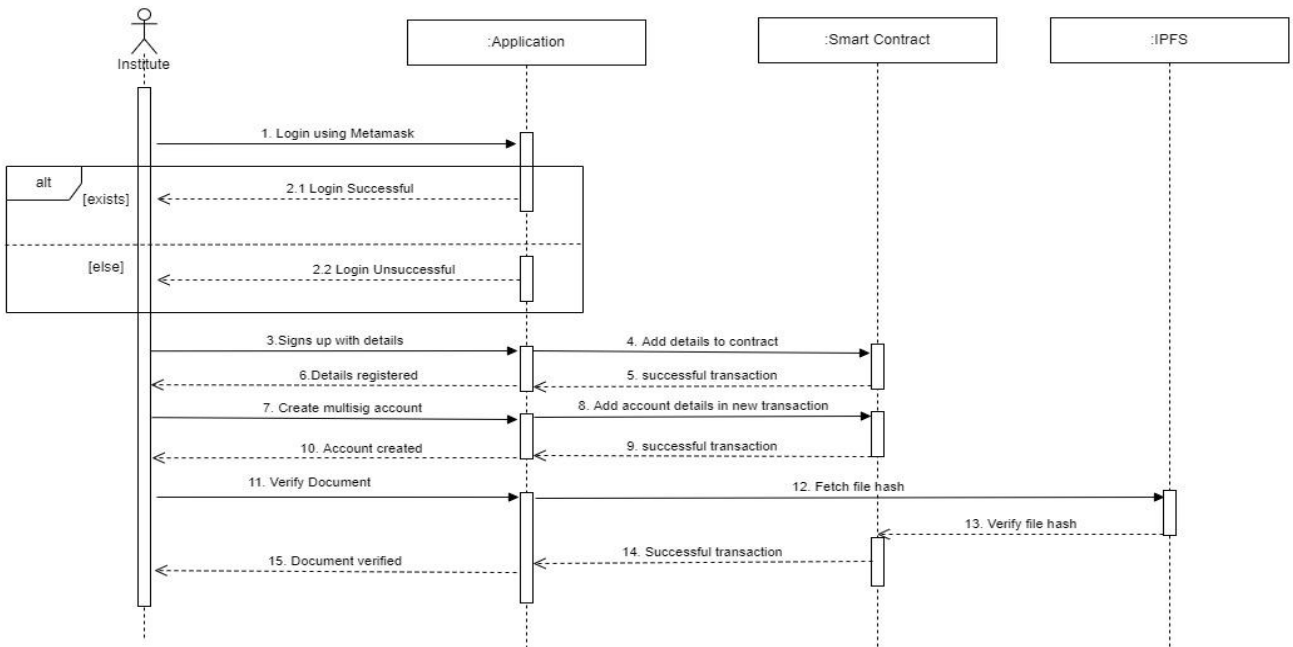


Fig 3. Sequence Diagram for verification of document by institute

5. SMART CONTRACT

1. Generate structs for our project:

```
struct Documents {
    string aadhar;
    uint8 aadharno;
    string aadharnoHash;
}

struct PersonallInfo {
    string name;
    string propic;
    string email;
    string phone;
}

struct Owners {
    uint8 max;
    address stud;
    address inst;
}

struct Request {
    bool stud;
    bool inst;
    bool aadhar;
}

struct ApprovalRequest {
    bool stud;
    bool inst;
}

struct ChangeOwnership {
```

```

    bool stud;
    bool inst;
    address studa;
    address insta;
    ApprovalRequest approve;
}

struct ChangeOwnerShipInst {
    bool inst;
    bool stud;
    address neinst;
    address nestud;
    uint256 timestamp;
}

struct Upload {
    ApprovalRequest approve;
    bool aadhar;
    string aadharhash;
    uint256 timestamp;
}

struct MultiSig {
    Owners owners;
    Documents doc;
    PersonalInfo profile;
    mapping(address => Upload) uploadreq;
    address[] listofuploadreq;
    address[] listofchangeowneri;
    mapping(address => ChangeOwnerShipInst) changeowneri;
    address[] ihaveaccess;
    address[] PastOwners;
}

mapping(address => MultiSig) wallets;
mapping(address => bool) listofwall;

```

```
mapping(address => address[]) iminwall;  
address public j;
```

2. Functions to create different entities for our project

```
function createNewMultiSigInst(address a) public {  
    wallets[msg.sender].owners.stud = a;  
    wallets[msg.sender].owners.inst = msg.sender;  
    listofwall[msg.sender] = true;  
    iminwall[msg.sender].push(a);  
}  
  
function createUploadRequestbyUser(bool ad, string memory hash) public {  
    address inst = wallets[msg.sender].owners.inst;  
    wallets[msg.sender].uploadreq[inst].approve.stud = true;  
    wallets[msg.sender].uploadreq[inst].approve.inst = false;  
    wallets[msg.sender].uploadreq[inst].aadhar = ad;  
    wallets[msg.sender].uploadreq[inst].aadharhash = hash;  
    wallets[msg.sender].listofuploadreq.push(msg.sender);  
    wallets[msg.sender].uploadreq[inst].timestamp = now;  
}  
  
function createUploadRequestbyInstitute(  
    address a,  
    bool ad,  
    string memory hash  
) public {  
    wallets[a].uploadreq[a].approve.inst = true;  
    wallets[a].uploadreq[a].approve.stud = false;  
    wallets[a].uploadreq[a].aadhar = ad;  
    wallets[a].uploadreq[a].timestamp = now;  
    wallets[a].uploadreq[a].aadharhash = hash;  
    wallets[a].listofuploadreq.push(msg.sender);  
    wallets[a].doc.aadhar = hash;  
}  
  
function createNEwAccess(address stud, address ad) public {
```

```

wallets[stud].owners.listofpartialowner.push(ad);

wallets[ad].ihaveaccess.push(stud);

wallets[ad].PastOwners.push(ad);
}

```

3. Create getters for our project:

```

function getUploadReqList(address ad)
    public
    view
    returns (address[] memory)
    {
        return wallets[ad].listofuploadreq;
    }

function getAadhar(address stud) public view returns (string memory) {
    return wallets[stud].doc.aadhar;
}

function getOwners(address a) public view returns (address, address) {
    return (wallets[a].owners.stud, wallets[a].owners.inst);
}

function getInstitutesWallet(address ad)
    public
    view
    returns (address[] memory)
    {
        return iminwall[ad];
    }

function getInstitutesUploadList(address ad)
    public
    view
    returns (address[] memory)

```

```

    {
        return wallets[ad].listofuploadreq;
    }

function getChangeOwnerList(address a)
    public
    view
    returns (address[] memory)
    {
        return wallets[a].listofchangeowneri;
    }

function getProfile(address a)
    public
    view
    returns (string memory, string memory)
    {
        return (wallets[a].profile.name, wallets[a].profile.propic);
    }

function getIhaveaAccess(address a) public view returns (address[]
memory) {
    return wallets[a].ihaveaccess;
}

function getUploadReqPic(address stud, address inst)
    public
    view
    returns (string memory)
    {
        return wallets[stud].uploadreq[inst].aadharhash;
    }

```

4. Functions for approvals:

```

function approveUploadbyUser(address ad) public {
    wallets[msg.sender].uploadreq[ad].approve.stud = true;
}

```

```

if (
    wallets[msg.sender].uploadreq[ad].approve.stud &&
    wallets[msg.sender].uploadreq[ad].approve.inst
) {
    wallets[msg.sender].doc.aadhar = wallets[msg.sender]
        .uploadreq[ad]
        .aadharhash;
    delete wallets[msg.sender].listofuploadreq[
        wallets[msg.sender].listofuploadreq.length - 1
    ];
}
}

```

```

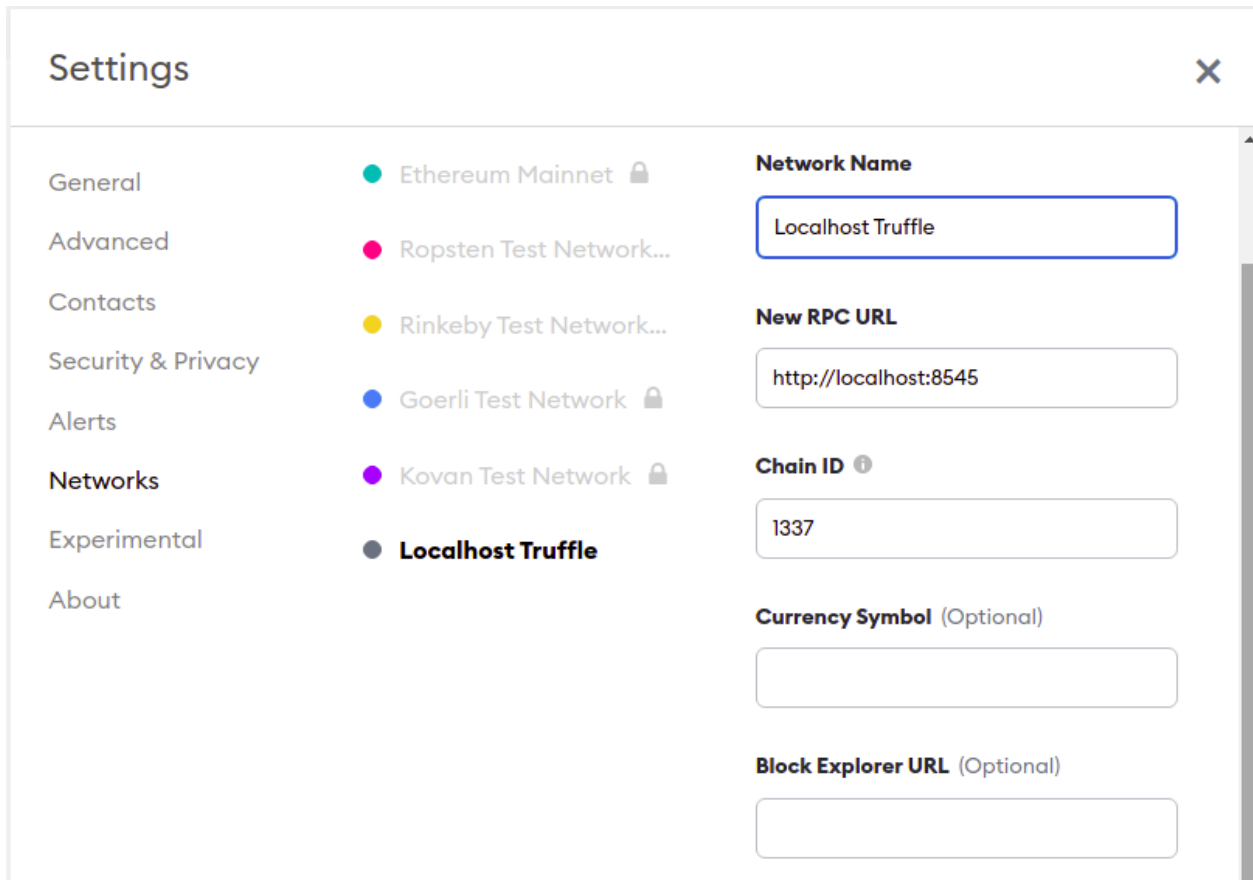
function approveUploadbyInstitute(address ad, address student) public
returns (bool) {
    wallets[student].uploadreq[ad].approve.inst = true;
    if (
        wallets[student].uploadreq[ad].approve.stud &&
        wallets[student].uploadreq[ad].approve.inst
    ) {
        wallets[student].doc.aadhar = wallets[student]
            .uploadreq[ad]
            .aadharhash;

        delete wallets[student].listofuploadreq[
            wallets[student].listofuploadreq.length - 1
        ];
        return true;
    } else {
        return false;
    }
}
}

```


6. DEMONSTRATION

1. We use truffle develop, which spins up a local blockchain for use Metamask to connect it to the truffle develop blockchain.



The screenshot shows the Metamask 'Settings' dialog with a sidebar on the left containing links: General, Advanced, Contacts, Security & Privacy, Alerts, Networks, Experimental, and About. The 'Networks' section is active, displaying a list of networks. The 'Localhost Truffle' network is selected and highlighted in bold. To the right of the list, the configuration fields for the selected network are shown: 'Network Name' (Localhost Truffle), 'New RPC URL' (http://localhost:8545), 'Chain ID' (1337), 'Currency Symbol (Optional)' (empty), and 'Block Explorer URL (Optional)' (empty).

Network	Network Name	New RPC URL	Chain ID	Currency Symbol (Optional)	Block Explorer URL (Optional)
Ethereum Mainnet					
Ropsten Test Network...					
Rinkeby Test Network...					
Goerli Test Network					
Kovan Test Network					
Localhost Truffle	Localhost Truffle	http://localhost:8545	1337		

Fig 4. Add new blockchain network

< Back

Restore your Account with Secret Recovery Phrase

Only the first account on this wallet will auto load. After completing this process, to add additional accounts, click the drop down menu, then select Create Account.

If you restore using another Secret Recovery Phrase, your current wallet, accounts and assets will be removed from this app permanently. This action cannot be undone.

Wallet Secret Recovery Phrase

☐

Show Secret Recovery Phrase

New password (min 8 chars)

Confirm password

Fig 5. Import wallet to access accounts

2. Once we have Metamask setup, we need to create 1 institute and 2 students. The Ethereum addresses of these are as follows:

Name	Address
VJTI (Institution)	0x7b425260bfd9315de6c816cf11418f729bb77826
Student1 (Student)	0x7b27c07af9f7872d75befbc1602743d8487fe3ec
Student2 (Student)	0xda559f55c151f07d74fce39f94fa9de3b1526725

Once we setup our DApp, we will see the landing page:



Fig 6. Landing Page

And on clicking Login, we see the Login page

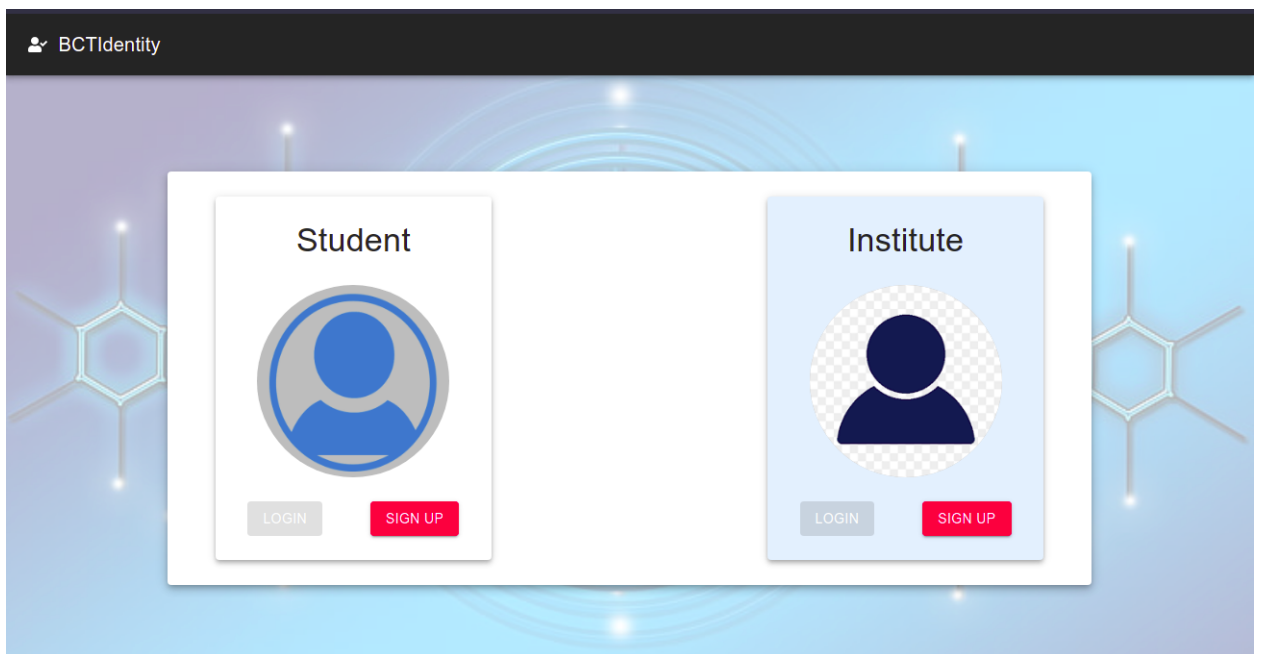


Fig 7. Login Page

Create New Account

Enter your Name

Name

VJTI

Enter your Email

Email

admin@vjti.ac.in

Enter your Phone Number

Phone

9856742130

Upload a picture

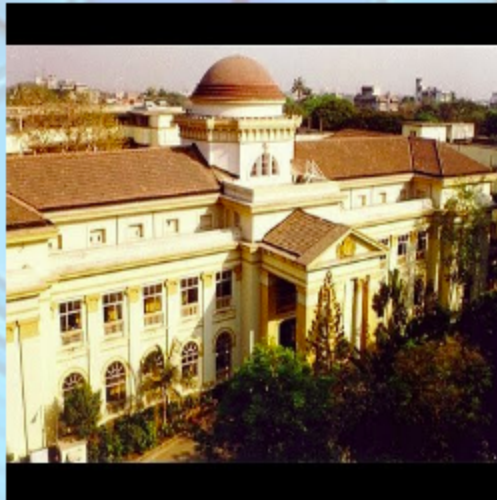


Fig 8. Add Institute Page

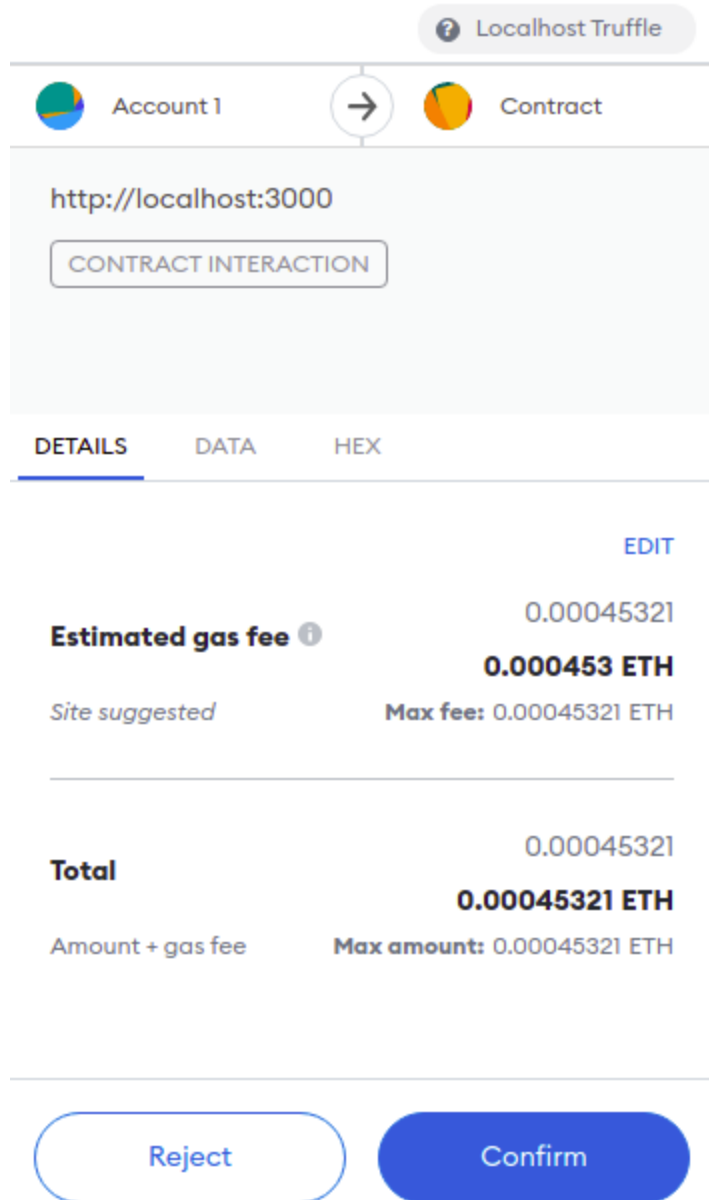


Fig 9. Confirm the transaction on Metamask

Similarly, we create accounts for Students. The only difference is that for students, we will create multisig wallets that will have the 2nd address as that of the Institute.

Create New MultiSig Wallet!

Enter Institute Address:

Your Address:

☒ I have read and Agree to All the [Terms And Conditions](#).

GO!

Fig 10. Create new Multisig account with institute

Now we can see the linked accounts on the Institute dashboard

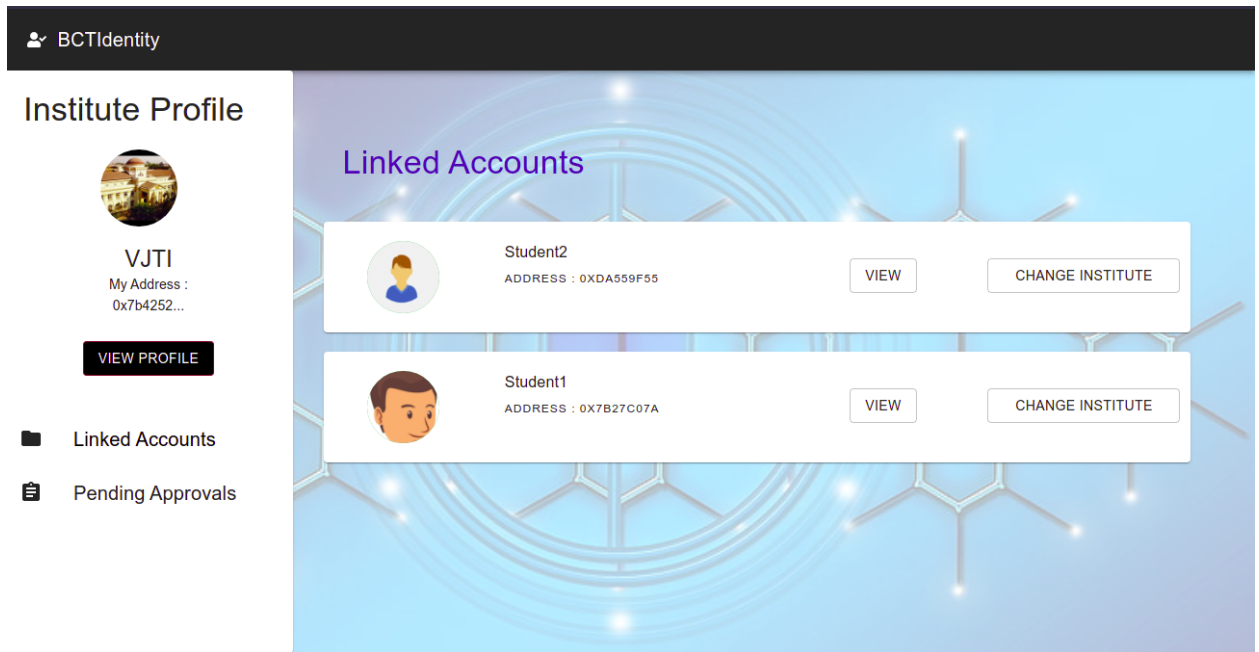


Fig 11. Institute Dashboard

3. Create and Upload Document flow:

First, a student uploads the document:

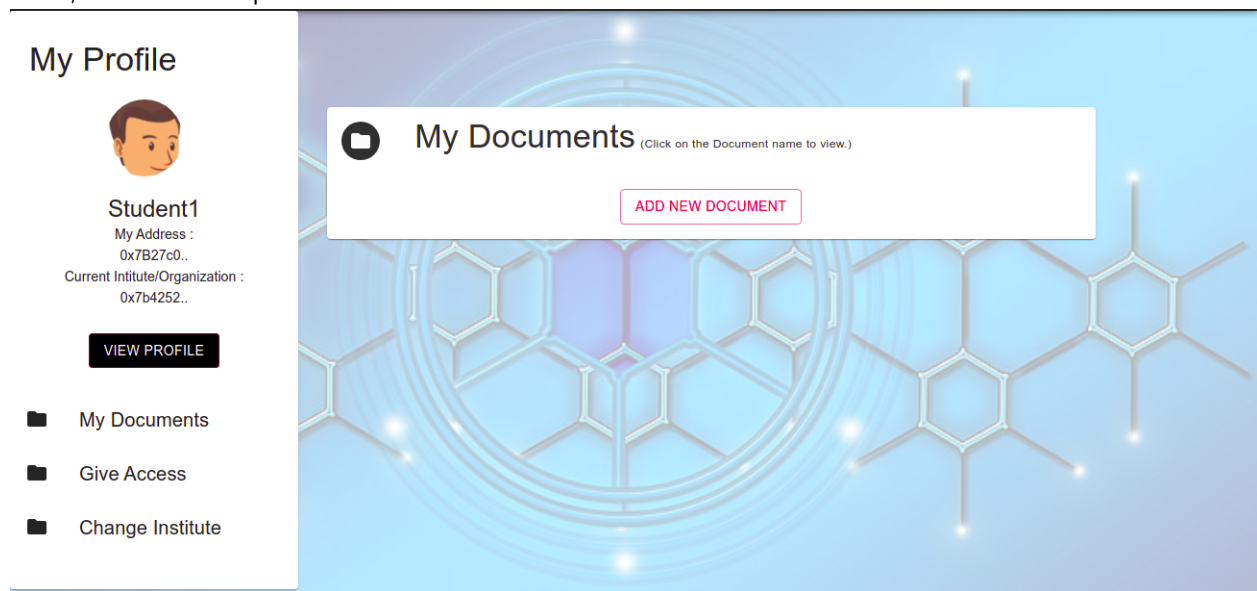
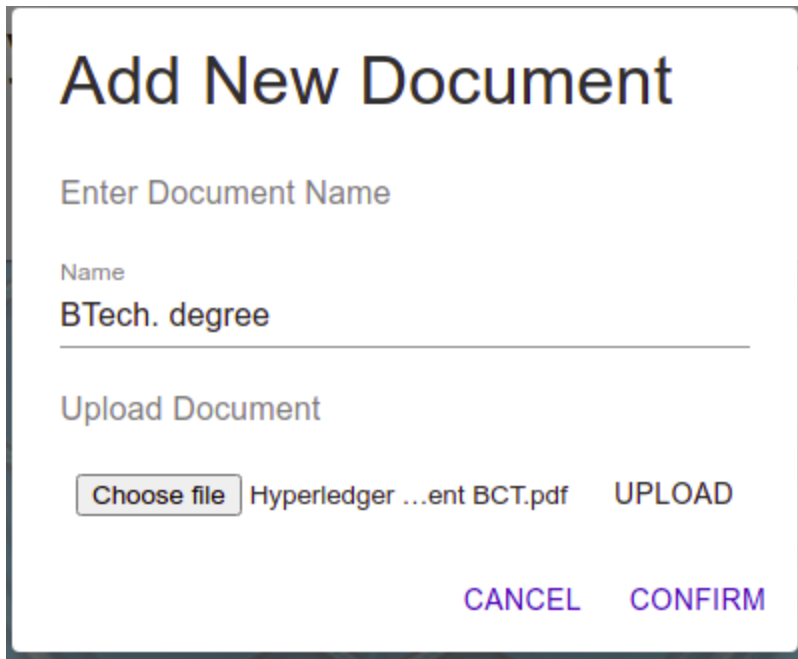


Fig 12. Student Dashboard



Add New Document

Enter Document Name

Name
BTech. degree

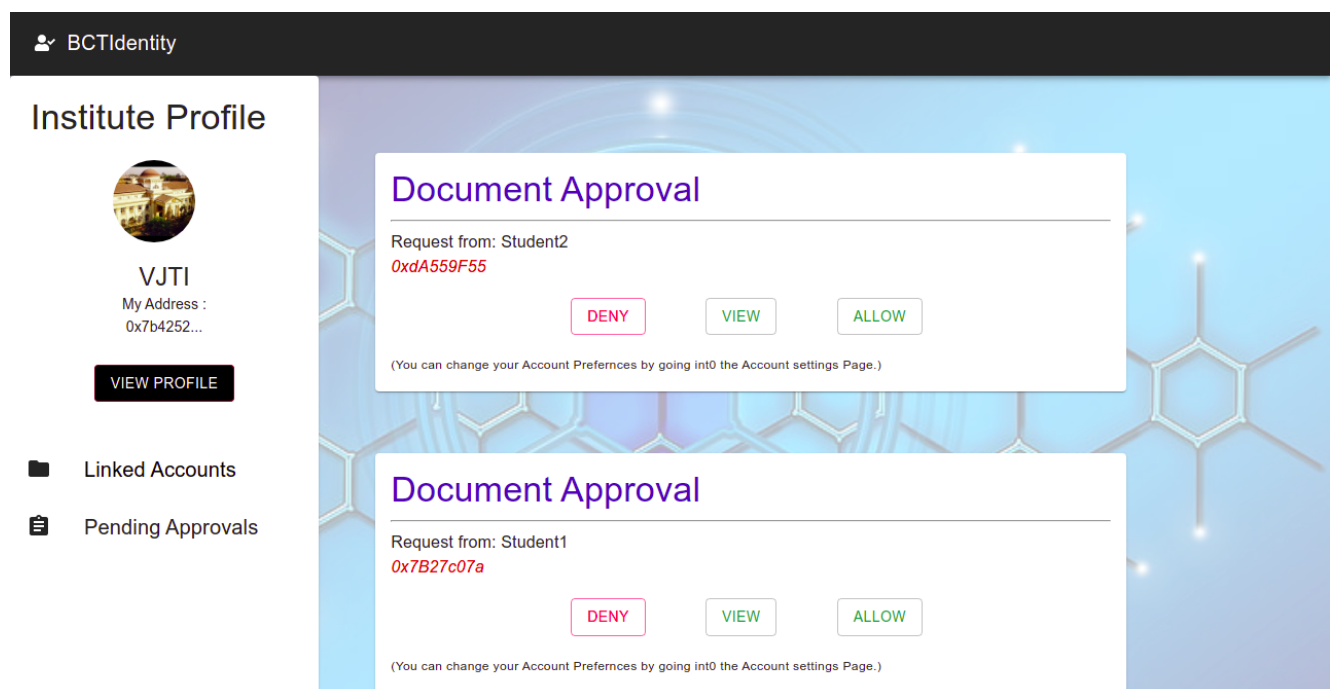
Upload Document

Choose file Hyperledger ...ent BCT.pdf UPLOAD

CANCEL CONFIRM


Fig 13. Add New Document Modal

We will do the same for Student2. We will now use the Institute's account for approving the documents.



BCTIdentity

Institute Profile


VJTI
My Address :
0x7b4252...

VIEW PROFILE

- Linked Accounts
- Pending Approvals

Document Approval

Request from: Student2
0xdA559F55

DENY VIEW ALLOW

(You can change your Account Preferences by going into the Account settings Page.)

Document Approval

Request from: Student1
0x7B27c07a

DENY VIEW ALLOW

(You can change your Account Preferences by going into the Account settings Page.)

Fig 14. Pending Approvals page.

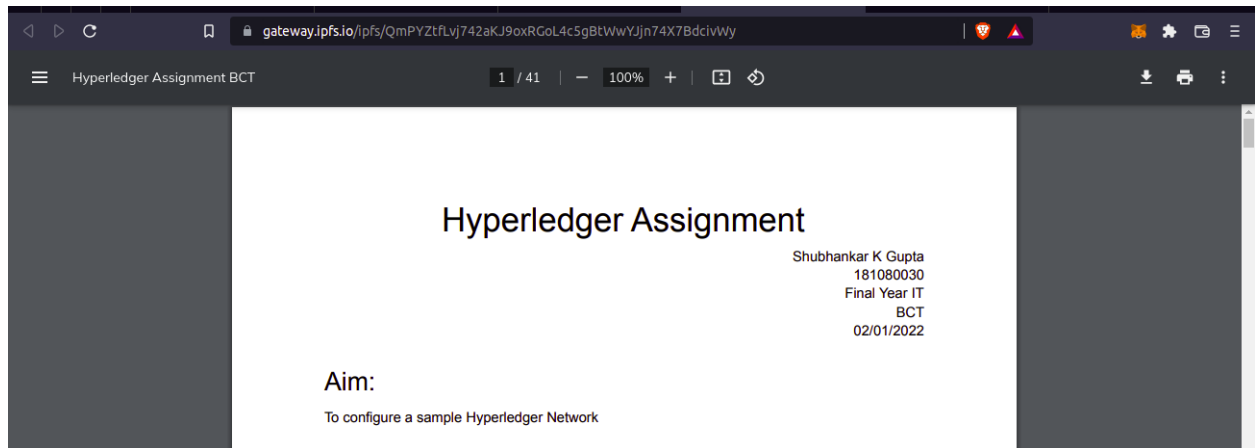


Fig 15. View Document from IPFS

On clicking on allow, we will approve the document. Since we have obtained 2 approvals, the Documents will now be available to the Students.

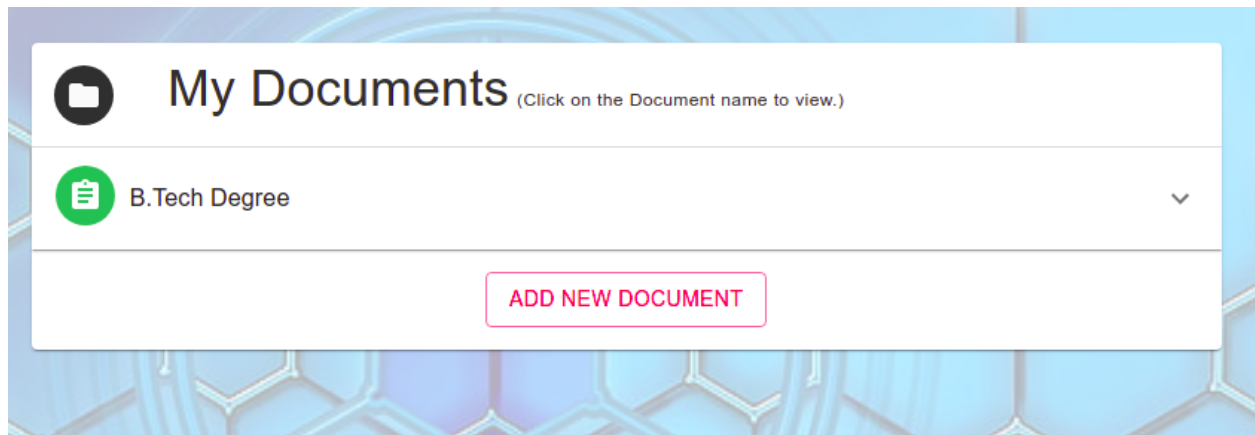


Fig 16. Document now available in My Documents for Student1

This completes the flow of our website.

7. CONCLUSION

Blockchain provides us a way to store data on a distributed ledger, and the ability to verify its contents using a distributed consensus algorithm. Ethereum allows us to extend the capabilities of our blockchain with the advent of smart contracts.

With newer ways of conning people to let out their crypto assets, a multisig wallet has proven to be the de-facto standard for securing assets. We have combined the power of a multisig wallet to securely upload and verify documents. We have developed a website and a smart contract that can be used by Institutions and Students for securely uploading and approving documents.