Introduction

by Pinaki Bhattacharjee

Submission date: 14-Apr-2022 07:08PM (UTC+0530)
Submission ID: 1810596643
File name: DigiBlock_merged.pdf (3.95M)
Word count: 13947
Character count: 73008

DigiBlock

Chapter 1

Introduction

People's lives have become increasingly reliant on social networking sites. Every day, it generates and processes a large amount of data. These services record a variety of sensitive user information such as age, gender, location, preferences, banking credentials, and forensics such as fingerprints, retinal scans, and faceIds. A user's virtual identity is made up of a combination of these and other data values.This information about the user is frequently transported across networks, making it accessible to people or organizations who do not have the necessary permissions to access, use, or modify the data. Such a breach of personal data could result in financial losses, as well as erode user trust and the integrity of online transactions.

According to a research, India placed third in the world in terms of data breaches, with a net of 86.63 million Indian consumers breached till November 2021. Surfshark, a cybersecurity service, employed its own breach detection mechanism in its report. In India, the number of affected accounts increased by 351.6% over the past year. Around 19.18 million Indian users' data was compromised in 2020.

Several situations in the real world, such as visiting a bank, signing up for network services, obtaining healthcare or health coverage benefits, voting, and so on, require users to present official identification documents such as a security card, driver's license, or proof of identity to authenticate their individuality. People must physically possess their credentials, and they may be required to submit duplicates of these manuscripts. Once the documents are forwarded for additional validation, there's always the risk that they'll end up in the wrong hands and the confidential material contained therein will be exploited for illegal or unethical purposes.

We propose a decentralized identity ledger based on blockchain to address the problem of identity management with regard to user's information. The wide range of functionalities that blockchain provides is why it was chosen to accomplish this functionality. Data is separated and disseminated among several nodes, just as the database's decentralization assures, and that there is no single point of failure. Furthermore, the necessity of consensus among all nodes or even the majority of nodes within the network makes changing the rules governing access to data a lengthy and complicated process for malevolent nodes. Due to blockchain's scalability, new users can join the chain, create an identity, and set access control policies for their private information. Because of the blockchain's openness, every information must be

hashed before being stored on the network. Because data is encrypted, the genuine information is kept safe and concealed from the other nodes.

Our suggested solution intends to empower end users by allowing them to own and govern their personal information. The end-user can grant, alter, and remove access to the private documents using the accessibility control mechanism. We have developed a Decentralized WebApp to illustrate our concept. The Interplanetary File System(IPFS) API was utilized to store data files, and the metamask extension must have been installed to access ethereum-enabled distribution applications.

Chapter 2

Basic Concepts / Literature Review

A blockchain is a decentralized, distributed, and public digital ledger composed of records called blocks that record transactions and addresses across many computers in such a way that any modification to one block affects all future blocks. This can be used for a variety of purposes, including transaction recording, asset tracking, and trust building. This technology allows participants to verify and perform transactions at a low cost on their own. A peer-to-peer network and a shared timestamping server are used to self-manage a blockchain database. In instances when participants' concerns about data security are low, a design like this fosters a stable workflow. The usage of a blockchain eliminates the possibility of a digital item to be indefinitely replicated. A blockchain is a protocol for transferring value.

Blocks in a blockchain network are used to store information of valid transactions that have been hashed and encoded into a Merkle tree. Each block contains the cryptographic hash of the address of the previous block in the blockchain, which connects the two. A chain is formed by linking these blocks. This recursive procedure verifies the integrity of each subsequent block, all the way down to the genesis block. A block is digitally signed to confirm the authenticity of the values contained within it.

Separate blocks which are produced parallely, can result in a temporary fork. Aside from a secure hash-based history, every blockchain features a preset method for scoring multiple versions of the past so that the one with the greatest score may be picked. Orphan blocks are those blocks that were not chosen for involvement in the chain. Peers in the database have various versions of the history from time to time. They only maintain the version of the database with the highest score that they have access to. When a peer receives a higher-scoring version (often the old version with one additional block added), they expand or edit their own database and retransmit the changes to their peers. There is never a guarantee that any particular entry will remain in the best version of history in the coming time. The score of new blocks is often added to the score of existing blocks, with incentives to extend with new blocks rather than overwrite old blocks. As a result, as more blocks are stacked on top of one entry, the chance of it being surpassed falls rapidly, finally becoming exceedingly low. Bitcoin, for example, has a proof-of-work method in which the network accepts the chain with the highest cumulative proof-of-work. There are numerous ways to show

that a minimal level of computation has been fulfilled. Instead of being separated and parallel, computing on a blockchain is done redundantly.

The average amount of time it takes the network to add another block in the blockchain is block time. Every five seconds or fewer, certain blockchains create a new block. The included data is verifiable by the time the block is finished. This is when the transaction really happens in bitcoin, hence a low block time equals speedier transactions. The block time for Ethereum is expected to be between 14 and 15 seconds, while the average block time for bitcoin is 10 minutes.

The web has evolved significantly in recent years, and its operating technologies are almost entirely unknown from its early days. Web appearance is typically classified into three levels: Web 1.0, Web 2.0, and Web 3.0.

The initial version of the internet was Web 1.0. Sites with static content rather than dynamic HTML were included. There is limited interaction between the sites, and data and content are supplied through a static file system rather than a database. Between 1991 and 2004, there was a period known as Web 1.0. You don't have to be a designer to participate in the Web2 design process. The majority of applications are created so that anybody can simply become a maker. Web2 and Web3 have a few essential changes, but power allocation is at the heart of the problem.

Web2 programmers design and execute apps on the same server or store their data in a single database, which is commonly hosted and managed by a single cloud provider. Instead, Web3 apps use the crypto-economic protocol's blockchains, servers , or a combination of the two.

Decentralized apps (DApps) with Web3 support are now accessible. DApps are programmes that exist and utilize a blockchain or peer-to-peer network of computers rather than a single computer, and are not subject to a single authority's view and control.

Web 3.0 is a version of the Internet in which users have control over their data, ownership, and money. Web3 can modify contracts and exchange rates. It alters the data structure underlying the Internet by introducing a global domain and incentivizing network participants with a brand.
A centralized network has a single point of failure because it has a single source of truth. The current state of the network is referred to as "Data Monarchy," whereas the current state of the decentralized web, where data is distributed, is referred to as "Data Democracy".

Dapps have their own backend programming that runs on a peer-to-peer network of their own, in contrast to an application where the backend code is consolidated on a single server. A dapp can have frontend code and a user interface written in any language that can connect with the backend (like the normal Webapp). Its front end code can also be hosted on a shared storage system like IPFS.

Off chain storage can enable scalability for block chain storage infrastructure: Off-chain data is any data that is too massive to be stored efficiently in the Block chain or requires the ability to be changed or wiped. Any organized or unstructured data that cannot be kept in a blockchain. Media and documentation files, such as JPEGs(images) and text files(pdfs), are a few examples. This workshop will also address the components of storage infrastructure essential for providing off-chain storage, as storage infrastructure is a critical feature in the block chain environment: Use Flash/NVME technology for performance. Off-chain storage relieves the block chain of the load of keeping huge datasets, but because real discs are sluggish, this might have an impact on performance. The ability to expand up/out to petabytes with ease We can simply scale up by adding more disks/memory, and scale out by adding more nodes to the storage cluster, allowing us to store petabytes of data. With storage product functionalities, backup and recovery through snapshot or continuous data synchronization are also conceivable. We recommend backing up off-chain data with virtual copies [snapshots] for quick recovery to avoid losing it. Replication solutions like these can provide continuous data synchronization. Data reduction capability built-in with deduplication/compression. Space can be saved by employing deduplication and compression technology. This will allow the system's overall storage to be used more efficiently. A vendor-specific solution will be demonstrated.

While blockchain is primarily a storage solution, it is not the same as a database. A database is a structured collection of information that represents the current state of a system. The primary purpose of a database is to facilitate data retrieval, fusion, and aggregation in response to user queries. Most blockchain implementations, on the other hand, constitute a ledger where a history of transactions (or, more broadly, changes to the system state) is kept. In Bitcoin, for example, there is no such thing as a user balance! While Ethereum keeps track of a contract's state, it only allows for restricted retrieval and processing of state data as stated by the contract. It lacks abstractions such as a flexible query language, data view, schema, join, and so on. Furthermore, Ethereum keeps track of all state changes, making blockchain space more expensive and storage less efficient when compared to a database.

As a result, a blockchain only stores specified data pieces (such as short transactions or indexes). The majority of blockchain-based applications include both blockchain and off-chain storage (databases or dedicated file systems). Many implementations,

such as StorJ, Filecoin, BigchainDB, and others, offer such a combination. IPFS is frequently cited as an off-chain storage option that can be integrated with blockchain storage in such hybrid systems.

In recent years, blockchain technology has gotten a lot of attention. The data volume of blockchain, on the other hand, continues to grow due to features that cannot be deleted and can only be added. The entire size of the Bitcoin blockchain record has now surpassed 200GB. Many nodes are unable to join the network due to the high demand for storage space and bandwidth to synchronize data with the network. This is not only inconvenient for the decentralized network's growth, but it also creates a roadblock for the advancement of blockchain technology. As a result, an IPFS-based blockchain data storage paradigm is developed to address this issue. The miners do this by depositing transaction data into the IPFS network and packing the transaction's returning IPFS hash into the block. The blockchain data is considerably decreased by utilizing the IPFS network's properties and the attributes of the IPFS hash. The Bitcoin blockchain is used to implement the plan. The compression ratio can reach 0.0817, according to the findings of the experiments. According to the analysis, it also performs well in terms of security and new node synchronization speed.

Protocol Labs' IPFS is a collection of subprotocols. It aspires to increase the efficiency of the web while also making it more decentralized. IPFS makes use of content-based addressing, in which content is identified by its content rather than by its location. IPFS's deduplication properties enable for efficient data storage due to the way it stores and addresses data. IPFS allows users to store and distribute files in a decentralized manner, boosting the content's censorship resistance. IPFS can be used to create a distributed web by deploying websites. It's utilized as a storage service to go along with blockchains, allowing for a variety of applications to be built on top of IPFS. IPFS focuses on immutable data since it employs content-based addressing. IPFS, on the other hand, uses the InterPlanetary Name System to support updatable addresses for material (IPNS). IPNS gives you the authority to link a name (the hash of a public key) to a file's content identification. Each peer keeps track of its own resolved items in an LRU cache (default 128 entries). After a certain amount of time has passed, an IPNS record is removed from cache (the default is 24 hours). File changes can be accomplished by modifying the mapping of fixed names to content identifiers. Please keep in mind that content identifiers are unique to each file. IPFS, on the other hand, is independent of Filecoin and vice versa. This is a great example of how to utilize a cryptocurrency to reward peers.

Digital identification is crucial to most commercial and social transactions in today's data-driven society. This determines how users engage in the digital world. Traditional identity systems, on the other hand, remain highly vulnerable, with single points of

failure that attract ongoing attempts to acquire access to the whole repository of high-value data. Customers' experience is dramatically harmed as organizations prioritize cybersecurity, identity protection, and compliance management.

Individuals are responsible for managing various internet IDs and passwords, as well as a variety of documents such as passports, driver's license, medical insurance cards etc.

If you want to use Ethereum to deploy a contract, you'll need gas, which you'll have to pay for in ether. So, gas is the charge that a user pays to conduct a transaction in Ethereum. Ether is a cryptocurrency that may be used to create decentralized applications, smart contracts, and regular peer-to-peer payments.

While Bitcoin's decentralized system and cryptocurrency were ground-breaking, Ethereum has built a global computer network that connects users to a marketplace of DApps that offer unparalleled user control, security, and efficiency building on its predecessor's vision of a decentralized payments system. Ethereum is utilized for n-number of creative applications in finance, gaming, identity management and supply chain management because of its ground-breaking mix of features including smart contracts.

The Ethereum blockchain is powered by its native cryptocurrency, ether (ETH), and it allows developers to build new sorts of ETH-based tokens that may be used to power dApps via smart contracts. The ERC-20 token standard is used by the majority of Ethereum-based cryptocurrencies. Smart contracts on a blockchain are self-executing which authorize, verify, and monitor transactions. They are a significant innovation in Ethereum and blockchain.

Chapter 3

We have developed a Software Requirements Specifications document, adhering to the IEEE 830 standard, which is attached herewith for further references.

Chapter 4

Implementation

4.1 Boilerplate Code

Because the original version hasn't been updated in a long time, the initial code base is bootstrapped with a customized version of Truffle react box, which has outdated dependencies. These out-of-date dependencies may cause clashes in the dependency

version we use later in our project. As a result, the initial boilerplate code included a react app with web3js enabled and a Truffle environment for solidity development using ethereum. We kept a private git repository for version control, so the boilerplate code was the "first commit" in our effort to complete the project.

4.2 Packages Used in Frontend

We used npm as the package manager for both the frontend and the backend.

package.json in the truffle environment has the following dependencies:

```
"dependencies": {
  "@truffle/hdwallet-provider": "^1.5.1",
  "dotenv": "^16.0.0"
},
"devDependencies": {
  "chai": "^4.3.4",
  "ganache-cli": "^6.12.2"
}
```

Fig. 4.1 - Packages Used in Truffle Environment

- **@truffle/hdwallet-provider** - Web3 provider with HD Wallet support. It's being used to sign transactions for addresses derived from mnemonic.
- **dotenv** - Keeps configuration separate from code in the environment.

package.json in the truffle environment has the following dev-dependencies:

- **chai** - Chai is an assertion library, similar to the built-in assert function in Node. It simplifies testing by providing a plethora of assertions that can be run against our code.
- **ganache-cli** - CLI version of Ganache, the personal blockchain for Ethereum development that is part of Truffle Suite.

package.json in the React environment has the following dependencies:

```
"dependencies": {
  "@chakra-ui/react": "^1.7.2",
  "@craco/craco": "^6.3.0",
  "@emotion/react": "^11.6.0",
  "@emotion/styled": "^11.6.0",
  "@hookform/resolvers": "^2.8.4",
  "@tippyjs/react": "^4.2.6",
  "axios": "^0.24.0",
  "buffer": "^6.0.3",
  "flatted": "^3.2.2",
  "framer-motion": "^4.1.17",
  "ipfs-http-client": "^56.0.0",
  "localforage": "^1.10.0",
  "react": "^17.0.2",
  "react-avatar": "^4.0.0",
  "react-countup": "^6.1.0",
  "react-custom-scrollbars-2": "^4.4.0",
  "react-dom": "^17.0.2",
  "react-dropzone": "^11.5.3",
  "react-helmet": "^6.1.0",
  "react-hook-form": "^7.21.0",
  "react-icons": "^4.3.1",
  "react-paginate": "^8.0.0",
  "react-pdf": "^5.7.0",
  "react-redux": "^7.2.5",
  "react-router-dom": "^5.3.0",
  "react-scripts": "^4.0.3",
  "react-scroll": "^1.8.4",
  "react-spring": "^9.3.0",
  "react-toastify": "^8.0.3",
  "redux": "^4.1.1",
  "redux-logger": "^3.0.6",
  "redux-persist": "^6.0.0",
```

```
    "redux-saga": "^1.1.3",
    "redux-thunk": "^2.3.0",
    "tailwind-scrollbar-hide": "^1.1.5",
    "validator": "^13.7.0",
    "web3": "^1.6.1",
    "yup": "^0.32.11"
  },
  "devDependencies": {
    "autoprefixer": "^9.8.8",
    "eslint": "^7.32.0",
    "eslint-config-airbnb": "^18.2.1",
    "eslint-plugin-import": "^2.25.2",
    "eslint-plugin-jsx-a11y": "^6.4.1",
    "eslint-plugin-react": "^7.26.1",
    "eslint-plugin-react-hooks": "^4.2.0",
    "gulp": "^4.0.2",
    "gulp-append-prepend": "^1.0.9",
    "postcss": "^7.0.39",
    "tailwindcss": "npm:@tailwindcss/postcss7-compat@^2.2.16"
  }
```

Fig. 4.2 - Packages Used in React Environment

- **@chakra-ui/react** - Chakra UI is a component library that is simple, modular, and easy to use, providing us with the building blocks we need to create React applications.
- **@craco/craco** - **C**reate **R**eact **A**pp **C**onfiguration **O**verride is a configuration layer for create-react-app.
- **@emotion/react** - This package is used by Chakra UI, and is installed along with it.
- **@emotion/styled** - This package is used by Chakra UI, and is installed along with it.

- **@hookform/resolvers** - Resolved library for react-hook-form to add custom library validations such as yup.
- **@tippyjs/react** - It is a tooltip UI provider.
- **axios** - Promise-based HTTP client for node.js that is used to make API calls.
- **buffer** - Buffer module from Node.js, for browser.
- **flatted** - A small utility for flattening arrays of arrays into a single array of recursively, infinitely, or to an optional depth.
- **framer-motion** - This package is installed alongside Chakra UI and is used by
- **ipfs-http-client** - A JavaScript client library for the IPFS API (/api/v0/*). This client library implements IPFS Core API. This client library also includes a set of utility functions.
- **localforage** - JavaScript storage library that is quick and easy to use.
- **react** - React is a JavaScript library that enables us to create UIs. It was installed when the create-react-app was run for the first time.
- **react-avatar** - The ability to fetch/generate an avatar based on the information we have about that user is enabled by universal avatar.
- **react-countup** - A React component wrapper for CountUp.js that counts up to a given number with animation.
- **react-custom-scrollbars-2** - React custom scrollbar styling package.
- **react-dom** - Provides access to React's DOM.
- **react-dropzone** - A simple React hook to create a file drag and drop zone that is HTML5 compliant.
- **react-helmet** - Will be used to manage changes to the document's head. For example, changing the title of a web page.
- **react-hook-form** - It reduces the amount of code we need to write while eliminating unnecessary form re-rendering.
- **react-icons** - We used react-icons to easily include popular icons in our react project. It uses ES6 imports to allow you to include only the icons that our project uses.
- **react-paginate** - We used ReactJS to render a pagination, but we had to write our own CSS to style it.
- **react-pdf** - Helps to display PDFs in ReactJS as if they were images.
- **react-redux** - React binding for Redux
- **react-router-dom** - Bindings for using React Router in web applications are included in the react-router-dom package.
- **react-scripts** - Contains the scripts and configuration files needed by Create React App.
- **react-scroll** - A smooth scrolling library for managing the scroll in a React application.
- **react-spring** - react-spring is an animation library.

- **react-toastify** - Package to show beautiful toasts in React application.
- **redux** - Redux is simply a store to store the state of the variables in our app.
- **redux-logger** - It is a middleware to log redux state changes.
- **redux-persist** - Redux Persist is a well-known library that allows us to add persistence to our redux store.
- **redux-saga** - Sagas enable a variety of approaches to dealing with parallel execution, task concurrency, task racing, task cancellation, and other issues.
- **redux-thunk** - The Redux Thunk middleware enables us to return functions rather than actions.
- **tailwind-scrollbar-hide** - Tailwind plugin for hiding scrollbars, although the element can still be scrolled if the element's content overflows.
- **validator** - Library to sanitize string.
- **web3** - This is the main package of web3.js.
- **yup** - JavaScript schema builder for data validation.

package.json in the react environment has the following dev-dependencies:

- **autoprefixer** - CSS Parser plugin.
- **eslint** - A tool linting JavaScript code.
- **eslint-config-airbnb** - This package provides an extensible shared config for Airbnb's.eslintrc. It includes the majority of the ESLint rules, as well as ECMAScript 6+ and React.
- **eslint-plugin-import** - This package is used by "eslint-config-airbnb", and is installed along with it.
- **eslint-plugin-jsx-ally** - This package is used by "eslint-config-airbnb", and is installed along with it.
- **eslint-plugin-react** - This package is used by "eslint-config-airbnb", and is installed along with it.
- **eslint-plugin-react-hooks** - This package is used by "eslint-config-airbnb", and is installed along with it.
- **gulp** - Helps us automate tasks in development.
- **gulp-append-prepend** - A gulp plugin for appending and prepending snippets in build.
- **gulp-inject-string** - A gulp plugin for injecting snippets in build.
- **postcss** - PostCSS is a style transformation tool.
- **tailwindcss** - Utility-first CSS framework for rapidly building custom user interfaces.

4.3 Frontend Architecture

The project began with the boilerplate code truffle. The directories were created with folders for testing, smart contracts, and the client folder, which contains boilerplate code for a simple react application. Following the completion of the boilerplate, the development of the application's homepage began. The figure below(Fig. 4.3) depicts the user interface.

DigiBlock

Blockchain-based Secure Document wallet

LOGIN SIGNUP

Keep Your Digital Identity Safe & Organised

A one-stop platform for your Digital Identity to be Safe and Organized. This platform is made using Blockchain Technology which makes it Fast, Trustworthy and Immutable.

About Us » Watch Video

How It Works?

Register Yourself → Get Documents → Search Documents → Verify Yourself

Documents

Become a DigiBlock Partner Organization

Get registered as a DigiBlock Issuer!

EXPLORE NOW

DigiBlock

Blockchain-based Secure Document wallet

DigiBlock

Fig. 4.3 - Homepage

After finishing the website's homepage and about page, we began integrating web3js into our application. This assisted us in connecting our frontend application to the Ethereum node. Here, we linked our smart contract to the application.

We began developing the login page (Fig. 4.4) for admins after receiving the functionality of our smart contract, which allowed them to access the dashboard by using a master key provided to them via email. We used web3js and metamask to connect to the ethereum node for our DApp. Admins would be redirected to the admin dashboard (Fig. 4.5). After successful login, we began working on the dashboard. The dashboard's content changes dynamically based on the URL that is accessed. The need for adding and deleting admins, as well as adding issuers, prompted us to use chakraUI for faster and easier development, as well as responsive designs. As a result, we refactored our codebase by utilizing chakraUI's components. At this point, we were dealing with a large amount of data from the blockchain that needed to be fed into the tables in order for them to display and make decisions. Using states and hooks to handle this volume of data is inefficient. To address this issue, we integrated redux into our application and wrote boilerplate code to allow it to begin working with this library.

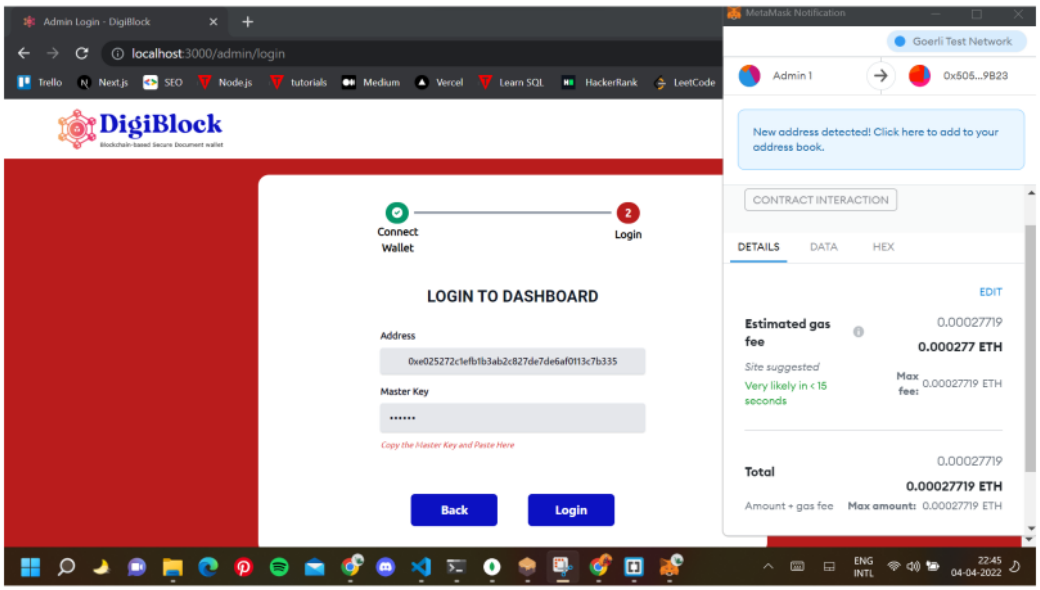

Fig. 4.4 - Admin Login Page

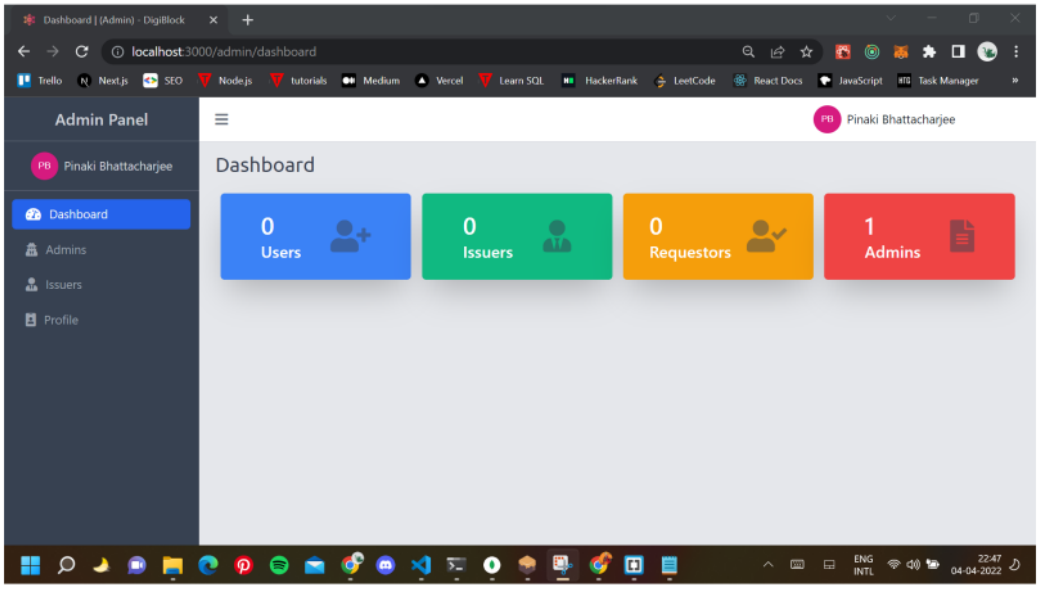

Fig. 4.5 - Admin Dashboard

We began working on issuer login after finishing the admin dashboard. We reused the code we had already written because the login method is the same for both issuer and admin. We began working on the issuer dashboard (Fig. 4.6) and the user login after successfully completing the login procedure. We reused the base code for login and dashboard throughout the application. After establishing the foundation code, we began implementing the issue document functionality, which required us to implement drag and drop functionality for the file. We began working on user signup and login concurrently because we couldn't test our functionality without actual users. We began working on the dropzone (Fig. 4.7) after successfully registering the user. We used a third-party library called react-dropzone to implement drag and drop in this section. We also added a view document functionality to the issue document page, where we used another library called ipfs-http-client to view the ipfs-saved document.

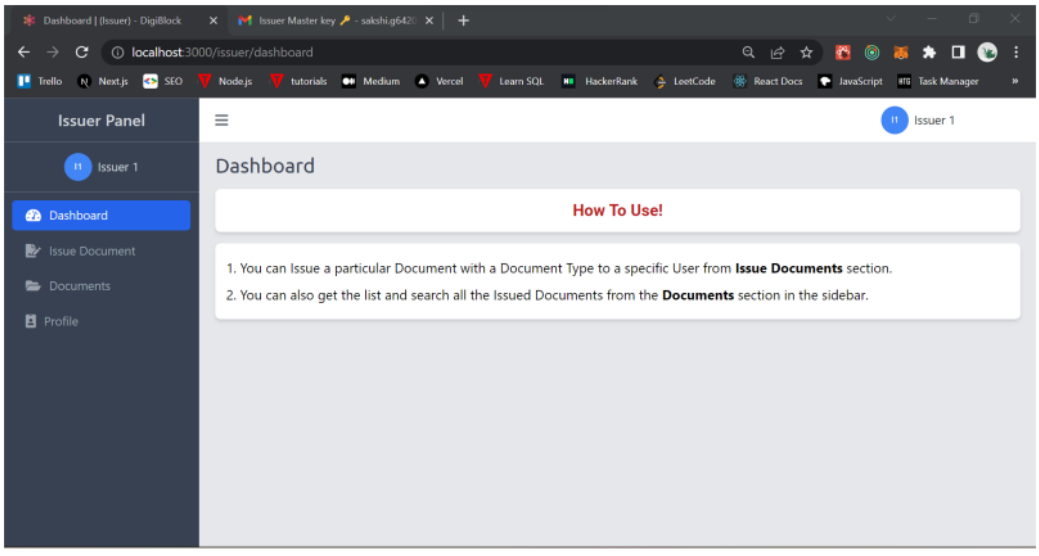

Fig. 4.6 - Issuer Dashboard

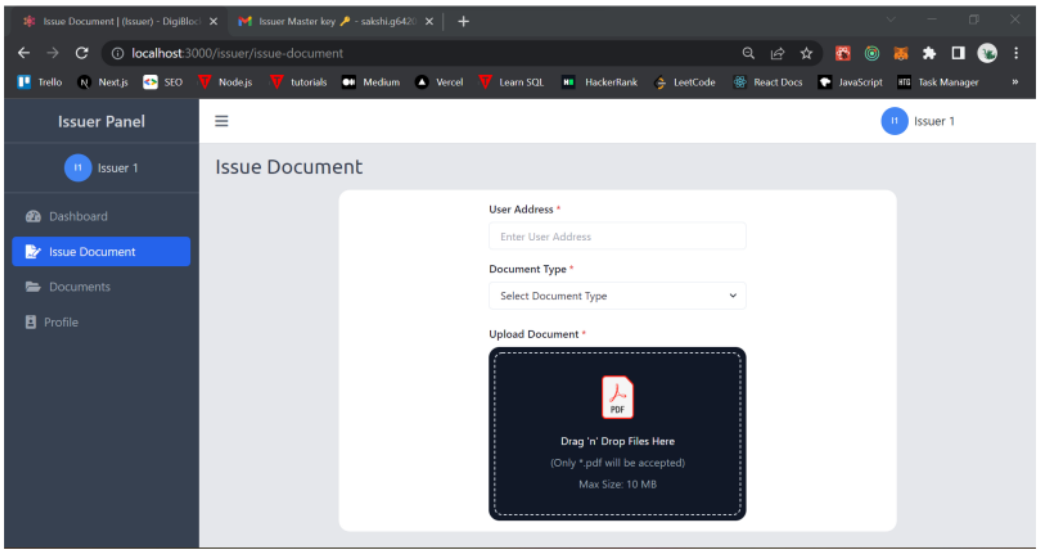

Fig. 4.7 - Issue Document

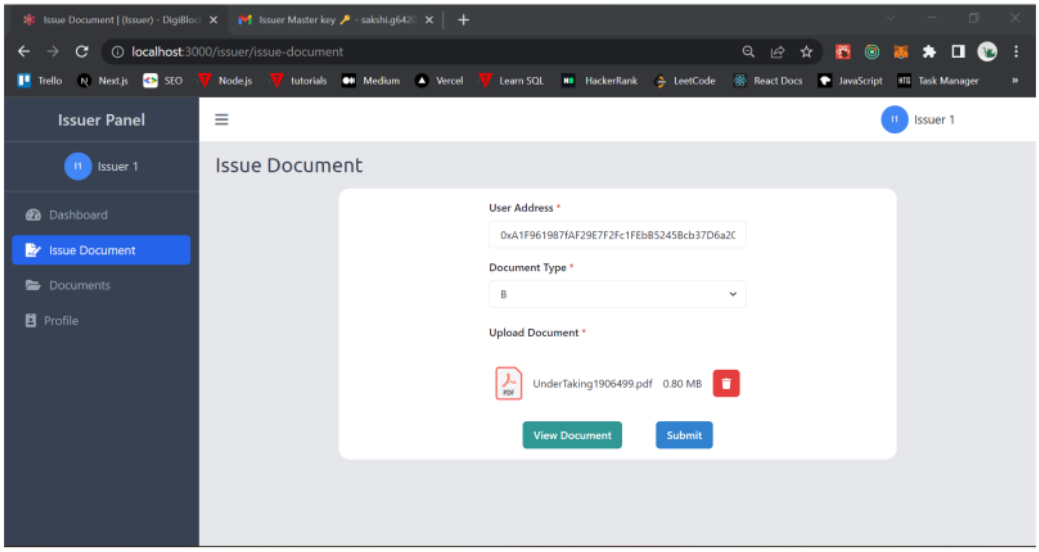

Fig. 4.8 - Issue Document Functionality

We checked our functions by issuing documents (Fig. 4.8) to registered users and after making sure everything was working correctly we started to work on the user dashboard. Tables on the user dashboard (Fig. 4.9) show issued, pending, accepted, rejected, and revoked documents. As soon as we were able to view all documents, we began working on requestor login and signup (Fig. 4.11) so that we could request user documents.

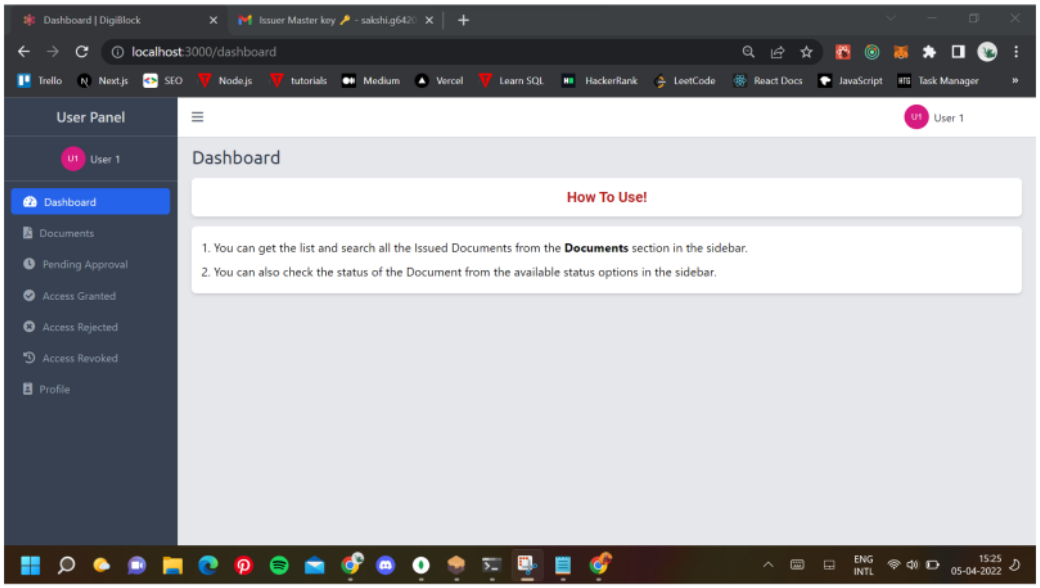

Fig. 4.9 - User Dashboard

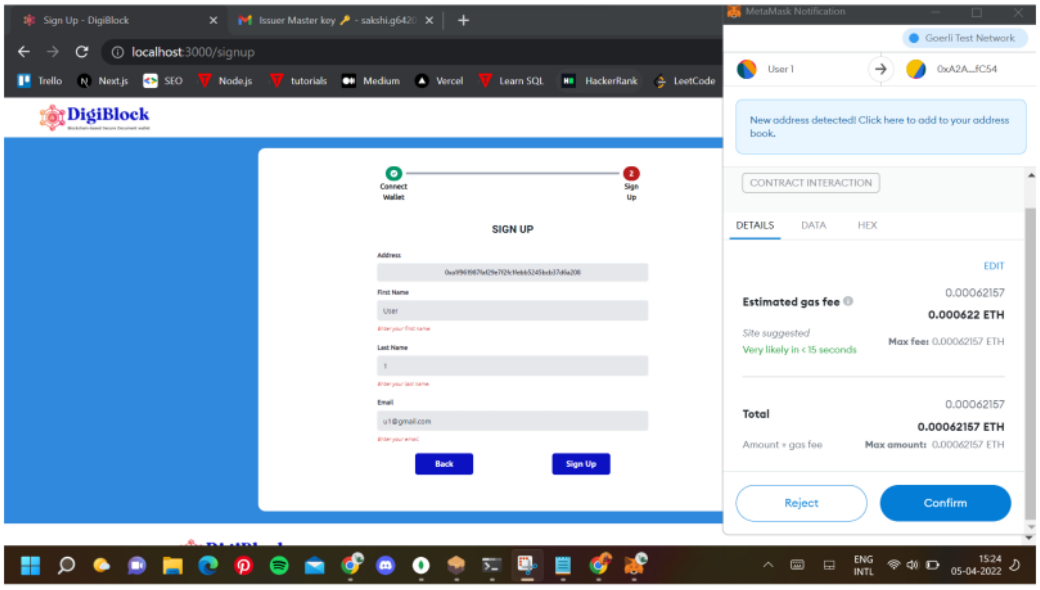

Fig. 4.10 - User SignUp

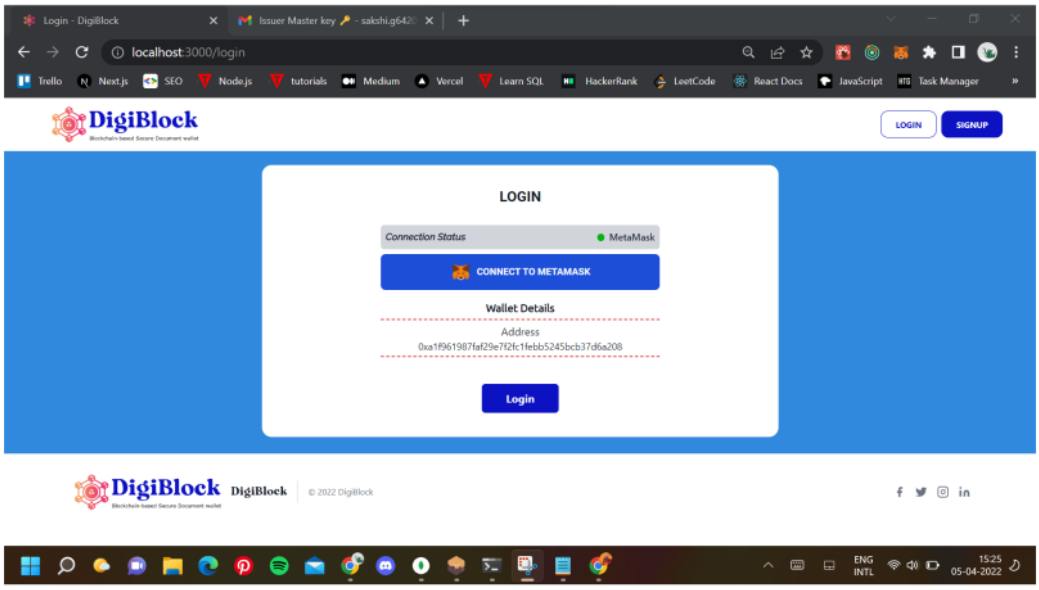

Fig. 4.11 - Requestor Login and SignUp

After completing the signup and login processes, we began working on the requestor dashboard (Fig. 4.12). We implemented the function for retrieving all user documents based on the wallet address here. We worked on making requests to the user for specific documents after we finished this functionality. After making requests (Fig. 4.13), the data is distributed throughout the application according to the request and is visible through tables on the dashboard that are separated based on the status of the request. After finishing the requestor dashboard, we moved on to the user dashboard, which displays the requests (Fig. 4.14) and gives users the option to accept or reject them, as they have complete control over the access to their data. Following successful request handling, data is dynamically visible in the tables based on the status of the request, i.e. granted, rejected, or revoked. Because of the data flow, the status of the requests is also displayed on the requestor dashboard at the same time.

After accepting any request, another functionality of revoking must be present (Fig. 4.15), which our team successfully handled.

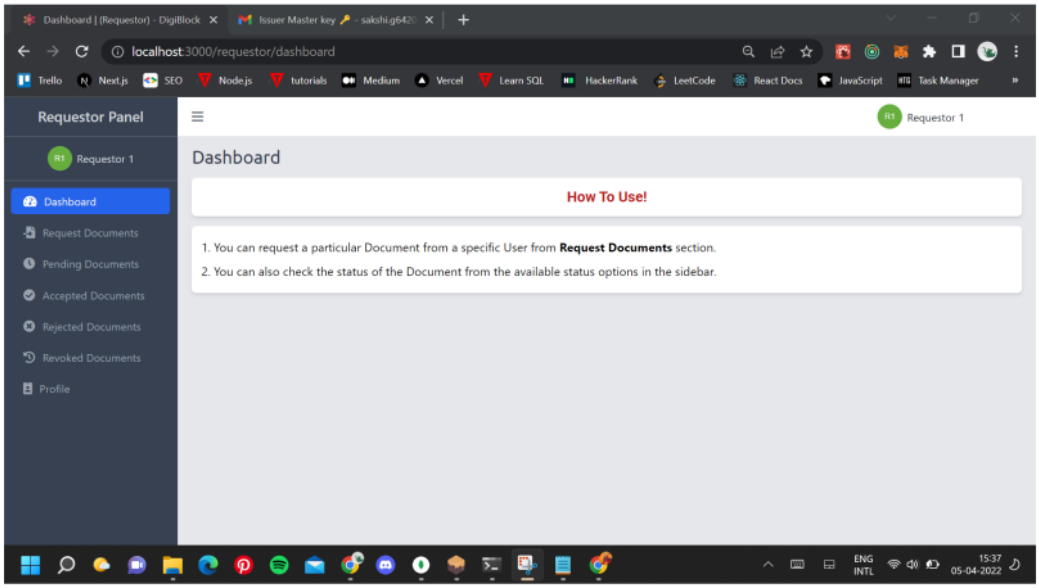

Fig. 4.12 - Requestor Dashboard

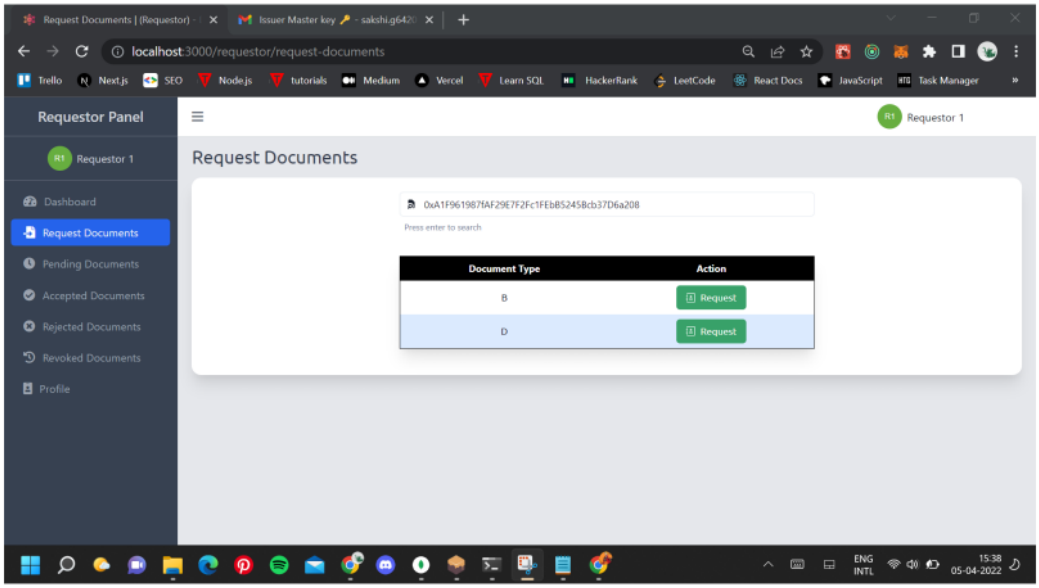

Fig. 4.13 - Request Documents

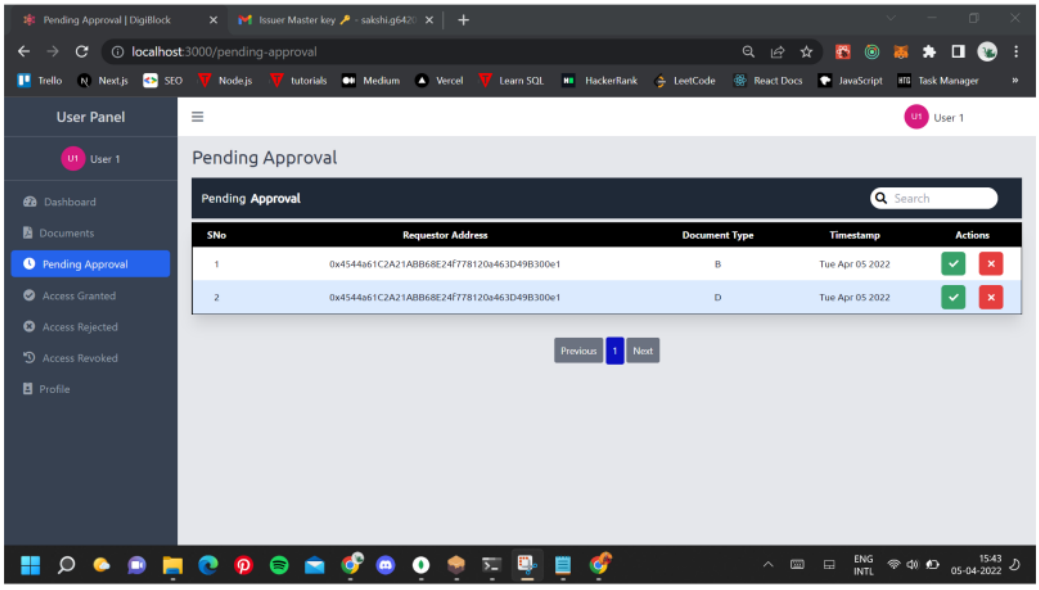

Fig. 4.14 - User Show Pending Requests

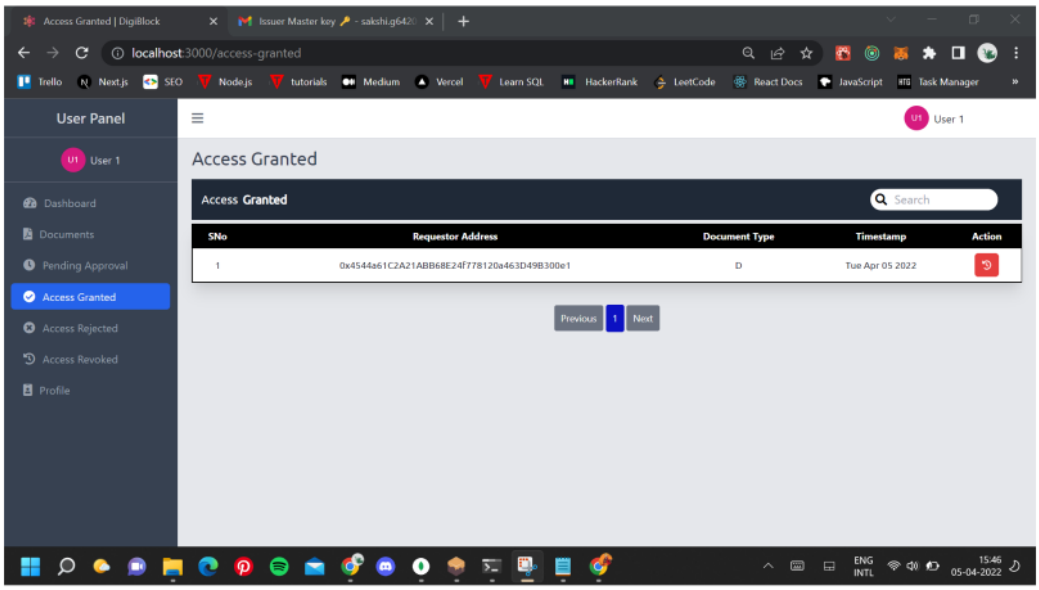

Fig. 4.15 - Revoke Access

We also handled metamask account and network changes, ensuring that only the logged in user can perform the transactions. This workflow is vastly different from the one used by web2. As a result, we devised our own logic and implemented UI level warnings when a user changes accounts while logged in to another account.

After finishing all of the nodes, namely admins, users, issuers, and requestors, we made minor changes to our UI by changing icons, fonts, spacing, padding, and so on. We then tested our application flow and removed any bugs that hampered the flow of functionality in our application.

The frontend can be accessed at the given URL : **https://digiblock.vercel.app**

4.4 Smart Contract Architecture

A smart contract is a blockchain-based programme. It is a set of data and functions that is executed whenever a transaction is initiated. Instead of being managed by any

user, smart contracts are deployed to the chain. This smart contract is running on the Ethereum blockchain.

To implement our contract, we used Solidity as a programming language. There are other programming languages that we could use to write our contract, such as Rust, JavaScript, and Vyper, but we chose Solidity because it is very similar to popular programming languages like C++, JavaScript, and Python.

It also includes fundamental concepts such as functions, string manipulation, classes, variables, and mapping data structures, which function as hash tables and contain key-value pairs.

A small portion of the smart contract containing the constructor has been shown on the next page (Fig. 4.16) :

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.10;

contract DigiBlock {
    // global variables
    address public immutable owner;

    // constructor
    constructor() {
        owner = msg.sender;
        Admin memory ownerAdmin = Admin(
            "Pinaki",
            "Bhattacharjee",
            "pinakipb2@gmail.com",
            msg.sender,
            "Pinaki"
        );
        registeredAdmins[msg.sender] = ownerAdmin;
        registeredAdminsAddresses.push(msg.sender);
    }

```

Fig. 4.16 - Portion of Smart Contract

The overview of all the functions present in our smart contract is listed below:

- **Constructor** - Starting with a constructor function which is executed when the contract is first deployed. We have set the owner in this constructor. This function will be invoked when someone will login for the first time from frontend on the owner dashboard with a predetermined password and then a 14 length master key will automatically set for that owner account and the same will be mailed to owner emailId.
- **addAdmin** - During the admin addition process, it is ensured that the account that will be associated as admin has not previously played any role in the

DApp. Once it is determined that the participant is a first-time admin, the address is linked to the admin role. This is an administrative-only action.

- **deleteAdmin** - Only the owner has the ability to delete an admin, but it cannot delete itself for obvious reasons. The role is removed once it is determined that the address exists as admin.
- **allAdmins** - It returns all the admins that are currently associated with the DApp.
- **singleAdmin** - The function returns the details of a single admin, of which the address is passed as a parameter.
- **updateMasterKey** - It is a function that is only called once during the DApp's lifetime. It is activated for the first time when the owner admin overwrites the default master key with the 14-letter key generated by the API during the first login.
- **adminsCount** - The function returns the total number of admins associated with the DApp.
- **addIssuer** - During adding of issuer, it is made sure that the account that is going to be associated as issuer, has not been participating in any role in the DApp. Once it is made sure that the participant is a first time issuer, then the address is associated with the issuer role. This is an admin only action.
- **singleIssuer** - The function returns the details of a single issuer, of which the address is passed as a parameter.
- **issuerCount** - The function returns the total number of issuers associated with the DApp.
- **addUser** - During adding of user, it is made sure that the account that is going to be associated as user, has not been participating in any role in the DApp. Once it is made sure that the participant is a first time user, then the address is associated with the user role.
- **singleUser** - The function returns the details of a single user, of which the address is passed as a parameter.
- **usersCount** - The function returns the total number of users associated with the DApp.
- **issueDocument** - An issuer-only action that is performed to issue documents to users whose wallet addresses are known ahead of time. A specific type of document can only be issued to a single user.
- **getDocsIssuedByIssuer** - An action for the issuers to get the list of all the issued documents he/she has issued.
- **isUserVerified** - A boolean function returns if the current user is verified. A user is considered verified if he/she has at least issued one document.
- **getallDocuments** - An action for the users to get the list of all issued documents along with the document type and timestamp in which the document was issued.

- **addRequestor** - During the requestor addition process, it is ensured that the account that will be associated as a requestor has not previously played any role in the DApp. When it is determined that the participant is a first-time requestor, the address is assigned to the requestor role.
- **singleRequestor** - The function returns the details of a single requestor, of which the address is passed as a parameter.
- **requestorCount** - The function returns the total number of requestors associated with the DApp.
- **requestDocumentFromUser** - A requestor only action in which a requestor can request a particular document of a particular type from a particular user.
- **getUserPendingDocuments** - This is a user-only function that returns a list of all outstanding documents that the requestors have requested.
- **getUserAcceptedDocuments** - A user-only function which returns a list of all accepted documents that are being granted to the requestors.
- **getUserRejectedDocuments** - This is a user-only function which returns a list of all rejected documents that are being requested by the requestors.
- **getUserRevokedDocuments** - This is a user-only function which returns a list of all revoked documents that are being requested by the requestors.
- **getRequestorPendingDocuments** - This is a requestor-only function which returns a list of all pending document requests that are being made to users.
- **getRequestorAcceptedDocuments** - This is a requestor-only function which returns a list of all accepted documents that are being granted to the particular requestor.
- **getRequestorRejectedDocuments** - This is a requestor-only function which returns a list of all rejected document requests that are being made to users.
- **getRequestorRevokedDocuments** - This is a requestor-only function which returns a list of all revoked document requests that are being made to users.
- **changeDocumentsStatus** - The status of a document is changed from one state to another depending on the workflow available, as shown in fig. x. And, following a successful change in document status, an event is emitted to keep track of the change in document access.

The costs associated with every function has been tabulated below :

Function	Cost (in ETH)
Initial Migration	0.00799822
updateMasterKey	0.00027719
addAdmin	0.00082406

addIssuer	0.00123869
deleteAdmin	0.00027113
addUser	0.00062157
addRequestor	0.0005339
issueDocument	0.00153126
issueDocument	0.00141541
requestDocumentFromUser	0.00152768
requestDocumentFromUser	0.00141183
changeDocumentStatus (reject)	0.00020582
changeDocumentStatus (approve)	0.00024208
changeDocumentStatus (revoke)	0.00024214

Table 4.1 - Transaction Fees of Various Functions

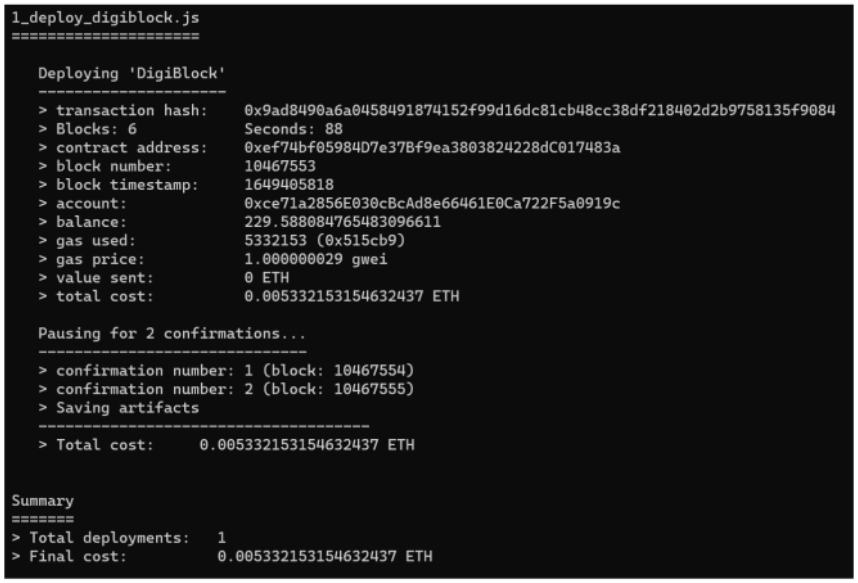

Fig. 4.17 - Smart Contract Migration

The smart contract is deployed in Rinkeby test network on the following address : **0xef74bf05984D7e37Bf9ea3803824228dC017483a**

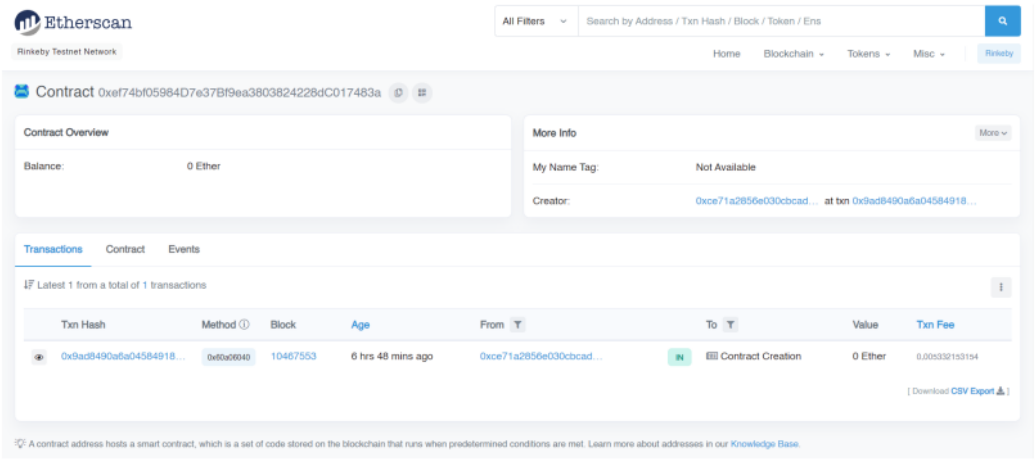

Fig. 4.18 - Deployed Smart Contract Details

and the transactions can be accessible here : **https://rinkeby.etherscan.io/address/0xef74bf05984d7e37bf9ea3803824228dc017483a**

4.5 Smart Contract Testing

Because smart contracts are executed within a blockchain, smart contract testing is critical. Using the Ethereum network would be prohibitively expensive, and while testnets are free, they are also slow (with block times between 5 and 20 seconds). We need something better if we intend to run hundreds of tests whenever we make a change to our code. What we'll use is a local blockchain, which is a scaled-down version of the real thing that runs on your machine and is not connected to the Internet. This will greatly simplify things: you won't need to obtain Ether, and new blocks will be mined instantly.

We used Truffle to test our contracts because it includes an automated testing framework that includes built-in smart contract compilation, automated contract testing for rapid development, and deployment of smart contracts to any number of public and private blockchain networks. Truffle makes use of the Mocha testing framework and the Chai assertion language. We used ganache as our local blockchain

for running and testing, which provides some test accounts and ethers to run our tests. Finally, we tested all of our solidity functions to see if they produced the expected results.

Fig. 4.19 - Smart Contract Testing

4.6 Packages used in Backend

The packages that we used in the backend are :

Dependencies :

- **bcrypt** - Library that helps in hashing passwords.
- **cors** - CORS middleware
- **dotenv** - Module that loads environment variables from a .env file into process.env. Storing configuration in the environment separate from code.
- **esm** -esm is an ECMAScript module loader.
- **express** - Web framework for NodeJS.
- **express-rate-limit** - Rate-limiting middleware for Express. Used to limit multiple requests based on IP to REST APIs.
- **handlebars** - Handlebars.js is a templating language.
- **helmet** - It secures our Express apps by setting various headers.
- **http-errors** - Easy error handling package for Express.
- **joi** - Schema description language and data validator for JS.
- **mongoose** - Mongoose is an object modeling tool for MongoDB that works in an asynchronous env.
- **mongoose-unique-validator** - Mongoose schema plugin which adds pre-save validations for fields that are defined as unique. This simplifies error handling because when we attempt to violate a unique constraint, we will receive a Mongoose validation error rather than an E11000 error from MongoDB.
- **morgan** - Middleware to log HTTP requests.
- **nanoid** - A secure unique string ID generator for JS.
- **nodemailer** - Send emails from Node.js.
- **nodemailer-sendgrid-transport** - This module is a transport plugin for Nodemailer that makes it possible to send through SendGrid's Web API.
- **winston** - Logger for the DApp.
- **winston-mongodb** - A MongoDB transport for winston.
- **xss-clean** - Node.js middleware to sanitize input from the user from POST bodies, GET queries, and URL parameters.

Dev-dependencies :

- **eslint** - ESLint is a tool for detecting and reporting on patterns found in JS.
- **eslint-config-airbnb-base** - This package provides Airbnb's JS .eslintrc as a shared config.
- **eslint-plugin-import** - Linting of ES6+ import/export syntax is supported by this plugin, preventing difficulties with misspelled file paths and import names.

- **nodemon** - nodemon is an utility that aids in the creation of node.js-based apps by restarting running node app automatically when file modifications in the subdirectory are observed.

4.7 Backend Architecture

The backend is built using REST architecture, the MVC pattern, and proper versioning. Because the APIs requirements were minimal, we decided against using Graphql, which would have increased the backend's complexity.

The barrel technique is used in the API to ensure that import statements do not increase significantly because exports are done from a single service. Because MongoDB is used for the database, the connection was initially established. The collections' schemas have been defined. After sanitizing user input and validating the request body structure, we proceeded to the next set of actions, which depended on the route and method used to make the request.

To avoid sending unnecessary data to the frontend, we used DTOs to serve data from the database. In addition, we have avoided sending the primary key (_id) and other additional data that could lead to a security vulnerability. A 14-letter random system generated key was generated for the master key by a library known as nanoid, which is two times faster than the legacy UUID. The generated key is then encrypted with the aes256-gcm algorithm and hashed with the bcrypt library, after which the hashed key is converted to an HMAC object and the two forms of the same key are sent to the frontend. The key is hashed twice because the data will be sent to the frontend, so it is done to ensure the master key's integrity. The smart contract stores the HMAC object. The encrypted key is then sent back to the backend, which sends mail to the user. The response payload consists of the recipient's name, email address, and the encrypted key. As mail is sent to the user, the encrypted key is decrypted. The key validation is straightforward; when the user provides the key, it is sent to the backend along with the key stored in the smart contract; the api then creates an HMAC object of the key and then invokes bcrypt to verify its authenticity.

The SendGrid API was used to send the master key. It allowed us to send up to 100 emails per day and was completely free in that range.

The backend also has error handling and logging capabilities; the system dumps errors and critical system startup and failures into the database with timestamps and proper messages. As a result, error recovery would be a piece of cake for the API.
In addition, to improve API security, we enabled CORS and rate limited the application. Various headers have also been added to improve the API's security.

4.8 Backend Testing

We used Postman and Insomnia for API testing, sending requests to a variety of endpoints and routes and comparing the expected results to different status codes (like 200,404 and 500 etc.). We also created custom error messages for easier debugging and comprehension. We tested each request method, such as GET, POST, and so on, for the type of action we intend to perform. A GET method is used to retrieve something, and a POST method is used to save something with the request.

The advantage of using a postman API is that it only responds to secure communication over HTTPS. HTTP requests will be redirected to the corresponding HTTPS resources via a 301 redirect. In addition, every response to a request is sent in JSON format. If the API request fails, the error is represented by an error message and status code as key in the JSON response.

The backend testing is mainly done on Insomnia, a picture depicting api tests has been shown below (Fig. 4.20) :

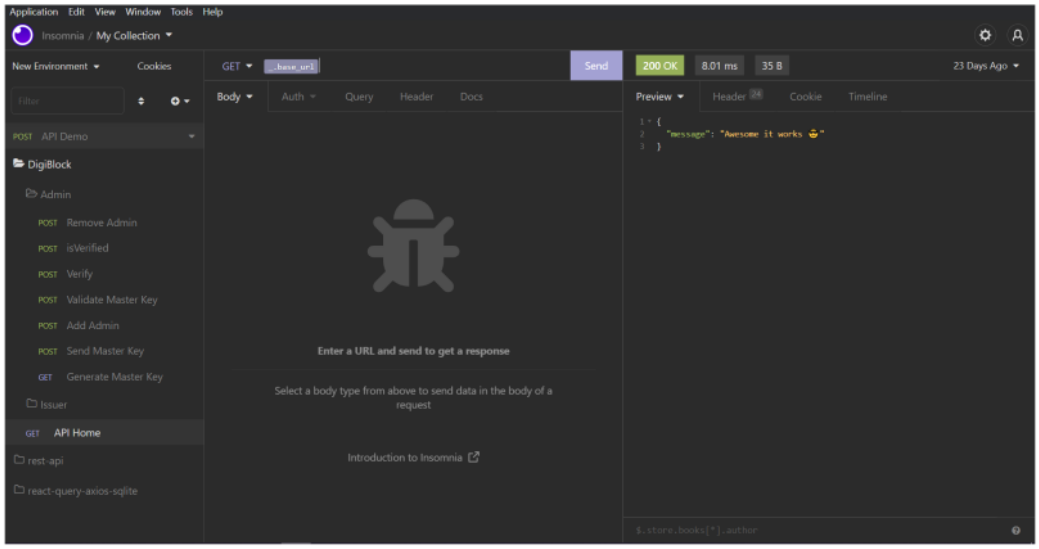

Fig. 4.20 - Backend Testing

4.9 Integration and Deployment

The frontend was integrated as the backend using the "axios" library.

The URL of the backend is injected into the application via an environment file. The payloads are delivered to the API via axios-created functions.

Deployment is a critical step in the successful development of an application. Code may work properly in the local development environment, but when deployed, it may encounter the dreaded "Deployment Bug." It usually takes hours of debugging to figure out even the most innocuous piece of code. Particularly in the case of DApps, where the ecosystem is growing at a faster rate, there is still a lack of a larger community to guide deployment. So, with the help of Infura and a free API key provided by them, we first deployed our smart contract in the Rinkeby test network. We also had a backend as well as a NoSQL database, MongoDB. For the developers, we hosted the MongoDB database in Atlas, which is powered by MongoDB itself. The backend, i.e. the REST API, was hosted in the free tier of Heroku, which has the disadvantage of putting the instance running it to sleep if it is not pinged for a certain amount of time. Also, the backend code has been added to Heroku as CI/CD, which means we don't have to redeploy the API every time we make a change to the application. For final deployment, we used the "main" branch, and the same is true for CI/CD.

Because it is a React app, the frontend is hosted in Vercel. It is also equipped with the CI/CD feature. The environment configurations were completed, and the application was tested to ensure that everything was in working order.

The DApp can be access from the URL : **https://digiblock.vercel.app/**

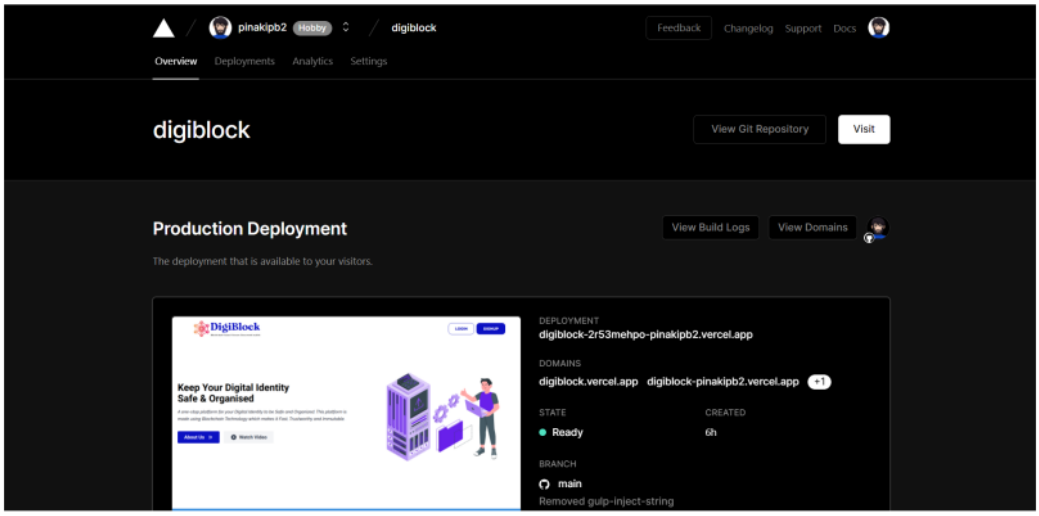

Fig. 4.21 - Frontend Deployed in Vercel

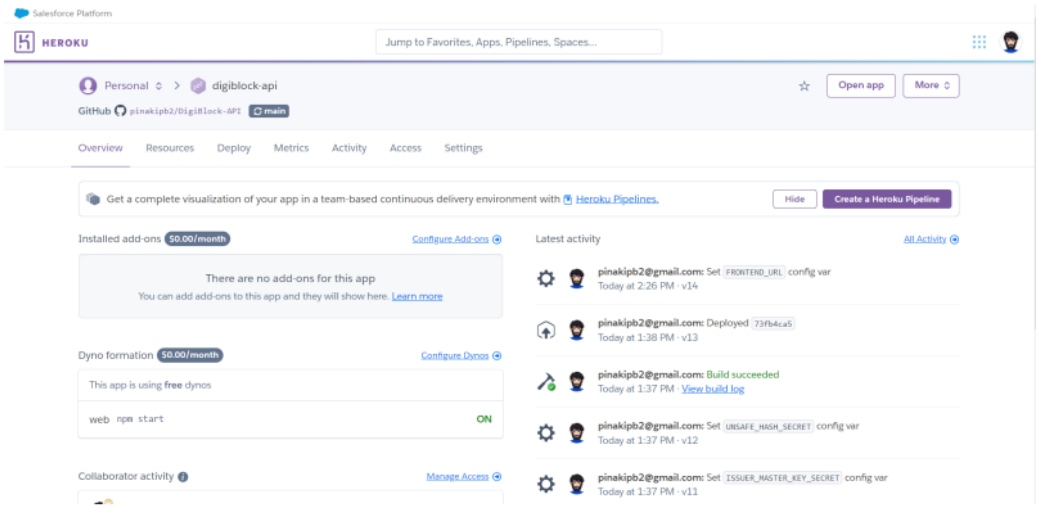

Fig. 4.22 - Backend Deployed in Heroku

4.10 Security Analysis

In this part, we evaluate the security of the proposed system. This is accomplished by analyzing our system against various threat models and providing solutions that promise greater security.

The system's purpose is to keep a user's personal data safe from illegal access and loss. We made various assumptions when assessing and building the DApp.

Assumption 1: We presume that the owner of the documents and the intended receiver have an off-chain interface channel through which they can share the recipient's public key.

Assumption 2: We presume that the only person who has access to the user's private key is the legitimate user. It is also the duty of the user to store the private key in a secure location.

Now we'll go through potential threat models for the DApp and how the it handles each of them.

Threat : An intruder obtains access to the hashed string used it to request files from the IPFS.

Argument: A non-user attacker might listen in on the system and requestor's communication and retrieve the hash string stored on the network. He is unable, however, to decode it in order to identify the string that may be used to request the file from the IPFS. He will require the private key of the legitimate recipient to decode the string, preventing the attacker from getting unauthorized access to the file.

Threat: A user who is not meant to have accessibility to a certain file may seek to acquire it. By acting as an authorized user, a registered user may simply read the shared ledger and read an authorized user's login Credential, as well as retrieve files from the DApp.

Argument: A person might read the login Credential that has permitted access to a file from the public ledger. He can use this to mimic the legitimate user and acquire access to the file's hash code. The user, however, is unable to decode this hash code in order to retrieve the file's contents address on the system files. As a result, the user will be unable to access the file.

Threat: A malicious user can try to prevent a legitimate user from receiving the file by launching a denial of service attack on the system. It may be in the user's best interests to postpone the service.

4.11 Problems and Bugs Faced

The main issue encountered during the development of the DApp is that the library that serves as the application's backbone, "web3js," does not have a large enough community to provide all the support we require in the event of a bug, so in order to resolve the issues, we either implemented some code to handle the cases or dropped the implementations.

Furthermore, "web3js" has conflicting packages with the most recent React version; specifically, the most recent React version uses the most recent version of webpack, which has conflicting dependencies with web3js. As a result, we had to stick with the React version that doesn't conflict with web3js.

Another major issue with DApps is determining which data should be stored in smart contracts and which should be stored in off-chain storage. Because a smart contract is

not a database, bulk data retrieval is not possible in solidity because it does not support dynamically sized arrays within a scope.

The main challenge we encountered when writing smart contracts was the size of the smart contract; as the number of features increased, so did the code. However, there is a limit to the ABI size of the code, which is 25KB, or else the contract will run out of gas and will be unable to deploy. So, instead of reducing the number of features, we optimized the code by writing efficient algorithms and employing optimisers that can be enabled via truffle-config.

We were also aiming for modern UI features that were clean, minimalistic, and responsive on the frontend, which would necessitate a lot of CSS, which is legacy and time consuming. So, in order to reduce our effort, we used Chakra UI, which could be used with Tailwind CSS, which we used to make minor changes. Because chakra is a newer UI library, it has fewer customizations and some limitations. So we had found a solution to the problem there as well.

Furthermore, making an application without breaking a code is nearly impossible due to the infamous JS console errors. We have tried to keep the console as error free and clean as possible. As React is a library based on DOM renders, any code that attempts to manipulate code that was not intended to behave in that manner will result in errors. We have fixed numerous bugs in the application relating to the React component lifecycle.
Another issue was that when the React application was hot-reloaded, it rendered a blank iframe on top of the application, rendering it unresponsive to any interactive events. We looked into the problem extensively, but nothing worked as of yet, and it is still an open issue in Github's React repository at this time. So, in order to resolve this issue during development, we wrote custom CSS to disable the iframe manually by making it invisible.

Furthermore, we were using the Google Mailing Service API in the backend to send mail, but the api key appears to expire after a certain time interval. There was also no way to renew the expired api key programmatically, so we decided to switch to another mailing service, so we chose SendGrid, which provided about 100 emails per day for free forever, and the API key never expires.

Chapter 5

Standards Adopted

5.1 Design Standards

The Agile methodology is a method of project management that breaks a project into stages. It demands continuing collaboration with stakeholders as well as ongoing development at each step. Once the project begins, teams cycle through a process of planning, implementing, and analyzing. Collaboration is essential among team members and project stakeholders.

We implemented the following strategies to meet our goal :

- Used trello for managing workflow and efficient and timely completion of tasks.
- Weekly meetings with the team for updates and ensuring new updates are dealt with.
- Biweekly meets for team building.

Hundreds of practitioners from around the world collaborated to create IEEE standards. Orgs, on the other hand, are not obligated to follow IEEE norms because no outside entity imposes a global standard.

Specification 830, like many other IEEE software development standards, offers guidelines and suggested ways for expressing software requirements. It isn't a comprehensive guide to requirement development, but it does provide some important information.

According to the IEEE standard, functional requirements can be organized by mode, user class, object, feature, stimulus, functional hierarchy, or a blend of these parameters. There is no one-size-fits-all solution to organizational structure; utilize whatsoever makes perfect sense for the project.
Test processes verify whether a specific activity's production products meet the activity's objectives and whether the systems and/or software meets its intended usage and user requirements. The IEEE 829-2008 standard covers the development, maintenance, and reuse of software-based systems.

5.2 Coding Standards

A style guide is a set of guidelines that describe how coding must be written and structured. Style guides are produced so that new developers may quickly become familiar with a code base and then generate code that other programmers can comprehend. The following are some of the resources we've used:

- The Airbnb Javascript style includes features such as camelcase naming, destructuring, always placing default arguments last, spaces between parentheses, and more.

- Airbnb Eslint - Airbnb maintains a prominent JavaScript Style Guide that many JavaScript developers use throughout the world. ESLint is a utility that "lints" the code. It may examine the code and alert users to any potential problems. We must set it up with particular rules in order to make it work. To preserve code consistency, we configured it with VScode.

- React File Structure - For better understandability and visualization of code, we separated react components into distinct folders based on their role in the application.

- Barrel - A barrel is a means to group together all of the same type or usage files in one location, then import and export them all at the same time using a single file.
  e.g. - // **LEGACY**
  ```
  // index.js
  import { Apple } from './interfaces/apple.interface';
  import { Mango } from './interfaces/mango.interface';
  import { Guava } from './interfaces/guava.interface';

  // **NOW**
  // index.js
  import { Apple, Mango, Guava } from './interfaces';
  ```

- Data Transfer Object(DTO) - It's a data-carrying entity that connects processes. Its use is motivated by the fact that communication amongst processes is typically accomplished through remote interfaces (e.g., web services), where every other call is a costly operation.

- MVC - Model-View-Controller (MVC) is an acronym for Model-View-Controller. The benefit of this is that it allows you to concentrate on a specific aspect of the application's name, namely the manner in which data is communicated to and accepted by the user. It aids in the effective reuse of code and concurrent development of the programme.

- Github - For version control and collaboration in the maintenance of our code, we used github. We employed the branching and merging technique, which involves a developer duplicating a portion of the code base. The developer can now modify that section of the repository without impacting the progress of the project.

  The developer can then merge their code back into the main branch once they have gotten their section of the code operating properly. All of these modifications can be tracked and, if necessary, reverted.

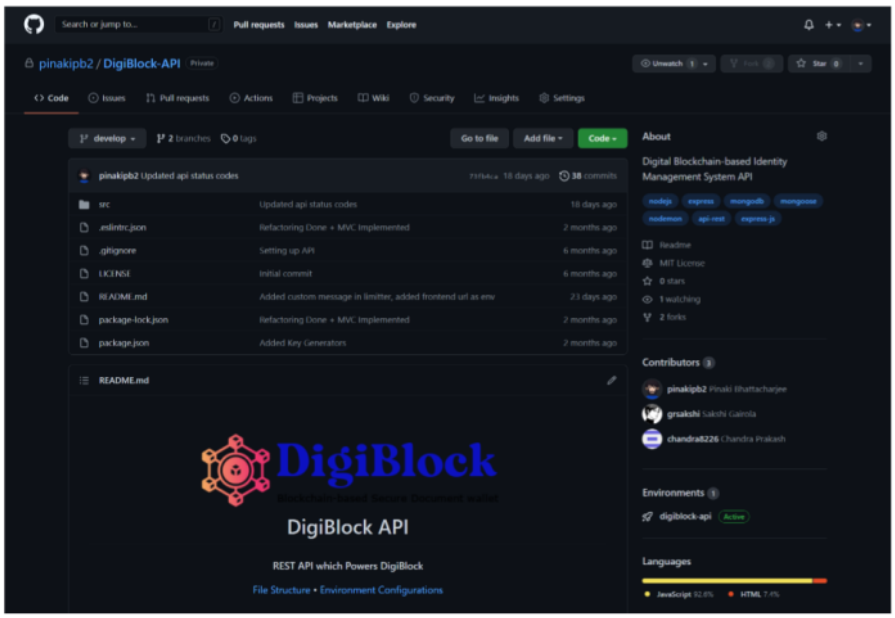

Fig. 5.2 - Backend Repository in Github

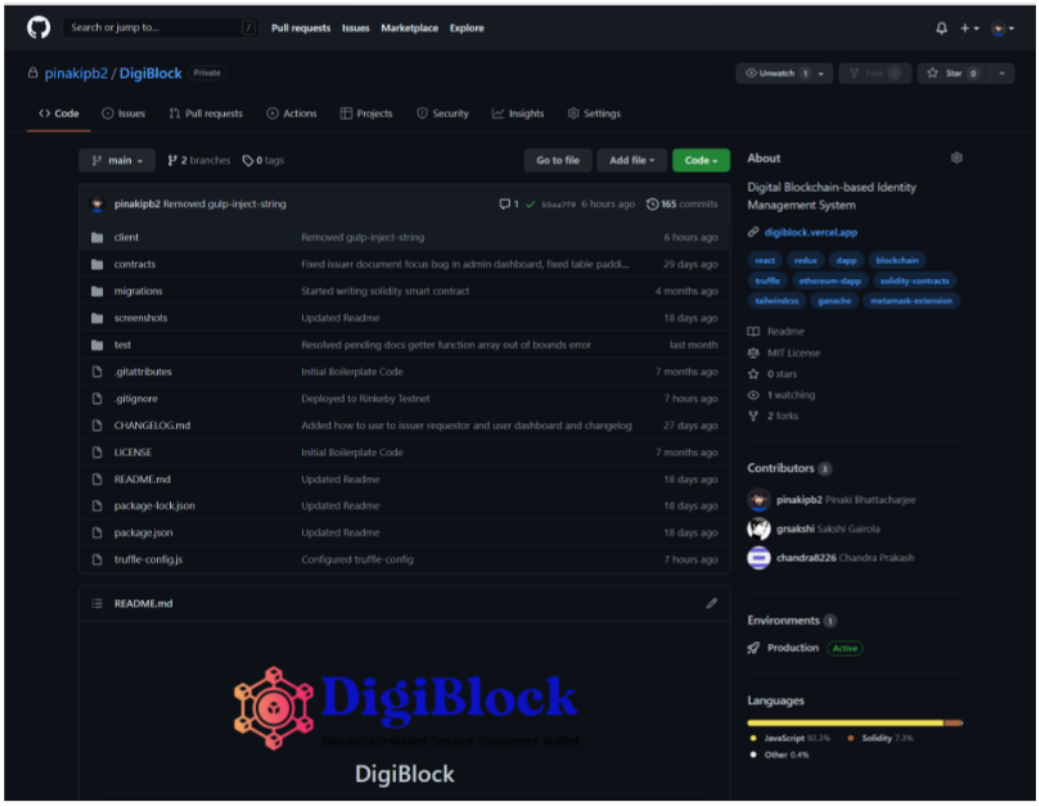

Fig. 5.1 - Frontend Repository in Github

- Semantic Versioning - Version numbers and how they change in this approach provide insight into the underlying code and what has changed from one version to the next. Consider the X.Y.Z version format (Major.Minor.Patch). Bug fixes in the API increase the patch version, API additions/changes that are backwards compatible increase the minor version, and API changes that are backwards incompatible increase the major version.

5.3 Testing Standards

Software, it's not like engineering something easy like a bridge where we start with a blueprint, build it to spec then forget about it. Software is dynamic with a lot of moving parts and requirements that evolve over time. Developers build apps on top of a mountain of abstractions and nobody fully understands how every layer works, that's okay because we just need to make sure that our code matches the requirements of the product. Test driven development is scientifically proven to reduce defects and

improve the maintainability of a code base but it does require some additional effort. One option is manual testing, where a human being clicks on every button and fills out every form then assigns a bunch of jira tickets so they can be backlogged by the developers but that's not very efficient for a large scale product. A better approach is to use automated testing tools that allow developers to write code for the sole purpose of testing the main application code.

In a code base we'll often find files that end in .test or .spec. Inside we'll first find a line of code that describes the feature or thing that's being tested that's known as a test suite and it contains one or more individual tests. An individual test usually starts with it followed by a description of what is being tested, this syntax is known as jasmine syntax, the idea is to describe the behavior of the code in human readable terms. Inside the test the code will be executed then one or more expectations or assertions are used to check that the code produces the expected result if the expectation returns false then the test fails if it's true it passes.

At the most granular level we have unit testing which is designed to test individual functions or methods like does this function return the proper value when given the arguments of a and b, then we have integration testing to determine how well different components or modules work together like is the component able to use the database service to get data from the server at the highest level we have end-to-end testing which usually happens in a mock browser or device and simulates actual user behaviors like clicking on buttons and filling out forms it's like having a robot to do all your manual testing for us and that's not all there are many other types like performance and smoke testing.

But in our scope of development, we have only tested the smart contact, which cannot fail at any cost, as it is the heart and soul of the DApp. We have used unit testing as well as integration testing. We have also created a test coverage report based on the results obtained.
We used the built-in testing tool in truffle to test the smart contract.

Truffle comes with an automated testing framework out of the box, making it simple to test your contracts. This framework provides two options for writing simple and manageable tests: Javascript and Solidity. We used Javascript testing in our scenario.

Truffle provides a strong foundation for writing JavaScript tests by using the Mocha testing framework and Chai for assertions.

```
PS C:\Users\KIIT\Desktop\digiblock> truffle test
Using network 'test'.


Compiling your contracts...
===========================
> Compiling .\contracts\DigiBlock.sol
> Artifacts written to C:\Users\KIIT\AppData\Local\Temp\test--18064-1ESsQrvrK0DE
> Compiled successfully using:
   - solc: 0.8.10+commit.fc410830.Emscripten.clang



  Contract: DigiBlock
    Deployment
      ✓ should deploy smart contract properly
      ✓ should set the right owner (146ms)
      ✓ validating first admin registration in constructor (194ms)
    Adding test accounts
      ✓ Adding admin (2204ms)
      ✓ Adding user (4113ms)
      ✓ Adding requestor (3446ms)
      ✓ Adding issuer & onlyAdmin check (7429ms)
    onlyOwner
      ✓ onlyOwner (4066ms)
    deleteAdmin
      ✓ deleteAdmin (1208ms)
    allAdmins
      ✓ allAdmins (236ms)
    Document issuing and access flow
      ✓ document issued correctly (7341ms)
      ✓ document requested correctly (8293ms)
      ✓ document status changed correctly (3867ms)
```

Fig. 5.3 - Tests in Truffle

The contract() function, which works similarly to describe() but supports Truffle's clean-room capabilities, is what distinguishes Truffle tests from those of Mocha. This is how it works:

- ➔ The contracts are redeployed to the running Ethereum client before each contract() function is called, ensuring that the tests are performed with a clean contract state.
- ➔ The contract() function returns a list of accounts that the Ethereum client has made available for testing.

Because Truffle is powered by Mocha, we can even use describe() to run regular Mocha tests anytime, the Truffle clean-room features aren't required.

Chapter 6

Conclusion and future scope

6.1 Conclusion

The power of blockchain would be harnessed with the arrival of web3.0. This technology would not be confined to a few use cases, but could be used by any web application. In comparison to a cloud provider, blockchain data storage provides more openness. Transactions are tamper-resistant, unchangeable, and verifiable. Moreover, because the data is dispersed across several nodes, blockchain technology can provide a higher level of flexibility and fault tolerance. Blockchain minimizes transaction costs and overhead by drastically reducing paperwork and mistakes, and it eventually eliminates the need for third parties and middlemen to authenticate transactions.

We've devised a technique for securing the management of a user's data on the blockchain. It solves the problem of data loss while allowing users to share physical copies of the documents. It also attempts to provide users more visibility and granular control over their data. In addition, we examined the system for vulnerabilities using a variety of threat models.

Self-sovereign identification isn't simply a beautiful idea; it has the potential to solve a number of challenges that affect consumer privacy, most notably identity theft. Identity fraud affected 16.7M people in the US last year, up 1.3M people from 2016. However, these figures only tell half of the story. Individuals frequently are unaware that their digital identities have already been stolen until they seek to purchase a home or obtain credit and discover that their financial lives have been turned upside down.

Managing identities on a blockchain ledger would make it incredibly difficult for scammers to wreck chaos without leaving a visible digital trail. The following is how it works: Each block in the blockchain builds on the previous one, and the cryptographic structure of these blocks makes it difficult to change data recorded in previous blocks. The resultant record is immutable, which means that any changes to

an individual's identity must be logged. This technology protects data custodians from hostile behavior, making identity theft extremely difficult to perpetrate.
Traditional data structures will be challenged by blockchain technology, which will return control of personal data to consumers, whose loss of trust is bad for business and fails to impress well-intentioned company leaders. Cyberattacks that lead to consumer fraud and identity theft, which is the basis of the problem and may be addressed with a promising solution, continue to be an enterprise's best defensive measure in this arena.

6.2 Future Scope

With the growth of technology and its pervasiveness, a person's identity will be preserved in digital forms in the future. Future versions of the programme could include files encoding a user's digital signatures, such as biometrics and retinal scans. The impacts of scaling the users on the system are still being researched. Furthermore, incorporating procedures to integrate the user's digital signature with the documents can help to increase the application's data's legitimacy. This ensures that the documents received as part of the application can be authenticated.

In comparison to CRA, Next.js takes a different strategy. The rendering portion of Next.js is moved towards the server, so the client does not even have to handle that knowledge. The server pre-renders the pages and then provides the finished HTML to the client, leading to decreased JS, or less code to load, which helps performance and SEO. Not only can visitors experience a speedier website, but crawlers will be able to see and index it more readily as well. Static Site Generation (SSG) and Server-side Rendering (SSR) are two ways to pre-rendering.

In terms of performance, static generation is and always will be the best. Pre-rendering refers to the process of creating a page at build time and then reusing it for each request. Static pages are typically supplied over a content delivery network (CDN), which makes them extremely fast. In some cases, such as when presenting frequently updated dynamic data, SSR is preferable: the page is generated over every request then sent to the client. This will improve the DApp's overall performance.

With an increase in the number of users, the DApp must scale correspondingly. For smooth performance, api pagination can be a lifesaver because it reduces the payload from API requests, allowing the DApp to obtain data from API calls faster. The number of network calls would increase, but the overall quantity of data provided from API would decrease due to the change in page.

API authentication for web3 is another challenge which could be a research topic, as basic Oauth based authentication fails in this case, and token based authentication fails because the authentication section is in the frontend. So, in the future, API authentication could be set up to protect off-chain storage from multiple spurious request calls.

The data in the IPFS might be encrypted, making it impossible for anyone with a content address (CID) to read the document content directly. Allows IPFS to connect only with peers who share a secret key. Each node in an IPFS private network determines which additional nodes it will connect to. Nodes within the network do not respond to signals from other nodes.

In addition, the current application supports authentication and authorization via a wallet provider, Metamask, but in the future, a number of wallet providers, such as Coinbase Wallet, WalletConnect, Fortmatic, and others, could be added so that any user with a wallet address can access it regardless of the wallet providing software.
Instead of using the upload pdf option, issuers might be given forms to fill out, which would then be converted to pdf so that the details could be filled up without leaving the DApp.

1. Introduction

1.1 Purpose

The motive of developing this software would be to take advantage of web3's capabilities to prevent document fabrication and misuse. We propose a blockchain-based system that will provide end-users a greater secure environment and fine-grained control over who has accessibility to their documents, limiting the accidental circulation of an individual's documents to people/entities.

1.2 Scope

"**DigiBlock**" is a DApp that uses blockchain technology to assist individuals manage document storage and access more efficiently. Our suggested system's backbone is blockchain.
The Issuer, who has the authority to issue certified papers to users, is added by the Admin. A Requestor can ask a user for a certain document and then wait for the user to approve access. The decision to provide access is entirely up to the user.
To access the DApp ecosystem, the user must have MetaMask (a free software cryptocurrency wallet) installed on their PC.

1.3 Definitions, acronyms and abbreviations

Term	Definition
Web3	Web3 (commonly referred to as Web 3.0) is a construct for a new version of the WWW based on blockchain technology and including ideals which include decentralization and token-based economy.
Blockchain	A blockchain is a decentralized, distributed, and usually public digital ledger consisting of documents which are used to record the transactions across many devices in a way that any one block can't be changed without impacting the subsequent blocks.
DApp	A decentralized application (DApp, dApp, Dapp, or dapp) is an application that runs on a decentralized,blockchain system and may operate autonomously, generally through the use of smart contracts.
Decentralized	Decentralization in blockchain refers to the transfer of power and course of action from a centralized entity (person, organization, or group thereof) to a dispersed network.

Metamask	MetaMask is a software-based cryptocurrency wallet that connects with the Ethereum network. Users can access their Ethereum wallet via a browser extension or a mobile app, which can subsequently be used to communicate with decentralized applications.
Crypto Currency	A virtual currency wherein transactions and records are confirmed and stored by a decentralized system instead of a central authority through the use of encryption.
Ethers (ETH)	Ethereum is a blockchain-based platform best known for its cryptocurrency, ETH.
IPFS	The InterPlanetary File System (IPFS) is a peer-to-peer network and distributed file system protocol for storing and transferring data. IPFS employs content-addressing to differentiate each file.
Off-Chain Storage	Any non-transactional data (e.g. images) that is vast to be stored efficiently onto the blockchain or requires the ability to update frequently or remove.
Admin	Someone who has special permissions to manage and operate the DApp.
User	Someone who has access to a document that has been issued to his or her account.
Issuer	Someone who has the ability to send a document to any user.
Requestor	Someone who would ask any user for a document.
API	The API (Application Programming Interface) is a interface that allows multiple programs to communicate with one another.
React	React (also known as React.js or ReactJS) is an open-source JavaScript front-end library for creating UIs.
Redux	Redux allows React components to read the data from a Redux Store Redux aids app scaling by offering a logical mechanism to handle state through a unidirectional data flow model.
NodeJs	It is a cross-platform, open-source backend JS runtime that uses the V8 engine of Chromium to execute JavaScript code outside of a browser.
Web3Js	web3.js is a set of modules that lets protocols communicate with an Ethereum node.
MongoDB	MongoDB is a cross-platform DB application that is open source. This database application is a NoSQL database.

Smart Contract	Smart contracts are software pieces that control how the accounts would behave in Ethereum env.
Solidity	Solidity is a high-level object oriented programming language used by developers for creating smart contracts.
Truffle	The Truffle Suite is a Web3 development ecosystem, asset pipeline, and testing platform for smart contract development.
Ganache	Ganache is a private Ethereum based blockchain which allows us to develop, deploy, and test our DApps in a secure and deterministic environment.
Master Key	It's a 14-character random string produced by the system and delivered to administrators and issuers only once through email to make their accounts more secure.
Testnet	Testnet is a blockchain instance that runs on the same or a newer version of the underlying software and is intended to be used for testing and experimenting without putting actual money or the main chain at risk.

2. Background

2.1 Existing System

The current system is a centralized system that aims to give citizens "Digital Empowerment" by giving them access to legitimate digital documents through their digital document wallets.

The following are some of the drawbacks of a centralized system :

- The central authority retains complete control, making it less trustworthy. The contract of trust is between service provider and client. However, that is a contract that can be readily violated. Large organizations face trust concerns with their clients. The central authority has control over network participation.
- If the centralized server fails, the entire system will come to a halt.
- Fraudsters may be able to modify or shut down the network by attacking the central authority, making it more vulnerable to cyberattacks and data leaks.
- Third-party engagement is possible, as many corporations have sold personal information to parties other than the major participants in the past.
- Because users are not anonymous, their data is at risk.

2.2 Proposed System

The suggested system is a decentralized system that intends to combat document fraud and misappropriation by providing end-users with better safe storage and fine-grained controls over who has access to personal data.

The following are the benefits of the proposed system :

- Users have complete control over their transactions, which are secure and governed by blockchain principles, and the network verifies the data using consensus mechanisms.
- There is no single point of failure since the network can continue to operate even if a significant number of participants are attacked or eliminated.
- The data structure of blockchain technology is append-only. This means that no one will be able to change or modify the data after it has been saved.
- Because of how systems manage data and transactions, decentralized networks are secure. Cryptography is used to ensure the security of the data ledgers. Furthermore, the information in the current block necessitates data from the preceding block in order to apply cryptography to verify the data..
- As a result of decentralization, there is less censorship. There are increased risks of information being restricted under a centralized system. However, because there is no central authority controlling the data, the decentralized network is less vulnerable to censorship.
- There is no need for a third party because all transactions are accessible, which is one of the blockchain's primary cornerstones.
- Users are anonymous in this environment, ensuring that their activities are not recorded.

Advantages for users:

- Users can create an account with only one click, with no personal information or password required.
- Although it is a decentralized system, it appears to be centralized to the user because all documents are safely managed and preserved.
- With a single click, users may access all of their documents in the dashboard. The documents can also be searched using a variety of criteria.
- When a requestor initiates a specific document request, the user can see a list of all pending documents in the dashboard.
- The user has complete control over the access. However, once access has been granted, it can be canceled at any time.

Advantages for issuers:

- The issuer is responsible for the documents' legality and authenticity. As a result, the issuers are chosen by the administrator.
- In only a few clicks, issuers can create any unissued document of their accessible types.
- The dashboard has a list of all issued documents that is safely arranged, and search operations may be performed using various parameters.

Advantages for requestors:

- Requesters can create an account with only one click, with no personal information or password required.
- If access is not previously granted, a requestor can request any document type from any user.
- The dashboard, which is properly arranged, has a list of all pending, accepted, rejected, and revoked documents, and search operations may be performed based on various parameters.

3. Overall Description

3.1 Product Perspective

Users will be able to register for this system through a web application. The sole need is that users have a Metamask account with ethers in it. This online application will serve as a secure and well-organized repository for all personal information. In addition, requestors/verifiers can register and make requests for users of any document of a specific type. It is the user's responsibility to control access.

The metamask extension, which is installed within the web browser, will need to communicate with the web application. Our web application will communicate with the metamask for user authentication using web3.js. The frontend would be built with React.js, and Redux would be used to better manage the application's state.
Because this is a data-centric product, we'll need to store the data off-chain, depending on its value and relevance. We'd need an API to interact with the off-chain data storage for this.

Truffle would serve as the application's blockchain layer, compiling, deploying, and injecting smart contracts into the webApp. IPFS is a distributed file system for storing, distributing, and identifying unique data. Solidity is used to develop smart contracts in the blockchain system that apply business logic and generate a chain of transaction records. Ganache for building up a personal Ethereum blockchain for testing solidity contracts, and ether (ETH) for laying the groundwork for a less hackable cryptocurrency that is utilized by hundreds of DApps (see fig.1).

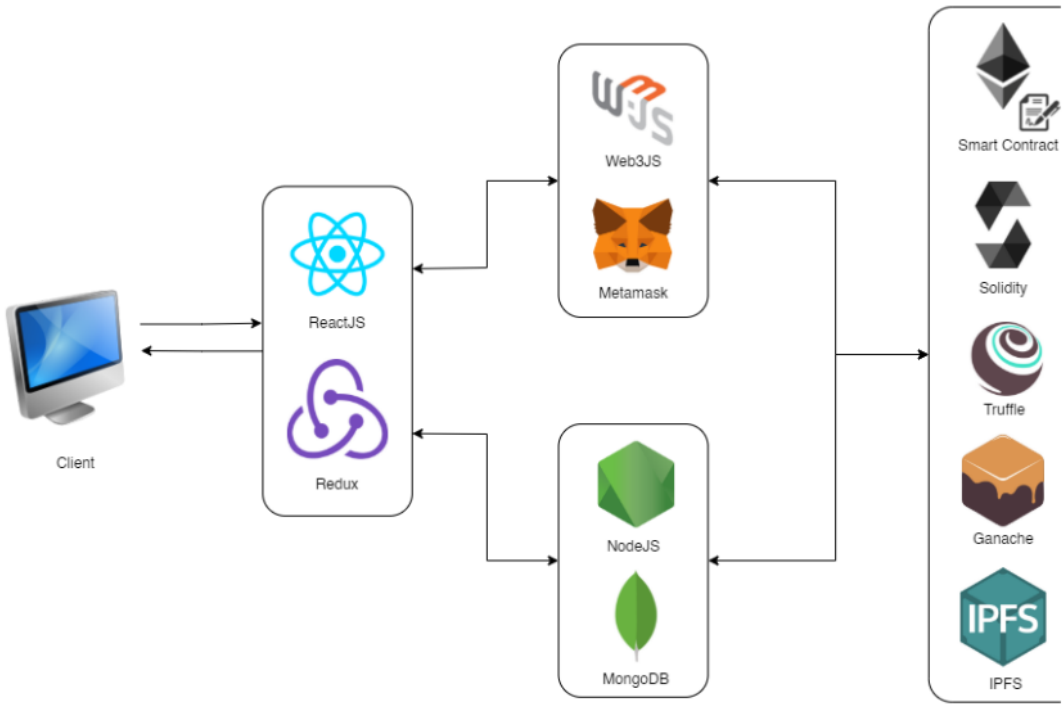

Figure 1 - Application Components

3.2 Product Functions

Users will be able to obtain personal documents in one place thanks to the online application. Users will no longer have to be concerned about document security and organization, as well as allowing access to others who will have access to their documents. Advanced searches based on multiple parameters are available. Furthermore, the interaction would be straightforward and user-friendly.
Issuers would be able to send documents to current users based on the document types to which they have access. Issuers can also examine the documents distributed to users in a tabular format, with search capabilities based on a variety of attributes.

A requestor would be able to ask a user for documents. The user has the authority to authorize access. The requestor would be alerted based on the action taken by the user.

The admin end of the decentralized web application will have the ability to manage the system and add issuers. It would also provide statistics on how many users are authorized to each role.

3.3 User Classes and Characteristics

Users of the DApp, admins, Issuers, and Requestors are the four sorts of consumers who interact with the system. Each of these four participants uses the system in a unique way, therefore they each have their own set of requirements.

Only administrators have access to the admin portal. This means that the admin will be able to see data in the dashboard for the number of users in each role. In addition, the administrator will be able to add issuers in response to requests from other organizations. The admin can also invite other admins to help with the process, making it go much more smoothly and quickly.

Users will be able to register themselves with a single click and view the numerous documents issued to them by issuers. And they'd be able to give the requestors access to the documents they requested.

Once the issuers have been added by the admins, they may log in to their dashboard and issue documents to users as well as view the documents that have been issued.
Requestors can also register in the application with a simple click and request any type of document from any user. If the user takes any action, it will be visible in the dashboard.

3.4 Operating Environment

Frontend:

- Hardware Platform: 1vCPU, 1GiB memory, t2 micro instance type
- Operating System: Linux Image (EC2 instance)

Backend:

- Hardware Platform:1vCPU, 512MB memory, t2 micro instance type
- Operating System: Linux Image (EC2 instance)
- Databases: MongoDB (Atlas)

3.5 Design and Implementation Constraints

The Metamask extension must be enabled in the user's browser for the DApp to work. Users can also switch between accounts in Metamask, but they must first log out of the system otherwise the system would come to a halt, preventing users from interacting with the DApp. To regain access to the DApp, the user must change the logged-in account's account from metamask.
The application's Internet connection is also a limitation. The DApp cannot work without an Internet connection since it retrieves and sends data from the database over the Internet.

3.6 Assumptions and Dependencies

One assumption regarding the product is that it will always be utilized on high-performance desktop applications. If the program does not have enough hardware resources, the DApp may not perform as planned, if at all. Another assumption is that the desktop application should have a browser with the most recent specifications installed, as well as Metamask with the most recent version accessible.

4. External Interface Requirements

4.1 User Interfaces

Once the first DApp user opens the web page, he or she should be able see the home page, as shown in fig. 2. If the user has not yet registered, then they must be able to do so through the signup page. The user will be able to connect to the user panel with a single click after completing the signup process. Because ReactJS conditionally updates the DOM, the UI will transition smoothly. No page refreshing would be necessary.

Users would also have a single-page application experience with smooth transitions from one page to the next.

Redux would also be used to maintain the DApp's state, with the states being saved to the user's device's local storage, eliminating the need for API calls for each action. The UI would not lag as a result, and the page waterfall would be consistent.

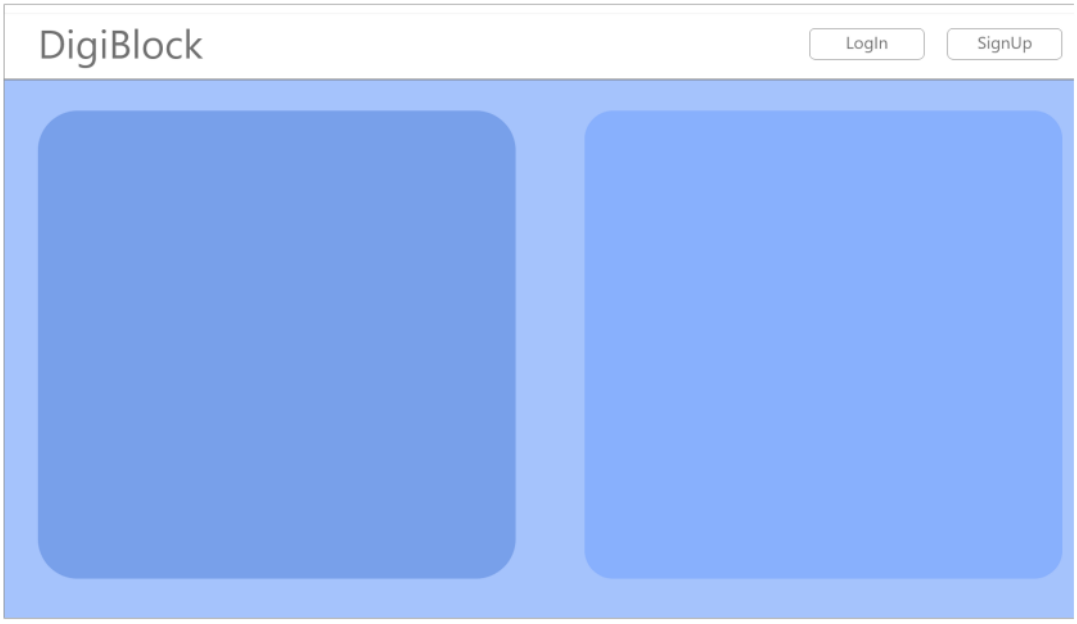

Figure 2 - Home Page

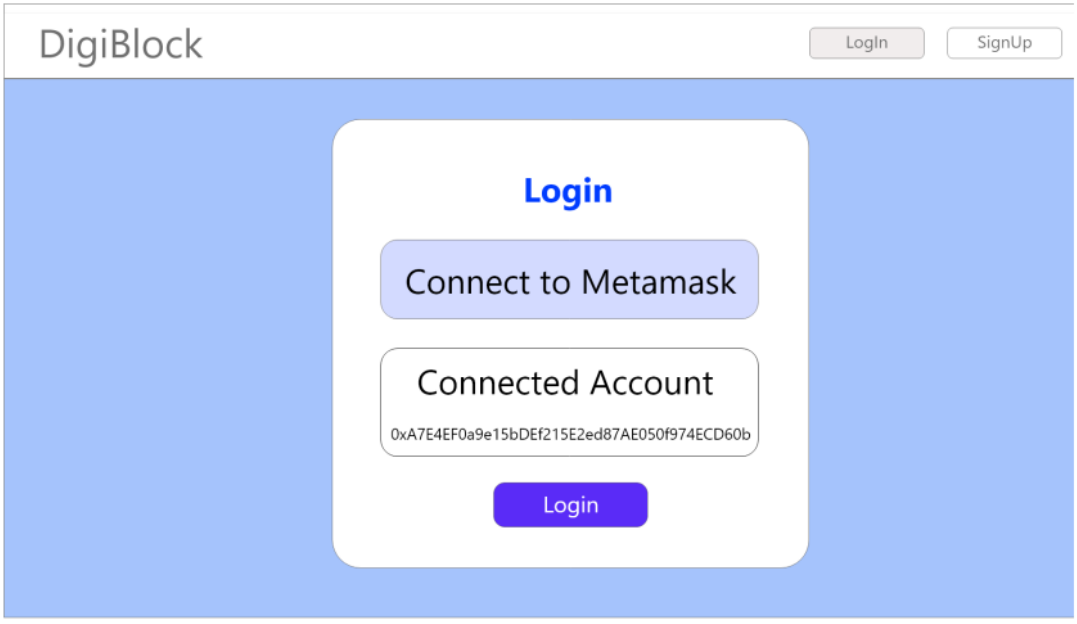

Figure 3 - Login Page

The user would be sent to the dashboard after logging in. Fig. 4 depicts the dashboard's fundamental appearance and feel.

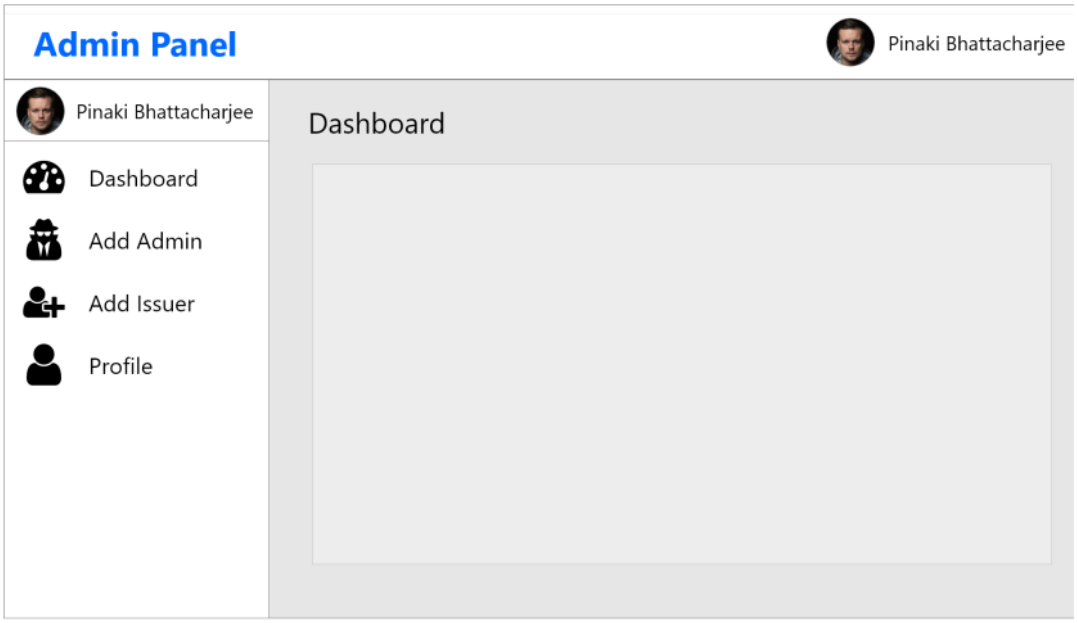

Figure 4 - Dashboard

The website's basic appearance and feel would be the same for the administrator, user, issuer, and requestor. The login would be the same, with the exception that the administrator and issuer would use a master key to ensure that the operations they execute are properly authenticated and allowed. Admins and Issuers would also not be required to register because the admin would add them.

4.2 Hardware Interfaces

There are no direct hardware interfaces in the DApp because it does not have any defined hardware. The OS on the web server manages REST API connection.

4.3 Software Interfaces

The DApp interacts with the web browser's metamask extension, and the web3 injected by the metamask is received by the website's web3js. The web application also communicates with the API to send emails, store data off-chain in the database, and perform some secret computations as in the frontend, i.e. client-side code is more vulnerable because it is easily visible at the

client-side, and most of the data generated by users is stored in the blockchain thanks to a smart contract.

4.4 Communications Interfaces

Because the numerous modules of the system are interdependent, communicating is essential. At various levels, communication can be accomplished in a variety of ways. It is the communication between multiple components in the frontend utilizing Redux, as well as the system's communication with the database in the backend. The master key generated for the admin and issuer must be delivered to them through email, which is also sent from the API that interfaces with the frontend and blockchain.

5. Functional Requirements

ID	DOMAIN	TITLE	DESCRIPTION
FR0	**Universal**	Detect metamask presence	If metamask isn't installed in the browser, the DApp should redirect the user to install it; otherwise, users should be able to log in.
FR1	**Universal**	One Account One Role	The system must not allow users who have already registered with one role to register with another one. Only one role can be connected with a metamask account.
FR2	**Universal**	Issuer Role	To request an issuer role, the organization must send an email to the admin.
FR3	**Admin**	Logging In	The system must allow administrators to log in with their wallet address and master key..
FR4	**Admin**	Add or View Admins and Issuers	Admins should be able to add and view other admins as well as issuers via the dashboard.
FR5	**Issuer**	Logging In	Issuers must be able to log in using their wallet address and master key, which must be delivered to them.
FR6	**Issuer**	Issue and View Documents	Issuers must be able to send documents to users and view the documents they've sent through the

			dashboard.
FR7	**User**	Register	Users must be able to register for the app using their name, email address, and wallet address.
FR8	**User**	Logging In	Only after signing up should the system let users log in using their wallet address.
FR9	**User**	View Documents and Control Access	Users must be able to examine all documents that have been issued to them through the system. They should also be able to see access requests for their documents, as well as the separation of the documents into pending, accepted, refused, and revoked statuses.
FR10	**Requestor**	Register	Requestors must be able to register for the app using their name, email address, and wallet address.
FR11	**Requestor**	Logging In	Only after completing the sign-up process should the system allow requestors to log in with their wallet address.
FR12	**Requestor**	Request Documents	Requestors must be able to request documents from users through the system. They should also be able to see how their requests are organized based on whether they are pending, approved, refused, or revoked.

40

6. Other Non-functional Requirements

6.1 Performance Requirements

ID	TITLE	DESCRIPTION
NFR1	Prominent Search Feature	The user must be able to see and utilize the search functionality.
NFR2	Response Time	The DApp's response time should not exceed 1 second 100 % of the time.
NFR3	System Dependability	The user should be notified if their Metamask account is

		changed, if they are logged out, or if they provide unexpected input.
NFR4	System Reliability	The system must be 100% reliable at all times.
NFR5	System Availability	More than 98 percent of the time, the system must be available.
NFR6	System Integrity	The data produced by the system must be accurate and unchangeable at all times.
NFR7	System Confidentiality	In order to meet the user's needs, the system must be confidential. Unless the user chooses to grant access to requestors, the user's information must be exposed only to the user.
NFR8	Internet Connection	The application must have access to the internet.
NFR9	Communication Security	Communication between the system and the server is secure. For sensitive data transfer, the message should be encrypted.
NFR10	Application Extendibility	Scalability should be a priority for the application. The code should be constructed in such a way that it is easy to add additional functions.
NFR11	Application Testability	To facilitate testing of the application's many functions, test environments should be created for it.
NFR12	Application Portability	The application must be adaptable to a variety of browser-enabled devices.

6.2 Safety Requirements

There are no safety standards. This DApp is available to people of all ages. Children under the age of 18 must transact under the supervision of their parents.

6.3 Security Requirements

In any scenario, users must not compromise their Metamask, as it is the basis for authentication. In the case of admins and issuers, the Master Key must be preserved in a secure location because there will be no way to update it. If the Master Key is lost, the account linked with it will become orphaned.

In any case, in the occurrence of a transaction, the front end of our application would take precedence. In the event of a disparity, any transaction made to the smart contract from any third-party sources will be ignored.

6.4 Software Quality Attributes

From the user's perspective, the DApp would require no upkeep. Furthermore, there would be no need to learn the dashboard because it is extremely user-friendly and simple to operate for people of all ages. The system would be reliable because it would pay close attention to metamask events, adjust the DApp's state accordingly, and notify the user of the occurrences. The DApp is also versatile and portable, as it can run on any device that has a web browser and Metamask installed.

The data in the DApp is proof of perfect correctness that cannot be questioned in any way, and the app is developed with numerous test scenarios in mind that may happen in practical day-to-day use.

Appendix A: Analysis Models

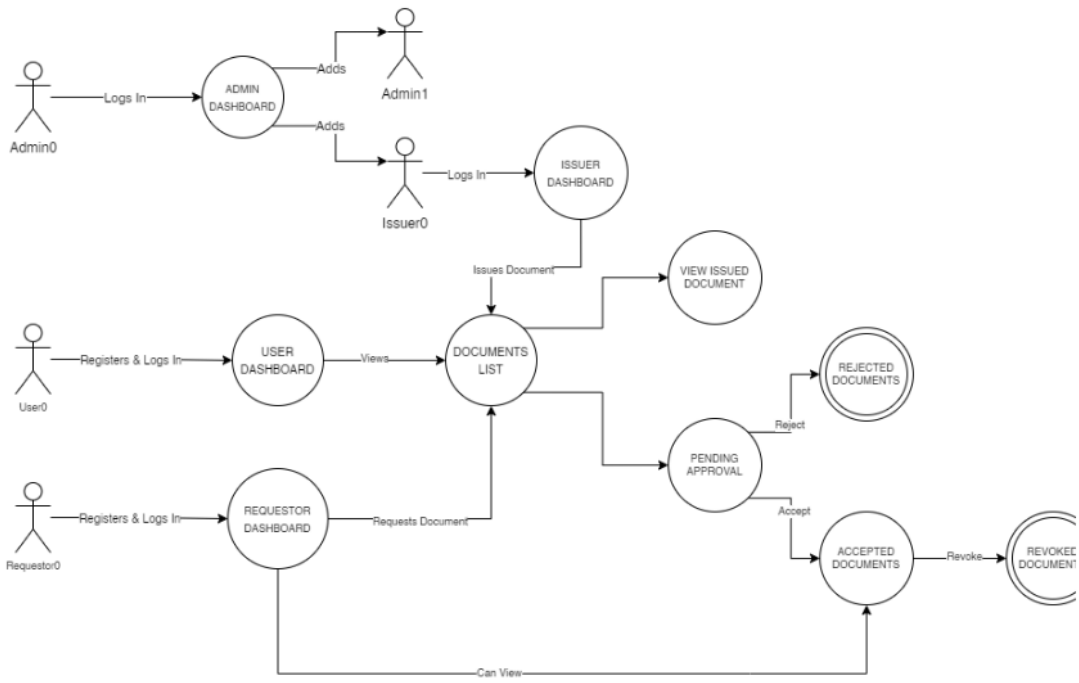

Figure 5 - System Architecture

The admin (admin0), whose wallet address was used to deploy the smart contract, is the first account who has access to this application. admin0 can add and delete both new admins and issuers after logging into the admin dashboard. The admins added by admin0 will receive their login credentials through email, and will then be able to login and access the admin-only functionality.

Similarly, issuers will receive their credentials in the mail and will be able to access the issuer dashboard afterwards. The ability to issue documents to users based on their wallet address will be available to issuers.

Users must first register before accessing the user dashboard. Users will be able to access all of the documents that have been issued to them by the issuers. Users will be able to see the status of their requests for access to their documents, such as pending, accepted, refused, and revoked. They have the ability to approve or deny outstanding requests as well as revoke previously given access at any time. After denying or canceling a request, no further options are available.

To access their dashboard, requestors will need to register and login. They would be able to seek access to any document from any user via this dashboard by providing their wallet address. Requestors would be able to check the progress of their requests and, if authorized, have access to the actual document.

Appendix B: To Be Determined List

- Decide the test network where the smart contract will be deployed.

Introduction

ORIGINALITY REPORT

8%	3%	2%	6%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Gitam University Student Paper	2%
2	en.wikipedia.org Internet Source	1%
3	Submitted to Taylor's Education Group Student Paper	1%
4	Erik Daniel, Florian Tschorsch. "IPFS and Friends: A Qualitative Comparison of Next Generation Peer-to-Peer Data Networks", IEEE Communications Surveys & Tutorials, 2022 Publication	<1%
5	Submitted to Middle East College of Information Technology Student Paper	<1%
6	wikizero.com Internet Source	<1%
7	Submitted to HCUC Student Paper	<1%
8	easychair-www.easychair.org Internet Source	<1%

9	www.johronline.com Internet Source	<1 %
10	Qian Wei, Bingzhe Li, Wanli Chang, Zhiping Jia, Zhaoyan Shen*, Zili Shao. "A Survey of Blockchain Data Management Systems", ACM Transactions on Embedded Computing Systems, 2022 Publication	<1 %
11	Submitted to WorldQuant University Student Paper	<1 %
12	www.CoinGecko.com Internet Source	<1 %
13	khuhub.khu.ac.kr Internet Source	<1 %
14	Submitted to University of the Cordilleras Student Paper	<1 %
15	api.research-repository.uwa.edu.au Internet Source	<1 %
16	Submitted to University of Derby Student Paper	<1 %
17	Submitted to University of Nebraska at Omaha Student Paper	<1 %
18	Submitted to Academy of Information Technology	<1 %

Student Paper

19	Submitted to King's College Student Paper	<1%
20	Submitted to Virginia Community College System Student Paper	<1%
21	www.geeksforgeeks.org Internet Source	<1%
22	www.maheshbhusanoor.com Internet Source	<1%
23	Submitted to Swinburne University of Technology Student Paper	<1%
24	Submitted to Colorado Technical University Online Student Paper	<1%
25	searchsoftwarequality.techtarget.com Internet Source	<1%
26	Submitted to Technological Institute of the Philippines Student Paper	<1%
27	Submitted to Segi University College Student Paper	<1%
28	Submitted to The University of Manchester Student Paper	<1%

29	Submitted to University of Lancaster Student Paper	<1 %
30	"Engineering Multi-Agent Systems", Springer Science and Business Media LLC, 2020 Publication	<1 %
31	Submitted to University of Technology, Sydney Student Paper	<1 %
32	www.investopedia.com Internet Source	<1 %
33	dspace.daffodilvarsity.edu.bd:8080 Internet Source	<1 %
34	32fdesignstudio.com Internet Source	<1 %
35	Submitted to Flinders University Student Paper	<1 %
36	www.theseus.fi Internet Source	<1 %
37	Submitted to University of Bridgeport Student Paper	<1 %
38	ionicframework.com Internet Source	<1 %
39	redux-saga.js.org Internet Source	<1 %

40	www.coursehero.com Internet Source	<1%
41	blog.bitsrc.io Internet Source	<1%
42	"Applications of Blockchain in Healthcare", Springer Science and Business Media LLC, 2021 Publication	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		