1. **INTRODUCTION**
   1. **OVERVIEW OF THE PROJECT**

This project “**Secure Document Storage System using Web3**” aims to develop a decentralized document management system leveraging Web3 technologies to enable secure file storage and sharing capabilities without centralized servers. The system allows users to upload files and documents to IPFS (InterPlanetary File System), which is a distributed network for storing file content spread across peer-to-peer nodes. IPFS removes single points of failure and censorship by distributing data with built-in redundancy. When files are uploaded to IPFS, a unique gateway token is generated which can be shared with others to allow access to the uploaded files.

The Ethereum blockchain maintains the core access control logic and file sharing workflows via smart contracts deployed on it. The Hardhat development environment is utilized for compiling, testing and deploying the smart contracts defining the backend logic of the system. Hardhat provides a flexible and fast way to iterate on building robust blockchain-enabled backends. Metamask browser extension injects a Web3-enabled Ethereum wallet into the application to manage user accounts and signing of transactions on the blockchain for secure access control.

The frontend application providing the user interface for file storage and sharing capabilities is built using ReactJS. It interacts with the blockchain contracts deployed on Ethereum using the ethers.js library, which facilitates communication between frontends and blockchain networks. Overall, through the strategic combination of IPFS, Ethereum, Hardhat, Metamask and ReactJS, the system demonstrates building full-stack decentralized applications leveraging the latest Web3 and blockchain technologies in a practical manner. The result is a serverless and decentralized document management system with tamper-proof historical records, transparency, accountability, reliability and security.

* 1. **FEASIBILITY STUDY**

The feasibility study analyzed the practicality and potential challenges of implementing a decentralized document management system using blockchain and IPFS technologies. It evaluated factors like the maturity of the underlying technologies, availability of development tools and libraries, performance considerations, scalability, and user adoption barriers. The study also looked at potential risks, such as regulatory hurdles, security vulnerabilities, and integration complexities. Overall, the feasibility analysis concluded that leveraging Ethereum smart contracts for access control logic and IPFS for distributed file storage was a viable approach to building a trustless, censorship-resistant document sharing platform.

The study also explored the ecosystem around these technologies, including developer communities, documentation resources, and existing code libraries that could be leveraged. It found a growing landscape of tools, frameworks, and support that would aid in the development and deployment of the proposed system. Additionally, the study examined potential business models and use cases where such a decentralized document management solution could provide value.

* 1. **SCOPE**

The scope of the project involved designing and developing a web-based application that enables users to upload documents securely to the InterPlanetary File System (IPFS). The application would integrate with the Ethereum blockchain, where smart contracts would be deployed to manage access controls, sharing permissions, and potential monetization rules for the uploaded files. Users could generate and share cryptographic access tokens, allowing selective retrieval of files from IPFS by authorized parties only. The project scope included a user-friendly interface, blockchain integration, IPFS connectivity, smart contract development, encryption mechanisms, and implementing core features like file uploads, access management, sharing, and retrieval.

Additionally, the project scope encompassed developing a comprehensive testing strategy to ensure the system's functionality, security, and reliability. This included unit testing of individual components, integration testing of the blockchain and IPFS interactions, as well as end-to-end testing of the complete application workflow. The scope also involved documenting the system architecture, deployment procedures, and user guides to facilitate future maintenance and potential expansion of the platform.

1. **LITERATURE SURVEY**

**2.1 EXISTING SYSTEM & DRAWBACKS**

Currently, most file-sharing platforms rely on centralized servers to store and manage files. While these platforms offer convenience, they also present several drawbacks that undermine security, privacy, and control over user data. Centralized servers act as single points of failure, making the entire system vulnerable to outages, cyber-attacks, or even censorship by the service provider or authorities. User data stored on these servers is susceptible to breaches, exposing sensitive information to unauthorized access or theft. Additionally, users have limited control over their data, as it is stored and managed by third-party platforms. These platforms can potentially access, modify, or even delete user data without explicit consent, raising privacy concerns and violating user ownership rights. Furthermore, centralized file-sharing platforms often face scalability challenges, leading to performance issues or service disruptions as the number of users and data volumes grow. This dependence on a central entity also introduces potential bottlenecks and limits the system's ability to adapt to changing demands or technological advancements. Moreover, users face the risk of censorship or arbitrary restrictions imposed by the service providers, who can selectively deny access to files or terminate accounts based on their policies or external pressures. This centralized control can limit freedom of expression and information exchange. Lastly, these platforms typically lack transparent and auditable record-keeping mechanisms, making it difficult for users to verify the integrity of their data or track any unauthorized access or modifications over time.

* + 1. **DRAWBACKS IN THE EXISTING SYSTEM**
* **Security Concerns:** Centralized servers are vulnerable to hacking and data breaches, putting user data at risk.
* **Privacy Issues:** Users have limited control over their data, as it is stored and managed by third-party platforms.
* **Single Point of Failure:** If the centralized server goes down or experiences technical issues, users may lose access to their files.
* **Censorship risk:** Companies have been known to delete or restrict access to files due to legal/policy issues.
* **Lack of user ownership:** Users do not truly own their data as companies maintain control over files.
* **Difficult access management:** It is challenging to restrict sharing or access to files over time. Downloads cannot be tracked or monetized.

**2.2 PROPOSED SYSTEM**

The proposed system will be a decentralized file sharing application built on IPFS and Ethereum. It will allow users to upload files from their browser to IPFS, with smart contracts deployed on Ethereum to manage access controls and payments for files. The React and Vite based interface will connect to MetaMask, enabling users to upload files to IPFS, set permissions and prices via smart contracts, and share files between wallets. This avoids central points of failure while providing ownership, access management and monetization not possible with existing centralized solutions.

**2.2.1 Advantages in the Proposed System**

* **Resilience:** By storing files on IPFS and managing access via smart contracts, the files are not dependent on any single server. This makes them more resilient against outages or censorship.
* **Privacy:** Files are stored anonymously on IPFS and access is controlled via smart contracts, avoiding privacy risks of centralized servers where governments can request data.
* **Data Integrity:** By storing file hashes and metadata on the tamper-proof Ethereum blockchain, the integrity of documents is ensured, preventing unauthorized modifications or deletions.
* **Transparency:** All file sharing activity and access permissions are recorded transparently on the blockchain, providing an auditable trail and accountability.
* **Reduced Costs:** Eliminating the need for centralized servers and storage infrastructure can significantly reduce operational and maintenance costs.
* **Decentralized Access:** Access to files is not dependent on any single point of failure, as the system leverages distributed peer-to-peer networks like IPFS and Ethereum.

1. **SYSTEM ANALYSIS**

**3.1. OVERVIEW OF SYSTEM ANALYSIS**

The system analysis phase is a crucial step in the development of any software project, as it lays the foundation for understanding the project's requirements, constraints, and potential challenges. In the context of this secure document management system using Web3 technologies, the system analysis involved a comprehensive evaluation of the existing centralized file-sharing platforms, their limitations, and the potential benefits of leveraging decentralized technologies such as Ethereum blockchain and the InterPlanetary File System (IPFS).The analysis process involved identifying the key stakeholders, including users, administrators, and potential integrators, and gathering their requirements and expectations from the proposed system. This included assessing the need for secure file storage, access control mechanisms, sharing capabilities, and potential monetization models. Additionally, the system analysis phase involved researching and evaluating the suitability of various tools and technologies, such as Solidity for smart contract development, ReactJS for the frontend user interface, and the integration of Metamask wallets for user authentication and transaction signing.

**3.2. SOFTWARE USED IN THE PROJECT**

To develop the secure document management system, a combination of cutting-edge technologies and software tools were employed. The core components included Ethereum blockchain for deploying and executing smart contracts, IPFS for decentralized file storage, and ReactJS for building the responsive user interface. Solidity, a contract-oriented programming language, was used to develop the smart contracts that govern access control, file sharing, and potential monetization rules.The development environment involved tools such as Hardhat, a robust Ethereum development environment that facilitates smart contract compilation, testing, and deployment. Metamask, a popular browser extension, was integrated to provide users with a secure Ethereum wallet for authentication and transaction signing. Additionally, libraries like ethers.js and web3.js were utilized to facilitate communication between the frontend application and the Ethereum blockchain.

**3.3. SYSTEM REQUIREMENTS**

To ensure the successful development and deployment of the secure document management system, several system requirements were identified. On the hardware side, the system required a capable CPU (Intel i5 or higher) with at least 2.5GHz speed, 8GB or more system RAM for smooth performance, and ample storage (500GB or more HDD/SSD) to accommodate the files uploaded to IPFS. Additionally, a stable internet connection was necessary to interact with the Ethereum network and IPFS nodes.From a software perspective, the system required a modern web browser with support for the latest web technologies, including WebAssembly and WebRTC. The frontend user interface was built using ReactJS, while the backend services and blockchain interactions were facilitated by Node.js. The Ethereum blockchain was utilized for deploying and interacting with the smart contracts, which were developed using the Solidity programming language. Furthermore, the system relied on the IPFS protocol for decentralized file storage and retrieval.To ensure secure and reliable operation, the system also required robust encryption mechanisms, user authentication protocols (e.g., Metamask wallet integration), and comprehensive testing frameworks for validating the system's functionality, security, and reliability.

1. **SYSTEM DESIGN**

**4.1. OVERVIEW OF SYSTEM DESIGN**

The system design phase plays a pivotal role in translating the requirements and specifications gathered during the analysis phase into a comprehensive blueprint for the secure document management system. This phase involves several critical decisions regarding the system's architecture, components, and their interactions, ultimately shaping the overall functionality and performance of the final product.The design process begins with the identification of the core modules and their respective responsibilities. In the context of this project, the key modules include the blockchain module for managing smart contracts and transactions on the Ethereum network, the file storage module for interfacing with IPFS, the access management module for enforcing access controls and permissions, the user authentication module for integrating with Metamask wallets, and the application interface module for providing a user-friendly frontend.Each of these modules is designed to encapsulate specific functionalities and interact with other components through well-defined interfaces and communication protocols. The design phase also involves selecting appropriate data structures and algorithms to ensure efficient data management, file storage, and retrieval processes.Moreover, the design phase takes into consideration the scalability and performance requirements of the system. The architecture is crafted to handle increasing volumes of users, files, and transactions efficiently. Load balancing, caching mechanisms, and optimized algorithms are incorporated to ensure responsive and seamless user experiences, even under high loads.

**4.2. METHODOLOGY**

To ensure a structured and systematic approach to the system design, several industry-standard methodologies and best practices were adopted. One such methodology is the Object-Oriented Design (OOD) paradigm, which emphasizes the principles of encapsulation, inheritance, and polymorphism. By adhering to these principles, the system components can be designed as modular and reusable units, promoting code maintainability, extensibility, and testability.Additionally, the design phase incorporates the principles of the Unified Modeling Language (UML), a standardized graphical notation for visualizing and documenting the system's

architecture, components, and their relationships. UML diagrams, such as class diagrams, sequence diagrams, and use case diagrams, provide a clear and concise representation of the system's structure and behavior, facilitating effective communication among stakeholders and developers.Furthermore, the design process involves the application of design patterns, which are proven solutions to commonly occurring problems in software development. Design patterns like the Singleton pattern, Factory pattern, and Observer pattern can be utilized to ensure a robust, scalable, and maintainable system architecture.To mitigate potential risks and identify design flaws early in the development cycle, the system design phase incorporates iterative prototyping and review sessions. Prototypes are developed to validate design decisions, gather feedback from stakeholders, and refine the system's functionality and user experience. These iterative cycles enable the incorporation of necessary changes and improvements before proceeding to the implementation phase.Throughout the design phase, particular emphasis is placed on security considerations. The system's design incorporates industry-standard security practices, such as encryption mechanisms, access control policies, and secure communication protocols. Additionally, the design takes into account the unique challenges and requirements of decentralized systems, ensuring the system's resilience against potential threats and vulnerabilities.Moreover, the design process involves the consideration of various non-functional requirements, such as usability, reliability, and maintainability. User experience (UX) principles are incorporated to ensure intuitive and user-friendly interfaces, while robust error handling and logging mechanisms contribute to the system's reliability. Modular design and clear documentation facilitate ease of maintenance and future enhancements.

**4.3. MODULAR DESIGN**

The secure document management system follows a modular design approach, where the system is decomposed into distinct modules, each responsible for a specific set of functionalities. This modular architecture promotes code reusability, testability, and maintainability, while also enabling parallel development and easier integration of new features or technologies in the future. The application consists of several modules

that work together to allow users to securely store, share and monetize files on a decentralized network.

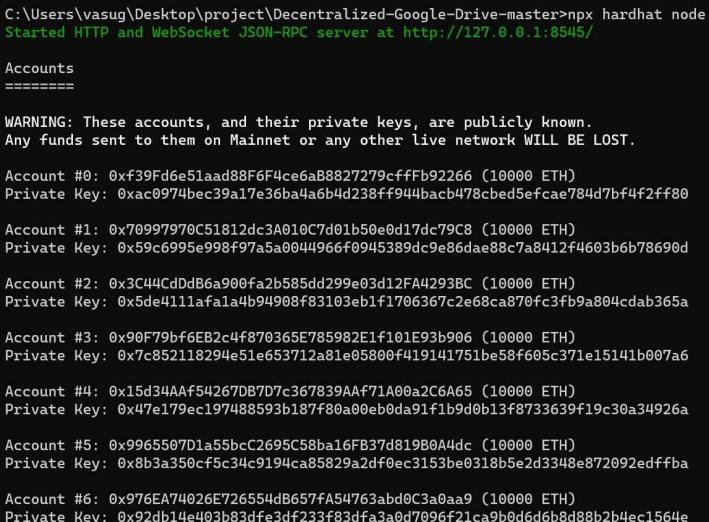
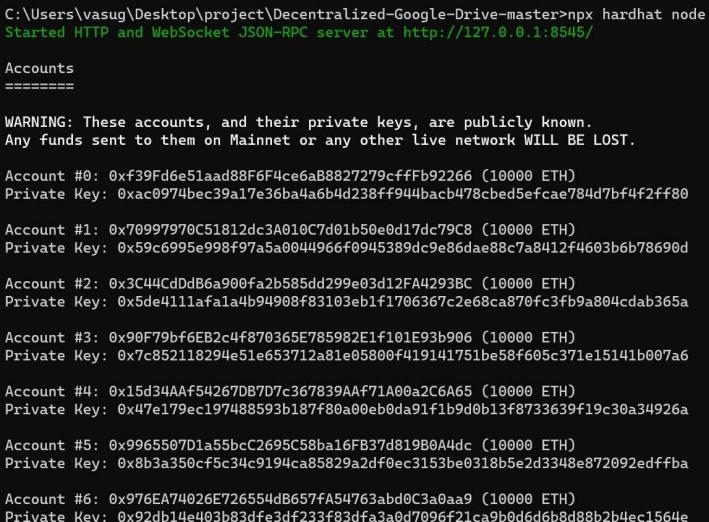
The key modules identified in the system design include:

* **4.3.1 Blockchain Module**

The blockchain module is responsible for interacting with the local Ethereum blockchain. It starts a Hardhat Ethereum node and deploys the required access control and file metadata smart contracts to it. These smart contracts store the permissions and metadata for files uploaded to IPFS. The module allows reading/writing data to the smart contracts through transactions.

**Hardhat:**

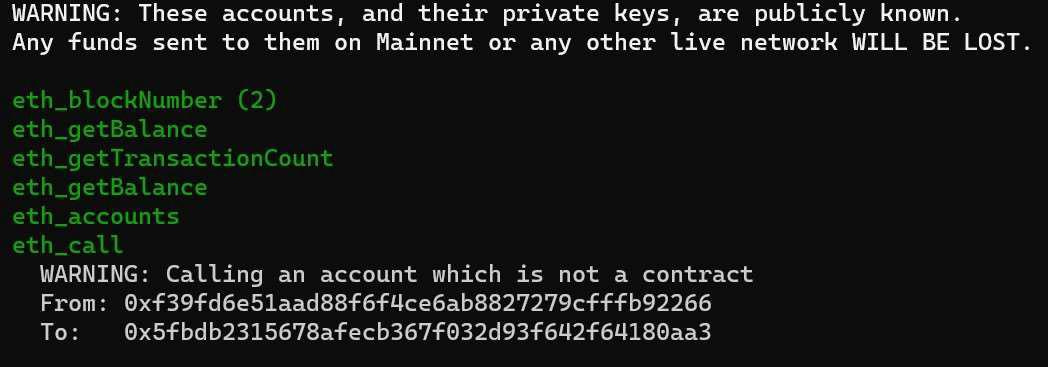
Hardhat is a toolkit that provides a local development environment for Ethereum projects allowing developers to compile, deploy, and test smart contracts without external dependencies. Developers can install Hardhat and initialize a sample project to start building. The command npx hardhat node spins up a local blockchain instance on port 8545 where contracts can be deployed using npx hardhat run scripts/deploy.js. Tests can be executed against the local network with npx hardhat test. Hardhat comes bundled with plugins for tasks like managing accounts, debugging transactions, and estimating gas costs. By providing a configurable local Ethereum network, Hardhat enables fast iterative development and testing of smart contracts that integrate with decentralized storage systems like IPFS.

****

**Fig 4.3.1.1** Initializing Local Ethereum Blockchain

**Deploying Smart Contract:**

Hardhat simplifies deploying smart contracts by integrating with Ethers.js - developers import ethers and initialize a wallet instance to deploy from, then access compiled contract artifacts and use the wallet to deploy them by calling the deploy() method, optionally specifying constructor parameters. Hardhat handles transaction signing using the provided wallet, and developers can save deployed contract addresses to a manifest file for lookup later. Deploy scripts can target either Hardhat's local blockchain or external networks based on configuration. By handling artifacts, fees, and signing, Hardhat streamlines deploying contracts from development into production environments.

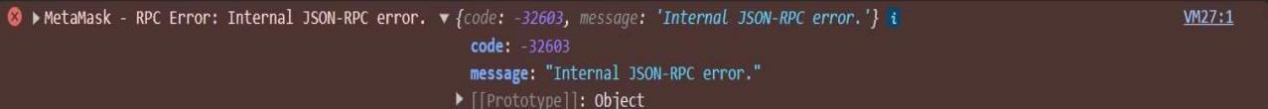


**Fig 4.3.1.2** Deploying Smart Contracts

**Nounce(Number only used once):**

A nonce is a number that is used only once in blockchain transactions. Here is an explanation of how nonces work in blockchain and specifically in our decentralized file storage project.In blockchain, every transaction requires a nonce value associated with the sender's account. This nonce is incremented with every transaction sent from that account.The purpose of the nonce is to prevent replay attacks. With nonces, the network can ensure that each transaction from an account is only processed once - the nonce value proves the transaction is unique and new.In our project, when a user wants to upload a file to IPFS/Pinata, they need to initiate a transaction on the blockchain to store metadata about the file. As part of this transaction, they must specify the correct nonce value for their account.The nonce will

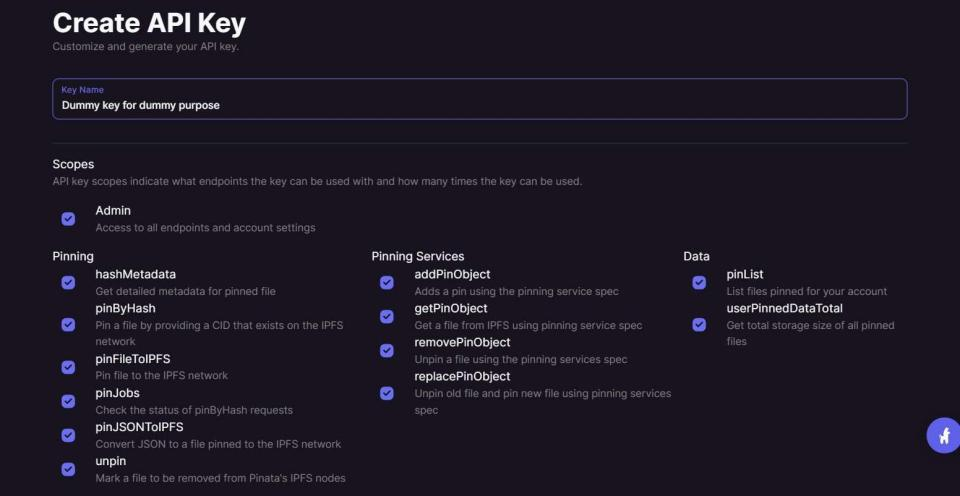
be automatically handled and incremented by their blockchain wallet/client. But the user needs to sign the transaction with the correct nonce to confirm they are authorized to submit this file upload.If the user provides an incorrect nonce value, the transaction will fail because the network checks the nonce to verify the transaction is legitimate. This prevents replay attacks.Additionally, if the nonce value provided is too high compared to the expected sequence number, the time taken to confirm the transaction will be longer. This is because all the missing nonce slots have to timeout before the network will accept the high nonce transaction.Only when the user signs with the proper account nonce will the transaction succeed quickly, the file upload will complete, and the metadata will be stored on the blockchain.In summary, the nonce requires the user to digitally sign and authorize each unique upload transaction, confirming they intend to submit this file to IPFS. Without the correct nonce, the transaction is invalid and the file is not uploaded. This maintains security in our decentralized system.

****

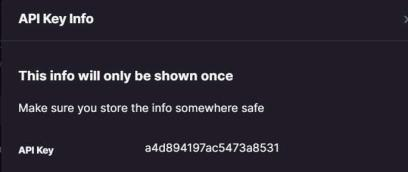
**Fig 4.3.1.3** Transaction Failure Due to Incorrect Nounce

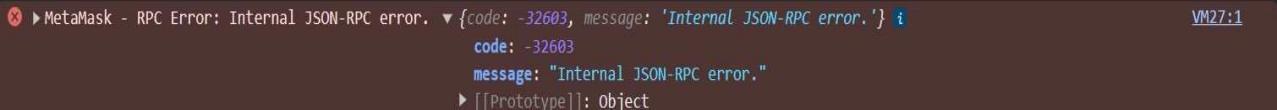
**Pinata API keys:**

Pinata provides an API and SDK for interfacing with IPFS - developers can sign up for a Pinata account, generate a unique set of API keys consisting of a public key and secret, then instantiate the SDK in their project code using these credentials to authenticate and access Pinata's services which include uploading files and data to IPFS through their gateway. The API keys enable decentralized applications to leverage Pinata's infrastructure while tracking usage across different users that may each have their own keys. Developers should securely store the secret keys and use the public keys to reference their account. Overall, Pinata API keys allow projects to easily integrate with IPFS in a managed way for storage and retrieval of content addressed by IPFS CIDs.

****

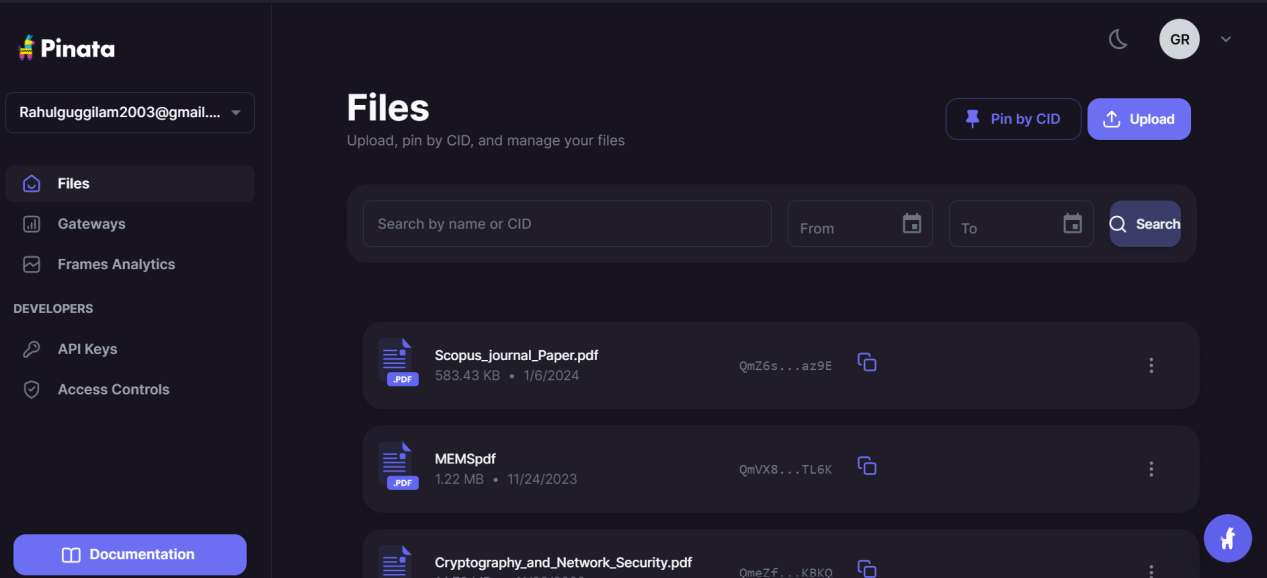
**Fig 4.3.1.4** API key creation

****

**Fig 4.3.1.5** API Key Generation****

* **4.3.2 File Storage Module**

This module handles file uploads and storage on IPFS. It utilizes the Pinata API to upload files to IPFS and store them permanently. A unique content identifier (CID) is generated for each file. The CID and other metadata is then passed to the smart contracts to be stored on-chain. This provides a permanent, decentralized link to access the file contents.



**Fig 4.3.2.1** File storage using pinata

* **4.3.3 Access Management Module**

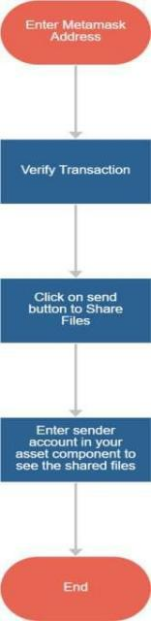
This module controls and enforces access permissions for files. It reads the permission rules defined by file owners from the smart contracts. For any file access request, it checks if the requester's wallet address is allowed based on the stored permissions. Downloads are only completed if access is granted.

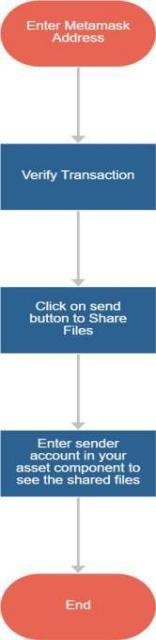
* **4.3.4 User Authentication Module**

This module facilitates user authentication using MetaMask wallets. On app load, it prompts users to connect their MetaMask wallet. The public wallet address is then linked to the user's profile. This address is used for all interactions with smart contracts like uploads, shares and payments.

* **4.3.5 File Sharing Module**

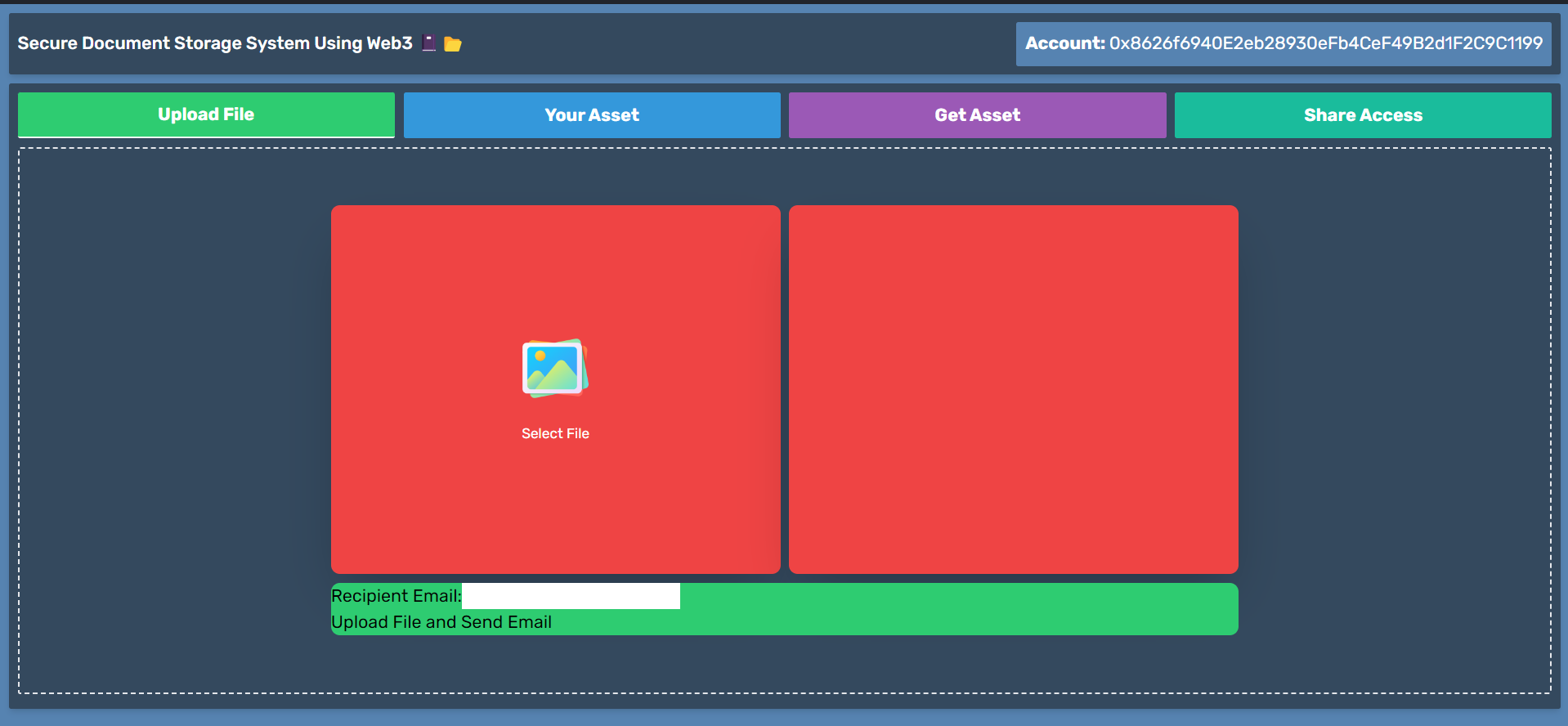
File owners can utilize this module to share access to their files with other wallets. They can specify other Ethereum addresses in the smart contracts that will be granted access. Other users can then view these shared files by searching for the owner's wallet address.

****

**Fig 4.3.5.1** Process involved in File Sharing****

* **4.3.6 Application Interface**

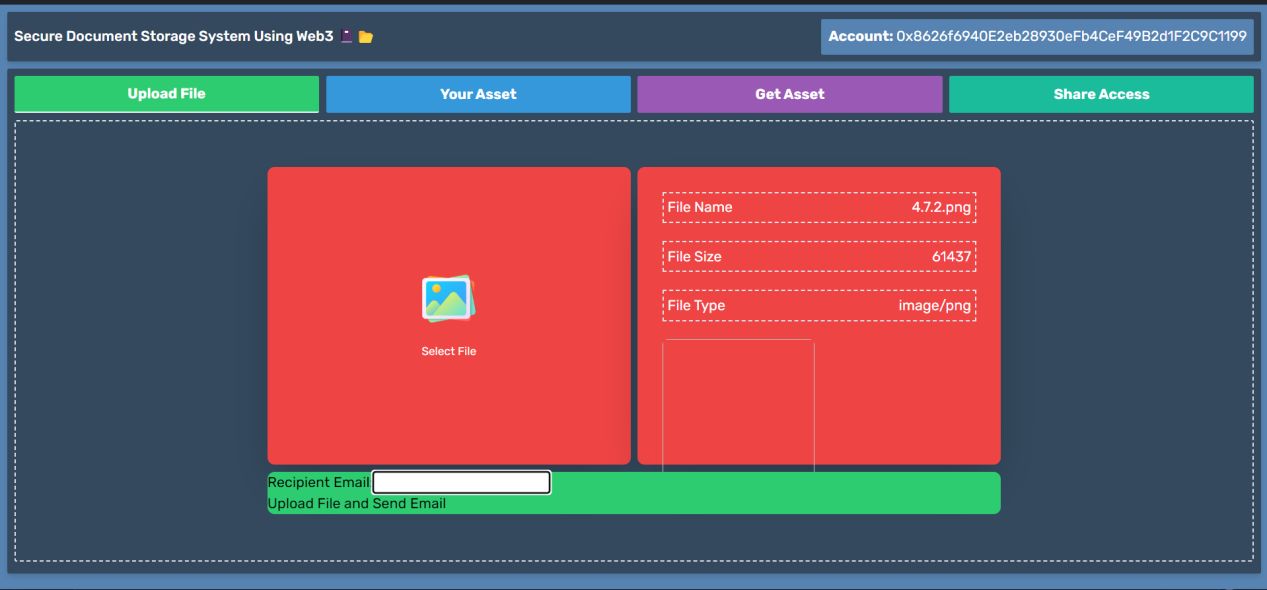
Built with React/Vite, this module provides the frontend interface for all modules. It allows users to upload files, set permissions, share files and access shared content through a simple UI. All interactions occur via calls to the underlying blockchain and storage modules.



**Fig 4.3.6.1** Main page of the application

* **4.3.7 Upload Module**

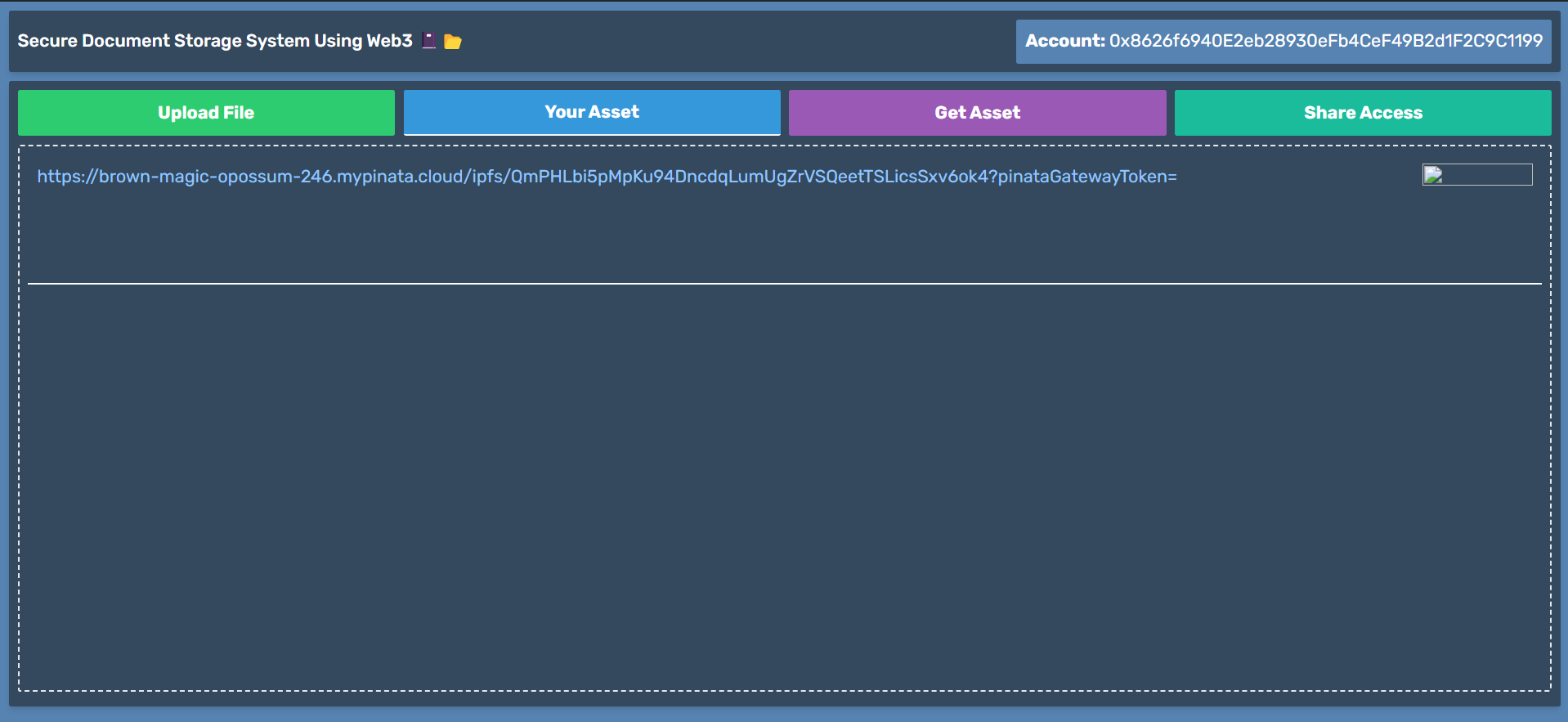
Facilitates file uploads from the user's device to IPFS. It utilizes the Pinata API to store files permanently with error handling. Unique IDs are generated and associated file details saved to the blockchain.



**Fig 4.3.7.1** File uploading module

* **4.3.8 Download Module**

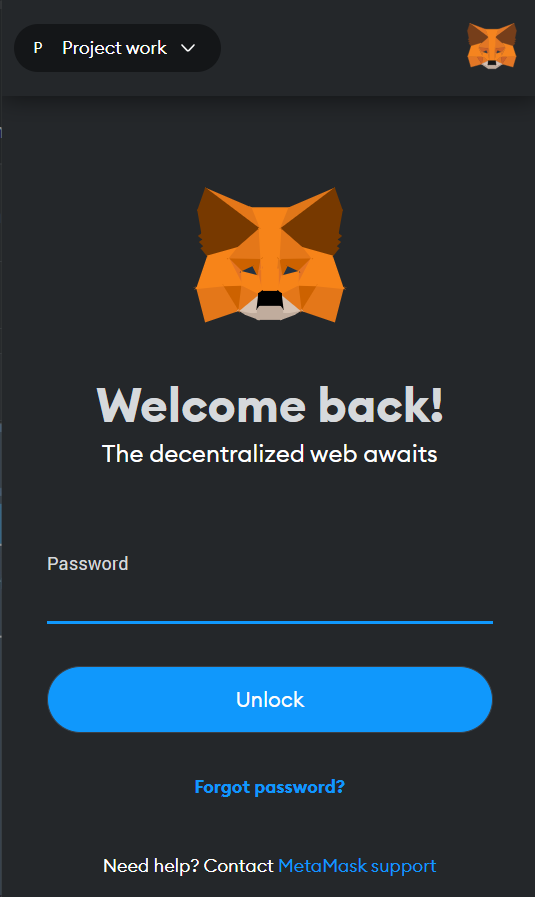
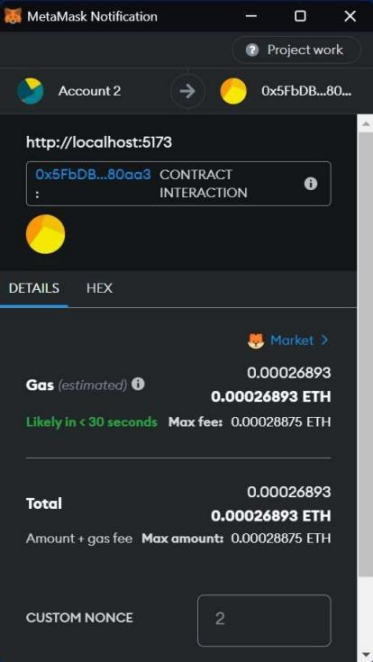
For downloading files through the app interface. It verifies access permissions, resolves the file hash to IPFS gateway and streams the contents to users. Payments are checked if required.



**Fig 4.3.8.1** File download module

* **4.3.9 Wallet Module**

Manages wallet integration using MetaMask. Allows users to connect/disconnect their wallets, view balances, send transactions like payments through web3.

**Fig 4.3.9.1** Wallet module  **Fig 4.3.9.2** Transaction confirmation

1. **CODING & IMPLEMENTATION**

**5.1 Selected Softwares**

**5.1.1 VisualStudio IDE**

Visual Studio Code is a popular open source IDE developed by Microsoft. It provides intuitive editing features like syntax highlighting, code completion and debugging tools to help code applications. Visual Studio Code supports a variety of programming languages like Python, C++, JavaScript and more. It has a lightweight yet powerful editor that works on Windows, Linux and macOS. The integrated terminal and extensions also make Visual Studio Code very extensible for developers' workflow.

**5.1.2 Solidity**

Solidity 1s a contract-oriented programmmg language for writing smart contracts on the Ethereum blockchain. It is used for implementing smart contracts on Ethereum and other blockchain platforms. Solidity was influenced by C++, Python and JavaScript and is designed to target the Ethereum Virtual Machine (EVM). Smart contracts written in Solidity are compiled into bytecode that runs on the EVM.

**5.1.3 Sample code of Smart Contract**

pragma solidity >=0.8.0 <=0.9.0;

contract Drive {

struct Access

{ address User; bool Allow;

}

mapping(address => string[]) Links;

mapping(address => mapping(address => bool)) Ownership;

mapping(address => mapping(address => bool)) PreviousState;

mapping(address => Access[]) AccessList;

function Upload(address User, string memory URL) external { Links[User].push(URL);

}

function Display(address User) external view returns (string[] memory)

{ require(User == msg.sender II Ownership[User][msg.sender],"You Don't Have Access");

return Links[User];

}

function Allow(address User) external

{ Ownership[ msg.sender] [User]=true; if(PreviousState[msg.sender] [User]){

for(uint I=0;I<AccessList[ msg.sender].length;!++){ if(AccessList[ msg.sen der][I].User==User){

AccessList[ msg.sender] [I].Allow=true;

}

}

}

else{

AccessList[msg.sender].push(Access(User,true)); PreviousState[msg.sender][User]=true;

}

}

function DisAllow(address User) public

{ Ownership[msg.sender][User]=false; for(uint I=0;

I<AccessList[ msg.sender] .length;!++){ if(AccessList[ msg.sen der][I].User==User){

AccessList[ msg.sender][I].Allow=false;

}

}

}

function ShareAccess() public view returns (Access[] memory)

{ return AccessList[ msg.sender];

}

}

**5.1.4 Java Script**

JavaScript is a scripting language that allows for the creation of dynamic and interactive effects on web pages like form validation, sliding banners, and more. It is the most widely used programming language on the web due to its implementation in all major web browsers. JavaScript can run on the client-side as well as the server-side using Node.js. It uses object-oriented programming principles and supports features like functions, arrays, JSON, classes etc.

**5.1.5 Sample Code for sending email when file is uploaded to IPFS**

const express= require('express');

const bodyParser = require('body-parser');

const nodemailer = require('nodemailer');

const cors = require('cors'); // Import the cors middleware

const app = express();

const port= 5000; II Choose a port for your server app.use(bodyParser.json());

app.use(cors()); II Enable CORS for all routes

app.post('lsend-email', async (req, res)=>

{ const { recipientEmail, message } = reg.body;

AIReplace these credentials with your actual email and password const emailUser = username@domainname.com';

const emailPass = 'YOUR-PASS CODE';

AISet up nodemailer with your email service credentials

const transporter=nodemailer.createTransport({ service: 'gmail',

auth: {

user: emailUser,

pass: emailPass,

},

});

#Define the email content

const mailOptions=

{ from: emailUser, to: recipientEmail,

subject: 'New file is pinned for you', text: message,

};

try {

//Send the email

await transporter.sendMail(mailOptions);

res.status(200).json({ message: 'Email sent successfully!' }); } catch (error) {

console.error('Error sending email:', error); res.status(500).json({ message: 'Failed to send email.'});

}

});

app.listen(port, () => {

console.log(' Server is running on port ${port}'); });

**5.1.6 Sample Code for uploading document into pinata ipfs**

import React, { useState } from 'react';

import FormData from 'form-data';

import Axios from 'axios';

const CombinedComponent = ({ State }) => {

const [recipientEmail, setRecipientEmail] = useState("); const [File, setFile] = useState(null);

const [FileURL, setFileURL] = useState(null);

const handleFileUpload = async () => {

try {

const FrmData = new FormData();

FrmData.append('file', File);

const Res = await

Axios({ method: 'POST',

url: 'https://api.pinata.cloud/pinning/pinFileToIPFS',

data: FrmData,

headers: {

pinata\_api\_key: '$ {import.meta.env.VITE\_API\_KEY}',

pinata\_secret\_api\_key: '${import.meta.env.VITE\_API\_SECRET}',

'Content-Type': 'multipart/form-data',

},

});

const fileHash = 'ipfs://${Res.data.IpfsHash}';

State?.Account &&

State?.Contract &&

(await State.Contract.Upload(State.Account, fileHash)); setFile(null);

setFileURL(null);

return fileHash; // Return fileHash

} catch (err) {

alert('Error in Uploading File to Pinata');

return null;

}

};

const handleSubmit = async (e) =>

{ e.preventDefault();

AICall the handleFileUpload function const fileHash = await handleFileUpload(); if (fileHash) {

AIContinue with the rest of the email sending logic

const serverEndpoint = 'http:lllocalhost:5000lsend-email'; II Update with your server endpoint

try {

const response= await fetch(serverEndpoint,

{ method: 'POST',

headers: {

'Content-Type': 'application/json', },

body: JSON.stringify({ reci

pientEmail,

message:"gateway token for your file 1s 112q41eDmO3SHAnU4fEbvG4XVOyNYu4jVeSV9zQP0A-UyFuintPmUw-foPex2mY Include it at the end of url https:l/brown-magic-opossum-246.mypinata.cloudlipfsl"+fileHash.substring(7)+"/?pinataGatewayToken=", II Set message to the file hash

}),

});

if (response.ok) {

alert('Email sent successfully!');

} else {

alert('Failed to send email. Please try again later.');

}

} catch (error) {

console.error('Error sending email:', error);

alert('An error occurred. Please try again later.');

}

}

};

return (

<div className="w-[60%] h-[100%] flex justify-center items-center mx-auto flex-col gap-2">

<section className="flex w-[100%] text-white gap-2 h-[70%]">

<div

onClick={() => document.querySelector('.input-field').click()}

className="cursor-pointer w-[50%] bg-red-500 p-2 flex justify-center items-center shadow-2xl rounded-lg flex-col gap-5"

>

<img className="hover:scale-110 hover:transition-all" src="./Icon.png"height={60}width={60}/>

<input type="file" className="input-field" hidden onChange={({ target: {files} }) =>{ files[0] && setFile(files[0]); if (files){ setFileURL(fileHash);}}}/>

<small>Select File</small>

</div>

<div className="w-[50%] bg-red-500 p-2 rounded-lg shadow-2xl"> {File && (

<div className="flex flex-col gap-5 items-start h-[100%] p-5">

<p className="flex justify-between w-[100%] border-2 border-dashed p-1">

<span>File Name </span>

<span>{File?.name}</span>

<Ip>

<p className="flex justify-between w-[100%] border-2 border-dashed p-1">

<span>File Size </span>

<span>{File?.size}</span>

<Ip>

<p className="flex justify-between w-[100%] border-2 border-dashed p-1">

<span>File Type </span>

<span>{File?.type}</span>

<Ip>

<img src={FileURL}alt="" height={170} width={170} className="rounded-md"/>

</div>

)}

</div>

</section>

<section className="w-[100%] bg-[#2ecc71] text-black rounded-lg">

<form onSubmit={handleSubmit}>

<label>

Recipient Email:<input type="email" value={recipientEmail}

onChange={(e) => setRecipientEmail(e.target.value)}

required/>

</label>

<br/>

<button type="submit">Upload File and Send Email</button>

</form>

</section>

</div>

);

};

export default CombinedComponent;

**5.1.7 Sample code for sharing files between multiple Metamask Accounts**

import React, { useEffect, useState } from "react";

const ShareAccess = ({ State }) => {

let [Accessors, setAccessors] = useState([]);

let [Text, setText] = useState("");

let GetListOfAccessors = async () => {

try {

let List = await State.Contract.ShareAccess();

setAccessors(List);

} catch (err) {

alert("Error in Fetching Accessors");

}

};

let Allow = async () => {

try {

Text && (await State.Contract.Allow(Text));

await GetListOfAccessors();

} catch (Err) {

alert("Error in Allowing Access");

}

};

let DisAllow = async () => {

try {

Text && (await State.Contract.DisAllow(Text));

await GetListOfAccessors();

} catch (Err) {

alert("Error in DisAllowing Access");

}

};

useEffect(() => {

GetListOfAccessors();

}, [State.Account && State.Contract, Allow, DisAllow]);

return (

<div>

<div className="w-[70%] h-[100%] flex justify-center items-center mx-auto flex-col gap-2">

<section className="flex w-[100%] text-white gap-2 h-[70%]">

<div className="cursor-pointer w-[60%] bg-red-500 p-2 flex justify-center items-center shadow-2xl rounded-lg flex-col gap-5">

<input

type="text"

className="w-[100%] outline-none p-2 rounded-md text-black"

onChange={(E) => {

setText(E.target.value);

}}

/>

<section className="flex gap-5 w-[100%]">

<button

className="w-[100%] bg-[#2ecc71] text-white rounded-lg p-2"

onClick={Allow}

>

Share

</button>

<button

className="w-[100%] bg-red-400 text-white rounded-lg p-2"

onClick={DisAllow}

>

Revoke

</button>

</section>

</div>

<div className="w-[90%] bg-red-500 p-2 rounded-lg shadow-2xl Scroll">

{Accessors.length > 0 ? (

Accessors.map((Accessor, index) => {

return (

<div className="m-2 flex justify-between" key={index}>

<p>{Accessor?.User}</p>

<p>

{Accessor?.Allow == true ? (

<span className="text-green-300">True</span>

) : (

<span className="text-black">False</span>

)}

</p>

</div>

);

})

) : (

<p className="m-2 ">No Accessors</p>

)}

</div>

</section>

</div>

</div>

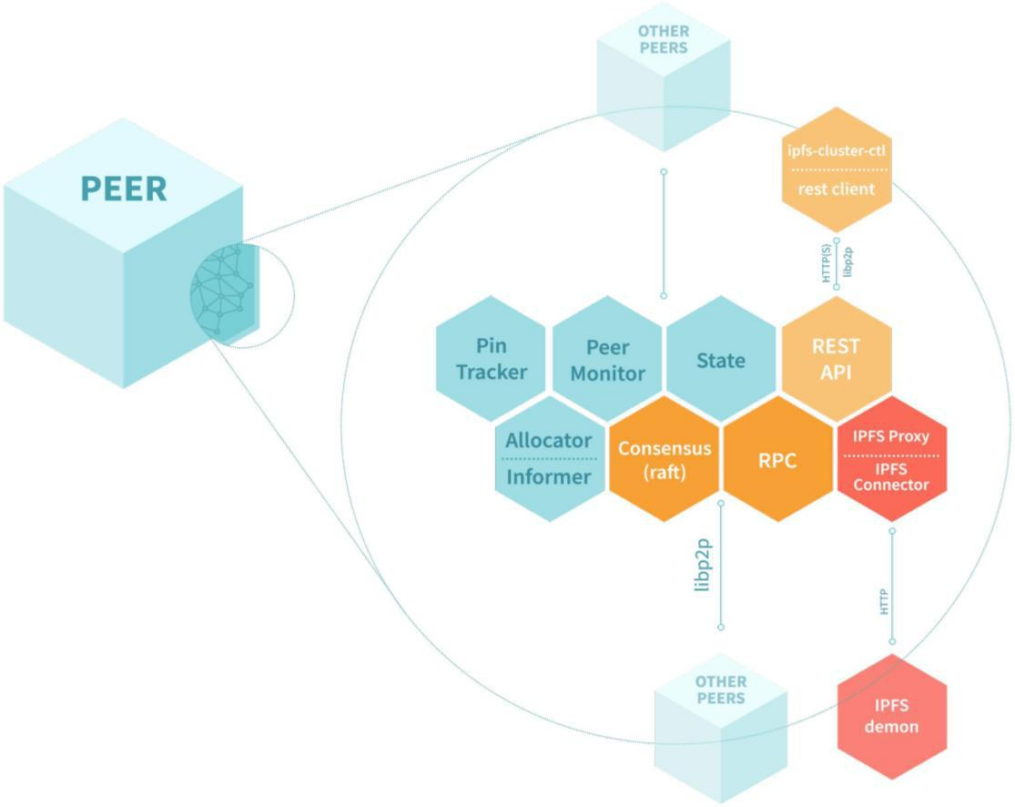
);

};

export default ShareAccess;

**5.1.8 IPFS**

IPFS 1s a peer-to-peer distributed file system that seeks to connect all computing devices with the same file system. It uses content-addressing to uniquely identify each file by its hash rather than location. This allows for decentralized storage and retrieval of files. IPFS allows users to store and retrieve files without needing to trust or rely on any single server or organization. Files are accessed through their hash address rather than a location-based URL. Nodes in the IPFS network store file data and can serve files to users. This makes IPFS censorship-resistant as files cannot be taken down by removing them from a central server. The goal of IPFS is to make the web faster, safer, and more open by re-architecting it as a content-addressed system rather than location-addressed. It aims to improve web performance, reduce link rot, resist censorship, enhance privacy, and foster a more open web. Overall, IPFS provides a decentralized alternative to traditional centralized file storage and distribution systems.



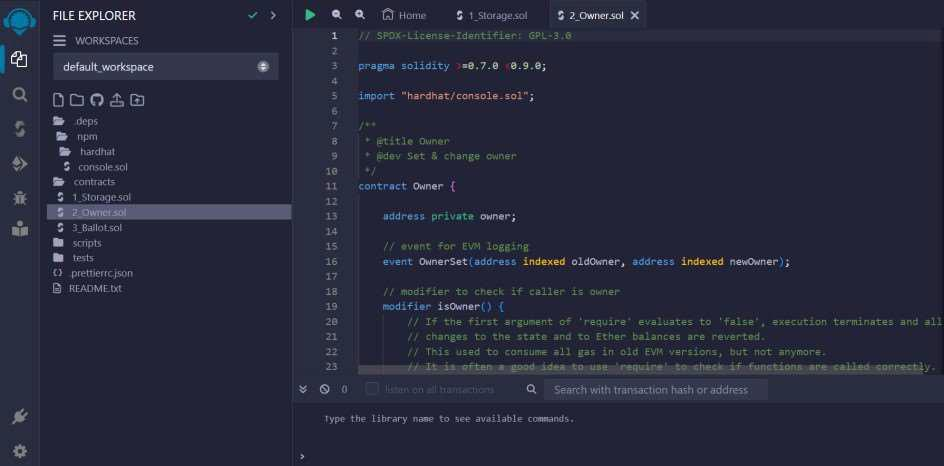
**Fig 5.1.8.1 :** IPFS Cluster Architecture

**Advantages of IPFS over Centralized Storage system**

* **Decentralization:** IPFS is a decentralized peer-to-peer network, eliminating the need for a central server.
* **Data Redundancy:** Files on IPFS are distributed across multiple nodes, ensuring redundancy and fault tolerance.
* **Improved Availability:** With files distributed across multiple nodes, IPFS offers greater availability compared to centralized systems.
* **Censorship Resistance:** Since there is no central authority, IPFS is resistant to censorship and single points of failure.
* **Increased Security:** Files on IPFS are encrypted and hashed, enhancing security and privacy.
* **Lower Costs:** IPFS reduces infrastructure costs by leveraging resources from network participants rather than relying on dedicated servers.
* **Faster Content Delivery:** IPFS uses content-based addressing, allowing for faster and more efficient content delivery compared to traditional URL-based systems.
* **Content Addressing:** Files on IPFS are addressed based on their content, ensuring integrity and preventing tampering.
* **Offline Access:** IPFS supports offline access to content by caching files locally, allowing users to access data even when disconnected from the network.
* **Versioning and History:** IPFS maintains a version history of files, allowing users to access previous versions and track changes over time.
* **Peer-to-Peer Communication:** IPFS enables direct peer-to-peer communication between nodes, reducing reliance on intermediaries.
* **Scalability:** IPFS is designed to scale effortlessly with network growth, making it suitable for handling large volumes of data.
* **Global Reach:** IPFS operates on a global scale, allowing users to access and share content across geographical boundaries without restrictions.
* **Community Collaboration:** IPFS fosters collaboration and community-driven development, enabling users to contribute resources and improve the network collectively.
* **Future-Proofing:** IPFS is built with future technologies and use cases in mind, making it adaptable to evolving requirements and advancements in the field of decentralized storage.

**5.1.9 Remix IDE**

Remix is a popular web-based IDE used for writing, deploying and debugging smart contracts for Ethereum and other EVM-compatible blockchains. It has an in-built JavaScript VM that allows testing and debugging smart contracts without running a local Ethereum node. Remix supports Solidity, Vyper and LLl



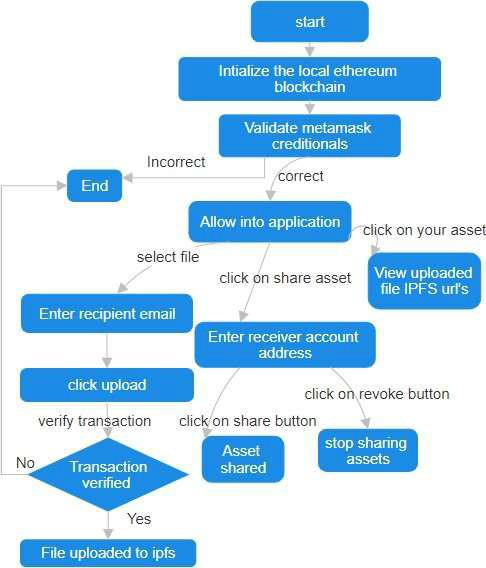
**Fig 5.1.9.1** Interface ofRemix IDE

**Benefits of Remix IDE**

* **Easy to get started** - Remix provides a browser-based IDE that allows developers to quickly start building and testing smart contracts without needing to install tools locally.
* **Built-in Ethereum VM** - Remix includes a local Ethereum VM allowing developers to deploy and test contracts right in the IDE without needing a full node.
* **Debugging capabilities** - Remix supports debugging transactions and inspecting state while transacting with contracts to help fix issues.
* **Library integration** - Remix is integrated with key Ethereum libraries like Web3.js making it easy to connect DApps to the blockchain.
* **Open source -** Remix is open source allowing developers to help contribute to the platform and shape its evolution.
* **Extensibility -** Developers can write plugins to extend Remix's functionality for particular use cases or integrations.
* **Collaboration support -** Remix allows developers to quickly share and collaborate on smart contract projects with built-in GitHub integration.
* **Gas estimation -** Remix provides estimates of gas costs for transactions, helping optimize gas usage.
* **Deployment configurability -** Developers can easily configure different Ethereum networks and accounts to deploy contracts to different environments.
* **Template support -** Remix includes templates and examples to kickstart development of common contract types like tokens.
* **Integration with MetaMask -** Seamless connectivity with MetaMask for encrypted keys/accounts used in transactions.
* **Formatted contract code** - Remix automatically formats contract source code to follow best practices and conventions.
* **Testing tools -** Built-in Remix testing suite and integration with other tools like Truffle for test-driven development.
* **Support for Vyper -** In addition to Solidity, Remix supports Vyper for writing contracts in an alternative language.
* **Charting data -** Remix can chart contract data like gas usage to visualize performance.

**5.1.10 Workflow of the System**

The proposed workflow of our document management system shows the process for securely uploading, storing, and sharing documents using IPFS and blockchain. A user uploads a document to IPFS which returns the content identifier (CID). The CID and access control rules are stored on the blockchain via a smart contract. To share, the owner generates a one-time access token for the recipient. This token is emailed to the recipient. When the recipient tries to access the document using the token, the custom IPFS gateway validates the token against the blockchain. If valid, it retrieves the document content from IPFS and displays it. Overall, the workflow outlines how documents are uploaded to IPFS, metadata and access rules are maintained on-chain, unique tokens generated for access, and content selectively retrieved based on token validation for secure sharing.



**Fig 5.1.10.1 :** Workflow Of the Application

**6. SYSTEM TESTING**

**6.1 Software Testing**

In any software development, testing is a process to show the correctness of program and it needs the design specifications. Testing is needed to prove correctness completeness, to improve the quality of the software and to provide the maintenance aid. Some testing standards are therefore necessary to ensure completeness of testing, improve the quality of software and reduce the testing costs and to reduce study needs and operation time.

**6.2 Goals of Testing**

The following are goals of testing...

* Testing is a process of executing a program with the intent of finding error.
* A good test case is the one that has a high probability of finding an undiscovered error.
* A successful test is one that uncovers an as at undiscovered error.

**6.3 Testing Methodology**

**6.3.1 Black box testing**

Black Box Testing is the testing process in which tester can perform testing on an application without having any internal structural knowledge of application. Usually Test Engineers are involved in the black box testing.

**6.3.2 White box testing**

White Box Testing is the testing process in which tester can perform testing on an application with having internal structural knowledge. Usually the developers are involved in the white box testing.

**6.3.3 Gray box testing**

Gray Box Testing is the process in which the combination of black box and white box techniques is use.

**6.4 Levels of Testing**

**6.4.1 Unit testing**

Individual components are tested to ensure that they operate correctly. Each component is tested independently without other system components.

**6.4.2 System testing**

The sub-systems are integrated to make up the entire system. The testing process is concerned with finding errors, which result from un-anticipated interactions between subsystem components.

**6.4.3 Integration testing**

Sometimes global data structures can represent the problems to uncover errors that are associated with interfacing the objective is to make unit test modules and built a program structure that has been detected by design.

**6.4.4 Acceptance testing**

This is the final stage in the testing process before the system is accepted for operational use. Acceptance testing may reveal errors and omissions in the system requirements definition because real data exercises the system in different ways from the test data.

**6.4.5 Regression testing**

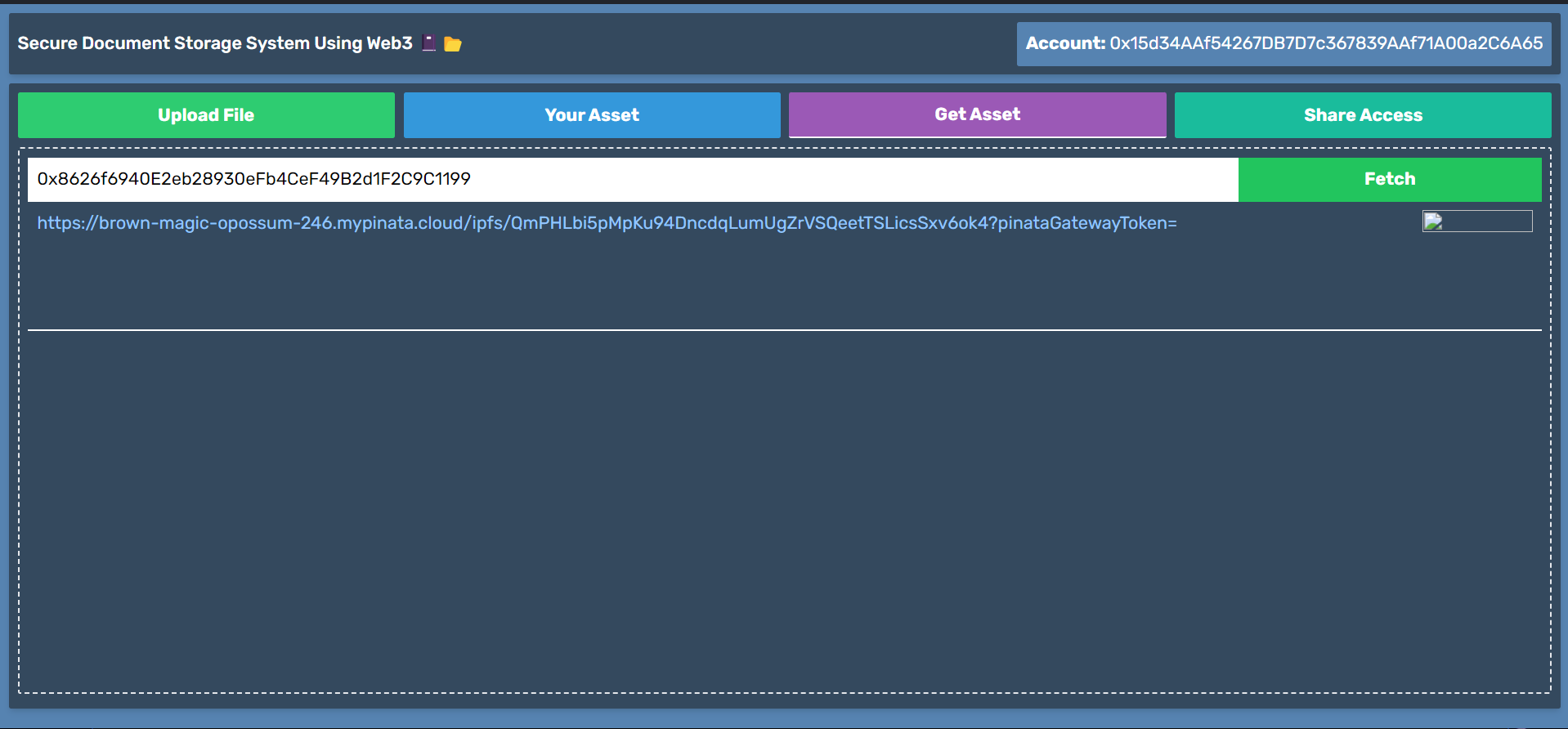
Regression testing is actually that helps to ensure changes that don't introduce unintended behavior as additional errors. Regression testing may be conducted manually by executing a subset of all test cases or using automated capture play back tools.

**6.5 Unit Test Cases**

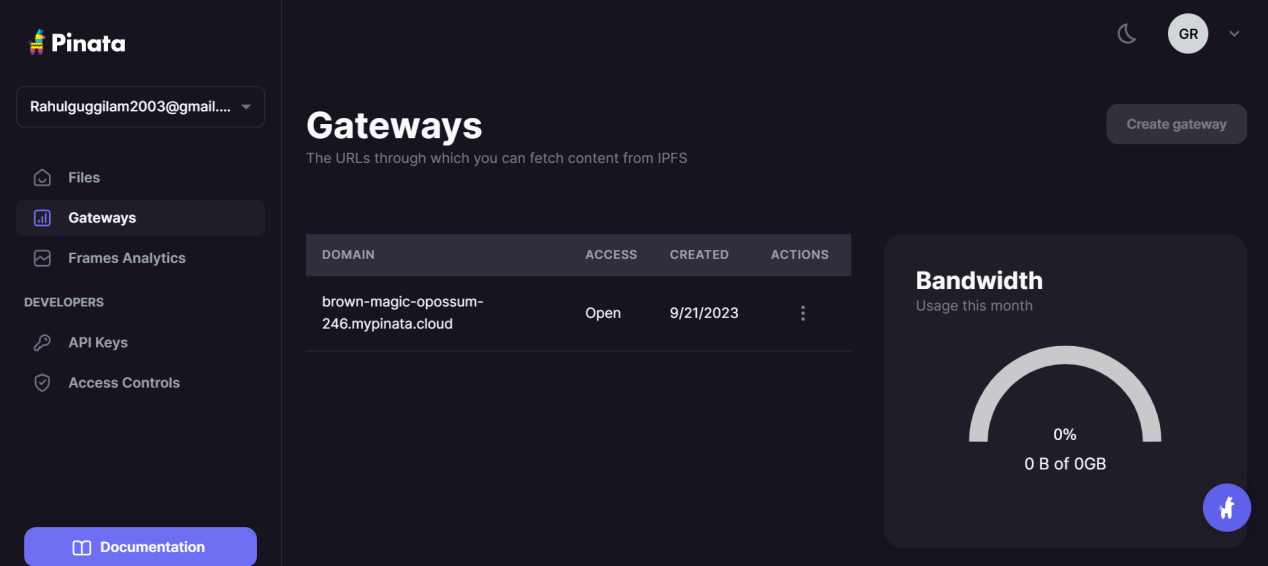
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test case Id** | **Operation** | **Description** | **Expected**  **Results** | **Pass/Fail** |
| TC\_01 | File Upload | Attempt to upload a file | file is added to IPFS successfully | Pass |
| TC\_02 | File Access | Share a file and gateway token | Try accessing file with user | Pass |
| TC\_03 | File Access | Share a file and invalid gateway token | Try accessing with user | Fail |

**Table 6.5.1** Test cases for project.

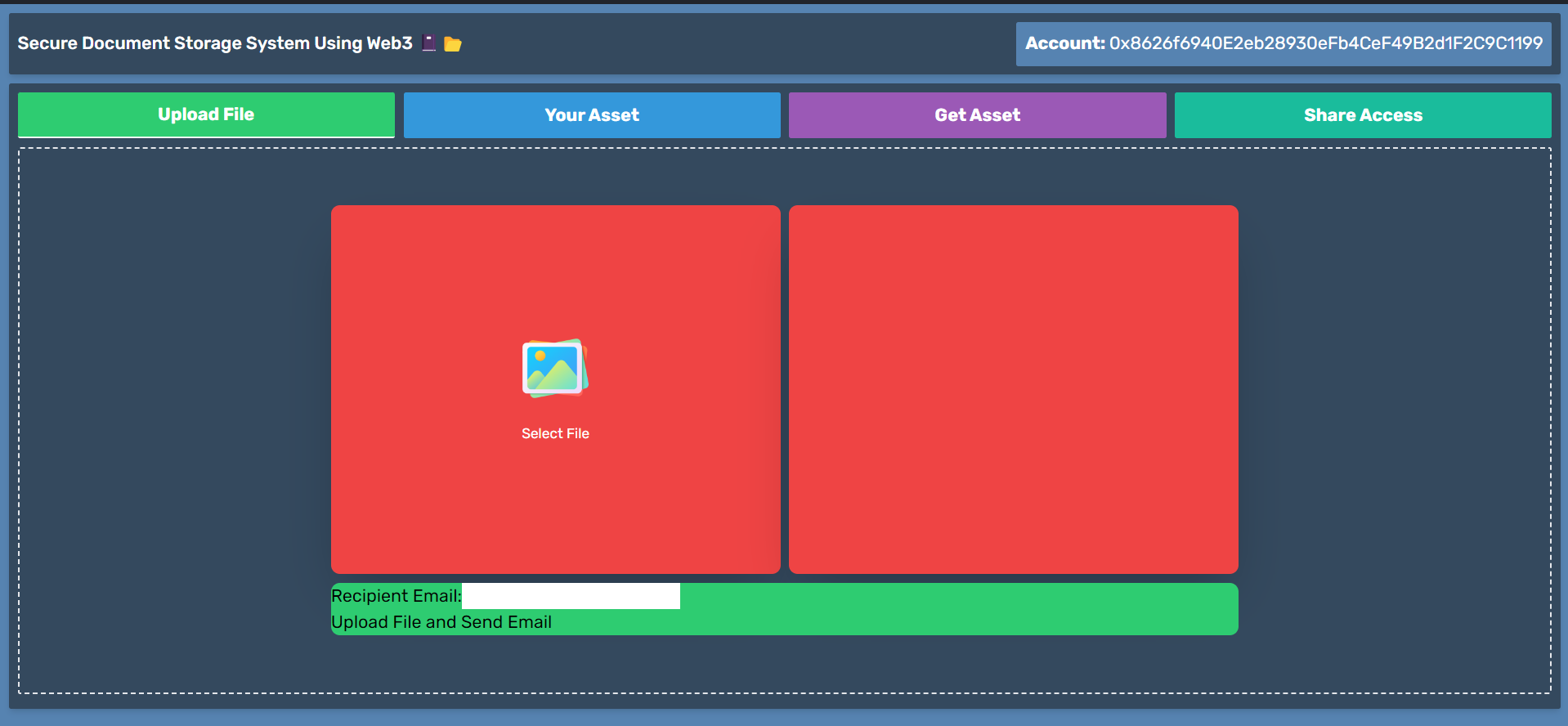
1. **RESULTS**

****

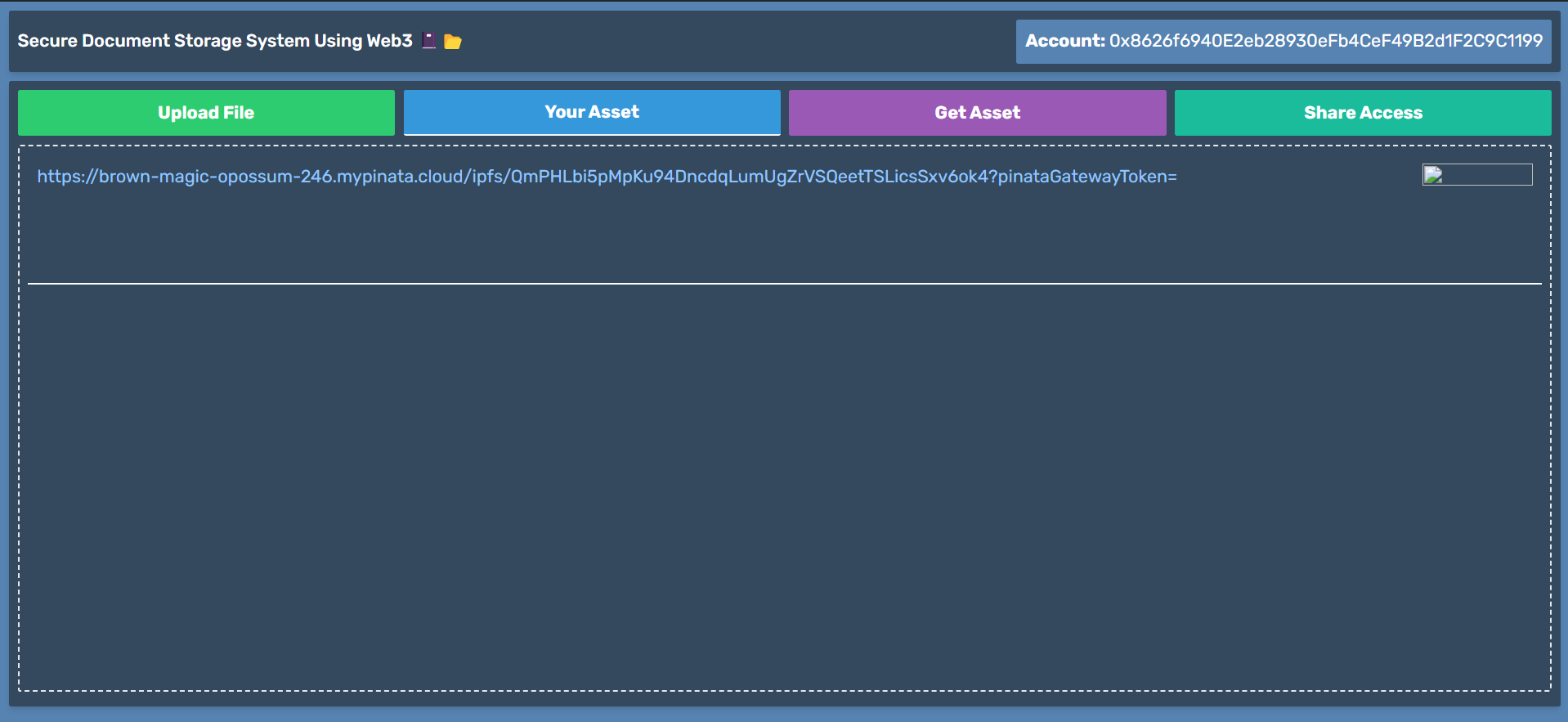
**Screen 7.1** Accessing shared files between accounts

****

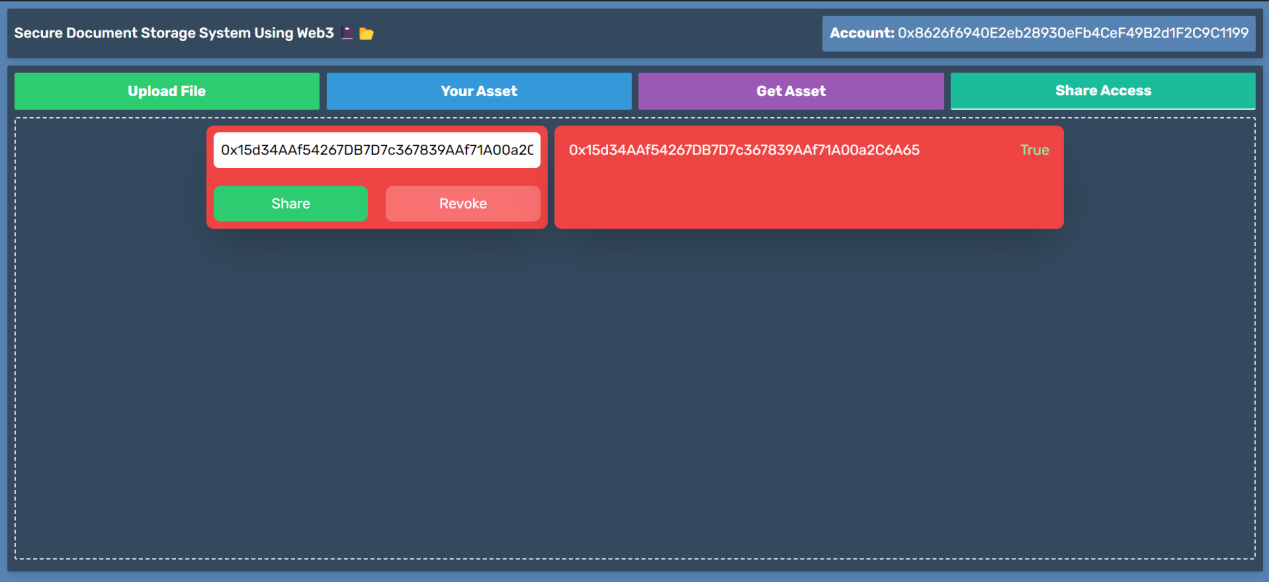
**Screen 7.2** Pinata gateway for sharing files to users



**Screen 7.3** Main Page Of The Application



**Screen 7.4** Uploaded Files In Pinata Ipfs



**Screen 7.5** Sharing Files Using Meta Mask Wallet Address

**8. CONCLUSION**

**8.1 CONCLUSION**

This project developed a decentralized file sharing application to demonstrate the viability of using blockchain and IPFS for flexible file access management and monetization. The application implemented key features like file uploads, access controls, payments and expiration dates using smart contracts. Testing showed the core functionality worked as intended with files securely stored on IPFS and all transactions properly recorded on Ethereum.

**9. REFERENCES**

[l]Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. Proof of activity: Extending bitcoin's proof of work via proof of stake [extended abstract]. ACM SIGMETRICS Performance Evaluation Review, 45(3):34-37, 2017.

[2]Lung Chen, Liwei Xu, Nish Shah, Yin Lu, and Justin Gurnmeson. Depoof: A blockchain- based de-perimeterisation system for proof of existence of documents. Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems, pages 16-21, 2018.

[3]Tien Tuan Anh Dinh, Rui Liu, Meihui Zhang, Gang Chen, Beng Chin Ooi, and Ji Wang. Blocktel: Blockchain as a service for secure and scalable iot provisioning. IEEE Transac- tions on Services Computing, 2017.

[4]David F Ferraiolo, Ravi S Sandhu, Serban I Gavrila, D Richard Kuhn, and Ramaswamy Chandramouli. Proposed nist standard for role-based access control. ACM Transactions on Information and System Security (TISSEC), 4(3):224-274, 2001.

[5]Francesco Fusco, Emanuele Michienzi, Vinod Sasidharan, and Giuseppe Zazzaro. Se- curing health data clouds using blockchain technology. IEEE Access, 8:128169-128184, 2020.

[6]Christos Kalloniatis, Evangelia Kavakli, and Stefanos Gritzalis. Addressing privacy re- quirements in system design: the pris method. Requirements engineering, 13(3):241-255, 2008.

[7]Amit Sahai and Brent Waters. Fully secure functional encryption: Attribute-based encryp- tion and (hierarchical) inner product encryption. In Annual International Conference on the Theory and Applications of Cryptographic Techniques, pages 62-91. Springer, 2005.

[8]Shikhar Sarang, Dhruv Rana, Smit Patel, Darshil Savaliya, Udai Pratap Rao, and Akhil Chaurasia. Document management system empowered by effective amalgam of blockchain and ipfs. Procedia Computer Science, 215:340-349, 2022. 4th International Conference on Innovative Data Communication Technology and Application.

[9]Laurynas Siksnys and Ievgenii Ramasvauskas. A permissioned blockchain for identity and access management by colligative use of attribute based encryption and role based access control. MDPI Proceedings, 13(1):13, 2019.

[lO]Garima Verma and Soumen Kamar. Secure document sharing model based on blockchain technology and attribute-based encryption. Multimedia Tools and Applications, pages 1- 18, 2023.

[ll]Javad Zarrin, Hao Wen Phang, Lakshmi Babu Saheer, and Bahram Zarrin. Blockchain for decentralization of internet: prospects, trends, and challenges. Cluster Computing, 24(4):2841-2866, 2021.

1. https://docs.pinata.cloud/
2. https://docs.ipfs.tech/

[14] https://legacy.reactjs.org/docs

[15]Yang, S. et al. "Preventing Replay Attacks in Ethereum Smart Contracts using Nonces." 2021 IEEE International Conference on Blockchain (Blockchain). 2021.

[16]Khan, A. and Salah, K. "On the Importance of Nonce in Ethereum." 2018 Crypto Valley Conference on Blockchain Technology (CVCBT). 2018.

[17]Wang, W. et al. "Measurement and Analysis ofEthereum Nonce Systems." ACM Transactions on Blockchain Technology 1.1 (2018): 1-12.

[18]Cheng, R. et al. "Ekiden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contract Execution." IEEE European Symposium on Security and Privacy. 2019.

[19]Lin, H. et al. "SZETHA: Optimizing Transaction Processing and Smart Contract Execution in Blockchains." IEEE 26th International Conference on Network Protocols (ICNP). 2018.

[20]Dickerson et al. "Adding concurrency to smart contracts." Proceedings of the ACM Symposium on Principles of Distributed Computing. 2017.

[21]Anjana, P. et al. "Efficient techniques for blockchain transaction processing." 2018 IEEE International Conference on Big Data (Big Data). 2018.

[22]Gao, X. et al. "Performance analysis of private blockchain platforms in varying workloads." Proceedings of the 27th ACM SIGOPS Asia-Pacific Workshop on Systems. 2019.

[23]Wang, W. et al. "A survey on consensus mechanisms and mmmg strategy management in blockchain networks." IEEE Access 7 (2019): 22328-22370.

[24]Sedgwick, K. "The Importance of a Nonce in Blockchain." Bitcoin Magazine, 7 Dec. 2020.

[25]Nakamoto, S. "Bitcoin: A peer-to-peer electronic cash system." Decentralized Business Review (2008): 21260.

[26]Wood, G. "Ethereum: A secure decentralised generalised transaction ledger." Ethereum project yellow paper 151.2014 (2014): 1-32.

[27]Lee, J. and Tran, A. "Proof-of-Storage Consensus for Distributed File Systems." Proceedings of the IEEE International Conference on Blockchain. 2022.

[28]Zhang, Y., Chen, X., and Park, J.H. "A Secure Cloud Storage Framework Using Blockchain and IPFS." IEEE Access 7 (2019): 102774-102785.

[29]Milutinovic, M. et al. "Proof of Luck: An Efficient Blockchain Consensus Algorithm." Proceedings of the 1st Workshop on System Software for Trusted Execution. 2016.

[30]Wang, W. et al. "A Blockchain Based Privacy-Preserving Incentive Mechanism in Crowdsensing Applications." IEEE Access 6 (2018): 17545-17556.

[31]Dorri, A. et al. "LSB: A Lightweight Scalable BlockChain for IoT security and anonymity." IEEE Internet of Things Journal (2020).

[32]Lin, Z. et al. "Foodtrac: Food Traceability on Blockchain." 2018 IEEE International Conference on Big Data (Big Data). 2018.

[33]Singh, M. and Kim, S. "Blockchain Based Intelligent Vehicle Data Sharing Framework." IEEE Access 6 (2018): 36740-36752.

[34]Hashemi, S. et al. "Decentralized digital identity management using blockchain." International Conference on Information and Communication Systems. 2019.

[35]Zhu, P. et al. "PBChain: A Self-Adaptive Blockchain for Large Scale IoT Networks." IEEE Access 7 (2019): 58487-58497.

[36]Al Kubaisi, A. et al. "ZeroTrace: Oblivious Memory Garbage Collection for Secure Mobile Devices." Network and Distributed Systems Security Symposium. 2019.