Al-Azhar University
Faculty of Engineering
Computers & Systems Engineering
Department

Devault:

A Blockchain-based, self-hosted, and end-to-end encrypted cloud storage.

A Project Submitted in Partial Fulfillment of the Requirements for the Degree of Bachelor of Science in Systems and Computers Engineering

Submitted by:
Abd El-Twab M. Fakhry *504055*
Hossam A. Eissa *504042*

Supervised by:
Dr. Abdurrahman Nasr

July 24, 2022

Letter of Approval

The undersigned certify that they have read, and recommended to the Faculty of Engineering for acceptance, a project entitled "Devault: A Blockchain-based, self-hosted, and end-to-end encrypted cloud storage" submitted by Abd El-Twab M. Fakhry and Hossam A. Eissa in partial fulfillment of the requirements for the degree of Bachelor of Science in Systems and Computers Engineering.

Examiner Committee President:

Dr. Ali ElSamary

Systems & Computers Engineering Department

Project Supervisor:

Dr. Abdurrahman Nasr

Systems & Computers Engineering Department

Examiner Member:

Dr. Muhammad Atef

Systems & Computers Engineering Department

Date of Approval:

Statement of Originality

This statement is to certify that to the best of our knowledge, the content of this thesis is our work. This thesis has not been submitted for any degree or other purposes.

We certify that the intellectual content of this thesis is the product of our work and that all the assistance received in preparing this thesis and sources has been acknowledged.

Abd El-Twab M. Fakhry
Hossam A. Eissa

July 22, 2022

To the person who helped...

Acknowledgements

In successfully completing this project, many people have helped me. I would like to thank all those who are related to this project.

Primarily, thanks to ALLAH (s.w.t), the Greatest, the Most Merciful, and the Most Gracious, Whose countless blessings bestowed upon me were kind, talented, and wise teachers, who provided me with sufficient opportunities and enlighten me towards this project.

I would like to express my deepest and most sincere gratitude to my family for everything they have done for me and all the love they gave to me. My mother, father, sisters, and brother. No words can express my love for them.

I would like to extend my deepest thanks to my supervisor, Dr. Abdurrahman Nasr for giving me the opportunity of undertaking this research work under his determined direction. His support, dedication, encouragement, excellent supervision, and guidance are what made this thesis possible.

Last but not least, I would like to thank my friends and colleagues, especially Al-Azhar ICPC Community (AIC) team members, who have helped me with their valuable suggestions and guidance and have been very helpful in various stages of project completion.

Thank You.

Abstract

We propose a Decentralized Application (DAPP) that uses the Ethereum smart contracts for data access control and uses the InterPlanetary File System (IPFS) as a distributed system for storing and accessing data. Moreover, the files get encrypted on the client side using AES-256-CBC symmetric encryption and split into smaller chunks, distributed across multiple computers, and assigned a hash to allow users to locate them. Its then served to the user via a peer-to-peer connection, similar to BitTorrent technology.

Our proposed solution overcomes the problem of the centralized web, data censorship, data hacking, and data loss.

KEYWORDS: *Blockchain; Cryptology; Peer-to-Peer Network; Decentralised Applications; Cloud Computing*

Contents

Chapter 1

Introduction

1.1 Background and Motivation

Data has become the biggest valuable asset for anyone. With the abundance of data and its ever-growing nature, its equally important to store it in an organized way such that its easily accessible from anywhere and secure. For this purpose, people use cloud storage to store and share their data with others.

Cloud storage has effectively replaced the traditional model of physical hardware storage for developers building apps and websites, as well as individual consumers storing their data. However, the centralized providers that provide cloud storage services have fostered a system with serious drawbacks like high fees, low flexibility, and a lack of alternatives.

One of the reasons that cloud storage is slow and encounters network bottlenecks is that the internet is now governed by HyperText Transfer Protocol (HTTP). And Its how you access websites, watch videos and download files. There are some problems with it, a lot of them stem from the fact that the current model is centralized, and this version of the web is called web 2.0.

Web 2.0 is the World Wide Web based on the concepts of social media, where the user can create content, post it online, and engage with other user-generated content. But the upcoming issue was that they did not own this content or the revenue being generated by it. The company that provided the platform for sharing the content has the maximum ownership of the revenue generated by that content. This led to the centralization of the data and traffic influence.

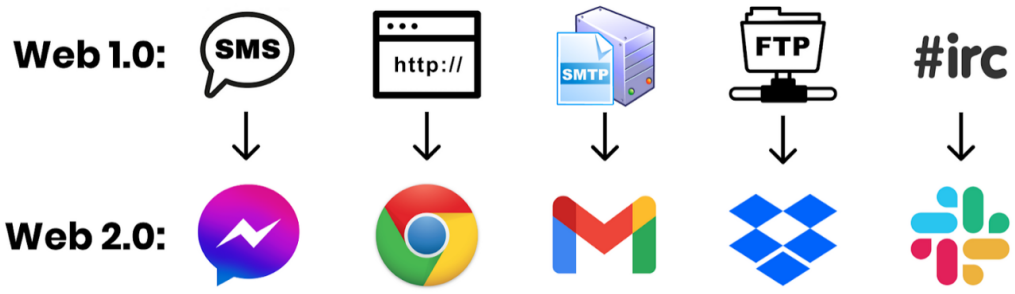


Figure 1.1: Web 2 build on top of the open protocols of Web 1, aggregating user data and creating an easy-to-use, simple user experience.

Web 3.0, unlike Web 2.0, has a decentralized distributed system. That means that all the nodes on the system in Web 3.0 have equal control and access. One of the key features of Web 3.0 is that it implements smart contract and Token using the Blockchain mechanism.

At its core, Web 3 aims to allow users to not only read and write content but also own their own content so that they cant get de-platformed or demonetized by centralized tech companies. Web 3 also aims to revert back to the open-source protocols of Web 1 where the rules are standardized for all market participants so that large tech companies cant stifle innovation and market competition.

1.2 Problem Statement

Different cloud service companies ensure data availability and safety. However, they have "terms of use" that allow the company to edit, modify, access, delete, view, and analyze your content. And that to provide the best possible service to the client, create an advertisement, manipulate it in some way to generate income, or use it for their purpose or analysis.

However, storing sensitive data only on local machines or drives can sometimes be very lamenting because once they are lost or destroyed by any other means, you cannot make a recovery. Moreover, most of the personal accounts of cloud storage also do not cover the insurance of data or take responsibility in case of data loss due to catastrophic failure. So, relying on data stored on your local machine only or the cloud storage is just not always safe and genuine.

As for the protocol itself, when you want to visit a website today, your browser (client) sends a request to the servers (host) that "serve" up that website, even when those servers are across the globe from your current location. This is location-based addressing, and it uses Internet Protocol (IP) addresses to show your location. That process eats bandwidth, and thus costs us a lot of money and time. On top of that, HyperText Transfer Protocol (HTTP) downloads a file from a single server at a time, which is way worse than getting multiple pieces of that file from multiple computers. It also allows for powerful entities to block access to certain locations, like Turkey did with the Wikipedia servers in 2017.

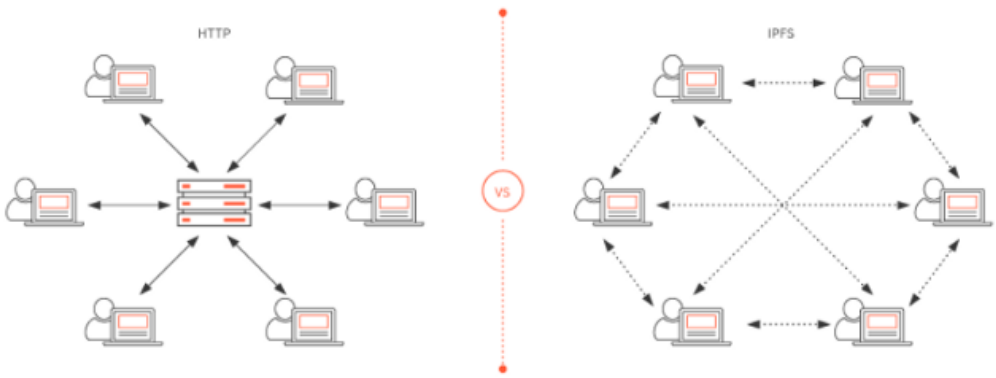


Figure 1.2: HTTP location-based addressing vs IPFS content-based addressing.

1.3 Aim of The Project

The project aims to design and implement a Decentralized Application that can store and retrieve files across the IPFS peer-to-peer network and uses the Blockchain as a public database. However, the Decentralized Application does not use traditional login. It uses a cryptocurrency wallet. Wallets give access to your funds and Ethereum applications. Only you should have access to your wallet. Moreover, we aimed to encrypt the files on the client side before uploading them so that it can prevent the data censorship and man-in-the-middle attack.

On the other hand, the project aims to research cryptography, peer-to-peer networks, web technology, and Blockchain and to contribute to the active research on decentralized applications and cryptography.

1.4 Organization of The Thesis

The work in this thesis is organized as follows:

- `Chapter 1`: gives a brief background and motivation, states the problem we are addressing, and introduces the project aims.
- `Chapter 2`: is a literature review about the integration between cryptography, Blockchain, and IPFS. And it states the workflow throughout the project and the objectives in every milestone.
- `Chapter 3`: gives an overview of the software structure, how it works, and a detailed account of the implementation and testing.
- `Chapter 4`: assesses the success of our project and summarizes the achievements and deficiencies of the project.
- `Chapter 5`: suggests ideas and enhancements that can be done and implemented in the future and give a brief statement of how the solution we provided addresses the problem.

Chapter 2
Background Materials

2.1 Introduction

The current standard for digital data storage is called cloud storage. With cloud storage, users looking to host data, applications, and websites on the internet are reliant on centralized providers like Amazon, Google, and Microsoft to provide storage services. This method of storage for which user data is stored on the centralized server farms of cloud storage providers is often cheaper, more scalable, and more readily accessible across geographic regions than the previous standard of storage on physical hardware.

Cloud service providers allow developers to launch their applications more quickly, without worrying about setting up and managing servers, but customers typically have limited options in terms of providers and functionality. The majority of cloud storage providers are subsidiaries of bonafide tech giants and dominate the cloud services market, accounting for about 70% of the total market share as of 2021.

Despite their popularity and widespread use, many centralized cloud storage providers have been criticized for their tendency to force end users into inflexible and expensive cloud services and storage plans due to a lack of viable alternatives. Studies have shown that many developers settle for fixed amounts of hosting space that remain underutilized. This often results in hefty and in many cases, unnecessary premiums paid for cloud services.

That said, perhaps the biggest concern with centralized data storage models is that users are required to place trust in the central authority of the provider to keep their data safe, keep websites online, and not tamper with or censor the content that the centralized data providers host. In response, blockchain technology and decentralized networks have fostered a whole new methodology for digital storage: decentralized cloud storage.

In contrast to centralized, permissioned cloud providers, decentralized cloud storage providers leverage infrastructure that is designed to mitigate undue control or influence. These providers typically also utilize a permissionless structure that enables developers to employ their services with reduced restrictions. Conceptually similar to a decentralized blockchain, decentralized storage models draw their security from their widely distributed structure. This overall architecture can help make these systems more resistant to the hackers, attacks, and outages that have plagued large, centralized data centers.

2.2 Need for decentralization

Decentralization is an ideology that advocates for a liberal style of administration in which no single authority has absolute power over all aspects of life. In a decentralized storage system, users may save their files without depending on large data hubs like the cloud.

Decentralization in data storage has gained recognition because of its user-friendly and trustable features like privacy and security. Decentralized data centers rely on a peer-to-peer network of users who each store small, encrypted chunks of the overall data. In this way, a reliable data storage and sharing system has created that can be founded on blockchain or any other peer-to-peer network.

Decentralized cloud storage is a storage system in which data is saved on various computers or servers. Its a decentralized peer-to-peer cloud storage system.

Some of the benefits of decentralized cloud storage are listed below:

- **Encrypted:** The nodes in a decentralized storage system are unable to see or modify your files since all data uploaded to a decentralized storage network is encrypted by default. As a result, you have unrivaled security and privacy, ensuring that your information is safe. Because of data encryption, nobody can access it without its unique hash. You can store personal and sensitive information without having any fear.

- **Secured:** Decentralized data storage systems, provide a high level of security. They split the data into smaller chunks, produce copies of the original data, and then use hashes or public-private keys to encrypt each portion independently. The entire procedure protects the data from malicious parties.

- **Flexible Load Balancing:** To make the process more efficient, blockchain-based decentralized storage systems allow the host to cache frequently-used data. It relieves server load and reduces network traffic. This eliminates the need for hosts to access the server on a regular basis to retrieve information.

- **Less Computer Power with Band Width:** Decentralized cloud storage encrypts data, breaks it up, and distributes it for storage on drives. It is operated by various organizations in a variety of locations, each with its own power supply and network connection, creating something much less wasteful. A decentralized file storage system reduces both hardware and software expenses. You also dont require high-performance equipment to use it efficiently. More significantly, a decentralized network may include millions, if not billions, of nodes. This significantly increases the amount of storage space accessible. Decentralized data storage does not need high power consumption to run on the system rather it uses less computer power with Bandwidth.

- **No dedicated Servers for Storage:** Decentralized cloud storage represents a paradigm shift to content-centric approach from a location-centric. One cannot access the database in decentralised cloud storage by just identifying where it is. Because data is distributed across a global network rather than being kept in a selected point, the principle of location becomes void in decentralised cloud storage.

  Unlike centralized storage systems where a finite few data centers host your data, decentralized storage networks are composed of a series of nodes eager to host the data in a secure manner. It does not only offer a wider range of storage bandwidth, but it also reduces the overall storage cost, making it a cost-effective option.

- **Fast:** It is commonplace to encounter network bottlenecks with centralized storage systems as the network traffic may sometimes overwhelm the servers. In a decentralized storage network, though, multiple copies of data are stored across various nodes. This eliminates the probability of network bottlenecks as you can access your data from a huge number of nodes, in a fast and secure manner.

  Above were some of the advantages of decentralized cloud storage over traditional cloud storage which do not need any explanation. Seeing above advantages we can say that it can be future of cloud storage in coming years.

2.3 IPFS, The Decentralized File System

2.3.1 What is IPFS

The Interplanetary File System (IPFS) is a bundle of subprotocols and a project-driven by Protocol Labs, IPFS aims to improve the webs efficiency and to make the web more decentralized and resilient.

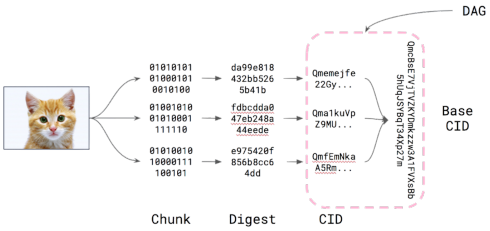
IPFS uses content-based addressing, where content is not addressed via a location but via its content. IPFS stores and addresses data with its deduplication properties, allowing efficient storage of data. It also can be used as a storage service complementing blockchains, enabling different applications on top of IPFS.

2.3.2 How IPFS works

IPFS is a peer-to-peer storage network. Content is accessible through peers located anywhere in the world, that might relay information, store it, or do both. IPFS knows how to find what you ask for using its content address rather than its location.

There are three fundamental principles to understanding IPFS:

(a) From raw image to cryptographic digest to content id (multihash).



(b) Large files are chunked, hashed, and organized into an IPLD (Merkle DAG object).

Figure 2.1: IPFS base CID construction

- Unique identification via content addressing.
- Content linking via directed acyclic graphs (DAGs).
- Content discovery via distributed hash tables (DHTs).

These three principles build upon each other to enable the IPFS ecosystem.

IPFS use cryptographic hashing functions to create fingerprints. Basically, we take the raw content (in this case, a cat photo), and run that data through a hash function, to produce a digest. This digest is guaranteed to be cryptographically unique to the contents of the file (or image or whatever), and that file only. If I change that file by even one bit, the hash will become something completely different.

Figure 2.1 shows how the content identifier is generated from the raw file.

The content is chunked up into smaller parts (about 256k each), each part is hashed, a CID is created for each chunk, and then these chunks are combined into a hierarchical data structure, for which a single, base CID is computed. This data structure is essentially something called a Merkle DAG, or directed acyclic graph.

2.3.3 Need for IPFS protocol

Web 3.0 is a long-term target that aims to replace the current internet infrastructure. As decentralization is the essence of Web 3.0. Many consider the distributed ledger technology (DLT), for example, blockchains, as the core building block of Web 3.0. A blockchain is an immutable and append-only ledger that stores the network state. Distributed consensus between all the network nodes is required in order to extend the blockchain and store the critical network data among the network nodes. Therefore, it could be prohibitively expensive to store any other kinds of data into the blockchain. For multiple use cases, it may be more efficient to store other non-critical data in a secure fashion close to the security level of the blockchain.

IPFS is the most suitable storage medium for this category of data. IPFS allows for distributed storage of data that is immune to altering and forgery. Data stored on the IPFS network cannot be altered without changing the data identifier. In IPFS, the identifier is a cryptographic hash of the data. This means non-critical

data can be stored to IPFS while storing this identifier to an underlying distributed ledger. This would result in less exhaustive operations over the distributed ledger.

2.3.4 The Optimal Storage Platform for dApps

Decentralized applications (dApps) are a class of applications that leverage decentralization to achieve unprecedented benefits. Among those are decentralized exchanges and marketplaces where centralized intermediaries are removed, hence eliminating/reducing any trading fees. Another example is decentralized social media and video platforms where content cannot be censored at the will of the operating company. Such dApps require the storage of a significant amount of data. IPFS allows this data to be stored in a distributed way that is censorship-resistant. For these reasons, IPFS is turning into a preferred storage platform for dApps.

2.4 The Necessity of Blockchain

2.4.1 What is Blockchain

A blockchain is a growing list of records, called blocks, that are securely linked together using cryptography. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data (generally represented as a Merkle tree, where data nodes are represented by leafs). The timestamp proves that the transaction data existed when the block was published to get into its hash. As blocks each contain information about the block previous to it, they form a chain, with each additional block reinforcing the ones before it. Therefore, blockchains are resistant to modification of their data because once recorded, the data in any given block cannot be altered retroactively without altering all subsequent blocks.

Blockchains are typically managed by a peer-to-peer network for use as a publicly distributed ledger, where nodes collectively adhere to a protocol to communicate and validate new blocks. Although blockchain records are not unalterable as forks are possible, blockchains may be considered secure by design and exemplify a distributed computing system with high Byzantine fault tolerance.

2.4.2 Blockchain in The Cloud Storage Use Case

For the system to be completely decentralized, we do not use centralized databases owned by a single entity, but we use Blockchain to keep track of users' files' metadata.

The immutable nature of blockchain, and the fact that every computer on the network is continually verifying the information stored on it, makes blockchain an excellent tool for storing data.

The blockchain will help reduce the security and data loss concerns associated with a typical public storage solution. Also, it will help reduce downtime on the

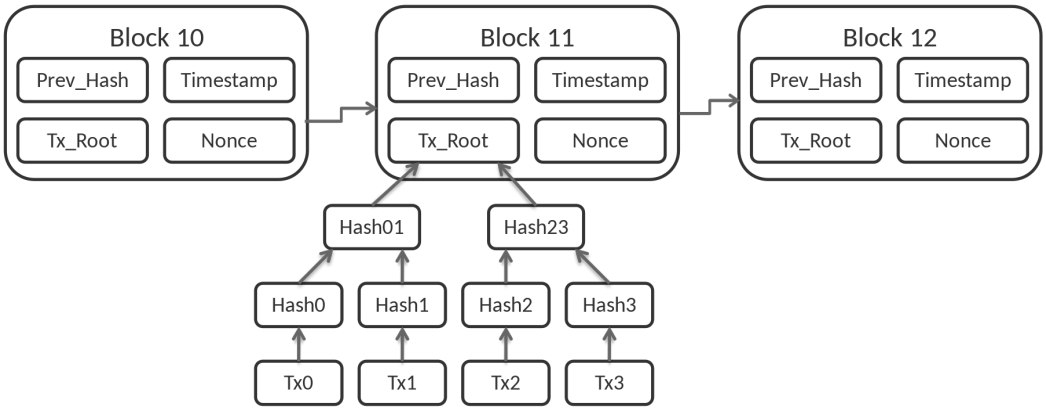


Figure 2.2: Bitcoin blockchain structure.

cloud. Users with excess storage will have the option of renting out their spaces for a fee.

There are many advantages to using blockchain technology compared to other traditional technologies listed below.

- **Transparency:** Blockchain makes transaction histories more transparent than they ever were. Because it is a type of a distributed ledger, all nodes in the network share a copy of the documentation. The data on a blockchain ledger is easily accessible for everyone to view. If a transaction history changes, everyone in the network can see the change and the updated record. Therefore, all information about currency exchange is available to everyone.

- **Security:** Blockchain is better than any other record-keeping system when it comes to security, by all standards. The shared documentation of transactions can only be updated and/or modified with consensus on a blockchain network. Only if everyone or a majority of nodes agree to update a record, the information is edited. Moreover, when a transaction is approved, it is encrypted and connected with the previous transaction. Therefore, no one person or party has the potential to alter a record.

- **Traceability:** In complex supply chains, it is hard to trace products back to their origins. But, with blockchain, the exchanges of goods are recorded, so you get an audit trail to learn where a particular asset came from.

- **Cost reduction:** As blockchain eliminates the need for third-parties and middlemen, it saves enormous costs for businesses. Given that you can trust the trading partner, you dont need anyone else to establish the rules and policies of exchange.

2.5 Infura, The Blockchain Node as a Service

When the transaction is signed, it has to be broadcasted to the blockchain network so miners know it and can include it in the block.

By connecting the wallet to a blockchain node, the node can run locally or connect to an Infura node. Infura takes care of keeping the nodes upgraded, online, and scaled; it handles as many transactions as you send to it. This is what Metamask does. Metamask sends all signed transactions through Infura nodes to the Ethereum network.

All blockchain nodes constantly communicate in-real time using peer-to-peer protocols. Peers exchange information about other peers, blocks, and transactions. When Infura nodes receive your signed transaction, Infura nodes propagate it further to other nodes and miners.

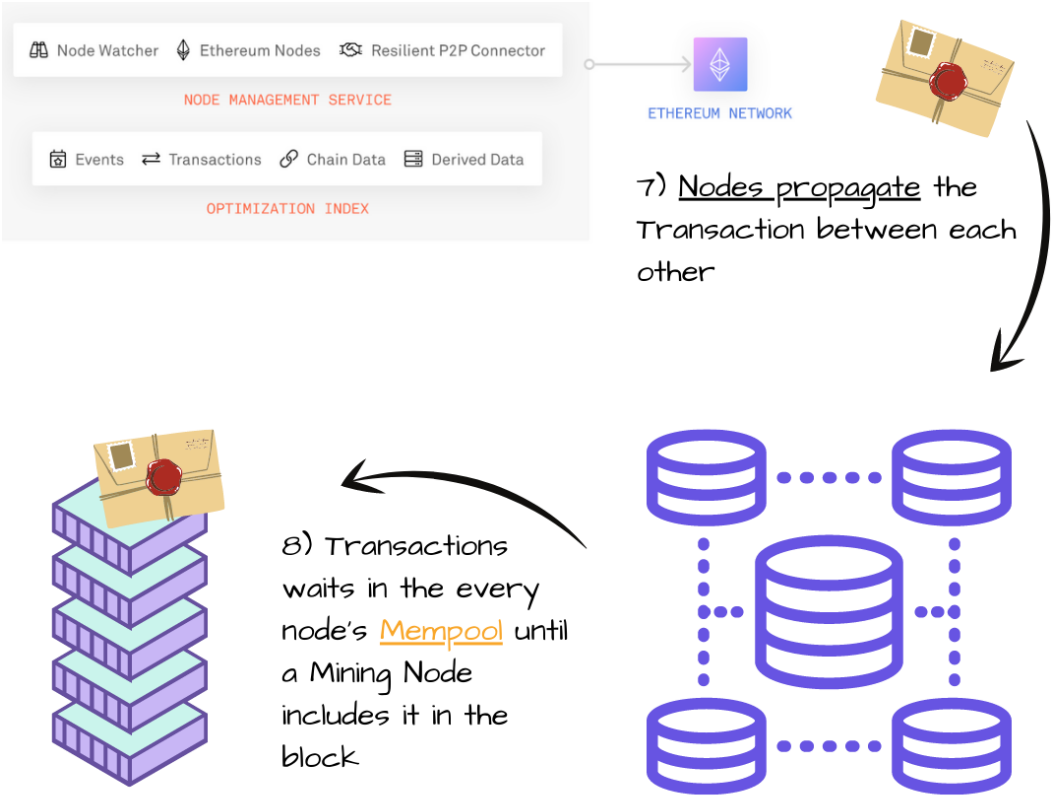


Figure 2.3: Infura nodes propagate the TX to other nodes and miners.

2.6 Metamask, The Wallet Auth

To transact on Ethereum, you need an account. There is no MySQL "users" table. There is no email/password login.

To create an Ethereum account, you need to set up a crypto wallet like Metamask. The wallet will be responsible for generating and securing your crypto keys for signing transactions.

Metamask generates your Private Key. You derive the Public Key from the Private Key. Your account's address is the last 20 bytes of a hashed Public Key.

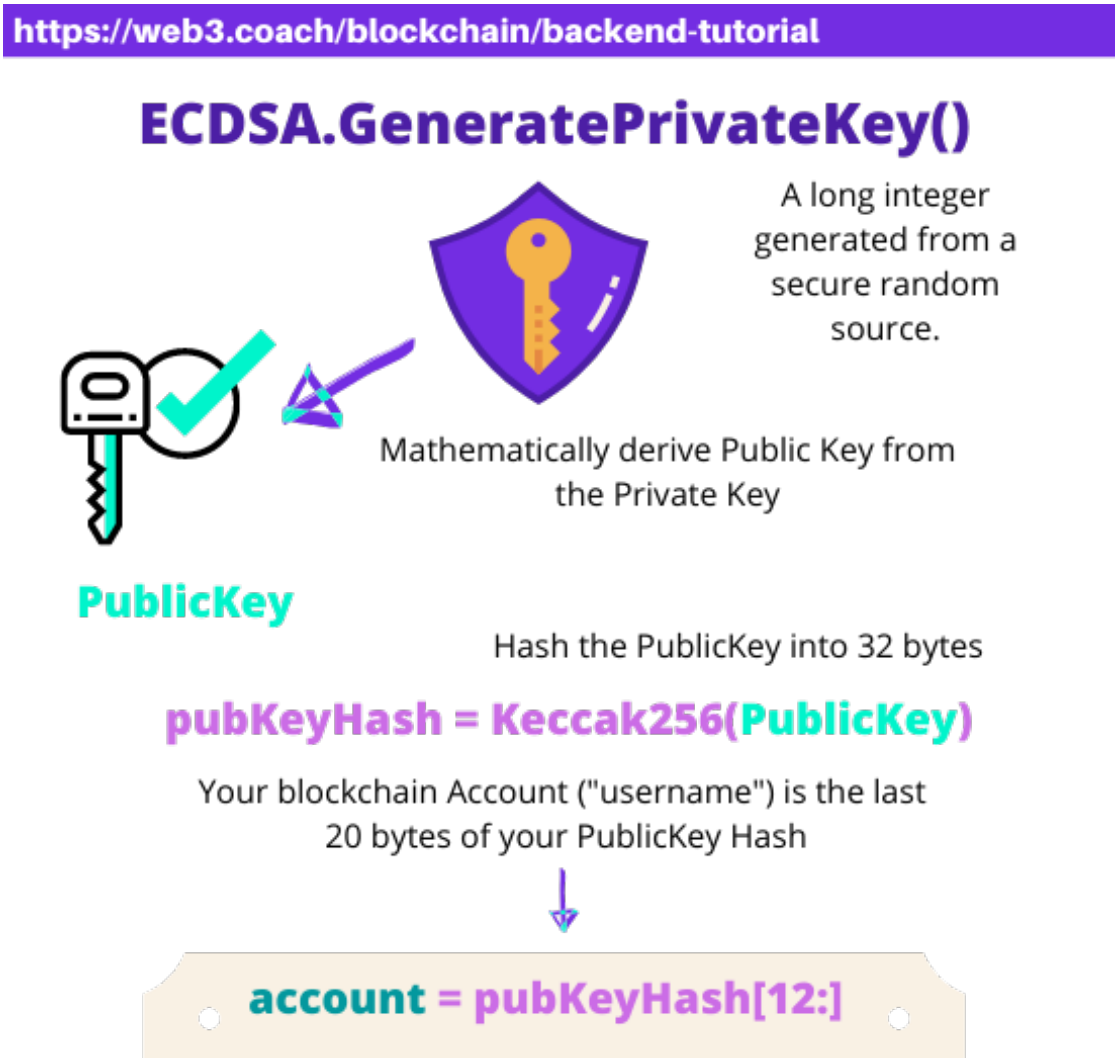


Figure 2.4: Generating keys and addresses.

The JS libraries will then connect to your Metamask wallet (if you permit them to do so), and they will ask you to sign Ethereum transactions. A decentralized application can't do anything without your signature.

Figure 2.5 show the transaction flow diagram.

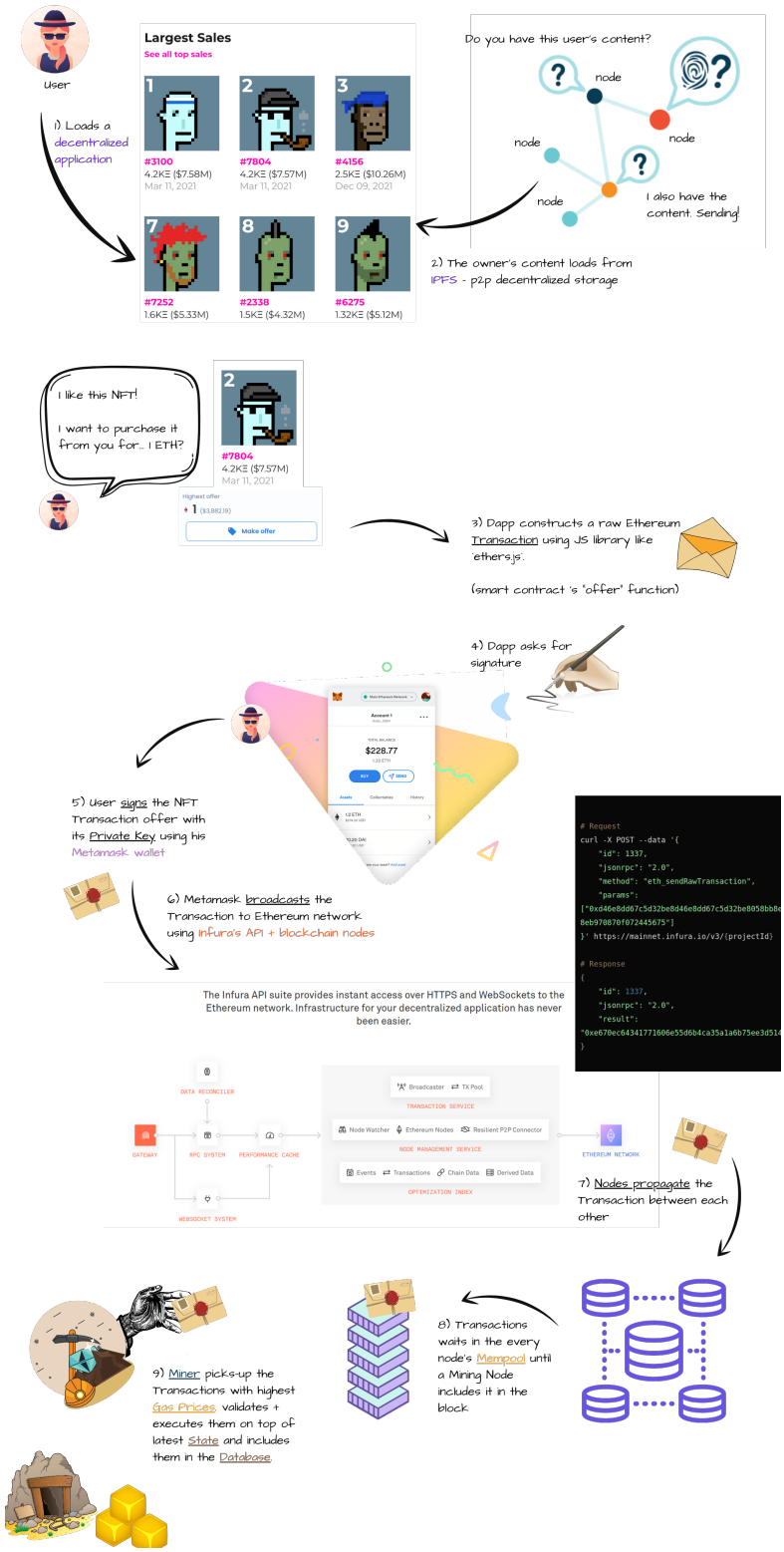


Figure 2.5: Transaction flow diagram.

2.7 Methodology

2.7.1 Development Methodology

In this project we use both waterfall and scrum methodologies. We used the waterfall methodology for developing the smart contract because we can not update it once we deploy it. And for the rest, we used the scrum methodology.

Scrum, is a framework for project management, with an initial emphasis on software development, although it has been used in other fields including research, sales, marketing and advanced technologies. It is designed for teams of ten or fewer members, who break their work into goals that can be completed within time-boxed iterations, called sprints, no longer than one month and most commonly two weeks. The scrum team assesses progress in time-boxed daily meetings of 15 minutes or fewer, called daily scrums (a form of stand-up meeting). At the end of the sprint, the team holds two further meetings: the sprint review which demonstrates the work done to stakeholders to elicit feedback, and sprint retrospective which enables the team to reflect and improve.

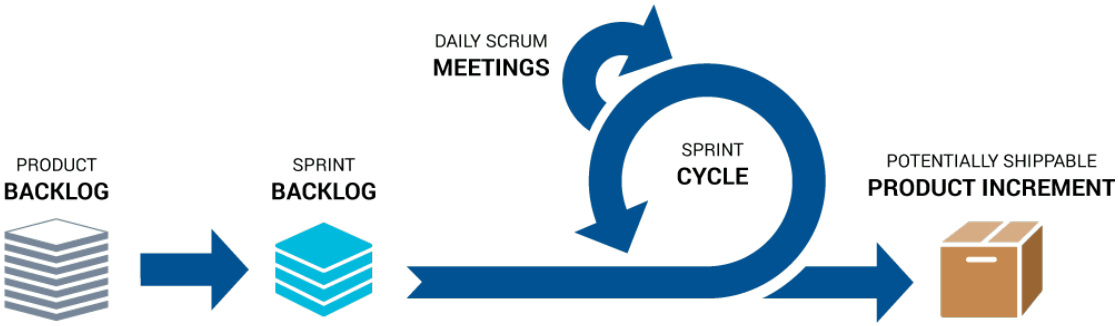


Figure 2.6: Agile scrum development process.

2.7.2 Use Case Modeling

A use case diagram is a graphical depiction of a user's possible interactions with a system. A use case diagram shows various use cases and different types of users the system has and will often be accompanied by other types of diagrams as well. In Devault the user can upload, download, share, delete, sort, and search files. Also the user can connect and disconnect their blockchain wallet.

Use Case Diagram

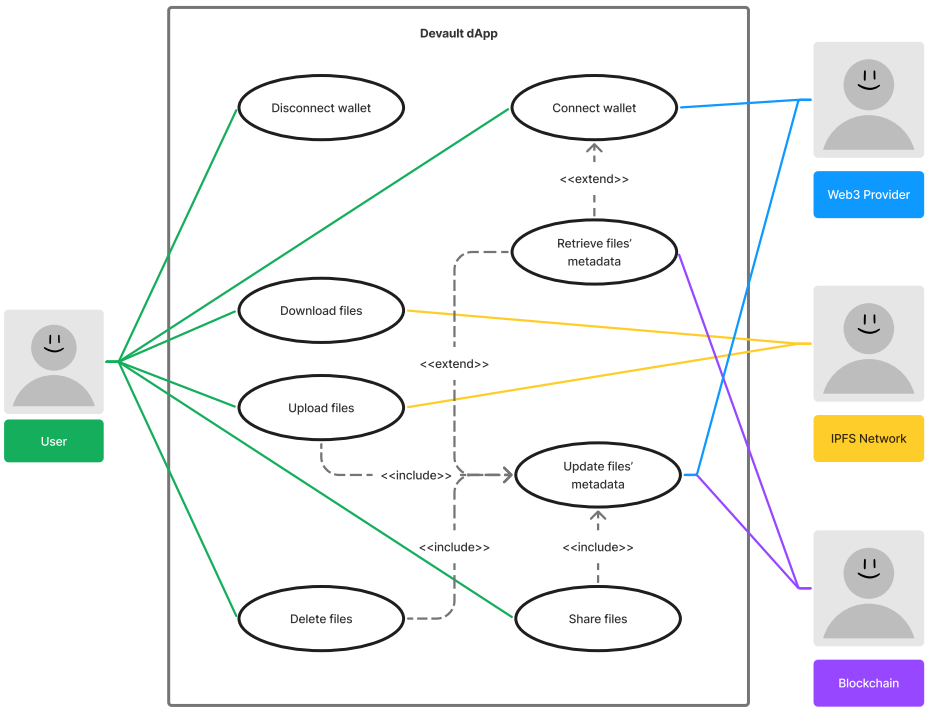


Figure 2.7: dApp use case diagram.

Use Case Model

Table 2.1: Use case 1: Connecting wallet

ID	UC_1
Title	**Connecting wallet**
Description	The user can connect with their Ethereum wallet to use the system.

Primary Actor	User.
Pre-Conditions	• The user must have internet connection. • The user navigates to `devault.vercel.app` or any other instance.
Main Success Scenario	1. The user clicks the connect wallet button. 2. The user confirms the connection. 3. The system then will retrieve the files from that account.

Table 2.2: Use case 2: Uploading files

ID	UC_2
Title	**Uploading files**
Description	The user can upload files or folders to the system.
Primary Actor	User.
Pre-Conditions	• UC_1
Main Success Scenario	1. The user navigates to the vault tab. 2. The user clicks on the upload button and picks a file or folder to upload. 3. The user enters a password to encrypt the files. 4. The system then will encrypt the files, store their metadata in the blockchain, and upload the encrypted files to the peer-to-peer network.

Table 2.3: Use case 3: Downloading files

ID	UC_3
Title	**Downloading files**
Description	The user can download files or folders from the system.
Primary Actor	User.
Pre-Conditions	• UC_1

Main Success Scenario	1. The user navigates to the vault tab. 2. The user selects the files they need to download. 3. The user enter a password to decrypt the files. 4. The system then will decrypt the files and download them.

Table 2.4: Use case 4: Sharing files

ID	UC_4
Title	**Sharing files**
Description	The user can share files or folders with other users.
Primary Actor	User.
Pre-Conditions	• UC_1
Main Success Scenario	1. The user navigates to the vault tab. 2. The user selects the files they need to share. 3. The user clicks the share button. 4. The user will be prompted to enter the addresses to share the file. 5. The system then will share the files with these addresses.

Table 2.5: Use case 5: Deleting files

ID	UC_5
Title	**Deleting files**
Description	The user can Delete files or folders form the system.
Primary Actor	User.
Pre-Conditions	• UC_1
Main Success Scenario	1. The user navigates to the vault tab. 2. The user selects the files they need to delete. 3. The user clicks the delete button. 4. The user will be prompted to confirm the deletion process. 5. The system then will delete the files with form the user address.

Table 2.6: Use case 6: Disconnecting wallet

ID	UC_6
Title	**Disconnecting wallet**
Description	The user can disconnect their Ethereum wallet from the system.
Primary Actor	User.
Pre-Conditions	• UC_1
Main Success Scenario	1. The user clicks the disconnect wallet button. 2. The system will log this user out.

2.7.3 Class Diagram

A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects. The following diagrams will show the class diagrams for the smart contracts.

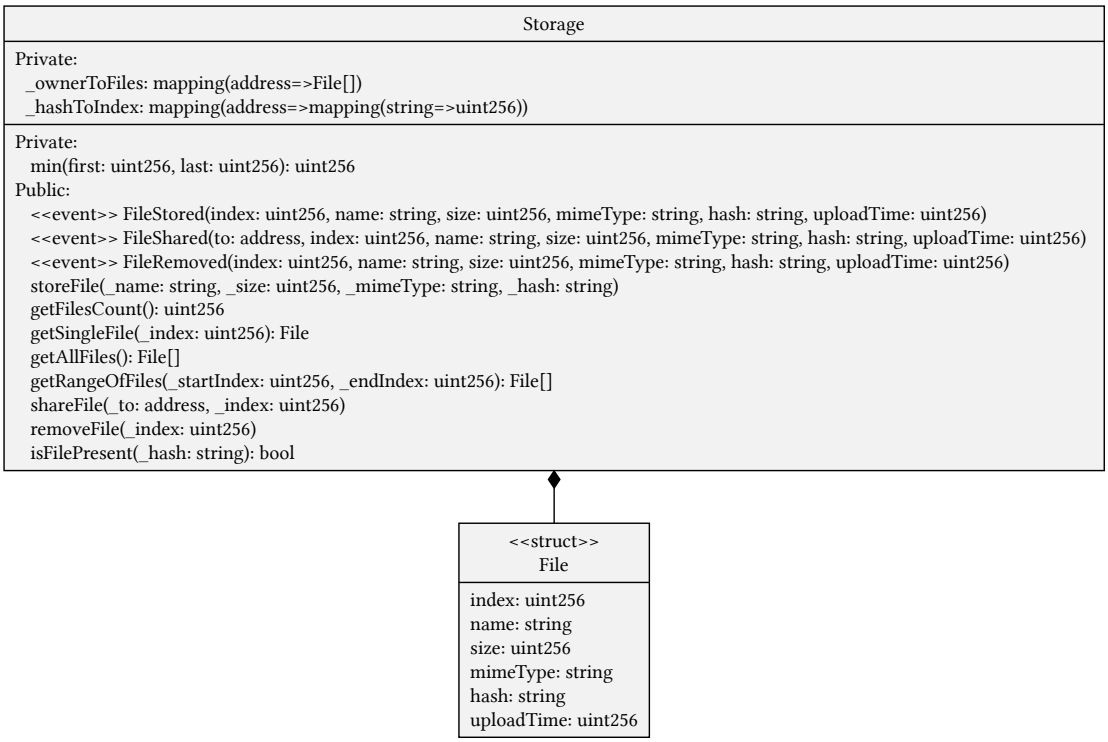


Figure 2.8: The smart contracts class diagram.

2.7.4 System Context Diagram

System context diagram represent all external entities that may interact with a system. Such a diagram pictures the system at the center, with no details of its interior structure, surrounded by all its interacting systems, environments and activities. The objective of the system context diagram is to focus attention on external factors and events that should be considered in developing a complete set of systems requirements and constraints.

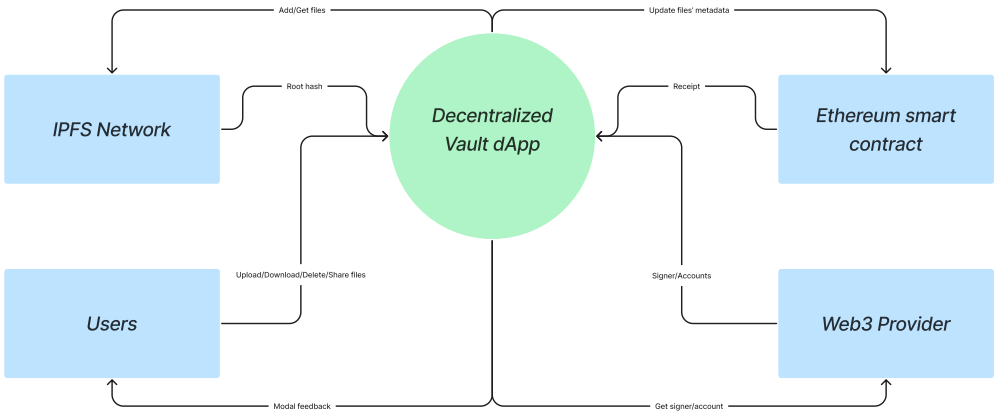


Figure 2.9: All external entities that may interact with a system.

2.7.5 Sequence Diagram

A sequence diagram or system sequence diagram (SSD) shows process interactions arranged in time sequence in the field of software engineering. It depicts the processes involved and the sequence of messages exchanged between the processes needed to carry out the functionality.

The following diagram shows the interaction between the user, dApp, blockchain, web3 provider, and IPFS network.

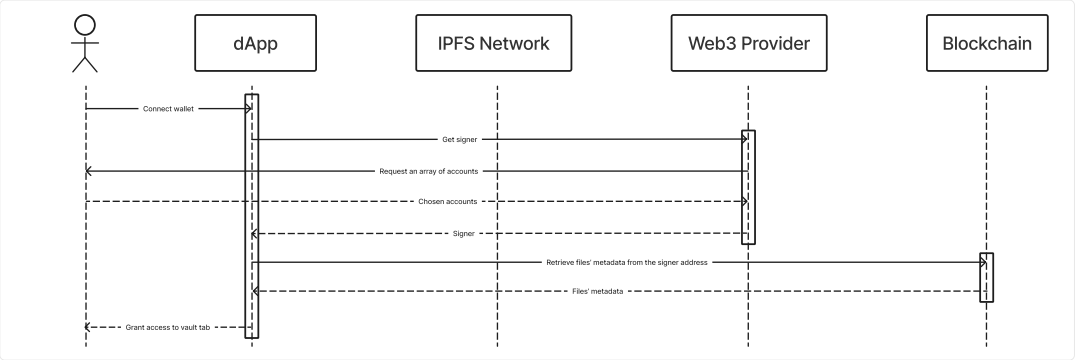


Figure 2.10: Connect wallet act in sequential order.

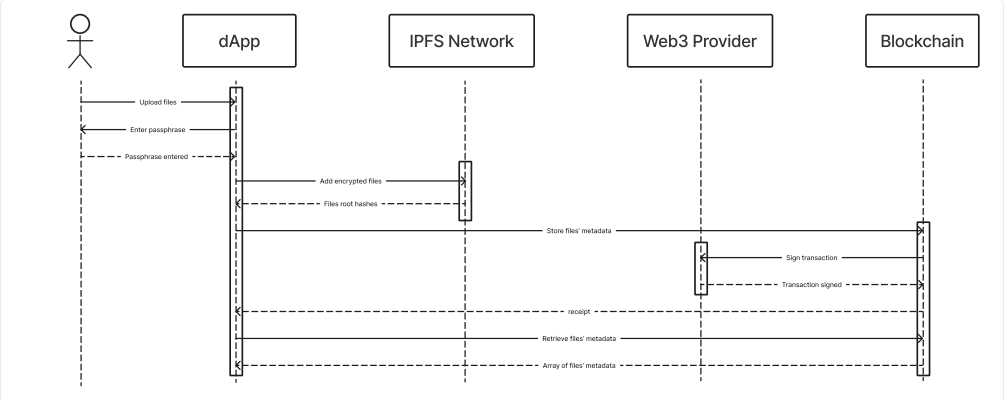


Figure 2.11: Upload files act in sequential order.

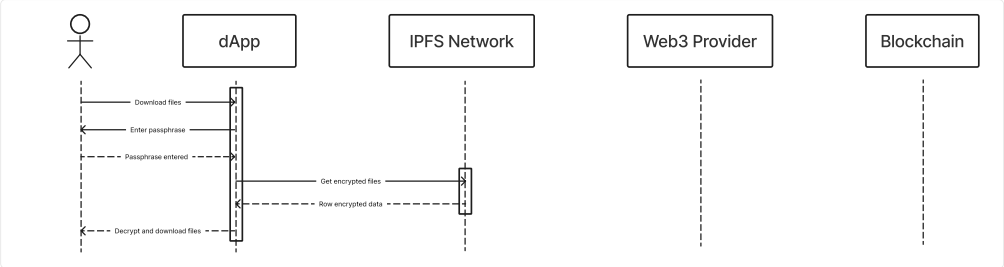


Figure 2.12: Download files act in sequential order.

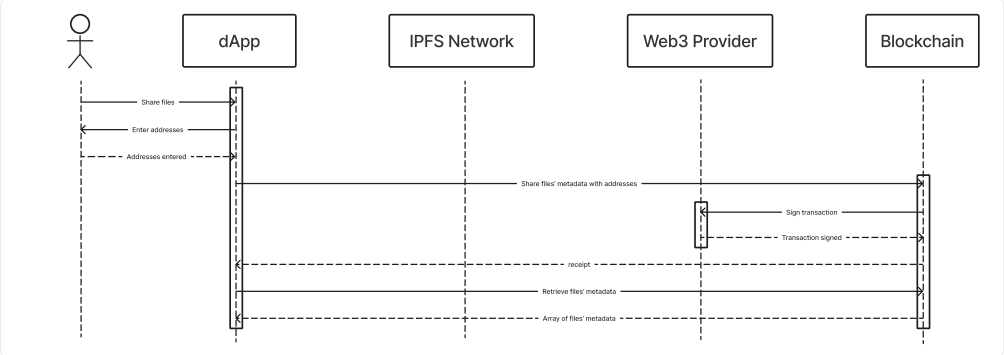


Figure 2.13: Share files act in sequential order.

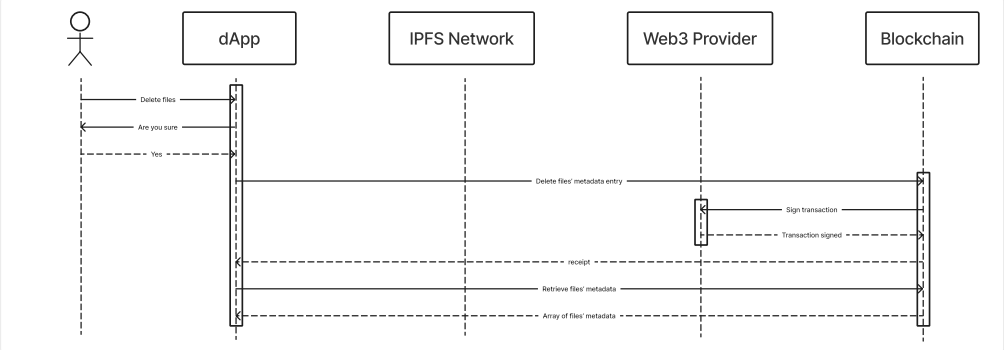


Figure 2.14: Delete files act in sequential order.

Chapter 3

System Design

3.1 Introduction

In this chapter, we will give an overview of the system and the software architecture. How each component in the system interacts with each other. Also, a brief explanation of the algorithms and data structure used and the tools and technologies involved. And implementation and testing details.

3.2 System Architecture

Here is the conceptual model that defines the structure, behavior, and more views of a system.

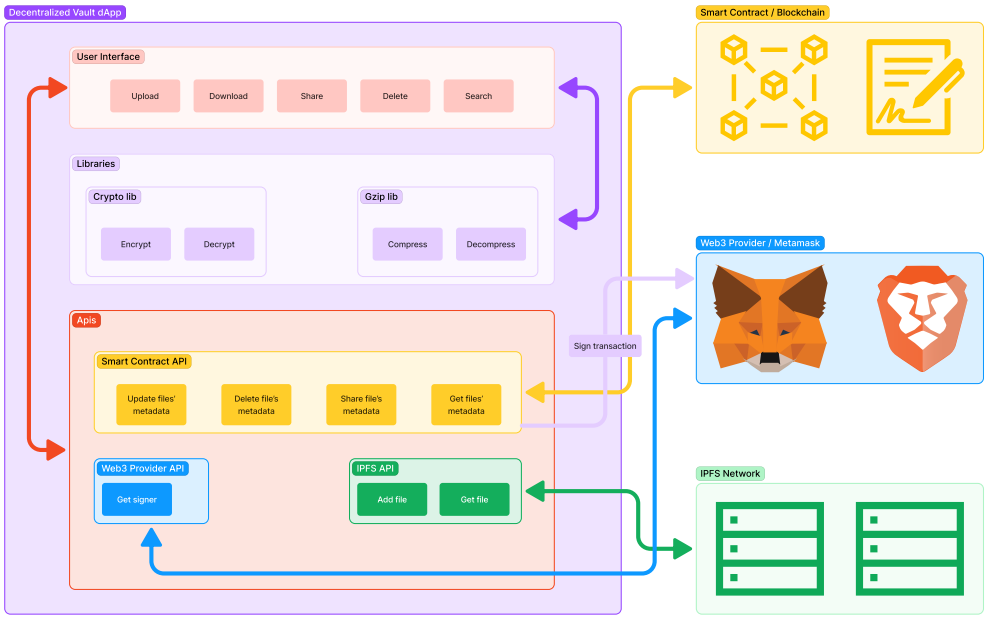


Figure 3.1: Decentralized vault dApp architecture.

3.3 Tools and Technologies

Here are the tools and technologies involved in this project.

JavaScript: JavaScript, often abbreviated JS, is a programming language that is one of the core technologies of the World Wide Web, alongside HTML and CSS. As of 2022, 98% of websites use JavaScript on the client side for web page behavior, often incorporating third-party libraries. All major web browsers have a dedicated JavaScript engine to execute the code on users' devices.

Next.js: Next.js is an open-source web development framework built on top of Node.js enabling React-based web applications functionalities such as server-side rendering and generating static websites. React documentation mentions Next.js among "Recommended Toolchains" advising it to developers as a solution when "Building a server-rendered website with Node.js". Where traditional React apps can only render their content in the client-side browser, Next.js extends this functionality to include applications rendered on the server-side.

Solidity: Solidity is an object-oriented programming language for implementing smart contracts on various blockchain platforms, most notably, Ethereum. It was developed by Christian Reitwiessner, Alex Beregszaszi, and several former Ethereum core contributors. Programs in Solidity run on Ethereum Virtual Machine.

Ethers.js: The ethers.js library aims to be a complete and compact library for interacting with the Ethereum Blockchain and its ecosystem. It was originally designed for use with ethers.io and has since expanded into a more general-purpose library.

Hardhat: Hardhat is an Ethereum development environment. Compile your contracts and run them on a development network. Get Solidity stack traces, console.log and more.

IPFS: The InterPlanetary File System (IPFS) is a protocol and peer-to-peer network for storing and sharing data in a distributed file system. IPFS uses content-addressing to uniquely identify each file in a global name-space connecting all computing devices.

Docker: Docker is a set of platform as a service (PaaS) products that use OS-level virtualization to deliver software in packages called containers. We use docker for shipping and self-hosting the dApp.

Ropsten: Ropsten Ethereum (also known as "Ethereum Testnet") is an Ethereum test network that allows for blockchain development testing before deployment on Mainnet, the main Ethereum network. Testnet ethers are separate and distinct from actual ethers, and are never supposed to have any value. This allows application developers or Ethereum testers to experiment, without having to use real ethers or worrying about breaking the main Ethereum chain.

3.4 Implementation and Testing

Here you will find the implementation details.

3.4.1 Smart Contracts

States

Persistent data is referred to as storage and is represented by state variables. These values get stored permanently on the blockchain. You need to declare the type so that the contract can keep track of how much storage on the blockchain it needs when it compiles.

```
/**
 * @dev The file's metadata.
 */
mapping(address => File[]) private _ownerToFiles;
mapping(address => mapping(string => uint256)) private _hashToIndex;
```

Store File

Whenever you call storeFile with its parameters, it pushes a new file to the array of files of the account that makes that call.

```
/**
 * @dev Creates a new file.
 * @param _name Name of the file.
 * @param _size Size of the file.
 * @param _mimeType Mime type of the file.
 * @param _hash Hash of the file.
 */
function storeFile(
                    string memory _name,
                    uint256 _size,
                    string memory _mimeType,
                    string memory _hash
                    ) public {
    require(bytes(_name).length > 0, "Name cannot be empty");
    require(
            bytes(_name).length <= 512,
            "Name cannot be longer than 512 bytes"
            );

    require(_size > 0, "Size must be greater than 0");
    require(_size <= 2**30, "Size must be less than 2**30");

    require(bytes(_mimeType).length > 0, "Mime type cannot be empty");
    require(
            bytes(_mimeType).length <= 512,
            "Mime type cannot be longer than 512 bytes"
            );

    require(bytes(_hash).length > 0, "Hash cannot be empty");

    File memory newFile = File(
                               _ownerToFiles[msg.sender].length + 1,
                               _name,
                               _size,
                               _mimeType,
                               _hash,
                               block.timestamp
                               );

    // check if the file's hash is already present in the hashes array
    if (_hashToIndex[msg.sender][_hash] == 0) {
        _ownerToFiles[msg.sender].push(newFile);
        _hashToIndex[msg.sender][_hash] = _ownerToFiles[msg.sender].length;

        emit FileStored(
                        newFile.index,
```

```
                        newFile.name,
                        newFile.size,
                        newFile.mimeType,
                        newFile.hash,
                        newFile.uploadTime
                        );
    }
}
```

Retrieve File

getFiles are a group of functions that retrieve the files' metadata.

```
/**
 * @dev Count the number of files owned by the caller.
 */
function getFilesCount() public view returns (uint256) {
    return _ownerToFiles[msg.sender].length;
}

/**
 * @dev Returns the file at the given index.
 * @param _index The index of the file to return. -- 1-based indexing
 */
function getSingleFile(uint256 _index) public view returns (File memory) {
    require(
        _index <= _ownerToFiles[msg.sender].length,
        "Index out of bounds"
    );
    require(_index > 0, "Index must be greater than 0");
    return _ownerToFiles[msg.sender][_index - 1];
}

/**
 * @dev Returns an array of files owned by the caller.
 */
function getAllFiles() public view returns (File[] memory) {
    require(_ownerToFiles[msg.sender].length > 0, "No files found!");
    return _ownerToFiles[msg.sender];
}

function min(uint256 first, uint256 last) private pure returns (uint256) {
    if (first < last) {
        return first;
    } else {
        return last;
    }
}

/**
 * @dev Share a file with another user.
 * @param _startIndex the start index of the range of files to return. -- 1-based indexing
 * @param _endIndex the end index of the range of files to return. -- 1-based indexing
 */
function getRangeOfFiles(uint256 _startIndex, uint256 _endIndex)
    public
    view
    returns (File[] memory)
{
    require(_ownerToFiles[msg.sender].length > 0, "No files found!");
    require(
        _startIndex <= _ownerToFiles[msg.sender].length,
        "Index out of bounds"
    );
    require(_startIndex > 0, "Index out of bounds");
    require(
        _endIndex > _startIndex,
```

```
        "start index must be strictly smaller than the end index"
    );

    _endIndex = min(_endIndex, _ownerToFiles[msg.sender].length + 1);
    uint256 range = _endIndex - _startIndex;
    _startIndex = _startIndex - 1;
    _endIndex = _endIndex - 1;
    File[] memory result = new File[](range);
    for (uint256 i = _startIndex; i < _endIndex; ++i) {
        result[i - _startIndex] = _ownerToFiles[msg.sender][i];
    }
    return result;
}
```

Share File

Whenever you call shareFile with the address parameter, it pushes a new file to the array of files of that address.

```
/**
 * @dev Share a file with another user.
 * @param _to The address of the user to share the file with.
 * @param _index The index of the file to be shared. -- 1-based indexing
 */
function shareFile(address _to, uint256 _index) public {
    require(
        _index <= _ownerToFiles[msg.sender].length,
        "Index out of bounds!"
    );
    require(_index > 0, "Index must be greater than 0!");

    require(_to != msg.sender, "To yourself!");
    require(_to != address(0), "Null address!");

    File memory file = _ownerToFiles[msg.sender][_index - 1];

    // check if the file's hash is already present in the hashes array
    if (_hashToIndex[_to][file.hash] == 0) {
        _ownerToFiles[_to].push(file);
        _hashToIndex[_to][file.hash] = _ownerToFiles[_to].length;

        emit FileShared(
            _to,
            file.index,
            file.name,
            file.size,
            file.mimeType,
            file.hash,
            file.uploadTime
        );
    }
}
```

Delete File

Whenever you call deleteFile with the index parameter, it removes the file with that index from the array of files of the account that makes that call.

```
/**
 * @dev Removes a file from the storage.
 * @param _index The index of the file to be removed. -- 1-based indexing
 */
function removeFile(uint256 _index) public {
    require(
        _index <= _ownerToFiles[msg.sender].length,
        "Index out of bounds!"
```

```
    );
    require(_index > 0, "Index must be greater than 0!");

    File memory file = _ownerToFiles[msg.sender][_index - 1];

    _ownerToFiles[msg.sender][_index - 1] = _ownerToFiles[msg.sender][
        _ownerToFiles[msg.sender].length - 1
    ];
    _ownerToFiles[msg.sender][_index - 1].index = _index;

    _hashToIndex[msg.sender][
        _ownerToFiles[msg.sender][_index - 1].hash
    ] = _hashToIndex[msg.sender][file.hash];
    _hashToIndex[msg.sender][file.hash] = 0;

    _ownerToFiles[msg.sender].pop();

    emit FileRemoved(
        file.index,
        file.name,
        file.size,
        file.mimeType,
        file.hash,
        file.uploadTime
    );
}
```

3.4.2 Front End

Initialize

The initialize function is executed whenever the user clicks the connect wallet button.

```
const getLibrary = () => {
  // A Web3Provider wraps a standard Web3 provider, which is
  // what MetaMask injects as window.ethereum into each page
  if (typeof window.ethereum !== 'undefined') {
    // The "any" network will allow spontaneous network changes
    const provider = new ethers.providers.Web3Provider(window.ethereum, 'any')

    provider.on('network', (newNetwork, oldNetwork) => {
      // When a Provider makes its initial connection, it emits a "network"
      // event with a null oldNetwork along with the newNetwork. So, if the
      // oldNetwork exists, it represents a changing network
      if (oldNetwork) {
        window.location.reload()
      }
    })
    return provider
  } else if (typeof window.web3 !== 'undefined') {
    const provider = new ethers.providers.Web3Provider(window.web3.
    currentProvider, 'any')
    provider.on('network', (newNetwork, oldNetwork) => {
      // When a Provider makes its initial connection, it emits a "network"
      // event with a null oldNetwork along with the newNetwork. So, if the
      // oldNetwork exists, it represents a changing network
      if (oldNetwork) {
        window.location.reload()
      }
    })
    return provider
  } else {
    return null
  }
}

```

```
const Initialize = async () => {
  // The MetaMask plugin also allows signing transactions to
  // send ether and pay to change state within the blockchain.
  // For this, you need the account signer..
  const provider = getLibrary()
  await provider.send('eth_requestAccounts', [])

  const _signer = provider.getSigner()
  const _account = await _signer.getAddress()
  const _balance = await provider.getBalance(_account)
    .then(_balance => ethers.utils.formatEther(_balance))

  setSigner(prevState => _signer)
  setAccount(prevState => _account)
  setBalance(prevState => _balance)

  const _contract = new ethers.Contract(contractAddress, Storage.abi, provider
    ).connect(_signer)
  setContract(prevState => _contract)

  const _chainId = await provider.getNetwork()
  setChainId(prevState => _chainId.chainId)

  const _blockNumber = await provider.getBlockNumber()
  setBlockNumber(prevState => _blockNumber)
}
```

UploadFile

When you press the upload button, this is what happens:

- The file gets encrypted using AES-256-CBC mode.
- The file is split into small chunks and uploaded to the IPFS node.
- The file's metadata is stored in the smart contract.

```
const uploadFiles = async () => {
  try {
    console.log('Encrypting & Uploading file to IPFS...')
    setIsUploading(prevState => TRUE)

    const _options = { from: account, gasLimit: 3000000 }
    const ipfs = getIpfs()
    // fileBuffer = ArrayBuffer
    const encryptedFileBufferWordArray = encryptAES256(fileBuffer, passphrase)

    const response = await ipfs.add(Buffer.from(encryptedFileBufferWordArray))

    console.log('response:', response.path)
    const isFilePresent = await contract.isFilePresent(response.path, _options
    )
    console.log(isFilePresent)
    if (isFilePresent) {
      console.log('File already exists')
      setIsUploading(prevState => FALSE)
      return
    }

    setSize(prevState => response.size)
    setHash(prevState => response.path)

    setIsUploading(prevState => FALSE)
    setIsReadyForTransaction(prevState => TRUE)
  } catch (error) {
    console.log('Cannot upload file to IPFS:', error.message)
    // TODO: show error message
```

```
30      setIsUploading(prevState => UNSET)
31      setIsReadyForTransaction(prevState => UNSET)
32    }
33  }
```

Chapter 4

Results and Discussion

4.1 Results

We tested the Devault with uploading, downloading, sharing, deleting, connecting, and disconnecting functionality on different browsers and devices. Below we show the results of three of these tests.

1. ***Brave browser, the desktop version and metamask***

   Figure 4.1 shows the result of connecting the wallet.

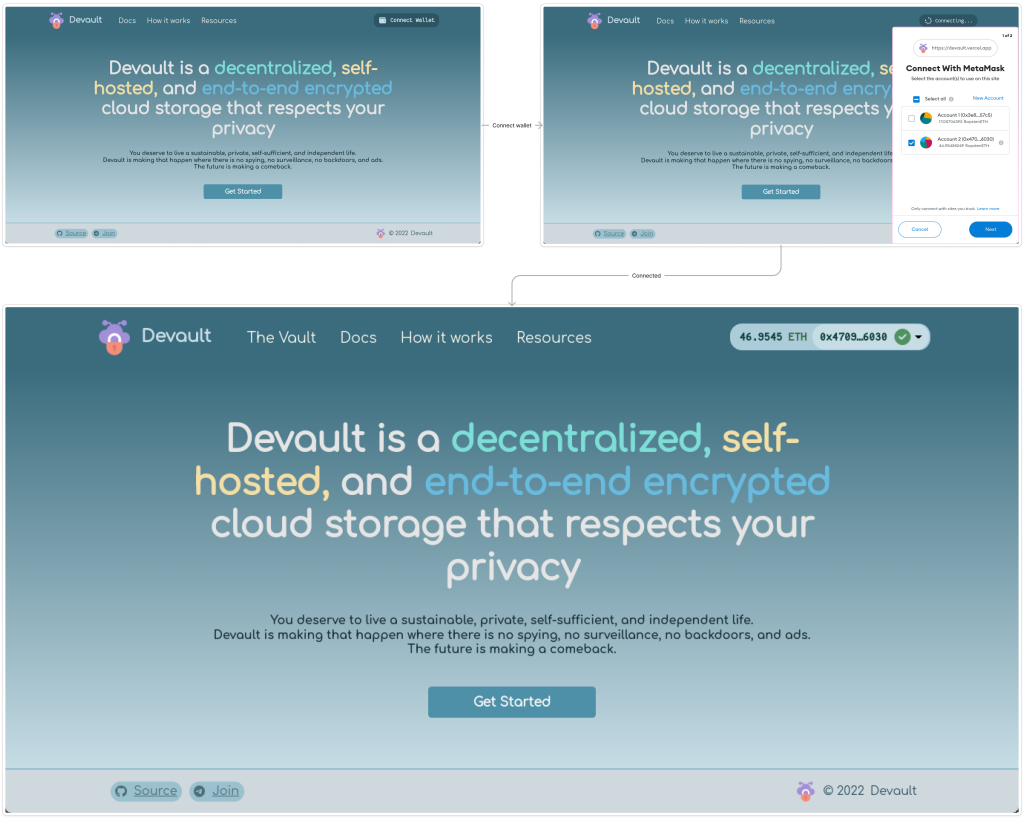


Figure 4.1: The user experience of connecting wallet.

Figure 4.2 shows the result of uploading files functionality.

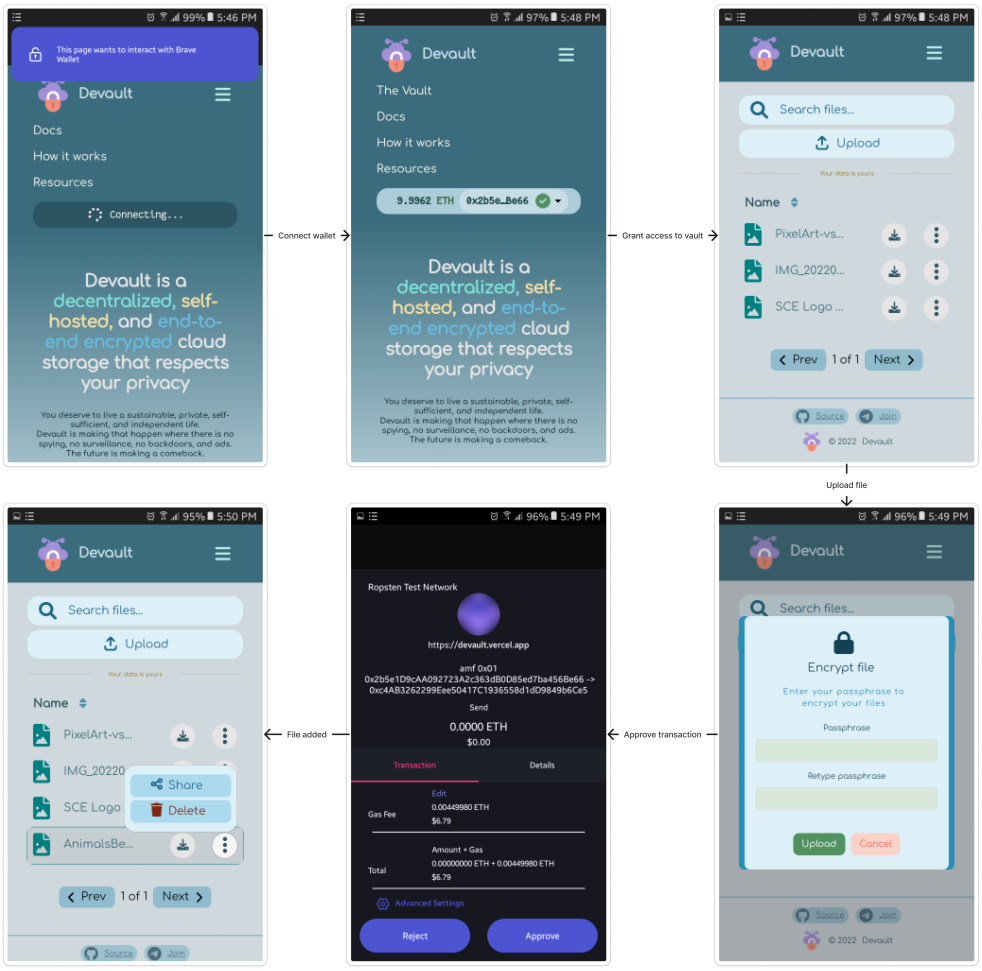


Figure 4.2: The user experience of uploading files.

3. *Firefox browser, the desktop version and metamask*

Figure 4.3 shows the result of sharing files functionality.

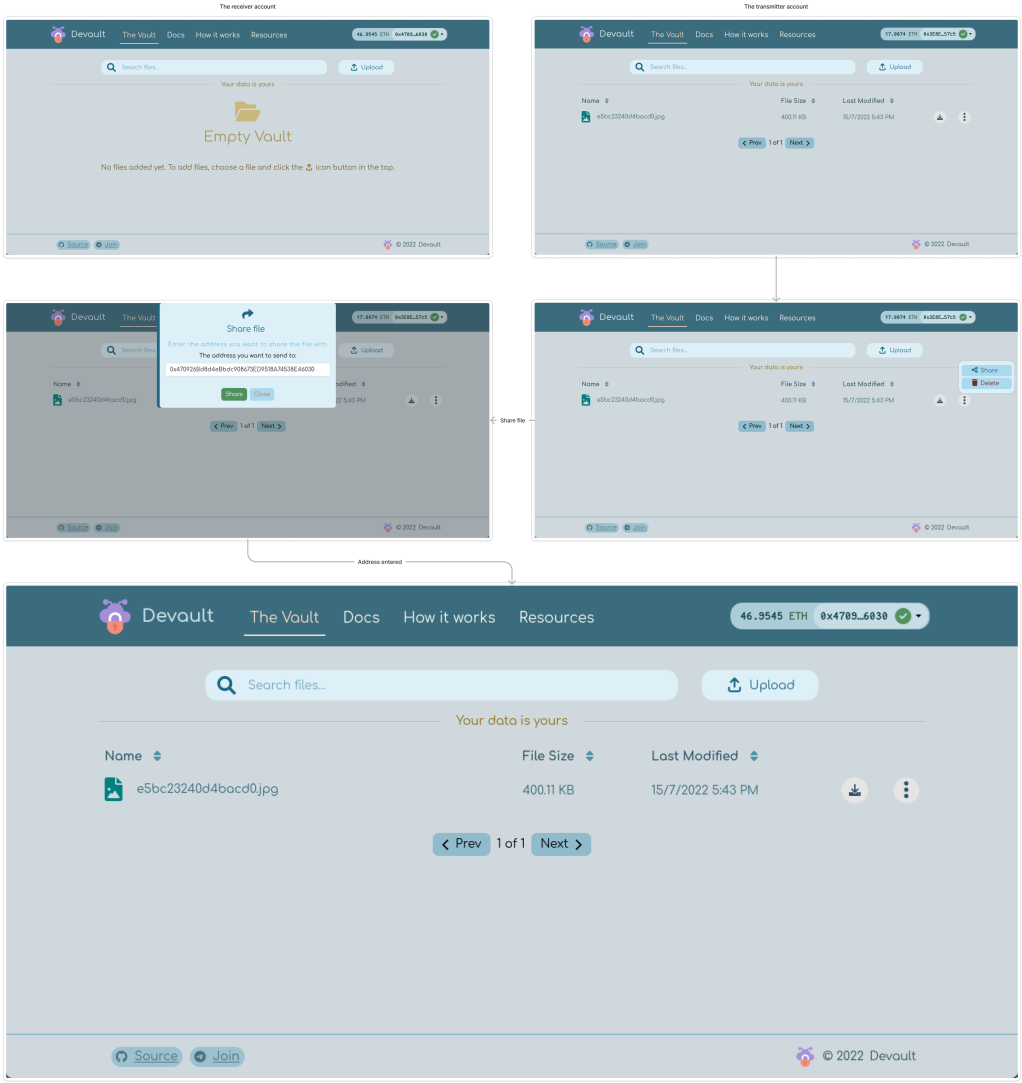


Figure 4.3: The user experience of sharing files.

4.1.1 Performance

For performance testing we used GTmetrix tool.

GTmetrix is one of the most popular tools for analyzing site speed performance. If you put a website to the test, it will provide a performance score and a report which shows the current state of the site along with some suggestions on what can be improved. Figure 4.4 shows performance of devault has reached the highest grade.

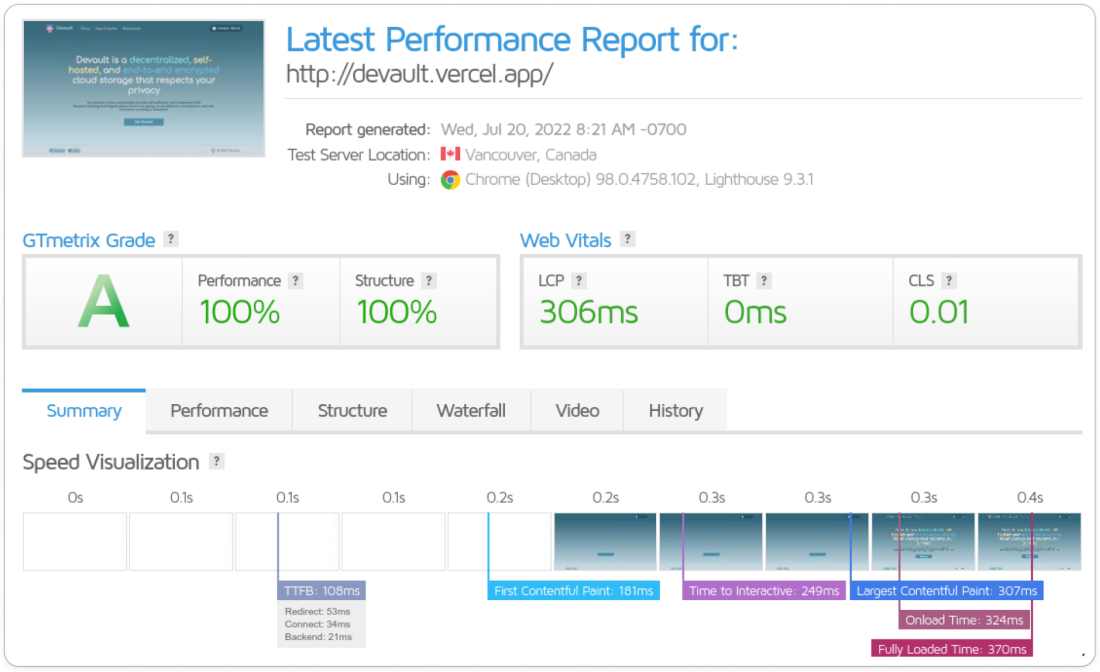


Figure 4.4: The performance report of devault generated by GTMetrix.

Table 4.1: The performance report details

Performance Metrics	The measure
Total Page Size	450KB.
Total Page Requests	31.
Fully Loaded Time	370ms.
Largest Content Element	306ms

4.2 Discussion

So far, we can upload files, download, share, delete, and search. The interface is pretty simple. If you are connected, it will show the account address and balance in the navbar. And you have been granted access to the vault tab. If not, you won't be able to access the vault tab. If we have time and resources, it would be great to deploy the smart contract to the mainnet to be ready for production. Also, we could replace the public IPFS gateway with Filecoin. As for Filecoin, the availability, reliability, true decentralization, privacy, and security are far more advanced than IPFS.

4.2.1 Limitation

The limitations of our system are listed below:

- The user should have cryptocurrency to make transactions and to reward the entity for renting their hard disk, which is a constraint.
- The user should install specific software such as metamask or brave wallet to use our system.
- Cryptocurrencies are illegal in some countries, whereas our system needs those cryptocurrencies to work.
- The uploading of a file may take a while because of the encryption and block mining, which is not practical.

Chapter 5
Conclusion and Future Work

5.1 Conclusion

Data becomes one of the most valuable assets to everyone. Storing data in a secure and decentralized way nowadays is a tough challenge. The big tech companies that provide the platform for storing the data have the right to edit, delete, modify, view, and analyze the content. Not to mention the data hacking, data loss, and the single point of failure, this is not a privacy-friendly platform.

Devault, unlike traditional cloud storage, has client-side encryption/decryption. Which prevents data censorship and man-in-the-middle attack. The decentralization nature of the blockchain and peer-to-peer networks make the devault even more secure and fast.

Remember, your data is yours. Do not let anyone take that from you.

5.2 Future Work

1. Support Arabic language.
2. Use Shamir's Secret Sharing algorithms for sharing files.
3. Add the search functionality.
4. Compress files before uploading.
5. Enhance the ui/ux.
6. Manipulate selected files and folders.
7. Deploy to the mainnet.
8. Use Filecoin instead of IPFS.
9. Use the advantages of private and public keys in encryption/decryption.
10. Give feedback if the key used for decryption is not the same key used for encryption.

Appendix

Glossary

Acronyms

ABI	Application Binary Interface
ACL	Access Control List
AES	Advanced Encryption Standard
AIC	Al-Azhar ICPC Community
ASIC	Application-Specific Integrated Circuit
CBC	Cipher block chaining
CID	Content Identifier
DAG	Directed Acyclic Graph
DAPP	Decentralized Application
DHT	Distributed Hash Table
ETH	Ether
EVM	Ethereum Virtual Machine
HTTP	HyperText Transfer Protocol
IP	Internet Protocol
IPFS	InterPlanetary File System
nft	Non-Fungible Token
P2P	Peer-to-Peer
PoA	Proof of Authority
PoS	Proof of Stake
PoW	Proof of Work
SC	Smart Contract
TX	Transaction

Terminology

51% attack	When more than 50% of the miners in a blockchain launch an attack on the rest of the nodes/users to attempt to steal assets or double spend.

ACL	In computer security, an access-control list (ACL) is a list of permissions associated with a system resource, also known as an object. An ACL specifies which users or system processes are granted access to objects, as well as what operations are allowed on given objects.
bandwidth	bandwidth is a measurement indicating the maximum capacity of a wired or wireless communications link to transmit data over a network connection in a given amount of time. Typically, bandwidth is represented in the number of bits, kilobits, megabits or gigabits that can be transmitted in 1 second. Synonymous with capacity, bandwidth describes data transfer rate.
Bitcoin	A cryptocurrency that uses a blockchain network to regulate the generation of coins/tokens and transfer of funds. Bitcoin is the most widely used cryptocurrency and is the most widely traded currency in the world.
BitTorrent	BitTorrent is a communication protocol for peer-to-peer file sharing, which is used to distribute data and electronic files over the Internet.
block	A block is a set of transactions that are recorded in a blockchain network.
Blockchain	A blockchain is a distributed database that is shared among the nodes of a computer network. As a database, a blockchain stores information electronically in digital format.
bytecode	Bytecode is the compiled code of a smart contract.
centralized	A system or process for which there is a singular (i.e., central) source of authority, control and/or truth.
chain	A chain is a sequence of blocks that are linked together by a hash of the previous block.
CID	A Content Identifier (CID) is a self-describing content-addressed label used to point to the data stored in IPFS.
consensus	The process used by a group of peers, or nodes, on a blockchain network to agree on the validity of transactions submitted to the network. Dominant consensus mechanisms are Proof of Work (PoW) and Proof of Stake (PoS).
crypto keys	A public key is a unique string of characters derived from a private key which is used to encrypt a message or data. The private key is used to decrypt the message or data.
cryptocurrency	Digital money which uses encryption and consensus algorithms to regulate the generation of coins/tokens and transfer of funds. Cryptocurrencies are generally decentralized, operating independently of central authorities.

cryptography	The science of securing communication using individualized codes so only the participating parties can read the messages.
Daemon	A Daemon is a computer program that typically runs in the background. The IPFS daemon is how you take your node online to the IPFS network.
dApp	Software which does not rely on a central system or database but can share information amongst its users via a decentralized database, such as a blockchain.
decentralized	A system with no single point where the decision is made. Every node makes a decision for its own behavior and the resulting system behavior is the aggregate response.
DHT	A Distributed Hash Table (DHT) is a distributed key-value store where keys are cryptographic hashes. In IPFS, each peer is responsible for a subset of the IPFS DHT.
digital signature	A mathematical scheme for verifying digital messages or documents satisfy two requirements - they have authenticity and integrity.
Ethereum	A public blockchain that supports smart contracts.
gas	A fee charged to write a transaction to a public blockchain. The gas is used to reward the miner which validates the transaction.
genesis	The first block in a blockchain network.
hash	A cryptographic hash function is a function that takes a message as input and produces a fixed-length output called a hash.
HTTP	HTTP is a protocol for fetching resources such as HTML documents. It is the foundation of any data exchange on the Web and it is a client-server protocol, which means requests are initiated by the recipient, usually the Web browser. A complete document is reconstructed from the different sub-documents fetched, for instance, text, layout description, images, videos, scripts, and more.
immutable	The property of being unchangeable. Once a transaction has been added to a block and written to a blockchain, it cannot be changed and therefore is immutable.
IPFS	A peer-to-peer hypermedia protocol for the Internet. It is used to store and retrieve information in a decentralized way.
IPFS Gateway	An IPFS Gateway acts as a bridge between traditional web browsers and IPFS. Through the gateway, users can browse files and websites stored in IPFS as if they were stored on a traditional web server.

mainnet	The production version of a blockchain.
Merkle Tree	A Merkle Tree is a specific type of hash tree used in cryptography and computer science, allowing efficient and secure verification of the contents of large data structures. Named after Ralph Merkle, who patented it in 1979.
mining	In a public blockchain, the process of verifying a transaction and writing it to the blockchain for which the successful miner is rewarded in the cryptocurrency of the blockchain.
node	A computer which holds a copy of the blockchain ledger.
nonce	A nonce is an abbreviation for "number only used once," which, in the context of cryptocurrency mining, is a number added to a hashedor encryptedblock in a blockchain that, when rehashed, meets the difficulty level restrictions. The nonce is the number that blockchain miners are solving for. When the solution is found, the blockchain miners are offered cryptocurrency in exchange.
off-chain	Data stored external to the blockchain.
on-chain	Data stored within the blockchain.
open-source	Software products that include permission to use, enhance, reuse or modify the source code, design documents, or content of the product.
peer-to-peer	A direct connection between two participants in a system.
pos	Proof-of-stake (PoS) protocols are a class of consensus mechanisms for blockchains that work by selecting validators in proportion to their quantity of holdings in the associated cryptocurrency. This is done to avoid the computational cost of proof-of-work schemes.
pow	Proof of work (PoW) is a form of cryptographic proof in which one party (the prover) proves to others (the verifiers) that a certain amount of a specific computational effort has been expended.
Satoshi Nakamoto	The name used by the person or entity who developed bitcoin, authored the bitcoin white paper, and created and deployed bitcoin's original reference implementation. As part of the implementation, Nakamoto also devised the first blockchain database.
seed phrase	A random sequence of words which can be used to restore a lost wallet.
smart contract	Self-executing computer code deployed on a blockchain to perform a function, often, but not always, the exchange of value between a buyer and a seller.

solidity	Solidity is a programming language for smart contracts.
testnet	A staging blockchain environment for testing application before being put into production (or onto the mainnet).
transaction	A transaction is a set of instructions that are sent to a blockchain network to be processed by the network.
trustless	The elimination of trust from a transaction.
wallet	A digital file that holds coins and tokens held by the owner. The wallet also has a blockchain address to which transactions can be sent.
Web 2.0	Web 2.0 is the World Wide Web based on the concepts of social media, where the user can create content, post it online, and engage with other user-generated content.
Web 3.0	Web 3.0 is an idea for a new iteration of the World Wide Web which incorporates concepts such as decentralization, blockchain technologies, and token-based economics.

List of Figures

List of Tables

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," www.bitcoin.org, techreport, 2009.

[2] P. Labs, "Filecoin: A decentralized storage network," https://filecoin.io, techreport, 2017.

[3] T. Laurence, *Blockchain For Dummies*, 1 **edition**. 111 River Street, Hoboken, NJ: John Wiley & Sons, Inc, 2017.

[4] N. Prusty, *Building Blockchain Projects*, 1 **edition**. B3 2PB, UK: Packt Publishing Ltd, 2017.

[5] D. Yaga, P. Mell, N. Roby **and** K. Scarfone, "Blockchain technology overview," National Institute of Standards **and** Technology, Gaithersburg, MD, techreport, 2018.

[6] S. Sarker, A. K. Saha **and** M. S. Ferdous, "A survey on blockchain & cloud integration," **in***International Conference on Computer and Information Technology* Sylhet, Bangladesh, 2020.

[7] P. Sharma, R. Jindal **and** M. D. Borah, "Blockchain-based decentralized architecture for cloud storage system," *Journal of Information Security and Applications*, **jourvol** 62, **number** 8, **page** 102 970, 2021.

[8] T. E. Community, *Ethereum development documentation*, Available at `https://ethereum.org/en/developers/docs/` (2022/07/20).

[9] Mozilla.org, *Web technology for developers*, Available at `https://developer.mozilla.org/en-US/docs/Web` (2022/07/20).

[10] G. Team, *Test the performance of webpages*, Available at `https://gtmetrix.com/` (2022/07/20).

[11] P. L. Team, *Ipfs documentation*, Available at `https://docs.ipfs.io/` (2022/07/20).