Florida International University
Knight Foundation School of Computing and Information Sciences

Software Engineer Focus

P2P Final Documentation

Project Title: P2P File System

Team Members:
Dominick Diaz, Bryan Alvarenga, Hamzah Masri, Anthony Delgado, Sonali Benni

Product Owner(s):
Dominick Diaz

Mentor(s):
Dominick Diaz

Instructor:
Masoud Sadjadi

Abstract

This university project aims to create a decentralized file system that utilizes the InterPlanetary File System (IPFS) network and Firebase to provide users with their own directories on the IPFS network. The project consists of a front-end application built in TypeScript, a back-end API developed in Golang, and an IPFS node running on the server. Upon registering with Firebase, users receive their own directory on the IPFS network through MFS, enabling them to store and access their files securely. Files are permanently stored on the server through pinning. The result is a robust and decentralized file system that offers users greater control over their data while leveraging cutting-edge technology.

INTRODUCTION

The digital era is marked by an exponential growth of data, which has intensified the need for reliable and efficient file storage and retrieval systems. Traditional file systems, despite being widely adopted, have inherent limitations and challenges that can compromise their effectiveness. This capstone project aims to create a cutting-edge file system built upon the InterPlanetary File System (IPFS) network, utilizing the Mutable File System (MFS) and

incorporating Firebase to overcome the limitations of traditional file systems. This innovative file storage and retrieval solution will not only address the shortcomings of existing systems but also offer users a secure, distributed, and resilient alternative.

Current System

Traditional file systems are built upon a centralized architecture that relies on servers to store and retrieve data. While these systems are prevalent, they suffer from several inherent drawbacks:

1. Data integrity: Centralized systems are vulnerable to data corruption, tampering, and unauthorized access. Inadequate security measures can compromise the integrity of stored data, leading to issues related to data privacy and regulatory compliance.

2. Single points of failure: Centralized systems are susceptible to server outages, hardware failures, and network disruptions, which can lead to data loss or inaccessibility. The reliance on a single server or a small group of servers creates a critical vulnerability in the system.

3. Scalability: As data volumes continue to grow, traditional file systems may struggle to maintain optimal performance and storage efficiency. Scaling these systems requires significant investment in hardware and infrastructure, which can be costly and time-consuming.

Purpose of New System

The primary objective of the proposed file system is to address the shortcomings of traditional systems by harnessing the power of the IPFS network and MFS, while incorporating Firebase for enhanced functionality. The new system aspires to provide the following benefits:

1. Data integrity: IPFS uses cryptographic hashing to ensure data integrity. Each file and its metadata are associated with a unique hash, which makes it virtually impossible for malicious actors to tamper with or corrupt stored data. This feature enhances the security and trustworthiness of the proposed system.

2. Scalability: IPFS is designed to handle massive volumes of data, allowing the proposed system to scale seamlessly with growing user demands. The distributed nature of IPFS also enables efficient data retrieval, even when dealing with large files or high levels of user traffic.

3. Decentralization: The IPFS network is based on a distributed architecture, which stores data across multiple nodes. This eliminates single points of failure and increases the overall reliability and resilience of the system. In the event of a node failure, data can still be retrieved from other nodes on the network.

USER STORIES

The following section provides the detailed user stories that were implemented in this iteration of the IPFS-based file system project. These user stories served as the basis for the implementation of the project's features. This section also shows the user stories that are to be considered for future development.

Implemented User Stories

1. As a user, I want to register for an account so that I can access the IPFS-based file system.

2. As a user, I want to log in to my account so that I can access my files and directories on the IPFS network.

3. As a user, I want to have a dedicated directory created for me upon registration, so that I can store and manage my files in an organized manner.

4. As a user, I want to upload files to my directory so that they are stored securely on the IPFS network.

5. As a user, I want to download files from my directory so that I can access them on my local machine.

6. As a user, I want to delete files from my directory so that I can manage my storage space on the IPFS network.

7. As a user, I want to create and manage subdirectories within my directory so that I can organize my files more effectively.

8. As a user, I want to search for files within my directory so that I can quickly locate the files I need.

9. As a user, I want to share files with other users securely so that I can collaborate on projects or share important documents.

Pending User Stories

Moving Files
To move files within the software, follow these steps:

4. Select the file(s) you wish to move.

5. Click on the "Move" icon or right-click the file and choose "Move" from the context menu.
6. Navigate to the destination folder where you want to move the file(s).
7. Click on "Move" to confirm the action.

Content Tracking
Content tracking allows you to monitor changes and updates made to files in real-time. To view content tracking:

1. Open the file you want to track.
2. Click on "File" in the menu bar and choose "Track Changes" from the dropdown menu.
3. Confirm your choice by clicking "View Content Addressing"

You can now view the tracked changes in the file, including additions, deletions, and modifications. This feature makes it easy to review and approve changes made by team members, ensuring the integrity and accuracy of the content.

Hardware and Software Resources

The following is a list of all hardware and software resources that were used in this project:

Hardware Resources:

1. Development machines: Laptops or desktop computers with sufficient processing power, memory, and storage capacity to develop, test, and run the project locally. These machines should be equipped with a modern processor, at least 8 GB of RAM, and adequate storage space for the codebase, dependencies, and project files.

2. Server infrastructure: A server or a cloud-based hosting solution to deploy the IPFS node, frontend, and backend components of the project. The server should have adequate processing power, memory, and storage capacity to handle user requests, store uploaded files, and maintain directories on the IPFS network.

Software Resources:

1. IPFS: The InterPlanetary File System (IPFS) is a core component of the project, providing the decentralized file storage and retrieval capabilities. Developers should install and configure IPFS on their development machines and the deployment server.

2. TypeScript: TypeScript is used for the frontend development of the project. Developers should have a working knowledge of TypeScript and have the necessary tools and libraries installed on their development machines.

3. Golang: Golang is used for the backend development of the project. Developers should have a working knowledge of Golang and have the necessary tools, libraries, and frameworks installed on their development machines.

4. Firebase: Firebase is used for user authentication and management. Developers should have a Firebase account and the necessary API keys and SDKs to integrate Firebase with the project.

5. Version Control System (e.g., Git): A version control system is essential for managing the codebase, tracking changes, and collaborating with other developers. Git is a widely used version control system that can be used for this project.

6. Integrated Development Environment (IDE): A suitable IDE is required for writing and debugging code. Developers can choose an IDE that supports TypeScript, Golang, and IPFS development, such as Visual Studio Code, IntelliJ IDEA, or WebStorm.

7. Operating System: Developers should use an operating system compatible with the chosen development tools and libraries, such as Windows, macOS, or Linux.

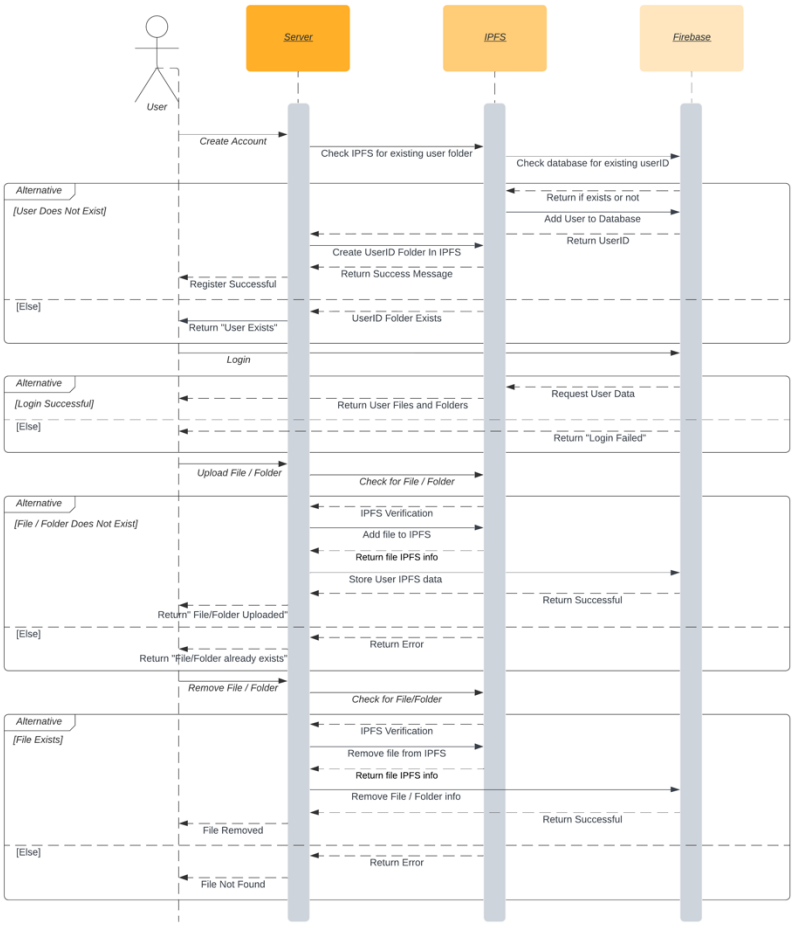
8. Additional Libraries and Frameworks: Depending on the project requirements, additional libraries and frameworks may be necessary to implement specific features, such as data encryption, file compression, or third-party application integration.

SYSTEM DESIGN

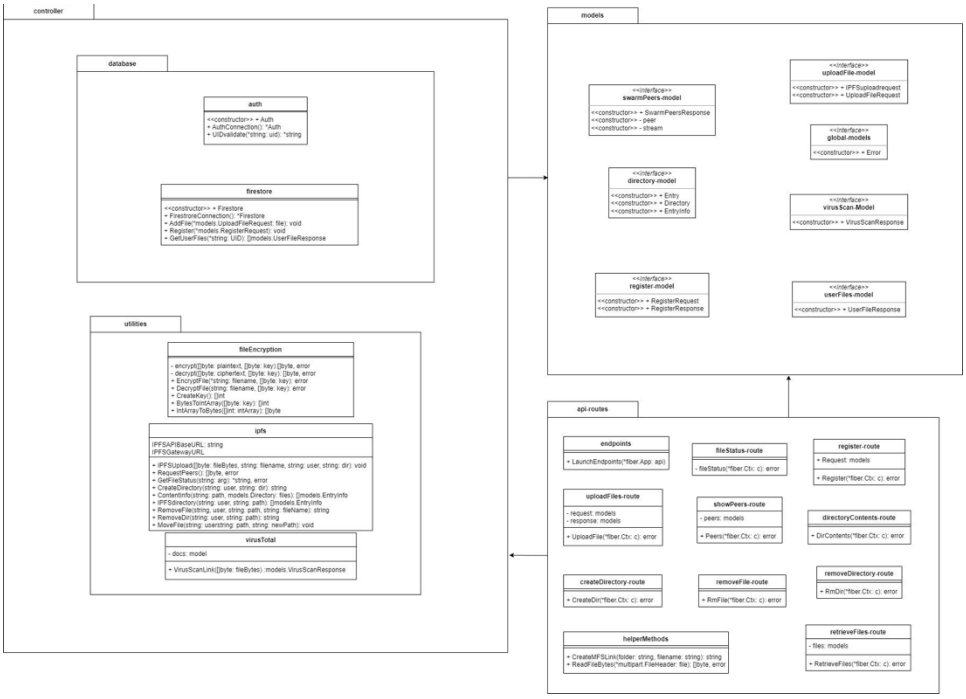
This system is a decentralized and permanent storage solution built on the IPFS network with MFS and Firebase. It provides users with secure and private decentralized storage by creating directories on the IPFS network utilizing MFS. The system uses Firebase for user registration, and the backend API creates a user directory on the IPFS network with their Firebase UID. The system also pins files in the server, ensuring their permanence for the users.

To visualize the functionality of the system, we have created sequence diagrams to illustrate the interactions between the front-end, backend, IPFS node, and Firebase. We have also designed relational diagrams to demonstrate the relationships between the entities in the system, such as users, directories, and files. Additionally, we have created use case diagrams to illustrate the possible user interactions with the system and how they relate to the system's functionality. Together, these diagrams provide a comprehensive understanding of the system's architecture and functionality, enabling developers to efficiently design, develop, and test the system.

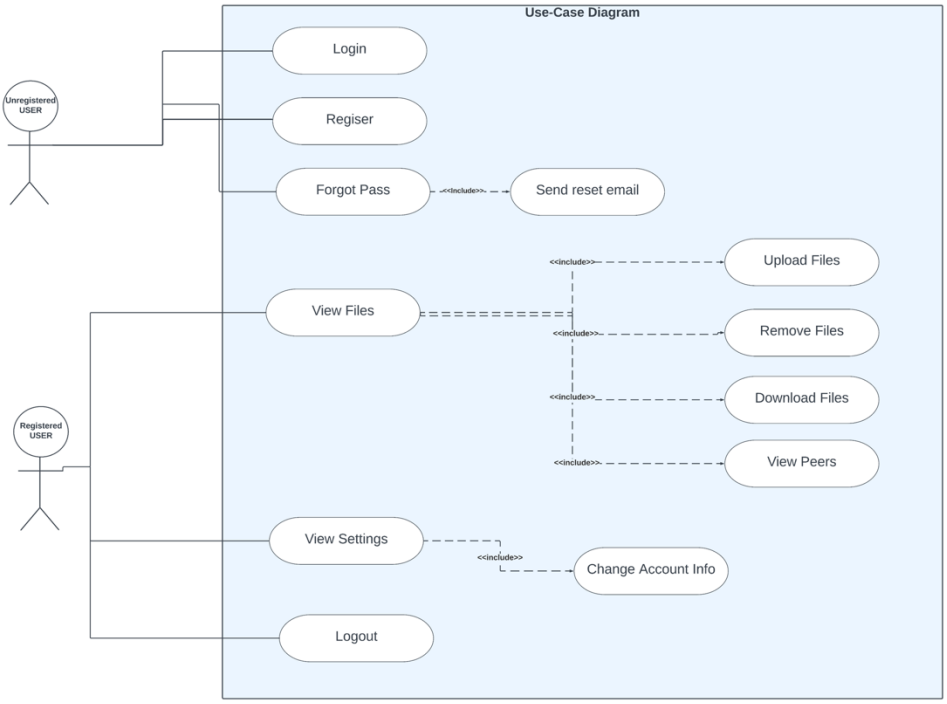
Sequence Diagram



Class / Relation UML Diagram

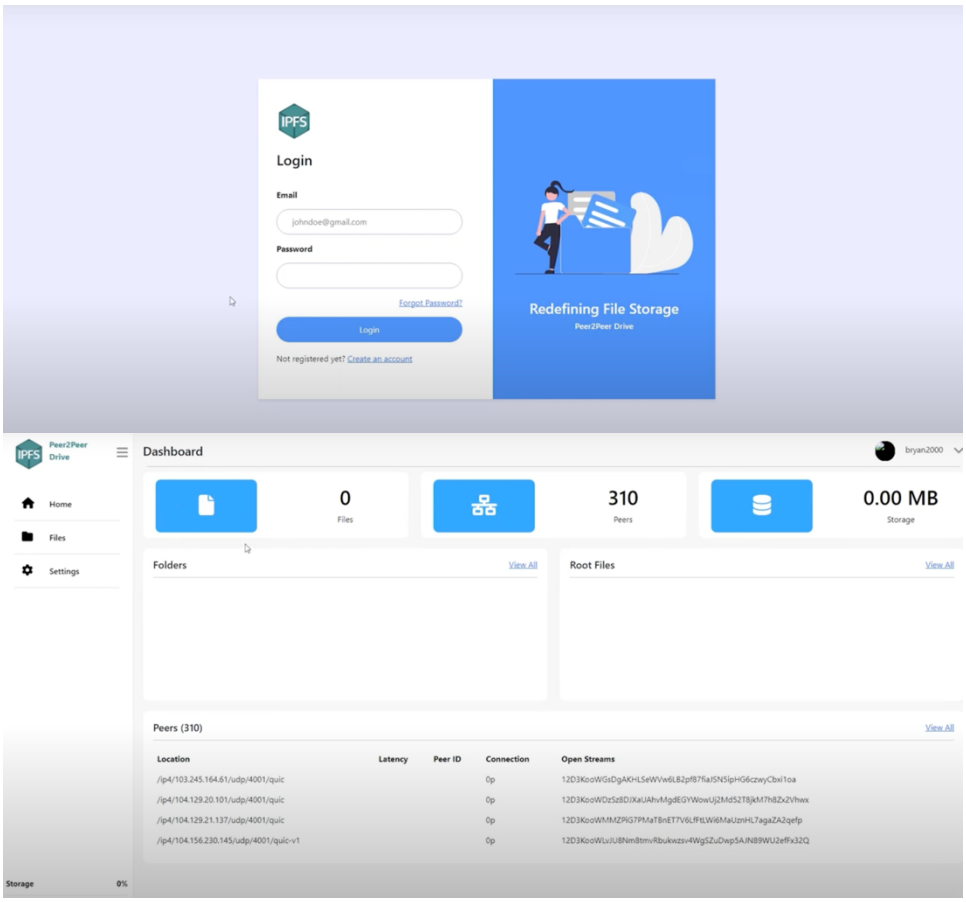

controller
database
auth
<<constructor>> + Auth
+ AuthConnection(): *Auth
+ UIDvalidate(*string, uid): *string
firestore
<<constructor>> + Firestore
+ FirestoreConnection(): *Firestore
+ AddFile(*models.UploadFileRequest, file): void
+ Register(*models.RegisterRequest): void
+ GetUserFiles(*string, UID): []models.UserFileResponse
utilities
fileEncryption
- encrypt([byte, plaintext, []byte, key): []byte, error
- decrypt([byte, ciphertext, []byte, key): []byte, error
+ EncryptFile(*string, filename, []byte, key): error
+ DecryptFile(string, filename, []byte, key): error
+ CreateKey(): []int
+ BytesToIntArray([byte, key): []int
+ IntArrayToByte([]int, intArray): []byte
ipfs
IPFSAPIBaseURL: string
IPFSGatewayURL
+ IPFSUpload([byte, fileBytes, string, filename, string, user, string, dir): void
+ RequestPeers(): []byte, error
+ GetFileStatus(string, arg): *string, error
+ CreateDirectory(string, user, string, dir): string
+ ContentInfo(string, path, models.Directory, files): []models.EntryInfo
+ IPFSdirectory(string, user, string, path): []models.EntryInfo
+ RemoveFile(string, user, string, path, string, fileName): string
+ RemoveDir(string, user, string, path): string
+ MoveFile(string, userstring, path, string, newPath): void
virusTotal
- docs: model
+ VirusScanLink([byte, fileBytes): models.VirusScanResponse
models
<<Interface>>
swarmPeers-model
<<constructor>> + SwarmPeersResponse
<<constructor>> - peer
<<constructor>> - stream
<<Interface>>
uploadFile-model
<<constructor>> + IPFSuploadrequest
<<constructor>> + UploadFileRequest
<<Interface>>
global-models
<<constructor>> + Error
<<Interface>>
directory-model
<<constructor>> + Entry
<<constructor>> + Directory
<<constructor>> + EntryInfo
<<Interface>>
virusScan-Model
<<constructor>> + VirusScanResponse
<<Interface>>
register.model
<<constructor>> + RegisterRequest
<<constructor>> + RegisterResponse
<<Interface>>
userFiles-model
<<constructor>> + UserFileResponse
api-routes
endpoints
+ LaunchEndpoints(*fiber.App, api)
fileStatus-route
- fileStatus(*fiber.Ctx: c): error
register-route
+ Request: models
+ Register(*fiber.Ctx: c): error
uploadFiles-route
- request: models
- response: models
+ UploadFile(*fiber.Ctx: c): error
showPeers-route
- peers: models
+ Peers(*fiber.Ctx: c): error
directoryContents-route
+ DirContents(*fiber.Ctx: c): error
createDirectory-route
+ CreateDir(*fiber.Ctx: c): error
removeFile-route
+ RmFile(*fiber.Ctx: c): error
removeDirectory-route
+ RmDir(*fiber.Ctx: c): error
helperMethods
+ CreateIPFSLink(folder: string, filename: string): string
+ ReadFileBytes(*multipart.FileHeader: file): []byte, error
retrieveFiles-route
- files: models
+ RetrieveFiles(*fiber.Ctx: c): error

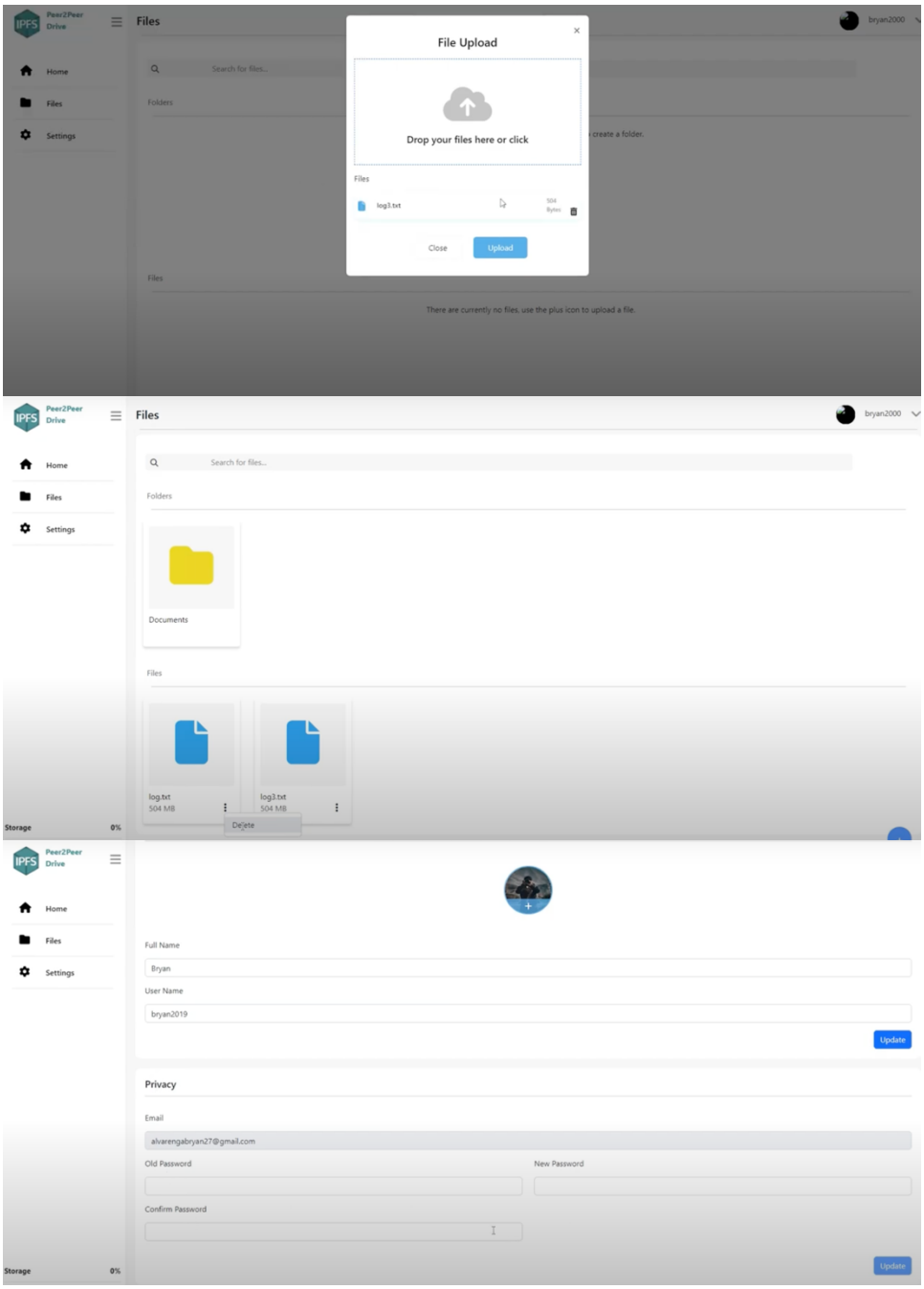
Use- Case Diagram



APPENDIX

Appendix A - User Interface Design




Peer2Peer Drive
Files
bryan2000
Home
Files
Settings
Search for files...
Folders
File Upload
Drop your files here or click
Files
log3.txt
Close
Upload
There are currently no files, use the plus icon to upload a file.
Documents
log.txt
504 MB
log3.txt
504 MB
Delete
Storage
0%
Full Name
Bryan
User Name
bryan2019
Update
Privacy
Email
alvarengabryan27@gmail.com
Old Password
New Password
Confirm Password

Appendix B - Sprint Plan Reports

Sprint Planning Meeting Minutes 1:
Project: P2P
Attendees: Diaz, Dominick; Bryan Alvarenga; Delgado, Anthony; Benni, Sonali; Masri, Hamzah;
Start time: 7:00pm
End time: 8:00pm

After discussion with the team and considering their velocity, the following 6 user stories have been selected for the next sprint.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.
User Story 1: As a user, I want to be able to securely store and access my files using Firebase, so that my files are safe and easily accessible.
User Story 2: As a software engineer, I want to integrate Firebase into the existing functions of the application's backend, so that users can store and access their files seamlessly.
User Story 3: As a software engineer, I want to create a UML diagram for the application, so that the architecture and design can be clearly understood.
User Story 4: As a software engineer, I want to improve the design of the front-end, so that the user experience is optimized and meets the needs of the users.
User Story 5: As a software engineer, I want to develop the landing page of the front-end, so that users have an engaging and easy-to-use first impression of the application.
User Story 6: As a software engineer, I want to understand Angular syntax to be able to effectively develop the front-end of the application and enhance the user experience.

Sprint Planning Meeting Minutes 2:
Project: P2P
Attendees: Diaz, Dominick; Bryan Alvarenga; Delgado, Anthony; Benni, Sonali; Masri, Hamzah;
Start time: 7:00pm
End time: 8:00pm

Sprint Goal: Launch and configure the server for the API, install IPFS, and launch our API. Conduct research on the creation of folders within our file system using IPFS Data Models and explore the implementation of mutable files on the IPFS network. Complete the UI for the client application.

User Stories:
User Story 1: As a team, we need to launch and configure the server for the API and install IPFS, so that we can have access to the server as soon as possible.
User Story 2: As a team, we need to launch our API, so that we can begin testing it and making any necessary changes.
User Story 3: As a team, we need to conduct research on the creation of folders within our file system using IPFS Data Models, so that we can implement this feature in our application by the end of the sprint.
User Story 4: As a team, we need to explore the possibility of mutable files on the IPFS network, so that we can implement this feature in our application by the end of the sprint.
User Story 5: As a team, we need to complete the UI for the client application, so that it matches our Figma design and all elements of the dashboard are complete by the end of the sprint.
Tasks:

- Launch and configure the server for the API
- Install IPFS
- Launch the API
- Conduct research on the creation of folders within our file system using IPFS Data Models
- Explore the implementation of mutable files on the IPFS network
- Complete the UI for the client application

Sprint Planning Meeting Minutes 4:
Project: P2P Attendees: Diaz, Dominick; Bryan Alvarenga; Delgado, Anthony; Benni, Sonali; Masri, Hamzah;
Start time: 7:00pm
End time: 8:00pm
After discussion, the velocity of the team was estimated to be 35 story points.
The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

User Story 1: As a team, we need to perform maintenance for the server, including port rules and installations (8 story points).
User Story 2: As a team, we need to work on the front-end application design (6 story points).
User Story 3: As a team, we need to establish communication between the front-end and back-end API (10 story points).

User Story 4: As a team, we need to conclude the development of v0.1 API documentation and perform code refactoring to achieve consistent design principles (5 story points).
User Story 5: As a team, we need to research and implement IPNS for naming and logging (6 story points).

Goals for the Sprint:

- Complete the maintenance tasks for the server, including setting up port rules and necessary installations.
- Finalize the design for the front-end application, ensuring a consistent and user-friendly interface.
- Establish smooth communication between the front-end and back-end API, allowing for seamless data exchange.
- Research and implement IPNS for naming and logging, improving system efficiency and organization.
- Complete all user stories within the sprint timeframe and deliver a high-quality product.

Sprint Planning Meeting Minutes 5:
Project: P2P
Attendees: Diaz, Dominick; Bryan Alvarenga; Delgado, Anthony; Benni, Sonali; Start time: 7:00pm
End time: 8:00pm
After discussion, the velocity of the team was estimated to be 30 story points.
The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

User Story 1: As a team, we need to perform maintenance for API Auth (5 story points).
User Story 2: As a team, we need to work on the front-end Home page design (5 story points).
User Story 3: As a team, we need to refine communication between the front-end and back-end API (10 story points).
User Story 4: As a team, we need to refine the development of v0.2 API documentation and perform code refactoring to achieve consistent design principles (5 story points).
User Story 5: As a team, we need to work on bug fixes and code refactoring (5 story points).

Sprint Planning Meeting Minutes 6:

Project: P2P
Attendees: Diaz, Dominick; Bryan Alvarenga; Delgado, Anthony; Benni, Sonali;
Start time: 7:00pm
End time: 8:00pm

After discussion from last week, the velocity of the team was estimated to be 30 story points, in addition to last backlog stories, we are finishing project docs.
The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

User Story 1: As a team, we need to perform maintenance for API Auth (5 story points).
User Story 2: As a team, we need to maintain the front-end Home page design (5 story points).
User Story 3: As a team, we need to refine communication between the front-end and back-end API (10 story points).
User Story 4: As a team, we need to refine the development of v0.2 API documentation and perform code refactoring to achieve consistent design principles (5 story points).
User Story 5: As a team, we need to work on bug fixes and code refactoring (5 story points).

Sprint Goals:

Continue to maintain API authentication, ensuring that the system remains secure and functional.
Maintain and improve the front-end Home page design, addressing any issues that arise and ensuring a user-friendly experience.
Further refine communication between the front-end and back-end API, enhancing the application's overall performance.
Complete the development of v0.2 API documentation, ensuring that it is accurate and up-to-date. Perform code refactoring to maintain consistent design principles throughout the project.
Address any bug fixes and continue with code refactoring to improve the overall quality and maintainability of the codebase.
By focusing on these user stories and sprint goals, the team aims to make progress towards completing the project and delivering a high-quality P2P application.

Appendix C - User Manuals, Installation/Maintenance Document, Shortcomings/Wishlist Document and other documents

Overview

The IPFS-MFS Filesystem is a decentralized file system built upon the InterPlanetary File System (IPFS) network, utilizing the Mutable File System (MFS) and Firebase to enable users to create their own directories on the IPFS network. This project consists of a front-end application built in TypeScript, a back-end API developed in Golang, and an IPFS node running on the server.

Installation

Before you begin, make sure you have Git and Golang installed on your machine. You can download them from the following links:

Git: https://git-scm.com/downloads
Golang: https://golang.org/dl/

You will also need an IDE to run the project, such as Visual Studio Code, which you can download from here: https://code.visualstudio.com/download

Getting Started

To get started with this project, follow the steps below:

1. Clone this repository to your local machine by running the following command in your terminal or command prompt:

Replace <username> and <repository-name> with your GitHub username and the name of the repository, respectively.

2. Open the cloned repository in your IDE.

3. Navigate to the frontend directory in the terminal or command prompt and run the following command to install the required dependencies:
   npm install
4. Navigate to the backend directory in the terminal or command prompt and run the following command to install the required dependencies:
   go mod download
5. Create a Firebase project and download the Firebase configuration file (google-services.json) to the frontend/src directory.
6. Start the IPFS node on your server by running the following command:
   ipfs daemon
7. Start the backend API by running the following command in the backend directory:
   go run main.go
8. Start the front-end application by running the following command in the frontend directory:
   npm start

Maintanence

To maintain this project, make sure to keep the dependencies up to date by running the following command in the respective directories:

frontend: npm update
backend: go get -u all

Additionally, make sure to keep the IPFS node up to date by running the following command on your server:

ipfs update

REFERENCES

- Backend References
  - Golang
    - https://go.dev/
  - Go-Fiber

        - https://gofiber.io/
    - Postman
        - https://www.postman.com/
    - IPFS
        - https://ipfs.tech/
    - Firebase / Firestore
        - https://firebase.google.com/

- Frontend References
    - TypeScript
        - https://www.typescriptlang.org/
    - Angular
        - https://angular.io/
    - Figma
        - https://www.figma.com/