THE DEVELOPMENT OF A FILE-SHARING DECENTRALIZED APPLICATION(DAPP)

PRESENTED BY

SARANSH KASLIWAL-2021A7PS2944H

RISHAB JAIN-2021A8PS1073H

NITIN JINAGAM-2021A8PS1056H

NITISH BRAT DAS-2021A8PS3185H

SURYA PAVAN-2021A8PS1054H

RUQQYAH KILLEDAR-2021A1PS2895H

CONTENTS:

5. Frontend Implementation

- React.js Components
- MetaMask integration
- Ether.js Usage
- Pinata Integration
- User Interface Elements
- Transaction Confirmation and Fees

1. Project Overview:

- This project focuses on the development of a decentralized file-sharing decentralized application (DAPP). The technology stack used for this project includes Solidity, React JS, and Ether.js, with a strong emphasis on achieving a decentralized file system.

- The project suggests an evolution beyond traditional cloud storage solutions like Google Drive (2.0). The objective is to create a decentralized alternative that enables users to upload images, share access with specific addresses, and revoke permissions when needed.

2. Technologies Used:

- **React.js**: A popular JavaScript library for building user interfaces, is employed for the frontend development. It offers a component-based architecture and efficient state management.

- **Ether.js**: A JavaScript library for interacting with the Ethereum blockchain, is utilized to enable seamless communication between the frontend and the Ethereum network. This library streamlines tasks such as transactions and contract interactions.

- **Solidity**: A programming language for writing smart contracts on the Ethereum blockchain, is chosen for the backend development. Smart contracts define the business logic of the decentralized file-sharing application, including access control and image management.

- **MetaMask**: A browser extension wallet, is integrated into the project, providing users with a secure way to manage their Ethereum accounts and interact with decentralized applications.

- **Hardhat**: A popular development environment for Ethereum smart contracts. It provides a flexible and extensible framework for compiling, testing, and deploying smart contracts. Hardhat also offers built-in support for tasks like debugging, testing, and scriptable deployment.

- **IPFS (Interplanetary File System)**: A decentralized protocol for storing and sharing data in a distributed file system. It uses content-addressing to uniquely identify files and allows for peer-to-peer file sharing. IPFS aims to create a more resilient and censorship-resistant way of storing and accessing information on the internet.

3. Key Features:

- **File Upload**: Users can upload images to the decentralized file system, providing a decentralized alternative to centralized cloud storage solutions.

- **Access Control**: The dApp enables users to share access to their uploaded images with specific Ethereum addresses. This feature allows for a controlled and permission-based approach to file sharing.

- **Access Revocation**: The project includes functionality to revoke access permissions previously granted. This adds a layer of security and control over shared content, enhancing user privacy and data management.

- **Decentralization Emphasis**: The overarching goal of the project is to emphasize decentralization, diverging from the centralized model of existing file storage solutions. The project aims to empower users with increased control and ownership over their data.

4. Smart Contract Functions:

- Functionality:

 - **Add Function**: The smart contract includes an "add" function, allowing users to grant access to their drive. This function likely involves updating the access control list or mapping to include the Ethereum address of the recipient.

 - **Access List Management**: Access permissions are maintained in a mapping structure, indicating which addresses have access to specific drives. This structure is likely utilized for efficient and secure access control.

- Allow function: The **allow** function is used to grant access to another user, updating the access control information in the **ownership** and **accessList** mappings.
- Disallow function: The **disallow** function revokes access for a user by updating the access control information in the **ownership** and **accessList** mappings.
- Share Access function: The **shareAccess** function returns the access control list for the calling user.

 - **Display Functions**: The smart contract includes display functions to allow users to view images associated with their Ethereum address on the decentralized file system. This function likely retrieves and presents relevant image data.

- The **display** function allows a user to view their stored URLs, but only if the requesting user has the necessary access rights.

- Access Control:
    - Access control is managed through the **ownership** mapping and the **accessList** mapping
    - Ownership Mapping: The **ownership** mapping tracks which users have access to specific data
    - AccessList: The **accessList** mapping stores a list of users and their access status

- Modifiers:
    - The **display** function includes **require** statement to check if the requesting user has access to the data.

5. Frontend Implementation:

- **React.js Components**: The frontend is developed using React.js, utilizing its component-based architecture. Components such as file upload, display, and modal are created to provide a seamless user experience.

- **MetaMask Integration**: MetaMask, a popular Ethereum wallet, is integrated into the frontend to enable account management and facilitate secure interactions with the Ethereum blockchain. Users can switch between accounts seamlessly.

- **Ether.js Usage**: Ether.js is employed in the frontend to interact with the Ethereum blockchain. This includes handling transactions, calling smart contract functions, and managing the state of the application.

- **Pinata Integration**: Pinata, a decentralized file storage service, is mentioned for image storage. The code encounters an issue related to slow uploads to Pinata, and troubleshooting steps are discussed to address this challenge. The process involves obtaining an API key from Pinata for authentication.

- **User Interface Elements**: The code demonstrates the creation of user interface elements, such as a share button and a modal, to enhance the overall user experience. The share button triggers the opening of a modal for managing access permissions.

- **Transaction Confirmation and Fee**: The frontend likely includes features to confirm successful transactions on the Ethereum blockchain. Users are informed about transaction fees associated with modifying the blockchain state, emphasizing the decentralized and trustless nature of the application.