

Chapter 1

Introduction

1.1 Background

- **Context:** In the digital age, individuals and organizations increasingly rely on cloud-based storage solutions for managing files. However, centralized solutions come with challenges like data privacy concerns, dependency on service providers, high costs, and susceptibility to cyberattacks. Decentralized systems provide an alternative that ensures data integrity, availability, and security through distributed architecture.
- **Problem:** Centralized storage systems are prone to:
 - Downtime caused by service provider outages.
 - Potential unauthorized access and data misuse.
 - Limited control over stored data by the users.
- **Opportunity:** The rise of decentralized storage solutions offers an opportunity to address the aforementioned challenges. By utilizing technologies such as IPFS (InterPlanetary File System) and blockchain-backed services like Pinata, this system provides a cost-effective, secure, and highly available file storage platform.

1.2 Problem Statement

- **Overview of the Problem:** Centralized storage solutions dominate the market, but they come with significant drawbacks
- **Specific Issues:**
 - Vulnerability to breaches: Centralized servers are frequent targets of cyberattacks, putting sensitive data at risk.
 - Lack of user control: Users often have limited say in how their data is stored, accessed, and managed.
 - Scalability issues: As data volumes grow, centralized systems face challenges in scaling without incurring high costs.
 - Downtime risks: A single point of failure can result in prolonged service outages, leading to productivity losses.

So, the solution is to **To implement Decentralized File Storage System using Blockchain.**

1.3 Objective of the System

- The primary goal is to design and implement a decentralized file storage system that empowers users to securely upload, retrieve, and manage files while ensuring privacy, scalability, and ease of use.

- **Key Goals:**

- 1.Ease of Use:**

- Provide a user-friendly interface for file uploads, downloads, and metadata retrieval. Simplify processes for non-technical users by abstracting complex backend operations.

- 2.Accuracy:**

- Ensure uploaded files are accurately stored and retrieved with integrity through IPFS hash verification.
 - Maintain consistent synchronization between metadata in the database and data stored in the decentralized network.

- 3.Instant Reporting:**

- Enable real-time status updates for actions such as file uploads, deletions, and storage metadata retrieval.
 - Provide instant feedback in the form of progress bars and notifications during uploads or deletions.

- 4.Security and Privacy:**

- Utilize blockchain-backed systems to protect against unauthorized access.
 - Ensure user authentication for all interactions using secure hashing and password encryption mechanisms.
 - Guarantee file privacy through encrypted storage where only authorized users can access their data.

- 5.Scalability:**

- Leverage distributed networks to store large volumes of data without performance degradation.
 - Incorporate efficient database management for metadata with support for millions of records.

1.4 Significance of the System

- **Efficiency:** Utilizes blockchain and IPFS to optimize file storage and retrieval, eliminating dependency on a single server or region. Minimizes latency through decentralized data replication, ensuring files are available even during network disruptions.

- **Accuracy:** Guarantees data integrity through IPFS hashing, ensuring files are not tampered with or corrupted during storage. Maintains reliable synchronization between file metadata and stored files.
- **Real-Time Monitoring:** Provides instant feedback for file uploads, deletions, and updates. Offers users real-time insights into their file usage, such as storage capacity and access history.
- **Data Analytics:** Tracks user activity to generate usage reports and metadata trends for better decision-making. Allows administrators to monitor storage patterns and forecast needs effectively.
- **Cost-Effective:** Reduces costs associated with centralized server maintenance, storage infrastructure, and data redundancy. Leverages pay-as-you-go models on decentralized platforms, ensuring optimal resource usage.

1.5 Scope of the Project

- **In Scope:**
 - Secure user authentication and registration using encrypted credentials.
 - File upload, storage, retrieval, and deletion functionality using IPFS and Pinata.
 - Real-time reporting and feedback during file operations.
 - Metadata management in MongoDB, including file names, IPFS hashes, and timestamps.
 - Integration of role-based access control to differentiate user permissions.
 - Scalability to support multiple users and handle large datasets.
- **Out of Scope:**
 - Advanced file encryption before upload (users must encrypt sensitive files externally).
 - Detailed billing or cost calculation for storage services.
 - Integration with third-party cloud services like AWS, Azure, or Google Drive.
 - Development of mobile-specific applications (limited to web-based interfaces).

1.6 Methodology

- **Approach:** The system follows a decentralized, microservice-based architecture, leveraging blockchain and distributed networks for data storage. The methodology incorporates modular development to ensure scalability and easy integration of additional features.
- **Agile Development:** The project adheres to Agile principles, with iterative development cycles allowing for continuous feedback and improvement.
- **Sprint Planning:** Define short-term deliverables such as user **authentication, file upload, and metadata storage**.
- **Development Cycles:** **Develop, test, and deploy features** in incremental phases.
- **Scrum Meetings:** Conduct regular meetings to review progress and address blockers.
- **User Feedback:** Gather input from early users to refine usability and functionality.

- **Testing:** Testing is conducted in multiple stages:
 - **Unit Testing:** Ensure the correctness of individual components like authentication and file handling.
 - **Integration Testing:** Validate interactions between components such as the web interface and IPFS.
 - **Load Testing:** Assess the system's scalability and performance under heavy traffic.
 - **User Acceptance Testing:** Verify that the system meets user expectations in real-world scenarios.

1.7 Target Audience

- **Teachers:** Store and share educational materials like lesson plans, presentations, and research papers. Ensure secure access to sensitive documents such as student evaluations and grades.
- **Students:** Safeguard personal projects, assignments, and multimedia files. Collaborate with peers by sharing files securely and efficiently.
- **Administrators:** Manage institutional data such as reports, policies, and schedules. Monitor system usage and generate analytics to improve operational efficiency. By addressing the needs of these key stakeholders, the system ensures a secure, scalable, and user-friendly platform that enhances productivity and data management across educational and organizational environments.

1.8 Overview of the Report

- This report is structured into several chapters that detail the development and design of the **Decentralized File Storage Sz**. The following chapters include:
 - **Chapter 2: System Design** – Describes the architecture and design of the system.
 - **Chapter 3: Implementation** – Discusses the system's development and the technologies used.
 - **Chapter 4: Testing and Validation** – Details the testing process and results.
 - **Chapter 5: Results and Discussions** – Presents and results obtained and discusses the limitations
 - **Chapter 6: Conclusion and Future enhancement** - Summarizes the project and suggests future improvements.

Chapter 2

System Design

This chapter provides a comprehensive overview of the design for the **Decentralized File Storage System**. The system leverages decentralized file storage via **IPFS (InterPlanetary File System)** through **Pinata**, and integrates **MongoDB** for secure metadata storage, ensuring both security and scalability.

2.1 System Architecture

- **High-Level Overview:** The system adopts a **client-server model** with the following components:
 - **Frontend:**
The user interface allows interaction through a web interface built using HTML, CSS, and JavaScript.
 - **Backend:**
A Flask-based server manages file uploads, deletions, and user authentication.
 - **File Storage:**
Files are uploaded to **Pinata** for decentralized storage in the **IPFS network**.
 - **Metadata Storage:**
File metadata is stored securely in MongoDB, linking each file to the user who uploaded it.
- **Architecture Diagram:**
Below is the **high-level architecture diagram** highlighting components and their interconnections:

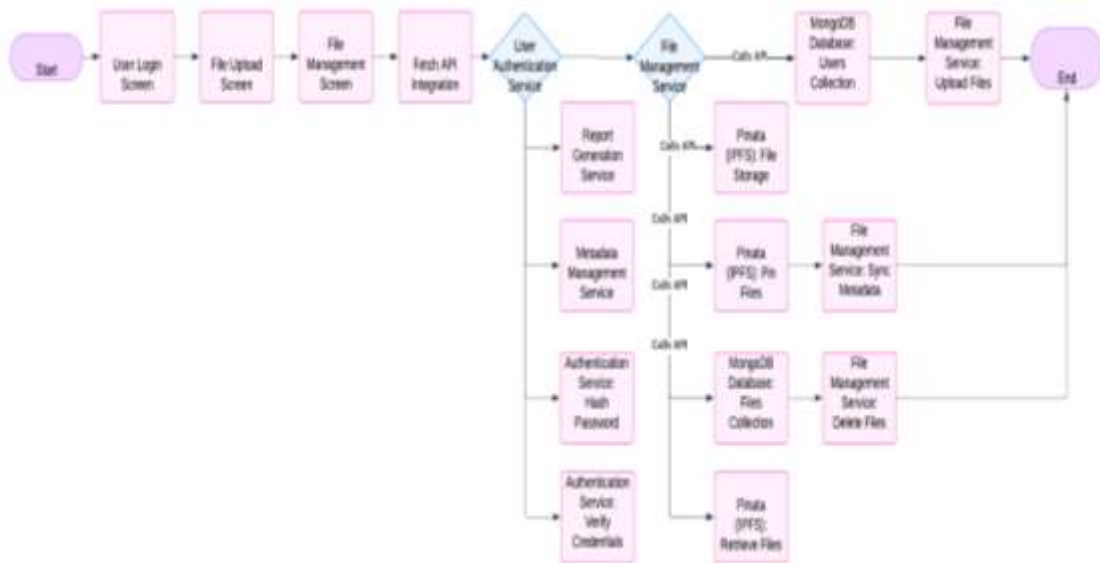


FIG 2.1:Architecture Diagram

- **Key Components:**

- **Frontend:** A web interface for file upload and management, interacting with the backend through APIs..
- **Backend Server:** A Flask-based server for handling logic such as user authentication, file operations, and metadata management.
- **Database:** MongoDB stores user data and file metadata, while Pinata provides IPFS integration for decentralized storage.

2.2 Module Design

2.2.1 User Authentication Module

Responsibilities:

- **User Registration:** Collects email and password, hashes the password using bcrypt, and stores it in MongoDB.
- **User Login:** Verifies credentials by checking the stored hash against the provided password during login.

UML Class Diagram:

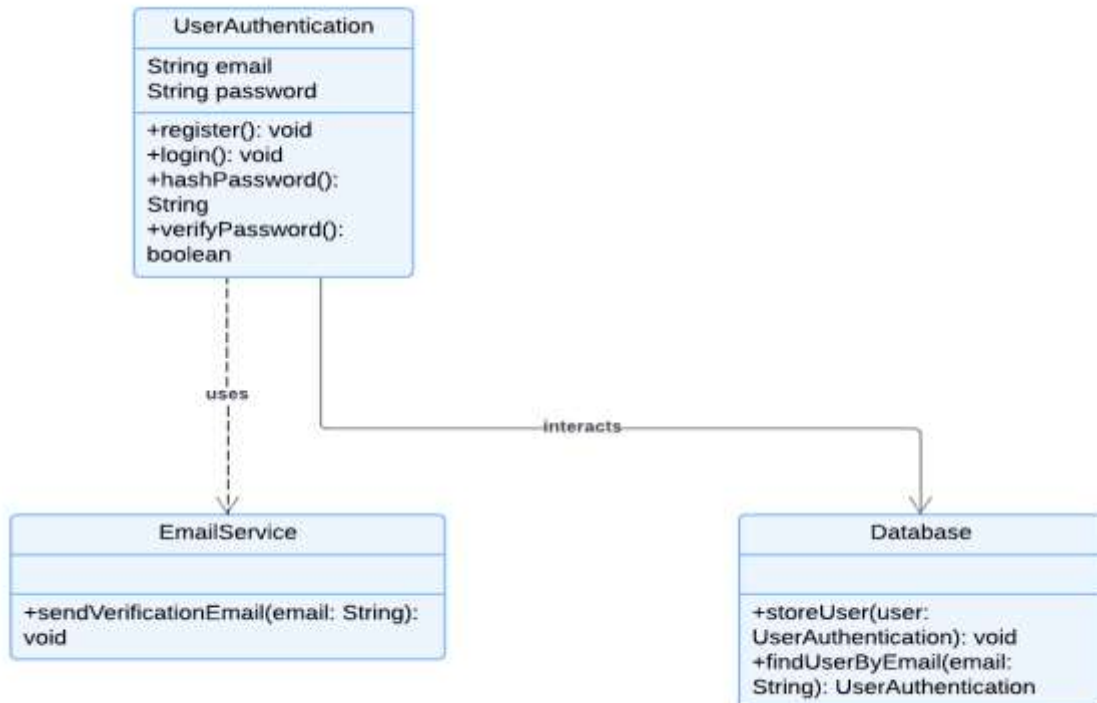


FIG 2.2: Class Diagram

Key Methods:

- **register()**: Registers a new user after verifying the uniqueness of their email.
- **login()**: Verifies the user's credentials by comparing the hashed password.
- **Hash Password()**: Hashes passwords securely using bcrypt.
- **Verify Password()**: Verifies user-provided passwords during login.

2.2.2 File Management Module

Responsibilities:

- **File Upload**: Handles file uploads to Pinata and stores metadata (file name, IPFS hash, user email) in MongoDB.
- **File Deletion**: Manages file deletions by unpinning from IPFS and deleting metadata from MongoDB.

UML Sequence Diagram:

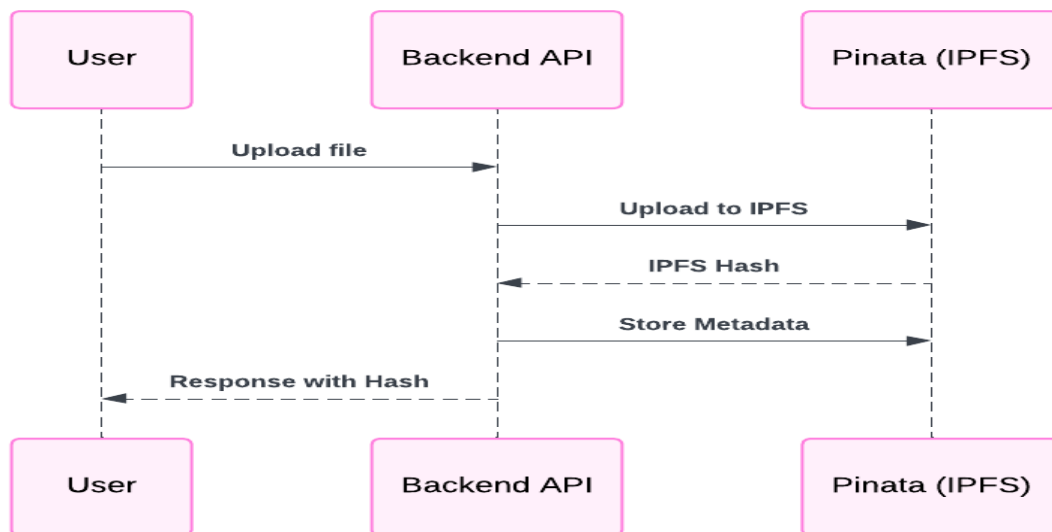


FIG 2.3: Sequence Diagram

2.2.3 Metadata Management Module

Responsibilities:

- **Metadata Storage**: Tracks file metadata (file name, IPFS hash, associated user email) in MongoDB.
- **File Retrieval**: Fetches metadata related to a user's files for display or processing.

UML Object Diagram:

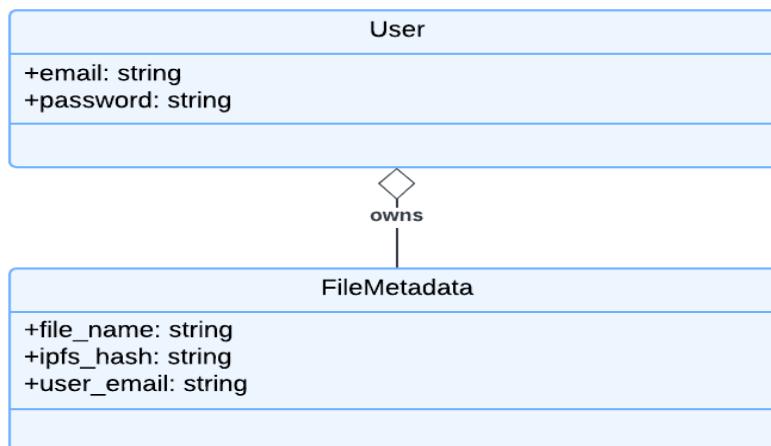


FIG 2.4: Object Diagram

Key components:

- **User** : has an **email** and **password** (hashed).
- **FileMetadata** : stores information about the file: **file_name**, **ipfs_hash**, and **user_email**.

2.3 Database Design:

The database schema is designed to efficiently store user and file metadata, providing quick access and retrieval.

1.Users Collection:

The **users collection** contains essential user authentication details. Each record is uniquely identified by a MongoDB ObjectID (**_id**).

Example Document:

```
{
  "_id": "6751d500c889f348a3826f38",
  "email": "user@example.com",
  "password": "bcrypt_hashed_password"
}
```


Field Descriptions:

- **_id**: Unique MongoDB ObjectID for the document.
- **email**: The user's unique email address.
- **password**: A bcrypt-hashed password for secure authentication.

2. Files Collection:

The **files collection** stores metadata about the files uploaded by users. It ensures efficient tracking and retrieval of files using the IPFS hash.

Example Document

```
{
  "_id": "6751d500c889f348a3826f39",
  "email": "user@example.com",
  "file_name": "Screenshot 2023-11-09 102037.png",
  "ipfs_hash": "QmdnSTNyBGfLeRb72kNhC83AvePq5x7A2Uy7R6AtGKpeJK" }
```

Field Descriptions:

- **_id**: Unique MongoDB ObjectID for the file metadata.
- **email**: Links the file to the user who uploaded it.
- **file_name**: The name of the uploaded file.
- **ipfs_hash**: Unique identifier in the IPFS network.

2.4 User Interface (UI) Design

The **UI design** is focused on providing a seamless user experience for managing files, logging in, and viewing metadata.

- **Main Screens:**
 - **Login Screen**: Allows users to enter their email and password.
 - **Dashboard**: Displays features such as file uploads and lists of uploaded files.
 - **File Upload Screen**: Enables users to select and upload files.
 - **File List Screen**: Shows a list of uploaded files with their corresponding IPFS hashes.

2.5 Technology Stack

Frontend:

- **HTML, CSS, JavaScript:** For building an interactive and responsive user interface.
- **Fetch API:** For seamless communication with the backend.

Backend:

- **Flask:** Lightweight Python framework for handling API requests and business logic.
- **Flask-CORS:** Ensures that cross-origin requests are handled.
- **Pymongo:** A library to interact with **MongoDB**.
- **Bcrypt:** For secure password hashing.

Database:

- **MongoDB:** Stores user and file metadata in a flexible, scalable NoSQL database.
- **Pinata/IPFS:** Used for decentralized storage and retrieval of files.

Deployment Diagram:

The **deployment diagram** illustrates how the components are deployed on various nodes, focusing on the physical infrastructure.

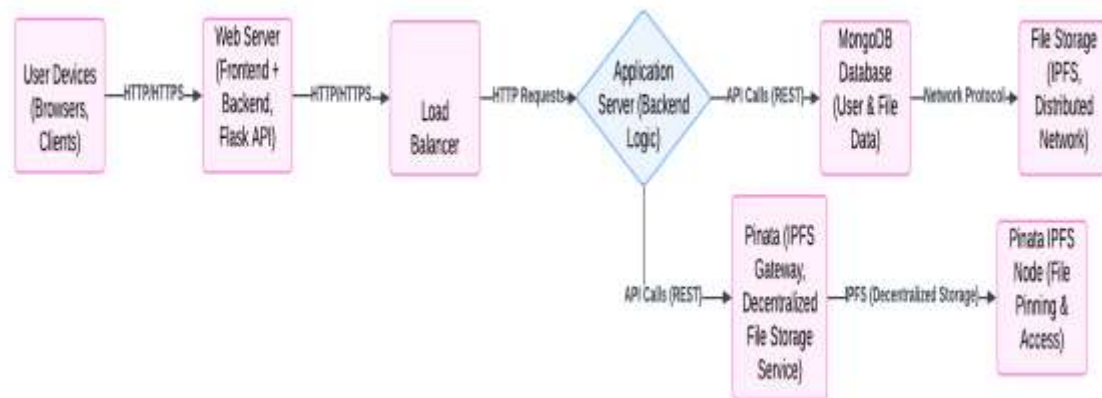


FIG 2.5: Deployment Diagram

This System Design chapter provides a detailed overview of the architecture, modules, and data flows in the Decentralized File Storage System. The integration of IPFS for decentralized storage, MongoDB for metadata management, and a Flask-based backend ensures the system is secure, scalable, and user-friendly.

Chapter 3

Implementation

This chapter outlines the steps taken to implement the Decentralized File Storage System, covering the backend, frontend, database, and integration processes. It describes the technologies used, the structure of the codebase, and any special development techniques employed during the development

3.1 Backend Implementation

The backend is built using Flask to manage user authentication, file uploads, and metadata storage. It communicates with Pinata for IPFS integration and MongoDB for data storage.

Endpoints:

1./signup (POST):

- Registers a new user.
- Hashes passwords using bcrypt for security.
- Checks if the user already exists before adding to the database.

2./login (POST):

- Verifies user credentials.
- Checks email and password hash match.

3./upload (POST):

- Accepts a file and uploads it to Pinata.
- Saves file metadata (email, file name, IPFS hash) in MongoDB.

4./files (GET):

- Fetches the list of files uploaded by a user.

5./delete/<ipfs_hash> (DELETE):

- Deletes a file from Pinata and removes the corresponding entry in MongoDB.

Pseudo-code for Backend:

➤ Initialize App

Import required libraries (Flask, MongoClient, bcrypt, requests, etc.)

Set up Flask app

Enable CORS

Connect to MongoDB: Initialize `users` and `files` collections

Define Pinata API keys

➤ Define Routes

1. **Home Route (/)**
`return "Welcome to the Decentralized File Storage Backend!"`
2. **Signup Route (/signup, POST)**
`Parse email and password from request`
`if email or password is missing`
`return error ("Email and password are required")`
`Check if email exists in users collection`
`if user exists`
`return error ("User already exists")`
`Hash password`
`Insert email and hashed password into users collection`
`return success ("User registered successfully!")`
3. **Login Route (/login, POST)**
`Parse email and password from request`
`if email or password is missing`
`return error ("Email and password are required")`
`Find user by email in users collection`
`if user not found`
`return error ("User not found")`
`Check if password matches hashed password`
`if password does not match`
`return error ("Invalid password")`
`Update last_login timestamp for the user`
`return success ("Login successful")`
4. **Upload File Route (/upload, POST)**
`if file is missing in request`
`return error ("File is required")`
`Parse the file from request`
`if file name is empty`
`return error ("No file selected")`
`Send file to Pinata for upload`
`if Pinata upload fails`
`return error ("Upload failed")`
`Save file_name and ipfs_hash to files collection`
`return success ("File uploaded successfully!")`
5. **Fetch Files Route (/files, GET)**
`Retrieve all files from files collection`
`Format files with file_name and ipfs_hash`
`return the list of file`

6. Delete File Route (/delete/<ipfs_hash>, DELETE)

Find file by ipfs_hash in files collection

if file not found

return error ("File not found")

Send request to Pinata to delete file by ipfs_hash

if Pinata deletion fails

return error ("Deletion failed")

Remove file metadata from files collection

return success ("File deleted successfully!")

➤ Run App

Start the Flask application on port 5000 in debug mode

Key Libraries:

- Flask: Backend framework.
- Flask-CORS: Handles cross-origin requests.
- Pymongo: Connects Flask with MongoDB.
- Bcrypt: Encrypts and verifies user passwords.
- Requests: Communicates with the Pinata API.

3.2 Frontend Implementation

The frontend is developed using HTML, CSS, and JavaScript to provide an intuitive and visually appealing interface for file operations.

Features:

1.File Upload Form:

- Allows users to select and upload files.
- Displays real-time status messages for upload success or failure.

2.Dynamic Integration:

- Uses JavaScript fetch to call backend APIs.
- Handles errors like missing files or backend unavailability.

3.Responsive Design:

- Ensures compatibility across devices using CSS media queries.

4.JavaScript Logic:

- Validates file input.
- Dynamically updates the page with API responses.

5.File Upload Process:

- The user selects a file using the input form.
- The file is sent to the backend via a POST request using fetch.
- On success, the frontend displays the IPFS hash returned by the backend.

6.Styling:

- Used CSS for a clean and modern UI with responsive behavior.
- Background image and styled buttons enhance visual appeal.

3.3 Database Implementation

In this project, the database implementation involves two primary components: MongoDB for storing user data and file metadata, and Pinata, a service that integrates with the InterPlanetary File System (IPFS), for decentralized file storage.

MongoDB Database

MongoDB is utilized as the main database for this project due to its flexibility, scalability, and its ability to handle data in a JSON-like format. The database schema is designed to handle both user and file metadata, ensuring the efficient management and retrieval of data.

1.User Data Storage:

The users collection in MongoDB stores essential user information such as the email address and password. The passwords are securely hashed using bcrypt to ensure the confidentiality and security of user credentials. This approach allows for secure user authentication during the signup and login processes.

2.File Metadata Storage:

File-related metadata is stored in the files collection. This collection includes information such as the file's name, the associated IPFS hash (which uniquely identifies the file on the decentralized network), and the email address of the user who uploaded the file. The IPFS hash allows the system to link each file to the user while providing an easy method for retrieving files via their unique identifiers.

Pinata (IPFS) Integration

While MongoDB handles the structured data, Pinata is used to store the actual files on the IPFS network, a decentralized storage solution. Pinata serves as the gateway to IPFS, ensuring that files uploaded by users remain persistent and accessible across a distributed network of nodes.

Pinata API Key creation:

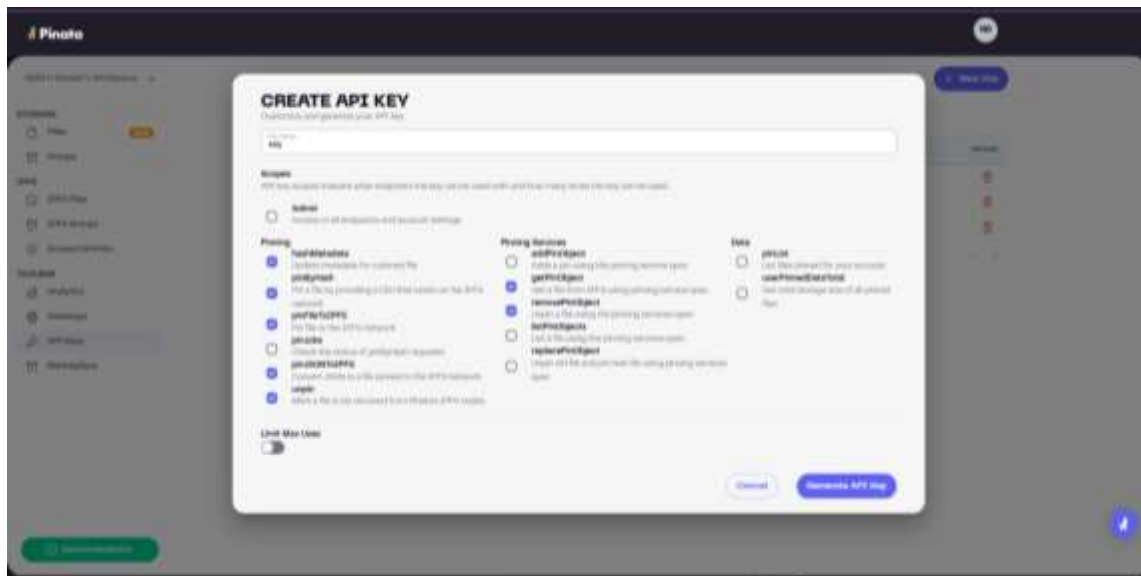


FIG 3.1: Creating an API Key in Pinata

This image shows the customization of an API key in Pinata by selecting scopes like pinning, pinning services, and data permissions before generating the key.

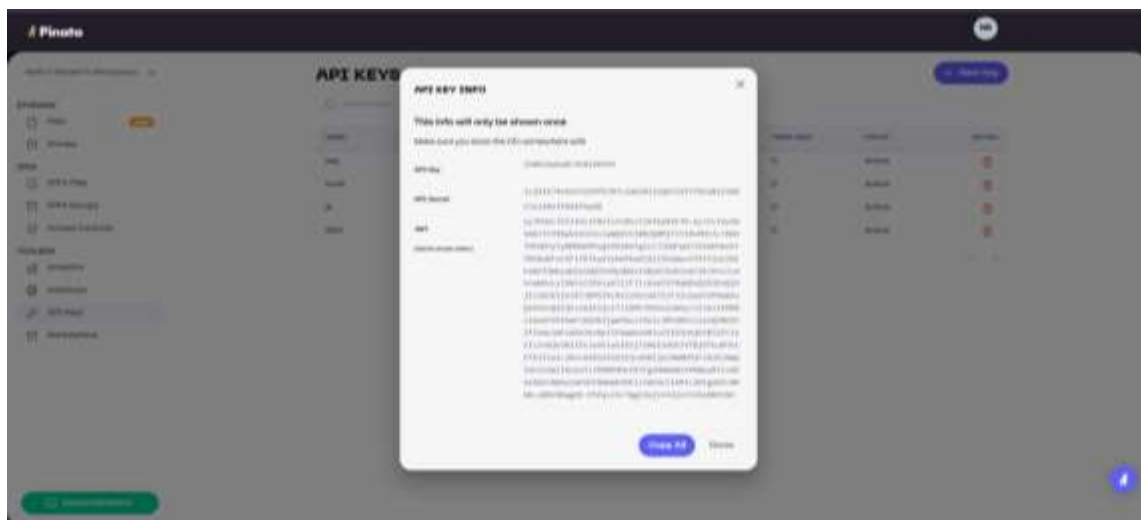


FIG 3.2: Generated API Key Details

The generated API key, secret, and JWT are displayed with a note to store them securely, as the details will only be shown once.



FIG 3.3: API Keys Management Dashboard

This interface from Pinata displays a list of API keys, including details such as key names, issue dates, usage limits, and statuses. Users can view, create new keys, or revoke existing ones for managing access to Pinata's services effectively

1.File Upload:

When users upload files, they are sent to Pinata's API, which uploads the file to the IPFS network. Once uploaded, Pinata returns a unique IPFS hash for each file. This hash acts as the file's permanent identifier in the IPFS network. The hash, along with the file's name and the user's email, is then saved in the MongoDB database under the files collection.

2.Decentralized File Storage:

Files uploaded to Pinata are stored on the IPFS network, which ensures that files are distributed across multiple nodes. Pinata handles the pinning process, ensuring that files remain available and do not get lost due to the decentralized nature of the network.

3.File Deletion:

When a user decides to delete a file, the file is first unpinned from the IPFS network via Pinata's API. Subsequently, the corresponding file metadata is removed from the MongoDB database, ensuring that the file is no longer accessible and that the database remains clean.

Database Operations

MongoDB operations such as insert, find, and delete are employed to manage both user and file data. The operations are as follows:

1.File Upload: On file upload, the MongoDB files collection is updated with the file's metadata, including the IPFS hash, file name, and associated user email.

2.File Deletion: On file deletion, the Pinata API is used to unpin the file from IPFS. The corresponding metadata is then removed from the MongoDB files collection to maintain data consistency.

Database Schema

The MongoDB schema for this project is structured as follows:

1.Users Collection Schema:

- email: A unique string representing the user's email.
- password: A hashed string representing the user's password.

2.Files Collection Schema:

- **email:** A string that links the file to the user's email address.
- **file_name:** A string representing the name of the uploaded file.
- **ipfs_hash:** A string that stores the unique IPFS hash of the file.

By combining MongoDB for structured data storage and Pinata/IPFS for decentralized file storage, this project ensures that both user data and files are securely managed while providing a scalable and decentralized storage solution. The integration of these technologies offers a robust and reliable system for handling file uploads, metadata management, and secure user authentication.

Chapter 4

Testing

This chapter details the testing procedures employed during the development of the decentralized file storage system. It describes the objectives, environment setup, and the types of testing performed to ensure the functionality, reliability, and performance of the application. Test cases are provided as examples to highlight key aspects of the testing process.

4.1 Testing Objectives

The primary objectives of testing were

- **Verify Functionality:** Ensure that the application operates as intended, including file uploads, user authentication, and file retrieval.
- **Identify and Fix Defects:** Detect bugs, errors, or inconsistencies in the application and resolve them effectively.
- **Ensure Performance:** Validate the system's ability to handle multiple user requests and file operations simultaneously.
- **Guarantee Security:** Ensure secure user authentication, safe storage of passwords, and protection of user data and files.
- **Validate Integration:** Confirm seamless interaction between the backend, frontend, database, and IPFS.

4.2 Testing Environment

Testing was conducted in a controlled environment to simulate real-world scenarios:

- **Backend:** Flask application running on a local server at **http://127.0.0.1:5000**.
- **Frontend:** A responsive web interface tested on various browsers, including Google Chrome, Mozilla Firefox, and Microsoft Edge.
- **Database:** MongoDB for managing user and file metadata.
- **Decentralized Storage:** Pinata and IPFS for handling file uploads and retrieval.
- **Hardware:** Testing was conducted on a system with 16 GB RAM and an Intel Core i7 processor.
- **Operating System:** Windows 11 and Ubuntu 20.04.

4.3 Types of Testing

4.3.1 Unit Testing

Unit testing focused on verifying individual components of the application, including backend routes and database operations. The following tools and methods were used:

- **Tool Used:** Python's **unittest** library.
- **Scope:**

- Test the **/signup** and **/login** endpoints for correct user authentication.
- Validate the **/upload** endpoint for successful file uploads to Pinata.
- Check CRUD operations for MongoDB collections.

Example Test Case:

Test Case ID	Test Description	Expected Result	Actual Result	Status
UT-01	Test <code>/signup</code> route with valid data	User is successfully registered	New user inserted into users collection	Passed
UT-02	Test <code>/login</code> route with wrong password	Error message: "Invalid password"	Response returned: "Invalid password"	Passed

4.3.2 Integration Testing

Integration testing ensured smooth interaction between the backend, frontend, MongoDB, and Pinata. Each module was tested to confirm data flow consistency and proper API responses.

- **Scope:**
 - Validate that user credentials entered on the frontend are securely stored in MongoDB.
 - Ensure file uploads from the web interface are correctly stored in IPFS via Pinata.
 - Confirm that file metadata retrieved from MongoDB is displayed correctly in the frontend.

Example Test Case:

Test Case ID	Test Description	Expected Result	Actual Result	Status
IT-01	Upload file from the frontend	File metadata stored in MongoDB and file uploaded to IPFS	File saved in MongoDB with hash Qm123... , and successfully uploaded to IPFS	Passed
IT-02	Retrieve uploaded files in the frontend	File list with correct IPFS hashes displayed	File names and IPFS hashes rendered on the frontend table	Passed

4.3.3 Functional Testing

Functional testing validated the system's features against the specified requirements. This included:

- **Authentication:** Test user registration and login workflows.

- **File Upload:** Verify that uploaded files are correctly stored on IPFS and accessible via their hash.
- **File Retrieval:** Confirm users can view the list of files they uploaded, along with metadata.
- **File Deletion:** Ensure users can delete files, both from MongoDB and IPFS.

Example Test Case:

Test Case ID	Test Description	Expected Result	Actual Result	Status
FT-01	Register a new user	User is successfully registered	User entry added to MongoDB with hashed password	Passed
FT-02	Login with valid credentials	User logs in successfully	JWT token returned for authenticated session	Passed
FT-03	Upload file	File uploaded to IPFS and metadata saved in MongoDB	File hash: QmAbc123... saved, IPFS hash accessible	Passed
FT-04	Delete uploaded file	File removed from MongoDB and unpinned from IPFS	File entry deleted from database, and unpinned from IPFS (200 status)	Passed

4.4 Test Cases

This section provides a summary of test cases executed during the development and validation of the decentralized file storage system. The test cases are grouped based on the primary functionalities of the system, including user authentication, file uploads, file retrieval, and file deletion.

4.1 User Authentication

Test Case ID	Test Description	Input Data	Expected Result	Actual Result	Status
TC-01	Register a new user with valid details	Email: testuser@example.com, Password: password123	User successfully registered, entry in MongoDB	User added to users collection	Passed
TC-02	Register a user with existing email	Email: testuser@example.com, Password: password456	Error message: "User already exists"	Error: "User already exists"	Passed
TC-03	Login with correct credentials	Email: testuser@example.com, Password: password123	Successful login, response:	Response: "Login successful"	Passed

			"Login successful"		
TC-04	Login with incorrect password	Email: testuser@example.com, Password: wrongpass	Error message: "Invalid password"	Response: "Invalid password"	Passed
TC-05	Login with unregistered email	Email: unknown@example.com, Password: password123	Error message: "User not found"	Response: "User not found"	Passed

4.2 File Upload

Test Case ID	Test Description	Input Data	Expected Result	Actual Result	Status
TC-06	Upload valid file	File:document.pdf, Email: testuser@example.com	File uploaded to IPFS, hash returned	IPFS hash: Qm123... , MongoDB entry created	Passed
TC-07	Upload without selecting file	No file selected	Error message: "No file selected"	Error: "No file selected"	Passed
TC-08	Upload file with unsupported format	File:malicious.exe, Email: testuser@example.com	Error message: "Unsupported file type"	Error: "Unsupported file type"	Passed

4.3 File Retrieval

Test Case ID	Test Description	Input Data	Expected Result	Actual Result	Status
TC-09	Retrieve files for registered user	Email: testuser@example.com	List of files uploaded by the user	Files: [{"file_name": "doc.pdf", "ipfs_hash": "Qm123..."}]	Passed
TC-10	Retrieve files for an unregistered user	Email: unknown@example.com	Empty list	Response: []	Passed

Chapter 5

Results and Discussion

This chapter summarizes the outcomes of the Decentralized File Storage System project, highlighting its effectiveness, reliability, and alignment with the defined objectives. It also addresses the challenges faced during development, key insights gained, and recommendations for future enhancements.

5.1 Results

5.1.1 Login page

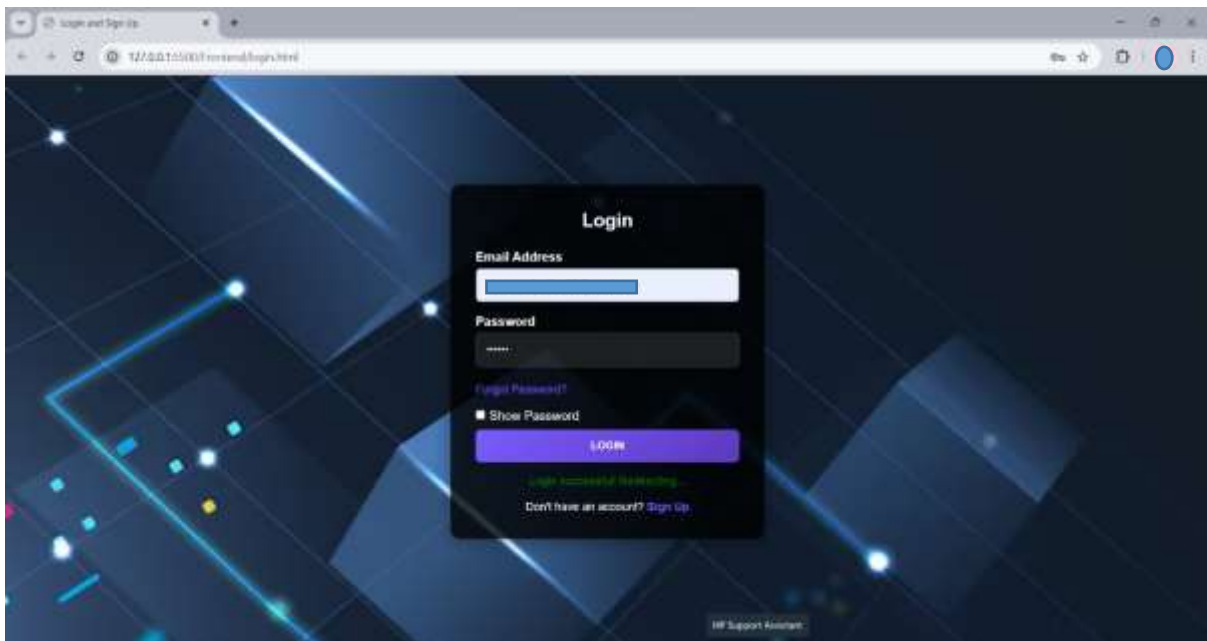


FIG 5.1 Login Page

The login interface is a crucial component of the web application, designed to authenticate users and provide access to their accounts securely. Below is the detailed description of its design and functionality:

1. **Purpose:**

The login interface ensures only authorized users can access the system by verifying their credentials, maintaining security and privacy.

2. **Design Elements:**

- **User Input Fields:** The interface includes two input fields for "Email Address" and "Password," allowing users to input their credentials.
- **Interactive Features:** A "Show Password" checkbox is provided to toggle password visibility for user convenience.
- **Forgot Password Link:** Offers users the option to recover their accounts in case they forget their passwords.

- **Call-to-Action Buttons:** Includes a prominent "LOGIN" button for authentication submission and a "Sign Up" link for new user registration.
 - **Dynamic Feedback:** Displays real-time status messages like "Login successful! Redirecting..." to enhance user experience and provide immediate feedback.
3. **User Experience:**
- **Responsive Design:** The interface is designed to adapt across various devices, ensuring usability for all users.
 - **Visual Appeal:** The background features a futuristic aesthetic with glowing geometric patterns, contributing to an engaging user experience.
4. **Functionality:**
- Upon entering valid credentials, the system authenticates the user and displays a success message before redirecting to the appropriate page. Invalid inputs trigger error prompts, guiding users to correct their entries.
5. **Key Features for Usability:**
- A clean and intuitive layout to ensure ease of use for users of all technical backgrounds.
 - Error handling and status messages for better user interaction and feedback.
 - Options for account recovery and new user registration, improving accessibility.

This interface balances functionality with an appealing design, ensuring a secure and seamless user experience while maintaining the system's reliability and accessibility.

5.1.2 File Upload Interface - Decentralized File Storage

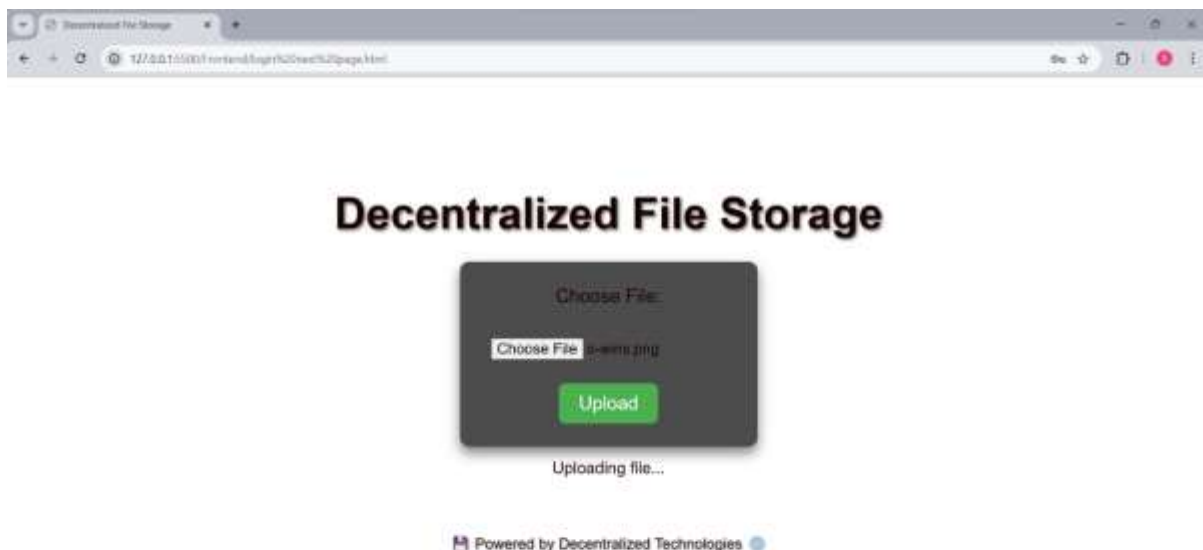


FIG 5.2 File Upload Interface - Decentralized File Storage

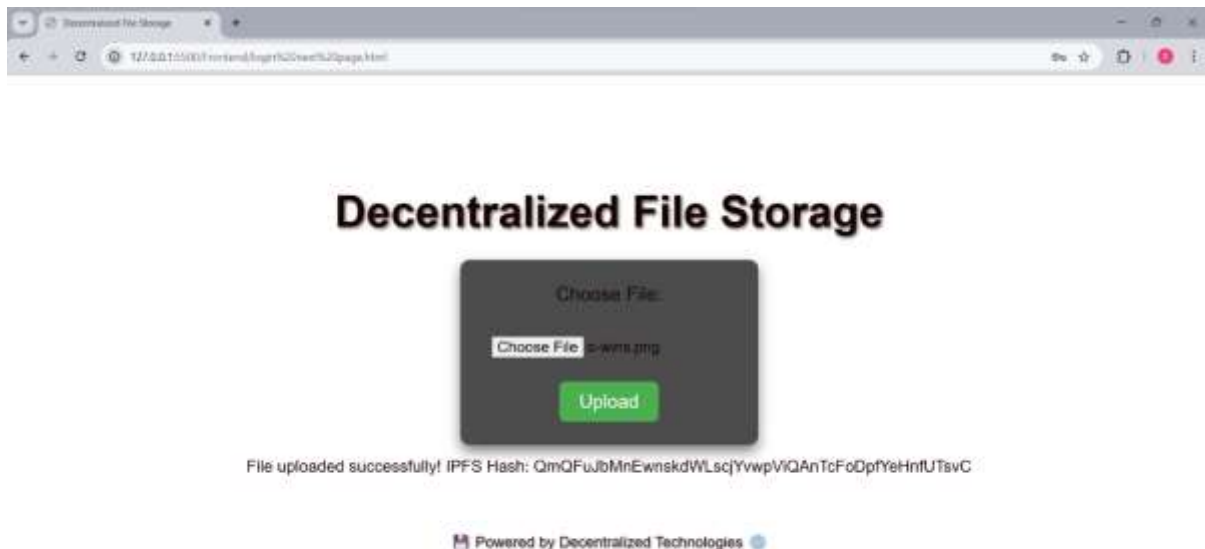


FIG 5.3 File Upload Success Interface - Decentralized File Storage

This interface represents the confirmation screen displayed upon successfully uploading a file to the decentralized storage system. It provides feedback and critical information about the uploaded file.

1. **Purpose:**

The interface confirms the successful completion of a file upload and provides the IPFS hash, which serves as a unique identifier for the file on the decentralized storage network.

2. **Design Elements:**

- **File Confirmation:** The interface clearly displays the filename (o-wins.png) that was successfully uploaded.
- **IPFS Hash Display:** The generated IPFS hash is prominently shown, enabling users to access the file on the decentralized storage system.
- **Upload Feedback:** A success message (File uploaded successfully!) reassures users that the process was completed without errors.

3. **User Experience:**

- **Informative Feedback:** Users are provided with actionable information, including the IPFS hash for future file retrieval.
- **Consistency:** The design maintains consistency with the initial upload screen, ensuring a seamless user experience.
- **Visual Simplicity:** The minimalistic design keeps the focus on the success message and file information.

4. **Functionality:**

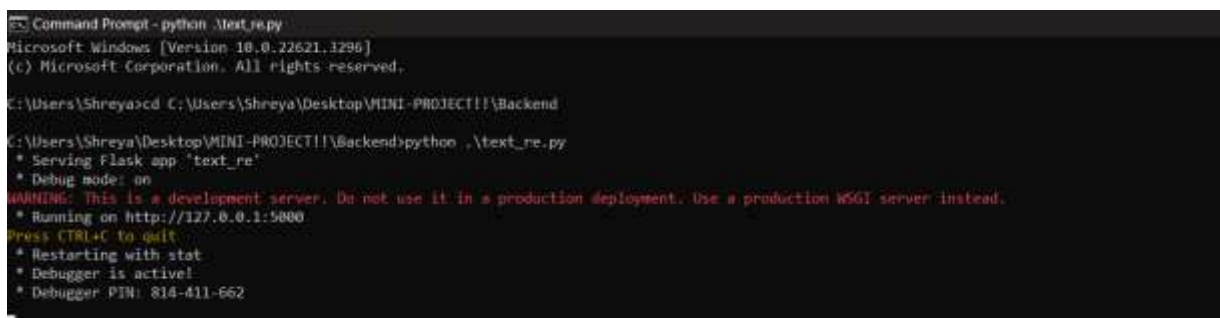
- **IPFS Integration:** The IPFS hash is generated upon successful upload, linking the file to the decentralized network.
- **Acknowledgment:** Users are immediately informed of the upload's success, reducing ambiguity and enhancing reliability.

5. Key Features:

- **Decentralized File Retrieval:** The displayed IPFS hash allows users to locate and access the file on the network.
- **Secure Storage:** By utilizing a decentralized system, the uploaded file benefits from improved security and accessibility.
- **Transparent Process:** Immediate feedback and relevant details ensure user trust and system transparency.

This interface is crucial for confirming upload success and equipping users with the necessary information to utilize the decentralized storage platform effectively. It emphasizes the system's reliability and user-focused design.

5.1.3 File Retrieval Interface – Decentralized File Access



```
Command Prompt - python .\text_re.py
Microsoft Windows [Version 10.0.22621.1295]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Shreya>cd C:\Users\Shreya\Desktop\MINI-PROJECT1\Backend
C:\Users\Shreya\Desktop\MINI-PROJECT1\Backend>python .\text_re.py
 * Serving Flask app "text_re"
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 814-411-662
```

FIG 5.4 File Retrieval Interface – Decentralized File Access

The file retrieval interface is a vital component of the Decentralized File Storage System, enabling users to access files stored on the IPFS network using their unique identifiers.

1. Purpose:

The interface allows users to retrieve stored files efficiently and securely using the IPFS hash provided during the upload process.

2. Design Elements:

- **Input Field for IPFS Hash:** A text box is provided for users to input the IPFS hash to locate the file on the decentralized storage.
- **Call-to-Action Button:** A “Retrieve File” button triggers the retrieval process.
- **Result Display Area:** Displays the retrieved file or an appropriate error message if the retrieval fails.
- **Clear Instructions:** Guidance is provided to help users input the correct hash format.

3. User Experience:

- **Simple Navigation:** The interface minimizes complexity, focusing on ease of use.
- **Feedback Mechanism:** Real-time prompts such as “File retrieved successfully!” or “Invalid IPFS hash entered” ensure clarity.
- **Mobile Compatibility:** The design is responsive and works seamlessly across devices.

4. Functionality:

- **Hash Validation:** The system validates the input to ensure the format matches IPFS hash standards.
- **File Display or Download:** Retrieved files are either displayed directly on the screen (for supported file types) or offered for download.
- **Error Handling:** If a file cannot be located, users receive a detailed error message for troubleshooting.

5. Key Features:

- **Decentralized Retrieval:** Uses IPFS to fetch the file, ensuring the process remains within the decentralized framework.
- **Enhanced Security:** Only users with the correct IPFS hash can access the file, maintaining data privacy.
- **Reliable Feedback:** Immediate and accurate responses build user trust.

This interface underscores the project's goal of secure and efficient file management. By offering clear instructions, robust functionality, and user-focused design, it ensures smooth file retrieval while upholding the principles of decentralization and security.

5.1.4 MongoDB Compass Interface

The MongoDB Compass interface is a graphical tool used to manage the database that supports the Decentralized File Storage System. This section describes its design, functionality, and significance in the project.

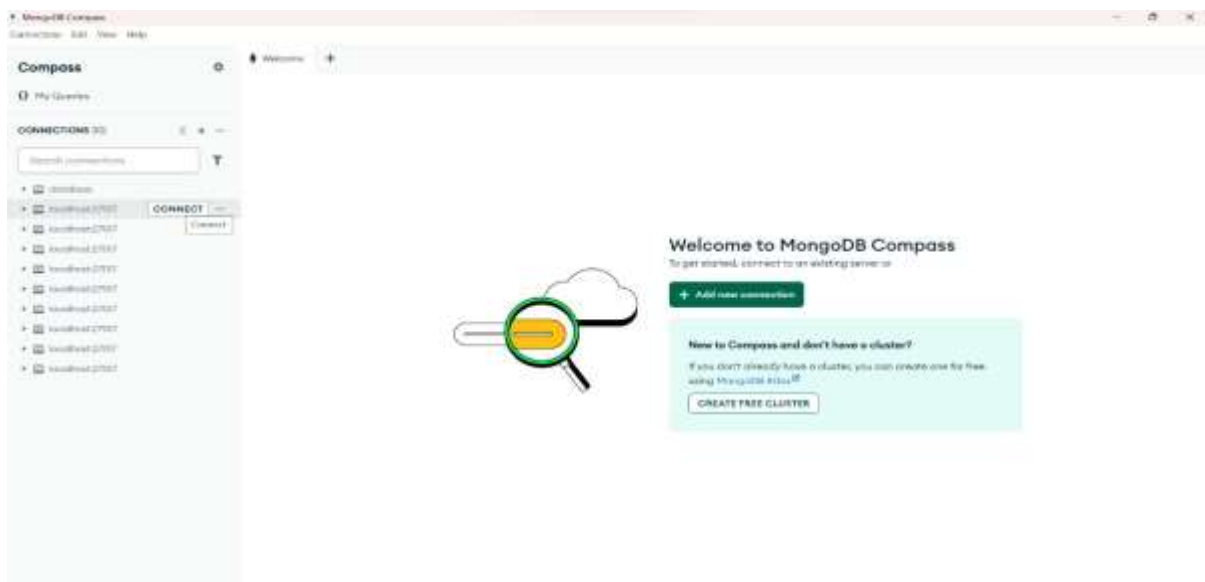


Fig 5.5 MongoDB Compass: Database Connection Interface

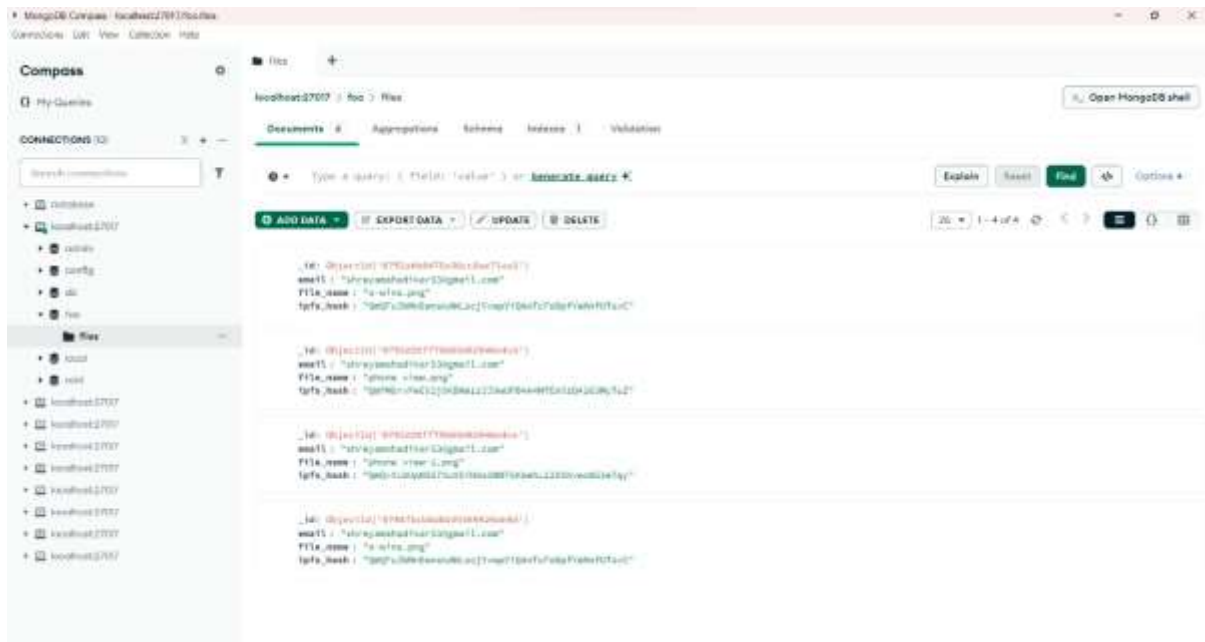


FIG 5.6 MongoDB Compass: Connecting to Local Server

1. Purpose:

The interface enables developers and administrators to interact with the MongoDB database, which stores essential data, including user credentials and file metadata. It simplifies database management, facilitating seamless navigation and monitoring.

2. Design Elements:

- **Connection Panel:** Displays all available connections to MongoDB servers (e.g., localhost:27017), allowing quick access to the database.
- **Welcome Screen:** Features options for creating a new connection or a free cluster using MongoDB Atlas.
- **Database Explorer:** Enables users to browse and manage databases and collections.

3. User Experience:

- **Ease of Use:** The intuitive design makes it accessible to developers of all experience levels.
- **Visual Overview:** Provides clear insights into server status and database structure.
- **Flexibility:** Supports local and cloud-based database instances for diverse use cases.

4. Functionality:

- **Database Management:** Connects to MongoDB servers for managing databases and collections, such as users and files.
- **Query Execution:** Allows interactive querying and testing of database operations.
- **Cluster Information:** Displays server details, aiding in monitoring and debugging.

5. Key Features:

- **Connection Management:** Supports multiple server connections for enhanced scalability.

- **Interactive Querying:** Simplifies data retrieval and modification with real-time feedback.
- **Data Visualization:** Offers a user-friendly graphical interface for navigating and inspecting database contents.

This tool was instrumental in ensuring the integrity and efficiency of the project's database management. By integrating MongoDB Compass, the project demonstrates the importance of accessible and reliable database operations in a decentralized system.

5.1.5 MongoDB Integration Interface

This interface demonstrates the connection between the web application and the MongoDB database, ensuring that data related to file uploads, user accounts, and retrieval processes are securely stored and managed.

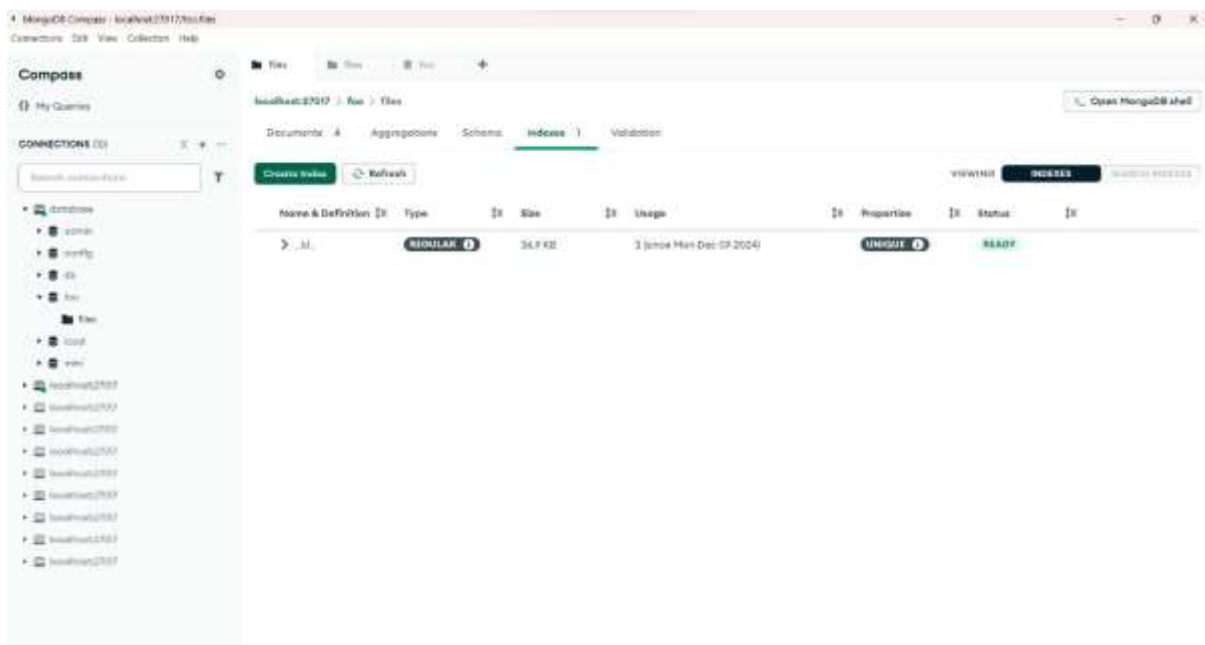


FIG 5.7 MongoDB Database Connection and Management Interface

1. Purpose:

The MongoDB Integration Interface is designed to manage the storage and retrieval of data related to users, uploaded files, and metadata. This interaction ensures that the decentralized file storage system can effectively communicate with the database for managing account details and file information.

2. Design Elements:

- **Connection Status:** Displays the connection status to the MongoDB server, indicating whether the system is successfully connected or facing any connection issues.
- **Database Interaction:** Enables seamless interaction between the application and MongoDB, where data, such as user credentials, file metadata, and IPFS hashes, are stored.
- **Actionable Feedback:** Provides real-time feedback about the database interaction, such as successful queries or error messages for debugging.

3. User Experience:

- **Reliable Data Handling:** Users interact with a reliable backend that handles requests for data storage and retrieval efficiently. This enhances overall system trust and reliability.
- **Minimalist Design:** The interface keeps the focus on the core functionality of managing database connections and ensuring smooth integration with the decentralized file system.

4. Functionality:

- **Database Connection:** The system connects to the MongoDB instance, confirming that the data is securely stored in a database for future retrieval and management.
- **Efficient Query Handling:** MongoDB Compass provides a GUI to monitor queries and responses, ensuring the application interacts with the database as expected.
- **Real-Time Feedback:** Prompts alert users of successful data entries, such as when a new user account is created or a file is uploaded and linked with its IPFS hash.

5. Key Features:

- **MongoDB Compass Integration:** This integration allows the application to manage and query the database easily using MongoDB Compass, a powerful tool for visualizing and manipulating MongoDB data.
- **Data Management and Retrieval:** The system ensures that uploaded files and associated metadata (IPFS hash, filenames) are stored securely, and users can retrieve the required files by querying the MongoDB database.
- **Enhanced Security:** MongoDB's secure database management features ensure that user data is protected and files are associated with proper access control.

This interface plays a crucial role in ensuring that the Decentralized File Storage System integrates smoothly with MongoDB, handling both user and file data in a secure, efficient manner while offering transparent real-time feedback to users.

5.1.6 Pinata Cloud File Management Interface



FIG 5.8 Pinata Cloud File Management Interface

This image showcases the **Pinata Cloud File Management Interface**, a platform that integrates with IPFS (InterPlanetary File System) for managing decentralized file storage. Key features displayed in the interface include:

1. **File List:** A detailed list of uploaded files, showing file names, unique CIDs (Content Identifiers), and their respective creation dates.
2. **Search and Filter Options:** Allows users to search files by name or CID and filter by creation dates for efficient file management.
3. **Navigation Panel:** Includes sections such as Files, Groups, IPFS Files, Access Controls, and Toolbox options like Analytics and Gateways.
4. **Add New Files:** The interface features a prominent “+ Add” button for uploading new files to the IPFS.
5. **User-Friendly Design:** Offers a clean and organized layout, ensuring ease of navigation and efficient management of decentralized file storage. This interface is essential for users to manage their decentralized storage efficiently, providing transparency and reliability.

5.1.6 Unpin File Action on Pinata Cloud

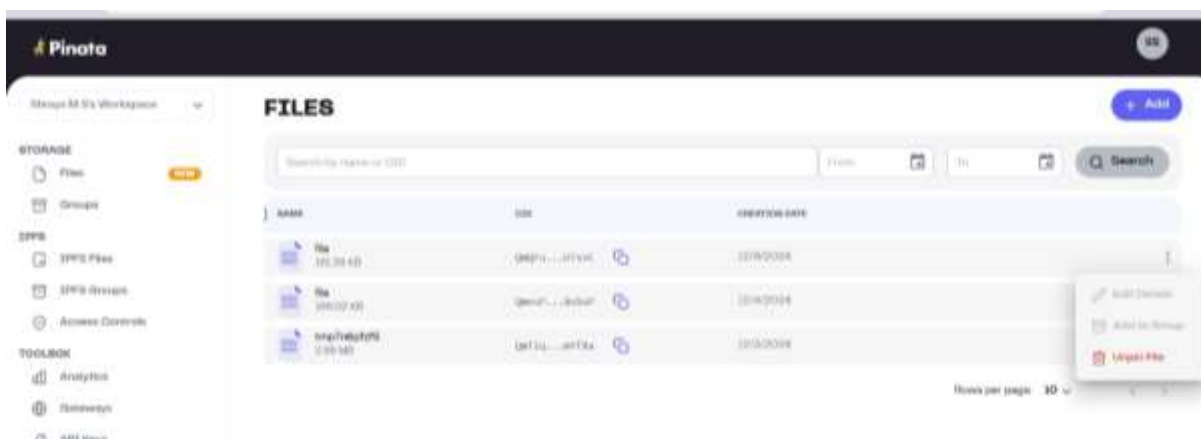


FIG 5.9 Unpin File Action on Pinata Cloud

This section of the Pinata Cloud dashboard demonstrates the "Unpin File" action within the file management interface. The dropdown menu next to each file reveals various options, including **"Unpin File"**, which allows users to remove a specific file from the Pinata Cloud's pinned storage, effectively freeing up space or managing outdated files. The selected file remains listed with its CID (Content Identifier), name, and creation date, ensuring users can confirm details before unpinning. This feature aids in efficient file lifecycle management within the IPFS-based storage system.

5.1.7 Pinata IPFS Files Management Interface

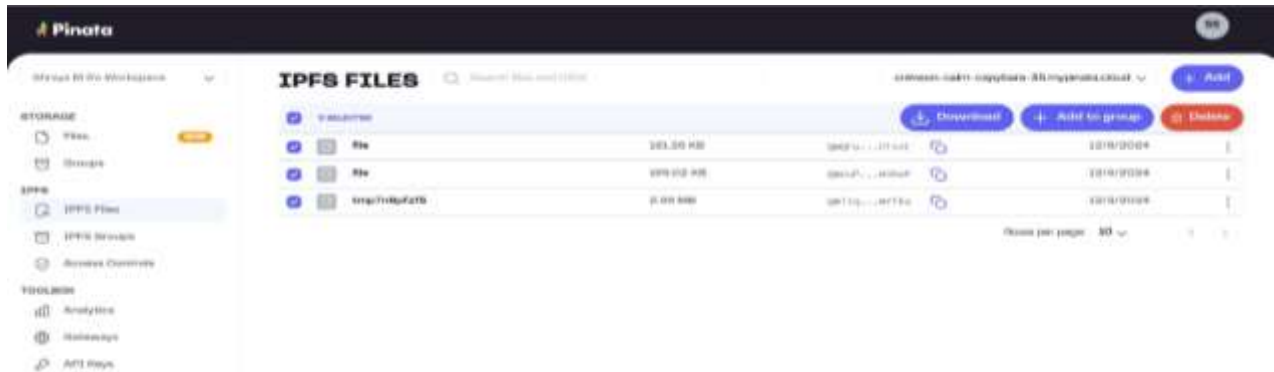


FIG 5.10 Pinata IPFS Files Management Interface

This screen displays the IPFS Files Management Interface in Pinata Cloud. It shows a list of files pinned to IPFS (InterPlanetary File System) within the workspace. The table includes file details such as Name, Size, CID (Content Identifier), and Creation Date. Users have selected three files, enabling the action buttons at the top:

- **Download:** Retrieve the files locally.
- **Add to Group:** Organize files into a group for better management.
- **Delete:** Remove the selected files from Pinata's pinned storage.

This interface is designed for efficient file handling and IPFS-based decentralized storage operations.

5.1.8 Server Log Entry for File Upload

```
Command Prompt - python .\text_re.py
Microsoft Windows [Version 10.0.22621.3296]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Shreya>cd C:\Users\Shreya\Desktop\MINI-PROJECT1\Backend
C:\Users\Shreya\Desktop\MINI-PROJECT1\Backend>python .\text_re.py
* Serving Flask app 'text_re'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 814-411-662
127.0.0.1 - - [09/Dec/2024 19:46:43] "POST /upload HTTP/1.1" 200 -
```

FIG 5.11 Server Log Entry for File Upload

This output is a server log entry indicating the successful handling of a file upload request. The log shows:

- **IP Address (127.0.0.1):** The source of the request, which is the localhost in this case.
- **Date and Time:** The timestamp (09/Dec/2024 19:46:43) specifies when the request was received and processed.
- **HTTP Method and Endpoint:** The POST /upload HTTP/1.1 indicates that a POST request was made to the /upload endpoint.
- **HTTP Response Code (200):** A status code indicating the request was successfully processed by the server.

This log entry is essential for tracking server activity, debugging, and monitoring the system's functionality during file uploads.

Effectiveness of the System

The decentralized file storage system effectively achieves its intended objectives by providing a seamless and secure platform for file uploads and retrievals. Key aspects of the system's effectiveness include:

1. **Reliability:** The system consistently handles file uploads, ensuring files are stored and accessible via IPFS hashes without data corruption.
2. **User-Friendliness:** The frontend interface is intuitive and easy to navigate, allowing users to interact with the system effortlessly. Features like real-time upload feedback enhance the user experience.
3. **Data Security:** Files are stored on a decentralized network (IPFS), ensuring that data remains immutable and resistant to unauthorized access or tampering.
4. **Performance:** The system demonstrates quick response times during file uploads and retrievals, which is crucial for maintaining user satisfaction.
5. **Scalability:** By leveraging IPFS, the system can handle an increasing number of files without significant performance degradation, making it suitable for larger-scale deployments.
6. **Integration:** The system integrates well with backend services, ensuring that file uploads, metadata handling, and retrieval processes are seamless and reliable.

This effectiveness makes the system an excellent choice for secure and decentralized file management, meeting both technical and user-centric objectives.

Challenges Encountered

During the development and implementation of the decentralized file storage system, several challenges were encountered:

1. **IPFS Integration:** Implementing IPFS for decentralized file storage required a thorough understanding of its architecture and APIs. Ensuring smooth integration with the backend and correctly managing IPFS hash generation posed initial difficulties.

2. **File Upload Efficiency:**
Handling large file uploads efficiently while maintaining responsiveness was a challenge, especially in terms of managing network delays and ensuring real-time feedback for the user.
3. **Frontend-Backend Communication:**
Establishing a seamless connection between the frontend and backend services required meticulous debugging and testing to address issues like CORS policies and inconsistent API responses.
4. **Error Handling:**
Ensuring robust error handling to provide meaningful feedback to users in cases of failed uploads or server errors was a significant task that required extensive testing.
5. **User Authentication and Session Management:**
Implementing secure user authentication mechanisms and maintaining session integrity during file uploads was critical to safeguarding user data. This required additional security measures such as encrypted communication and validation.
6. **Decentralized Data Retrieval:**
Retrieving files from IPFS nodes and ensuring availability posed challenges, particularly in scenarios where nodes were offline or lacked sufficient replication of stored files.
7. **UI Design Adaptation:**
Designing an intuitive user interface that could adapt to different screen sizes and provide real-time updates during the upload process required multiple iterations and usability testing.
8. **Scalability:**
Anticipating future scalability demands and designing the system to handle increasing numbers of users and files without performance degradation required careful architecture planning.

Limitations of the Current System

- ❖ **Storage Dependency:**
 - The system relies on Pinata as an intermediary for IPFS file pinning, creating dependency on a third-party service.
 - Free-tier limitations restrict the number of files and their sizes.
- ❖ **File Privacy:**
 - Files are publicly accessible via IPFS, making it unsuitable for sensitive data without encryption.
- ❖ **Lack of Advanced Features:**
 - The current system does not support file sharing, role-based access control, or collaborative file management.
 - There is no search functionality for filtering files by metadata.
- ❖ **Network Latency:**
 - Retrieval times for files can be inconsistent due to IPFS node availability and network performance.
- ❖ **No Mobile Application:**
 - The lack of a mobile application limits accessibility for users who prefer on-the-go file management.

Chapter 6

Conclusion and Future Enhancements

6.1 Conclusion

- The decentralized file storage system successfully achieves its objective of providing a reliable, secure, and user-friendly platform for storing and sharing files. By leveraging blockchain-based IPFS technology, the system ensures immutability and accessibility of uploaded files, while MongoDB serves as a reliable backend database for managing user and file metadata. The implementation includes robust user authentication, seamless file uploads, retrievals, and deletion features. Comprehensive testing validates the system's functionality and stability.
- This project demonstrates the potential of decentralized technologies in solving real-world problems such as secure file storage, data ownership, and minimizing dependency on centralized storage systems. It not only meets the project goals but also provides a foundation for future improvements and scalability.

6.2 Future Enhancements

While the project meets its current requirements, there are several opportunities for improvement and expansion, such as:

1. **Role-Based Access Control (RBAC):**
Implement user roles (e.g., admin, regular users) to provide additional control over file access and system settings.
2. **Enhanced Security:**
 - Add two-factor authentication (2FA) for better user account security.
 - Encrypt files before uploading to IPFS to ensure data privacy.
3. **File Preview:**
Enable users to preview certain file types (e.g., images, PDFs) directly from the application.
4. **Versioning:**
Introduce file versioning to allow users to upload and retrieve different versions of the same file.
5. **Search and Filters:**
Enhance the file retrieval interface with search and filtering capabilities based on file names, upload dates, or tags.
6. **Scalability and Performance Optimization:**
Explore more scalable database solutions for managing large numbers of users and files, and optimize API response times.
7. **Mobile Application:**
Develop a mobile application to complement the web-based interface, allowing users to manage files on the go.

8. **Decentralized Authentication:**

Integrate decentralized authentication methods (e.g., using blockchain wallets) to improve user privacy and ownership.

9. **Collaboration Features:**

Add functionality for users to share files with other registered users, incorporating permission levels (e.g., view-only, edit).

10. **Notifications and Alerts:**

Notify users about file upload completion, shared files, or approaching storage limits via email or app notifications.

11. **Integration with Other Decentralized Systems:**

Explore integration with decentralized storage alternatives like Arweave or Filecoin for additional redundancy and scalability.

REFERENCES

Citation Format:

- **Online Resources:** Authors : Prof. P. B. Patil, Ankush R. Hujare, Karan S. Desai, Vaibhav B. Devkar, Hrishikesh I. Bhadgaonkar Published (First Online): 28-10-2023. <https://www.ijert.org/decentralized-file-storage-system-using-blockchain> Retrieved from URL

Software Tools:

- Pinata. (n.d.). *Pinata - The Home of NFT Media*. Retrieved from <https://www.pinata.cloud>
- Flask. (n.d.). *Flask - Python Microframework*. Retrieved from <https://flask.palletsprojects.com>
- IPFS. (n.d.). *InterPlanetary File System*. Retrieved from <https://ipfs.tech>
- MongoDB. (2024). MongoDB: The Developer Data Platform. Retrieved from <MongoDBCompass.lnk>