Software Design Specification

for

CLOUDPI

CLOUD BASED STORAGE SYSTEM USING RASPBERRY PI

Version 1.0

Prepared by

A Anas
Aswathy Krishnan
Jayalekshmi K S
Nandakumar

Team- CloudPi

27 March 2023

Table of Contents

Table of Contents **ii**
Revision History ii
Table of Contents

1. Introduction **4**
1.1 Purpose 4
1.2 Scope 4
1.3 Definitions, Acronyms, and Abbreviations 4
1.4 References 4
1.5 Overview 4

2. System Architecture Design
2.1 Description 4
2.2 System architecture 5

3. Application architectural design **5**

4. GUI design **6**
4.1 User Interface Design 6
4.2 Visual Design 10
4.3 Navigation Design 10

5. API Design **10**

6. Technological stack **10**

7. Component Design **12**
5.1 Component Description 12
5.2 Component Interface 12
5.3 Component Implementation 13

8. Algorithm Design **13**

9. Data Design **15**
9.1 Data Description 15
9.2 Data Dictionary 15
9.3 Data Storage 16
9.4 Data Schema 16
9.5 Data Access 17

10. Error Handling Design **17**

11. Performance Design **18**

12. Security Design **19**

13. Testing Design **19**
6.1 Testing Approach 19
6.2 Testing Scope 20
6.3 Test Cases 20
6.4 Testing Tools 21

14. Maintenance Design **22**

15. Deployment Design **23**

Appendix A: Record of Changes 35 **24**

Appendix B: Acronyms 36 **25**

Appendix C: Referenced Documents 26

1. Introduction

2. Purpose

The purpose of this document is to provide a detailed design for the Cloud Pi system, outlining its architecture, features, data design, component design, deployment design, and operational and support issues.

3. Scope

This document is intended for developers and stakeholders of the Cloud Pi system, providing a comprehensive design for its implementation and maintenance.

4. Definitions, Acronyms, and Abbreviations

API: Application Programming Interface
Django: A high-level Python web framework
Raspberry Pi: A series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation
SDD: Software Design Document

1.4 References

https://www.ljnrd.org/papers/IJNRDA001020.pdf
Private Cloud using Raspberry Pi by Bhavani.G, Akshaye J, Christ Annson M, Cyril j wilson

1.5 Overview

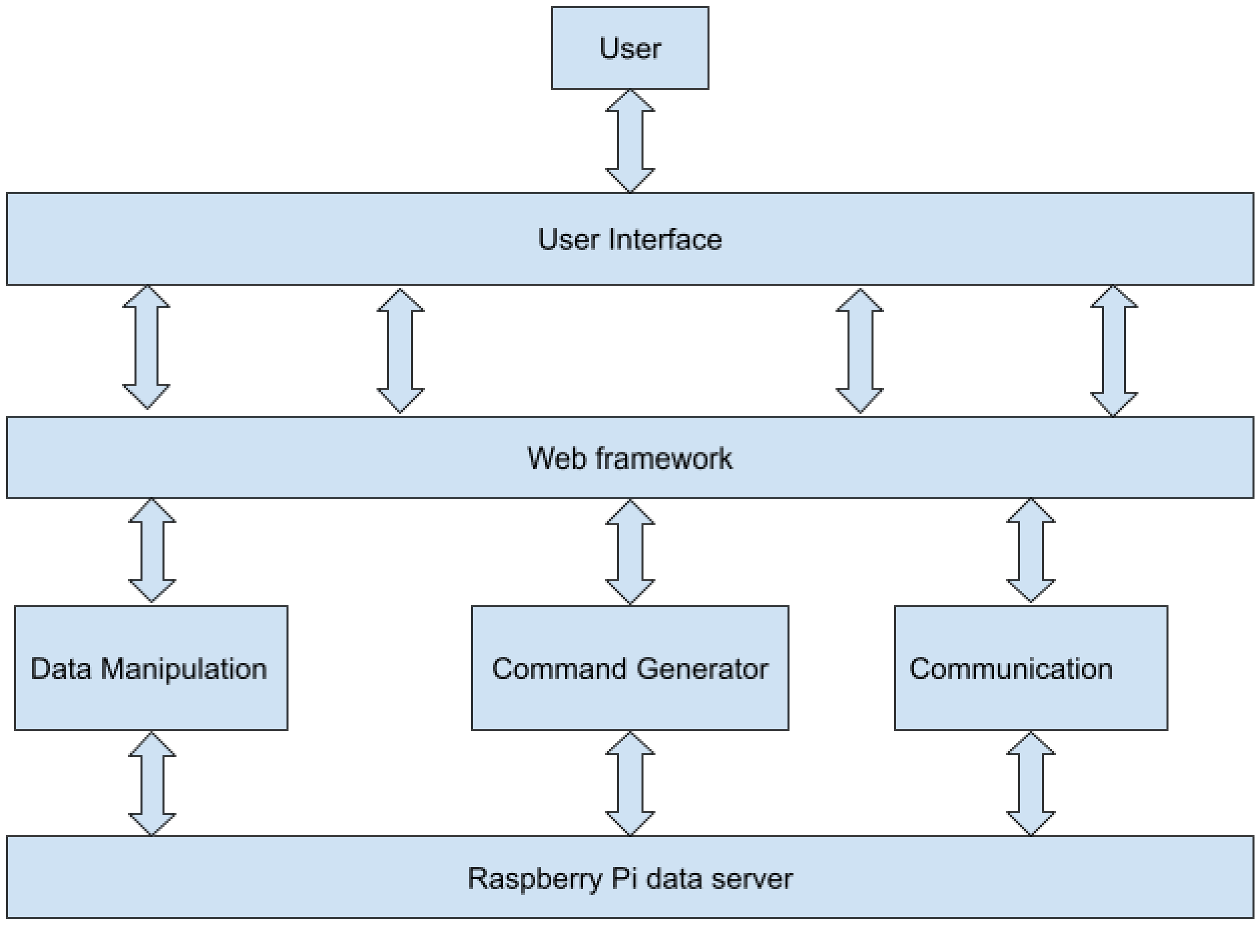
The Cloud Pi project aims to create a cloud-based storage solution using a Raspberry Pi server and a web-based user interface. The user interface will be built using Django web framework and will allow users to access and manage their files stored on the Raspberry Pi server.

2. System Architecture Design

2.1 Description

The system architecture design for a cloud-based storage platform using Raspberry Pi should include details on the hardware components, operating system, cloud storage provider, file management, and other key components necessary for the system to function properly. This document should provide a high-level overview of how the system is designed to work, including how the different components interact with each other to provide the desired functionality. It should also include any security considerations or other important design decisions that were made during the system architecture planning phase.

2.2 System Architecture



3. Application Architecture Design

1. Frontend: The frontend of the application is responsible for presenting the user interface to users and handling user interactions. It will typically be built using web technologies such as HTML, CSS, and JavaScript, and will communicate with the backend using a REST API.

2. Backend: The backend of the application is responsible for processing user requests, managing files and data, and communicating with the Raspberry Pi and cloud storage provider. It will typically be built using a server-side programming language such as Python, and will use a web framework such as Flask or Django to handle incoming requests.

3. Database: The database is responsible for storing user data and file metadata, such as file names, sizes, and creation dates. It will typically be a relational database such as MySQL and will be accessed by the backend through an ORM (Object-Relational Mapping) library.

4. API: The REST API is responsible for allowing the frontend to communicate with the backend and perform CRUD (Create, Read, Update, Delete) operations on files and data. It will typically be built using a web framework such as Django, and will provide endpoints for uploading, downloading, and managing files on the Raspberry Pi and cloud storage provider.

5. Authentication and Authorization: The application will need to implement authentication and authorization to ensure that only authorized users can access their files and data. This may involve using a third-party authentication service such as OAuth or building a custom authentication system.

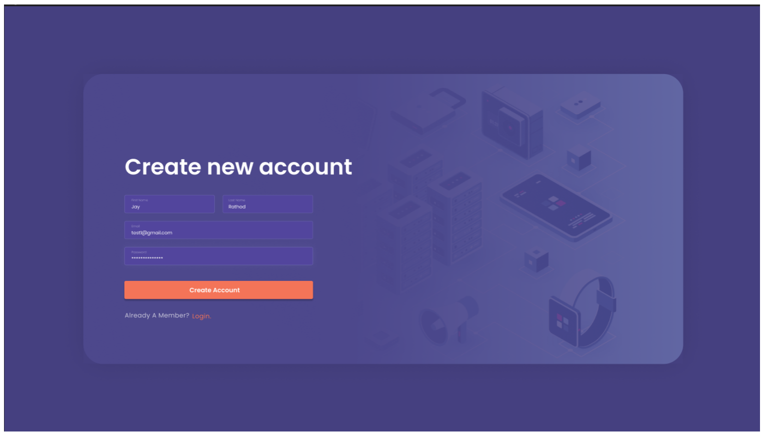
6. Security: The application will need to implement security measures such as encryption and secure communication protocols to protect user data and prevent unauthorized access.

7. Scalability: The application should be designed to be scalable, so that it can handle a large number of users and files. This may involve using a load balancer and multiple instances of the backend and database.

4. GUI Design

User Interface design

Create an account page


Create new account

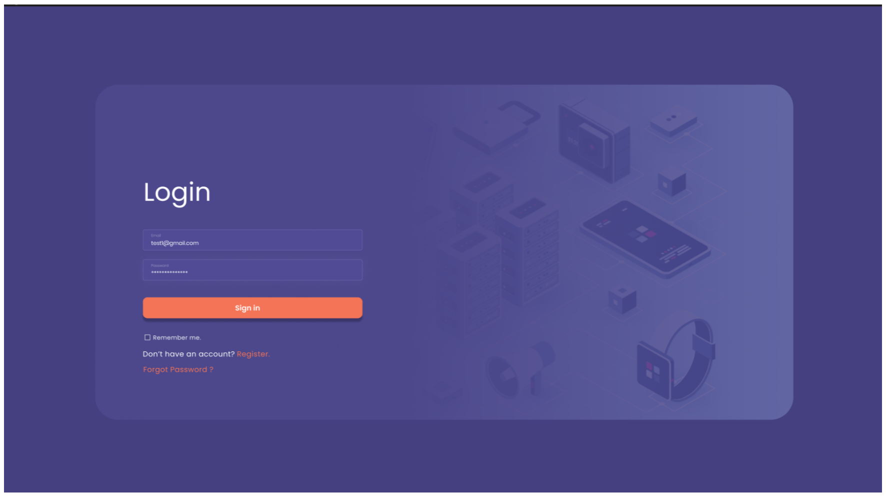
First Name: Jay

Last Name: Rathod

Email: test1@gmail.com

Password: ••••••••••••••

Login page



Login

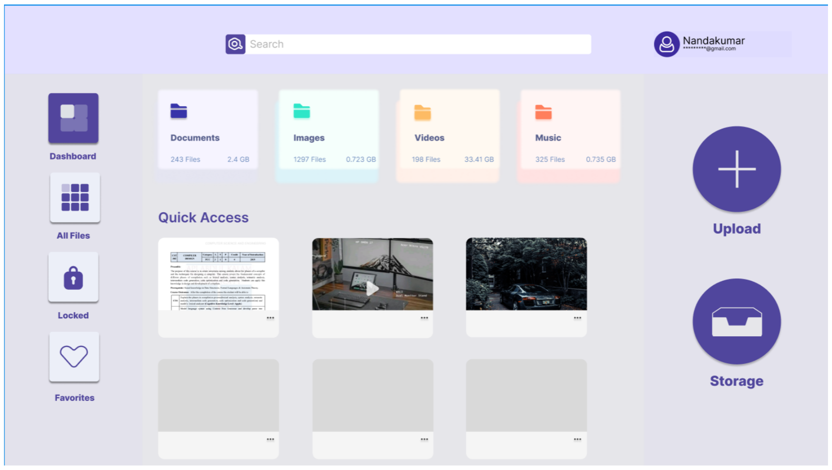
Email
test1@gmail.com

Password
****

Dashboard

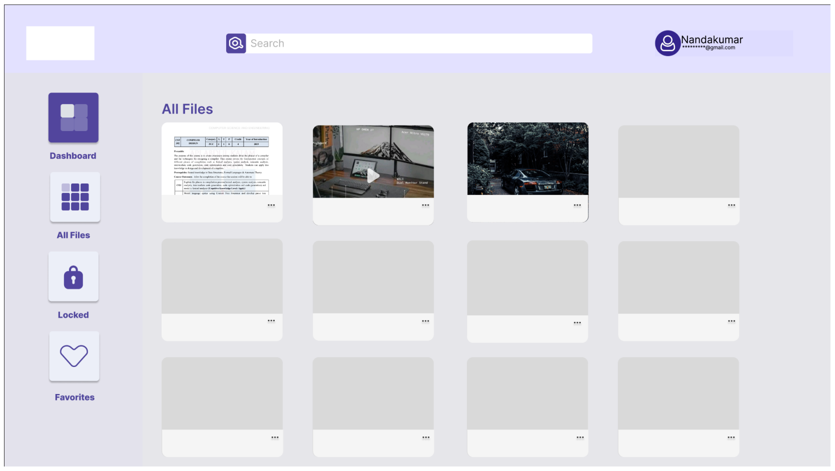
Dashboard consist of 5 buttons

- All files
- Locked
- Favorites
- Upload
- Storage



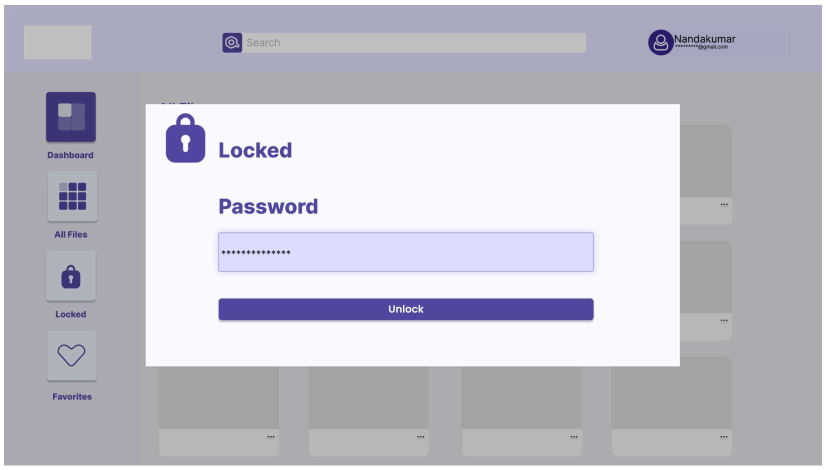
All Files

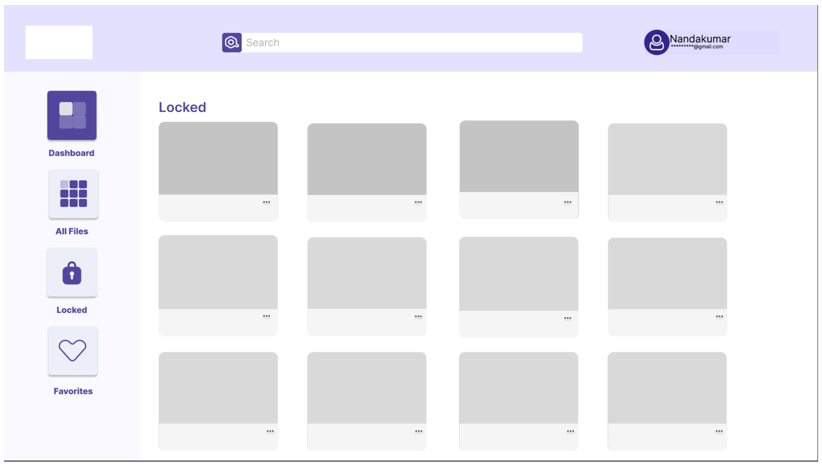
All files page includes the files that are stored without being locked.



Locked

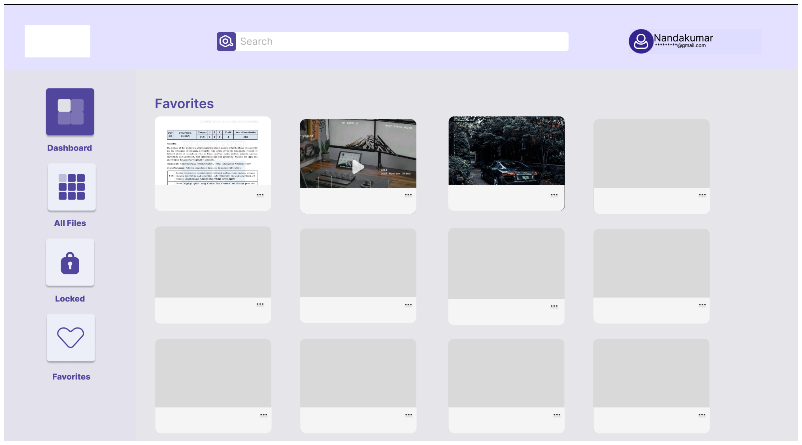
Locked displays an overlay page that asks for an encrypted password that when entered unlocks all the locked files





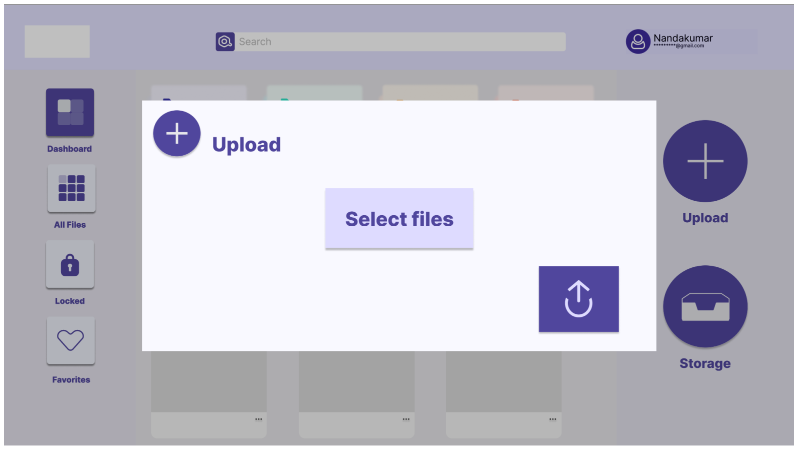
Favorites

Favorites displays the files that are added as favorites.



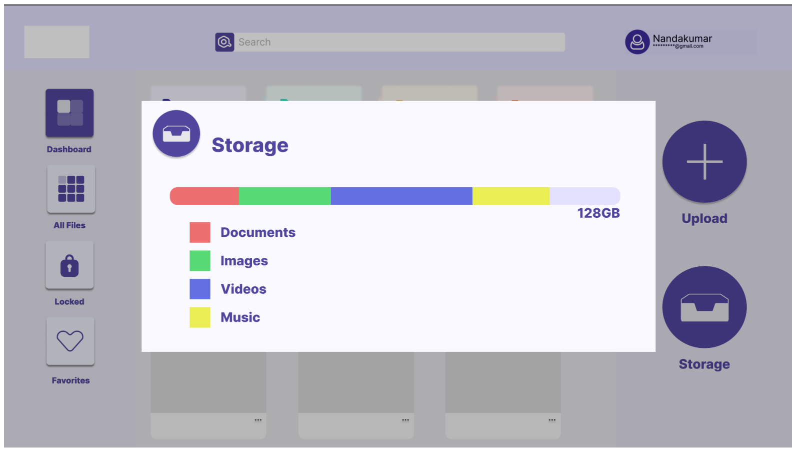
Upload

Upload shows a pop-up window that asks to select files



Storage

Storage page shows the allocation of each type of files



Visual design

- The GUI design will use high-contrast colors and larger font sizes for better visibility and readability.
- Icons and buttons will have descriptive labels in addition to their visual representation

Navigation Design

- The application will use a consistent navigation structure throughout.
- Navigation will be designed in such a way that each page can navigate to every other page.

5. API Design

REST API :

A REST API can be used in the cloud-based storage platform project to allow users to interact with the system by uploading, downloading, and managing files on the Raspberry Pi. This API provides a standardized and easy-to-use interface for users to access the system from a variety of applications and devices.

6. Technological Stacks

Front-end

The front-end of the cloud-based storage platform will be built using the following technologies:

ReactJS : A JavaScript library for building user interfaces.

HTML/CSS : Markup language and styling for the web application.

Back-end

The back-end of the cloud-based storage platform will be built using the following technologies:

Django : A Python-based web framework for building back-end components of web applications.
Django REST framework : A powerful and flexible toolkit for building APIs with Django.
SQLite : A lightweight relational database management system for storing user account data and metadata for uploaded files.

Libraries and Tools

The following libraries and tools will be used to enhance the functionality of the platform:

Django-cors-headers : A Django middleware that allows cross-origin resource sharing (CORS) for client-side web applications.
Django-storages : A Django library that provides support for storing files in remote storage services.

Hosting and Deployment

The cloud-based storage platform will be hosted on the following platforms:

Apache web server : A widely-used open source web server software that is known for its flexibility, stability, and security

Security

The following security measures will be implemented to ensure the security of the cloud-based storage platform:

HTTPS : All web traffic will be encrypted using HTTPS to protect against eavesdropping and tampering.
Access control : User authentication and authorization will be implemented using Django's built-in authentication system.
Encryption : Files uploaded to the platform will be encrypted at rest using server-side encryption with Apache default encryption.

Raspberry Pi

The Raspberry Pi will be used as a hardware component of the cloud-based storage platform. It will be running a local instance of the web application and will be responsible for managing file uploads and downloads from users on the local network. The Raspberry Pi will be configured with the following software:

Raspbian : A Debian-based operating system for the Raspberry Pi.
Apache : A popular web server software for serving web pages and applications.
Python : The Python programming language will be used to write the code for managing file uploads and downloads on the Raspberry Pi.

The Raspberry Pi will be connected to the Apache web server to sync user data and files to the cloud-based storage platform.

7. Component Design

7.1 Component Description :

- Raspberry Pi: This component is the heart of the project, responsible for running the Django web application..
- Django Web Framework: This open-source framework is used to develop the web application that serves as the user interface for the project.
- Python Programming Language: Python is used as the primary programming language for developing the project, including the Django application and sensor interface.
- HTML/CSS/JavaScript: These are the standard web development technologies used to create the web interface for the project.
- Linux Operating System: The Raspberry Pi runs on the Raspbian OS, which is a Linux-based operating system.
- Wi-Fi Module/Ethernet: The Wi-Fi module or Ethernet is used to connect the Raspberry Pi to the internet, enabling users to access the web interface from any device with an internet connection.
- Power Supply: The Raspberry Pi requires a power supply to operate, which can be provided through a USB cable connected to a power adapter.
- USB Storage Device: A USB storage device can be used to store the data collected by the project, providing an additional backup option.

7.2 Component Interface:

- Interface between the Django web application and the Raspberry Pi: This interface includes the communication protocols used for sending and receiving data between the web application and the Raspberry Pi.
- Interface between the database and the Django application: This interface includes the methods used for accessing and modifying data stored in the database, as well as the database schema used for storing the data.
- Interface between the remote storage system and the Django application: This interface includes the methods used for uploading and downloading files from the remote storage system, as well as the protocols used for communication.

7.3 Component Implementation

- Django application: The Django application component is implemented using the Python programming language and the Django web framework. The application is developed using the Model-View-Controller (MVC) architectural pattern.

- File upload and storage: The file upload and storage component is implemented using the Python Django framework's built-in file handling features. When a file is uploaded by the user, the file is temporarily stored in a buffer and then written to the file system of the Raspberry Pi.

- Remote storage integration: The remote storage integration component is implemented using the Python Requests library. The library is used to send HTTP requests to the remote storage system and to retrieve files from the remote storage system.

- Database: The database component is implemented using the SQLite database management system. SQLite is a lightweight and fast database management system that is ideal for small-scale applications.

- User interface: The user interface component is implemented using HTML, CSS, and JavaScript. The user interface is designed using the Bootstrap framework, which is a popular front-end development framework.

8. Algorithm Design

File upload algorithm :

1. User selects file(s) to upload.
2. The front-end sends a request to the back-end to initiate file upload.
3. Back-end generates a unique identifier for the file and stores it in a local directory on the server.
4. Back-end sends the unique identifier to the front-end.
5. Front-end uses the unique identifier to upload the file(s) directly to the server's local directory.
6. After the file(s) upload is completed, front-end sends metadata to back-end for storage in a PostgreSQL database.

File download algorithm :

1. User requests to download a file.
2. Front-end sends a request to the back-end to check if user is authorized to download the file.
3. Back-end retrieves the metadata of the file from PostgreSQL to verify user authorization.
4. If user is authorized, back-end sends the unique identifier of the file to the front-end.
5. Front-end uses the unique identifier to download the file from the server's local directory.

Access control algorithm :

1. User logs in to the system.
2. Back-end verifies user credentials and generates an access token.
3. Front-end stores the access token in local storage.
4. For each subsequent request to the back-end, front-end sends the access token in the request header.
5. Back-end verifies the access token before serving the request.
6. Access token expires after a certain period of time, and the user is required to log in again to generate a new access token.

9. Data Design

9.1 Data Description:

- The project involves storing and managing various types of files such as images, videos, and text files.
- Each file is associated with metadata such as the file name, date of upload, and file size.
- The metadata is stored in a relational database using Django's built-in ORM.
- The database schema includes tables for file metadata, user information, and user permissions.
- The file data is stored on the Raspberry Pi's file system, organized in directories based on user and file type.
- The file system is accessed using Python's built-in file handling libraries.
- The application uses HTTP POST requests to upload files from the client-side to the server-side.
- The file upload process involves checking file size limits and file type restrictions before allowing the upload to proceed.
- Uploaded files are stored in a temporary buffer before being written to the file system to prevent data loss in case of server crashes or errors.
- When a file is requested by a user, the file's metadata is retrieved from the database and used to locate the file on the file system.
- The requested file is served to the user using Django's file handling mechanisms.

9.2 Data Dictionary:

Field Name	Data Type	Description
Id	Integer	Unique identifier for each file
file_name	CharField	Name of the file
file_size	integer	Size of the file in bytes
data_created	DataTime	Date and time when the file was uploaded
is_deleted	Boolean	Indicates whether the file has been deleted
is_private	Boolean	Indicates whether the file is private or public
file_path	CharField	Path to the file in the server

9.3 Data Storage :

- The application stores the uploaded files in a local directory on the Raspberry Pi.
- The file metadata including file name, size, and upload timestamp is stored in a SQLite database that is integrated with the Django application.
- The database schema consists of a single table with columns for file ID, file name, file size, upload timestamp, and file path.
- The file path column stores the absolute path to the location of the uploaded file on the Raspberry Pi.
- The SQLite database is stored on the Raspberry Pi's file system and is accessed through the Django ORM.
- The data stored in the SQLite database can be queried and displayed in the application's user interface.

9.4 Data Schema

USER

Username	Email id	password
Varchar Unique Primary key	Varchar Unique	Encrypted At least 8 characters

Data

SlNo	File Name	DateOfUpload	Recently Used Date	Type	Path	Encryption Status	Size(in mb)	Favorite
int	varchar Unique	date	date	varchar	varchar	int	int	boolean

9.5 Data Access

- Data retrieval methods: This includes the methods used to retrieve data from the database such as SQL queries, Django's Object Relational Mapping (ORM), or other data retrieval libraries.

- Data manipulation methods: This includes the methods used to manipulate data stored in the database such as adding, updating, or deleting records. The methods used to ensure data consistency and integrity should also be mentioned.

- Data security: This includes the security measures taken to protect data stored in the database. This includes user authentication and authorization, encryption of sensitive data, and regular data backups.

- Data backup and recovery: This includes the backup and recovery procedures in place to ensure that data is not lost in case of a system failure or other unforeseen events.
- Data concurrency: This includes the mechanisms used to manage data concurrency such as locking mechanisms or timestamp-based mechanisms to prevent data inconsistency issues when multiple users are accessing the same data simultaneously.
- Data archiving: This includes the procedures for archiving old or outdated data to reduce the size of the database and improve the performance of data retrieval and manipulation operations.
- Data versioning: This includes the procedures for tracking changes made to the data stored in the database over time. This can help to maintain a history of data changes and enable rollback to previous versions of the data if necessary.

10. Error Handling Design

- File upload error handling:
- Front-end:
  - If the file size is too large, display an error message to the user.
  - If the file type is not supported, display an error message to the user.
  - If the file upload fails, display an error message to the user.
- Back-end:
  - If the request to initiate file upload fails, return an appropriate error response to the front-end.
  - If the metadata upload fails, return an appropriate error response to the front-end.

- File download error handling:
- Front-end:
  - If the user is not authorized to download the file, display an error message to the user.
  - If the file download fails, display an error message to the user.
- Back-end:
  - If the user is not authorized to download the file, return an appropriate error response to the front-end.
  - If the pre-signed URL generation fails, return an appropriate error response to the front-end.

- Access control error handling:
- **Front-end:**
  - If the access token is invalid or expired, redirect the user to the login page.
- **Back-end:**

- If the access token is invalid or expired, return an appropriate error response to the front-end.

11. Performance Design

- Caching:
- Implement caching to reduce the number of requests to the server and improve response times.
- Use client-side caching for frequently accessed static files such as images and stylesheets.
- Use server-side caching for frequently accessed data such as user details.

- Load balancing:
- Use load balancing to distribute the workload across multiple servers to improve performance and prevent overload.
- Implement horizontal scaling by adding more servers to the server pool as the load increases.

- Database optimization:
- Optimize database queries to minimize response times.
- Use indexes to speed up query execution and improve performance.
- Consider using a NoSQL database to improve performance for certain types of data.

- Content Delivery Network (CDN):
- Use a CDN to distribute content globally and reduce the load on the server.
- Store static files such as images and videos on the CDN for faster access.

- Compression:
- Use compression to reduce the size of files that are sent from the server to the client, thus reducing the time required for data transfer.

- Image optimization:
- Optimize images to reduce their size and improve loading times.
- Use tools to compress images and serve them in the appropriate format.

12. Security Design

- Authentication and Authorization:

  All users will be required to create an account with a unique username and password to access the application. Passwords will be encrypted using industry-standard encryption algorithms to prevent unauthorized access to user accounts. Additionally, only authorized users will be allowed to perform sensitive actions within the application.
- Secure Communication:

  All communication between the client and server will be encrypted using HTTPS to protect user data from interception and tampering by third parties. This will be achieved by installing a valid SSL certificate on the server.
- Data Security:

  All user data will be stored in a secure database, with appropriate access controls in place to prevent unauthorized access. Access to the database will be restricted to authorized personnel only.

13. Testing Design

Testing Approach

The testing approach for Cloud Pi will be a combination of manual testing and automated testing. Manual testing will be used for functional and integration testing, while automated testing will be used for regression testing.

Testing Scope

The testing scope for Cloud Pi will include the following areas:

1. User authentication and authorization
2. File upload and download functionality
3. File management functionality (e.g. deletion, renaming)
4. Integration with Raspberry Pi
5. Performance and scalability

Test Cases

- User authentication and authorization
    1. Verify that users can create accounts and log in with valid credentials
    2. Verify that users cannot log in with invalid credentials
    3. Verify that only authorized users can access protected areas of the application
- File upload and download functionality
    1. Verify that users can upload files to the cloud storage
    2. Verify that users can download files from the cloud storage
    3. Verify that files are uploaded and downloaded correctly, without corruption or data loss
- File management functionality
    1. Verify that users can delete files from the cloud storage
    2. Verify that users can rename files in the cloud storage
    3. Verify that files are deleted and renamed correctly, without data loss or corruption
- Integration with Raspberry Pi
    1. Verify that the application can communicate with the Raspberry Pi
    2. Verify that files can be uploaded and downloaded to and from the Raspberry Pi
    3. Verify that the Raspberry Pi can be managed through the application
- Performance and scalability
    1. Verify that the application can handle a large number of simultaneous users
    2. Verify that the application can handle large file uploads and downloads without performance degradation
    3. Verify that the application can scale to accommodate growth in user base and file storage requirements

Test Tools

The following test tools will be used for testing Cloud Pi:

- Selenium for automated testing of web interface
- JMeter for performance testing
- Python unittest framework for unit testing

Test Environment

The Cloud Pi application will be tested in the following environment:

1. Operating System: Ubuntu Linux 20.04 LTS
2. Web Browser: Google Chrome version 90
3. Python version 3.9.5
4. Django version 3.2.4

Test Data

The following test data will be used for testing Cloud Pi:

1. Test user accounts with varying levels of permissions
2. Test files of varying sizes and formats

Acceptance Criteria

The acceptance criteria for Cloud Pi are as follows:

1. All test cases pass without error
2. Performance testing meets specified requirements for maximum response time and throughput
3. The application can handle a large number of simultaneous users
4. The application can handle large file uploads and downloads without performance degradation
5. The application can scale to accommodate growth in user base and file storage requirements

14. Maintenance Design

Maintaining software involves fixing bugs, improving performance, adding new features, and ensuring compatibility with new hardware and software systems. Proper maintenance design ensures that software remains up-to-date, secure, and efficient, and it can greatly enhance the lifespan of software.

The following strategies will be implemented to ensure effective maintenance:

1. Regular software updates: The system should be updated regularly to address any security vulnerabilities, software bugs, and to introduce new features as required. It is important to plan and schedule these updates regularly, to ensure that the system remains up to date and secure.

2. User feedback: User feedback is an essential component of any software project. By gathering feedback from users, the development team can identify any issues or bugs with the system and prioritize them for future updates. A dedicated feedback mechanism should be provided to users to provide feedback to the development team.

3. Automated testing: Automated testing should be implemented in the system to catch bugs early on in the development process. This helps to minimize the need for expensive manual testing and allows the development team to identify and address issues quickly.

4. Scalability: The system should be designed with scalability in mind. This means that it should be able to handle a growing number of users and documents without compromising performance. As the user base and the number of documents grow, the system should be able to scale up accordingly.

5. Documentation: The system should be well-documented, with clear instructions for installation, configuration, and maintenance. This includes user manuals, developer documentation, and API documentation.

6. Regular backups: The system should be backed up regularly to ensure that user data is not lost in case of a system failure or error. Backups should be stored in secure, off-site locations to prevent data loss in case of a disaster.

7. Security: Security should be a top priority for the system. The development team should follow industry-standard security practices, such as encryption, multi-factor authentication, and access controls, to ensure that user data is secure.

8. Performance monitoring: The system should be monitored regularly for performance issues, such as slow response times, high server load, or database issues. Performance monitoring tools should be used to track system performance and identify any issues that need to be addressed.

9. Release management: Release management is a critical component of software maintenance. It involves managing the release process, including testing, documentation, and deployment. A well-defined release management process can help to ensure that new features are introduced smoothly and without disrupting the system's stability.

15. Deployment Design

The deployment design for our project involves making it available for our target audience through the use of a web application.

The web application will be deployed on a Raspberry Pi, which will act as a web server. The Raspberry Pi will run the Apache web server software, which will be responsible for handling incoming requests and serving web pages to users. The Apache server will be configured to run as a reverse proxy, which will allow it to forward incoming requests to the Django application running on the same Raspberry Pi.

To ensure optimal performance and user experience, the Django application will be regularly updated with bug fixes and new features. These updates will be pushed to the Raspberry Pi, and users will be able to access the latest version of the application by visiting the server's URL. The application will also include a built-in feedback mechanism to allow users to report bugs and suggest new features.

In terms of security, the application will employ industry-standard security practices to protect user data and prevent unauthorized access. This will include secure storage and transmission of user data, as well as strict access control measures for the application's backend services. The Raspberry Pi will be configured to use HTTPS encryption, which will ensure that all data transmitted between the server and users is secure.

To ensure scalability and availability, the application will be designed to work with a scalable cloud infrastructure in the future, if needed. This will allow the application to handle increasing numbers of users and requests without compromising performance or availability. Additionally, the Raspberry Pi's resources will be monitored regularly to ensure that it is operating optimally.

Finally, the deployment design will include a maintenance plan to ensure that the application continues to function as intended over time. This will involve ongoing monitoring of the application's performance and user feedback, as well as regular updates to ensure that the application remains compatible with new versions of the operating systems and other dependencies. Additionally, regular backups of the application's data will be taken to prevent any data loss in case of hardware failure

APPENDIX A: Record of Changes

Version	Date	Author	Description of Change
1.0	28/03/23	Anas A	Added System and Application Architecture
1.0	28/03/23	Jayalekshmi K S	Added Data and Algorithm Design
1.0	04/04/23	Aswathy Krishnan	Started UI Design and Testing Design
1.0	04/04/23	Nandakumar	Edited Component and Error Handling Design
1.0	06/04/23	Anas A	Updated Maintenance Design
1.0	06/04/23	Jayalekshmi K S	Added Security Design
1.0	11/04/23	Aswathy Krishnan	Edited performance design part
1.0	12/04/23	Nandakumar	Proofread document format
1.0	12/04/23	Jayalekshmi K S	Finalized document for submission

APPENDIX B: Acronyms

ACRONYM	**LITERAL TRANSLATION**
API	Application Programming Interface
GUI	Graphical User Interface
ORM	Object Relational Mapping
HTTPS	Hypertext Transfer Protocol
SQL	Structured Query Language
GPIO	General Purpose Input/Output

APPENDIX C: Referenced Documents

DOCUMENT NAME	DOCUMENT URL	ISSUED DATE
Preventing Pollution Attacks in Cloud Storages, Aswin Viswas V, Philip Samuel	https://www.sciencedirect.com/science/article/pii/S1877050918320829	2018
Secure distributed adaptive bin packing algorithm for cloud storage Irfan Mohiuddin, Ahmad Almogren, Mohammed Al Qurishi, Mohammad Mehedi Hassan, Giancarlo Fortino, Iehab Al Rassan	https://www.sciencedirect.com/science/article/abs/pii/S0167739X18304035	2018
An integrity verification scheme of cloud storage for internet-of-things mobile terminal devices , Xiuqing Lu, Zhenkuan Pan, Hequn Xian	https://www.researchgate.net/publication/337816943_An_Integrity_Verification_Scheme_of_Cloud_Storage_for_Internet-of-Things_Mobile_Terminal_Devices	2019