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

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 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.

1.2 Scope

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

1.3 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 Overview

2.1 System Context

The Cloud Pi system will consist of a Raspberry Pi server connected to the internet and a web-based user interface. The user interface will be accessible to authorized users over the internet.

2.2 System Architecture

The Cloud Pi system architecture consists of a web interface built with the Django framework, a Python script to perform file upload and deletion operations on the remote server, and a Raspberry Pi as the server. The system uses a RESTful API to communicate between the web interface and the remote server.

2.3 System Features

The Cloud Pi system includes the following features:

1. User authentication and access control
2. File upload and deletion operations
3. File download and viewing capabilities
4. Secure data storage using encryption
5. Scalability for increased storage capacity

2.4 User Classes and Characteristics

User classes refer to groups of users who will interact with the system in different ways. For a cloud-based Raspberry Pi storage system, the following user classes can be identified:

System Administrators: These are users who will have complete control over the system. They will be responsible for setting up the system, managing user accounts, configuring the system, and monitoring its performance.

End Users: These are the users who will use the storage system to store and retrieve their data. They may be individual users or members of an organization.

Developers: These are users who will be involved in the development of the system. They may be responsible for developing and maintaining the software that runs on the Raspberry Pi, or for integrating the storage system with other applications.

As for user classifications, they refer to different types of users within each user class. For example, within the end user class, we can have different classifications such as:

Basic Users: These are users who will use the system for simple file storage and retrieval.

Power Users: These are users who will use the system for more advanced tasks such as sharing files with other users, setting permissions, and creating backups.

Business Users: These are users who will use the system for business purposes such as storing and sharing documents, collaborating with team members, and accessing files remotely.

By identifying user classes and classifications, we can better understand the different requirements and needs of each user group, which can then be used to inform the design and development of the system.

2.5 Design and Implementation Constraints

- Hardware constraints: The system needs to be designed to work with a specific hardware configuration, such as the Raspberry Pi. This could include limitations on processing power, memory, and storage capacity.
- Software constraints: The system needs to be designed to work with specific software components, such as the Python programming language and the Django web framework.
- Time constraints: There may be limitations on the amount of time available to complete the project, which could affect the design and implementation choices made.
- Security constraints: The system needs to be designed to protect against potential security risks, such as unauthorized access to sensitive data.
- Usability constraints: The system needs to be designed to be user-friendly and easy to use, with a clear and intuitive interface.
- Scalability constraints: The system needs to be designed to handle a potentially large number of users and files, and to scale up as necessary.
- Budget constraints: There may be limitations on the amount of resources available to implement the system, which could affect the design and development choices made.

2.6 Assumptions and Dependencies

Assumptions:

- Users have access to a web browser and an internet connection to use the application.
- Users have basic knowledge of how to upload and delete files.
- The remote storage system is available and accessible to the application.
- The Raspberry Pi is properly set up and configured to receive and process the uploaded files.
- The uploaded files will not exceed the available storage capacity of the remote storage system or the Raspberry Pi.

Dependencies:

- The application relies on the Django web framework and its built-in functionality for handling file uploads and forms.
- The application depends on the remote storage system's API or SDK to programmatically upload and manage files.
- The Raspberry Pi must have the necessary software and libraries installed to process the uploaded files.
- The application may depend on third-party packages or modules for specific features or functionality.

3. Architectural design

This section provides an overview of the Cloud Pi system's architectural design, including subsystem decomposition, hardware/software mapping, persistent data management, access control and security, global software control, boundary conditions, interfaces, and communications.

3.1 Overview

The Cloud Pi system's architectural design consists of a web interface, a RESTful API, a Python script to perform file upload and deletion operations, and a Raspberry Pi server. The system is designed to be scalable, secure, and user-friendly.

3.2 Subsystem Decomposition

The Cloud Pi system's subsystems include:

- User authentication and access control
- File upload and deletion operations
- File storage and encryption
- File sharing with other users
- Web interface and RESTful API
- Raspberry Pi server

3.3 Hardware/Software Mapping

The Cloud Pi system requires a Raspberry Pi server, as well as a device with an internet connection and a web browser to access the web interface. The system is built using the Django web framework and Python programming language.

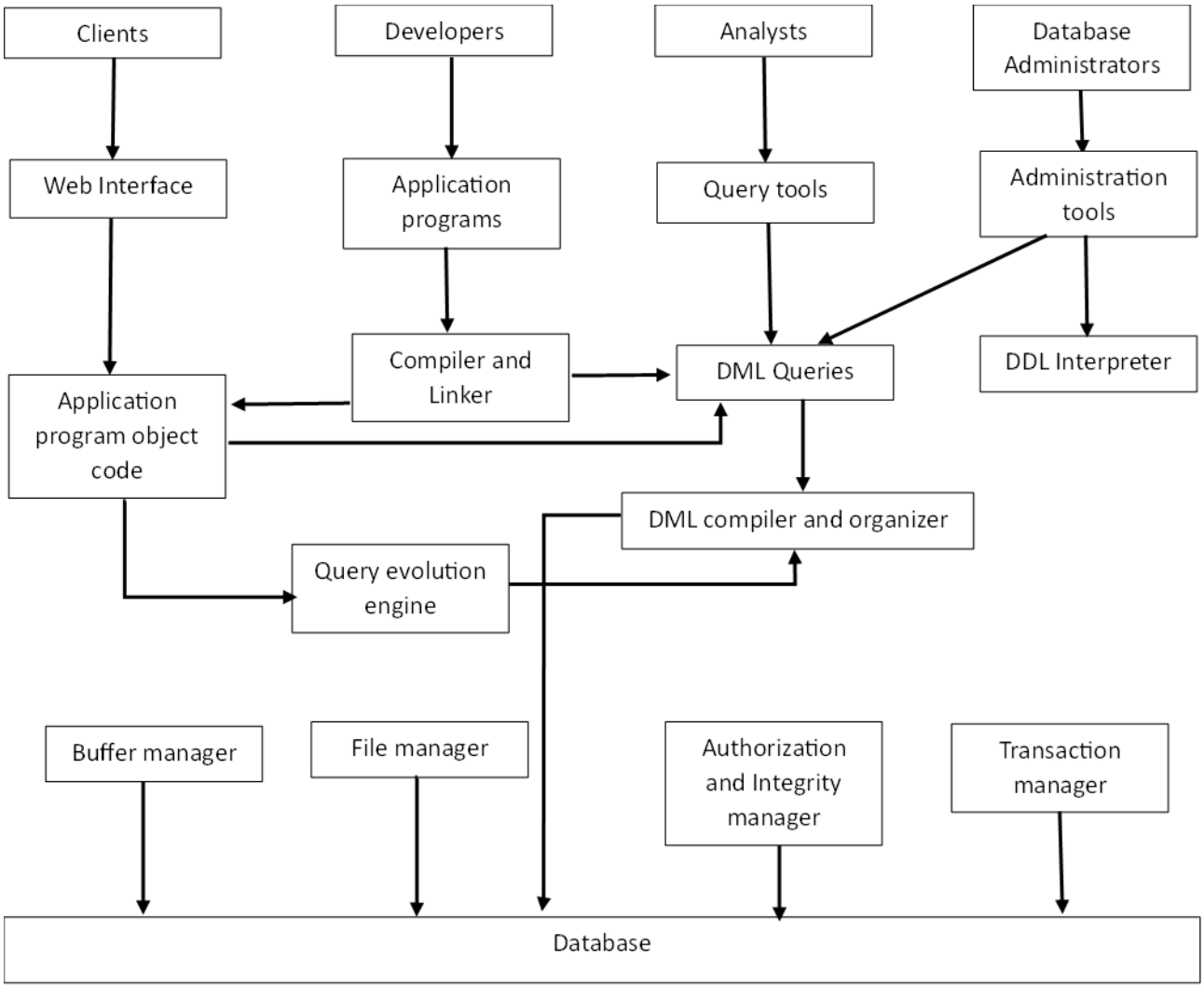
3.4 Interfaces

- Web interface: A user-friendly interface for accessing and managing user files.
- RESTful API: A communication interface between the web interface and the Raspberry Pi server.
- Python script: A script for performing file upload and deletion operations on the remote server.

3.5 Communication

The Cloud Pi system uses a RESTful API to communicate between the web interface and the Raspberry Pi server. The system uses encryption to ensure secure data transmission.

3.6 UML



4.GUI design

4.1

5. Data Design

5.1 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.

5.2 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

5.3 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.

5.5 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

5.5 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.

6. Component Design

6.1 Component Description:

- Raspberry Pi: This component is the heart of the project, responsible for running the Django web application and handling the hardware components such as camera and sensors.
- 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: The Wi-Fi module 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.

6.2 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, as well as the methods used for accessing and controlling the GPIO pins of 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.

6.3 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.

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

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.

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

Appendix B: Acronyms

Appendix C: Referenced Documents