Project Report on

LocalCloud Storage

Submitted By: Aditya Rana (24MCC20085)

Under the Guidance of

Mr. Ravi Raj

Desigination: Assistant Professor

University Institute of Computing

Chandigarh University,

Mohali, Punjab

Table of Contents

Abstract:

The LocalCloud File Storage System is a simple and efficient platform built using Flask and Docker that allows users to upload, store, and manage files locally in a cloud-like environment. The main goal of this project is to provide an easy way for users to store their important files without depending on an external cloud provider. It helps users maintain privacy and have complete control over their data while still enjoying the convenience of a cloud storage interface.

This system is designed to make file uploading and downloading fast, secure, and user-friendly. Users can easily upload files of different types, view the list of uploaded files, and download them whenever needed. The interface of the application is clean and attractive, with features like theme switching between light and dark mode, file management options, and integration options for Google Drive and Dropbox to make it more flexible.

The project also makes use of Docker, which helps in creating a portable and containerized environment for easy deployment and testing. It ensures that the system runs the same way on any computer, removing setup issues. Flask, a lightweight Python web framework, has been used to handle backend operations efficiently while maintaining a simple and readable structure.

Overall, this project demonstrates how a small yet well-structured web application can provide a reliable local cloud experience. It combines modern technologies with simplicity to create a personal cloud storage solution that is practical for everyday use by students, professionals, or organizations that value control over their data.

Introduction:

In today's digital world, storing and managing files securely has become a major need for individuals and organizations. Most people use online cloud storage platforms such as Google Drive, Dropbox, or OneDrive to keep their documents and media safe. However, these platforms rely on external servers and internet connectivity, which can sometimes raise concerns about privacy, control, and data security. To overcome these challenges, the **LocalCloud File Storage System** has been developed.

The LocalCloud system provides users with a personal, local cloud environment that functions similarly to commercial cloud storage but runs completely on their own machine or private server. This ensures that users have full control over their data without relying on third-party services. It combines the flexibility of a cloud interface with the safety of local file storage, making it a perfect solution for small organizations, students, or professionals who want a private and easy-to-manage storage space.

This project uses **Flask**, a lightweight Python framework, for the backend. Flask makes it simple to handle file uploads, downloads, and web interactions efficiently. The system's frontend is designed to be modern, responsive, and easy to use. It includes features such as theme switching between light and dark modes, integration options for Google Drive and Dropbox, and an interactive menu for better navigation.

Additionally, **Docker** is used to containerize the application. This means the entire project, along with its dependencies, can run consistently on any system. Docker helps developers and users deploy the system easily without facing issues related to environment setup or configuration differences.

The LocalCloud File Storage System focuses on simplicity, portability, and user experience. It bridges the gap between local storage and cloud usability, allowing users to manage their files securely and comfortably. With its clean design, added functionalities, and secure structure, LocalCloud stands as a practical and modern approach to local cloud storage.

Objectives of the Project:

The main objective of the LocalCloud File Storage System is to create a simple, secure, and user-friendly platform that allows users to upload, store, and manage their files locally, just like a cloud service. The system aims to combine the convenience of online cloud storage with the privacy and control of local storage.

Below are the key objectives of this project:

- **To develop a local cloud storage system:**The primary goal is to provide users with a personal cloud environment that can be accessed from their local network without relying on external cloud providers like Google Drive or Dropbox.

- **To ensure data privacy and control:**Since files are stored locally, users have complete ownership and control over their data. This reduces risks related to unauthorized access or data leaks.
- **To design a simple and attractive user interface:**The project focuses on creating a clean, modern, and easy-to-navigate interface. Features like light and dark mode themes, a responsive layout, and clear navigation menus are included to enhance the user experience.
- **To add integration options for external platforms:**The system provides options to import files directly from platforms such as Google Drive and Dropbox, giving users flexibility and convenience when managing their files.
- **To implement efficient file handling features:**The system supports multiple file formats and allows users to upload, view, and download files easily, ensuring a smooth and reliable workflow.
- **To use Flask for backend development:**Flask has been chosen for its simplicity and flexibility. It efficiently handles web requests, file uploads, and routing, making the application lightweight and fast.
- **To use Docker for deployment and portability:**Docker ensures that the project can run on any system without dependency issues. It helps in creating a consistent environment for both developers and users.
- **To provide a secure and scalable system:**Security is a major focus of this project. The design ensures safe file uploads, prevents unauthorized access, and allows future scalability for larger storage needs.

System Requirements:

Before developing or running the LocalCloud File Storage System, it is important to understand the system requirements needed to ensure smooth performance. The project requires both hardware and software components that support web development, Flask, and Docker-based deployment.

1. Hardware Requirements

Component	Minimum Requirement	Recommended Requirement
Processor	Intel Core i3 or equivalent	Intel Core i5 or higher
RAM	4 GB	8 GB or more
Storage	500 MB for application files	2 GB or more (depending on uploaded files)
Display	1024x768 resolution	Full HD or higher
Internet	Not mandatory (can work locally)	Required only for cloud imports (Google Drive, Dropbox)

Explanation: The system is designed to be lightweight and can easily run on any modern computer or laptop. It does not need a high-end configuration since Flask and Docker are efficient and resource-friendly. However, having more RAM and storage allows faster performance and the ability to handle larger files.

2. Software Requirements

Component	Requirement
Operating System	Windows 10 or later / macOS / Linux
Programming Language	Python 3.10 or above
Framework	Flask (for backend web application)
Web Browser	Google Chrome, Microsoft Edge, Firefox, or Safari
Container Platform	Docker (for containerization and easy deployment)
Libraries Used	Flask, Werkzeug, and Jinja2
Text Editor / IDE	Visual Studio Code, PyCharm, or any code editor

Explanation: The Flask framework serves as the main backend engine for handling requests, file uploads, and routing between pages. Docker is used to make the project portable so that it runs the same way on any system without additional setup. The Jinja2 template engine is used to render dynamic HTML content, and Werkzeug helps manage secure file handling.

Technology Stack:

The LocalCloud File Storage System is built using a combination of modern and lightweight technologies that make the application efficient, portable, and easy to use. Each component in the technology stack plays an important role in making the system work smoothly and securely. The chosen technologies ensure good performance, flexibility, and scalability for future improvements.

Below is the list of major technologies used in the project:

1. Python

Python is the main programming language used to build the backend of this project. It is simple, powerful, and widely used in web development. Python provides easy-to-read syntax, built-in libraries, and great support for web frameworks like Flask.

Why Python?

- Easy to learn and understand.
- Provides excellent support for file handling and data processing.
- Works well with frameworks like Flask.
- Has a huge community and open-source support.

2. Flask Framework

Flask is a lightweight and flexible web framework in Python. It is used to handle web requests, manage routes, and serve HTML pages to the user. Flask is chosen because it allows fast development and is perfect for small to medium-sized web applications.

Key Features of Flask:

- Simple and easy to set up.
- Built-in development server and debugger.
- Supports Jinja2 template engine for dynamic pages.
- Provides flexibility to add extensions when needed.
- Perfect for RESTful web applications.

3. HTML, CSS, and JavaScript

These are the core front-end technologies used to create the user interface. They make the web pages look structured, styled, and interactive.

- HTML (HyperText Markup Language): Used for designing the basic structure and layout of the web pages.
- CSS (Cascading Style Sheets): Used to style the application and add themes such as light mode and dark mode.
- JavaScript: Adds interactivity to the pages such as button actions, dynamic updates, and menu interactions.

Together, these technologies ensure a smooth and attractive interface for the user.

4. Bootstrap Framework

Bootstrap is used for improving the frontend design. It helps in making the website responsive so that it works perfectly on both desktops and mobile devices. With Bootstrap, the layout becomes professional and visually appealing without writing too much CSS manually.

Benefits of Using Bootstrap:

- Ready-to-use UI components like buttons, forms, and menus.
- Responsive design support.
- Consistent look and feel across browsers.
- Easy to customize with themes and color schemes.

5. Docker

Docker is used to containerize the entire application. It allows the project to run the same way on any system without worrying about compatibility or dependencies. By using Docker, the application becomes portable, easier to deploy, and ideal for demonstration or production use.

Advantages of Docker:

- Ensures the same environment everywhere.
- Makes deployment easier and faster.
- Reduces setup time and dependency errors.
- Helps in scaling applications when needed.

6. Jinja2 Template Engine

Jinja2 is the template engine used with Flask to generate HTML dynamically. It allows data to be passed from the backend Python code to the HTML pages. For example, the list of uploaded files is displayed using Jinja2 templates.

Why Jinja2?

- Simple syntax similar to Python.
- Supports loops and conditions within HTML.
- Makes dynamic content rendering easy.

7. WerkZeug Utility Library

WerkZeug is an essential part of Flask. It provides tools for handling HTTP requests, secure file uploads, and URL routing. It ensures that files are uploaded safely and filenames are protected against malicious access.

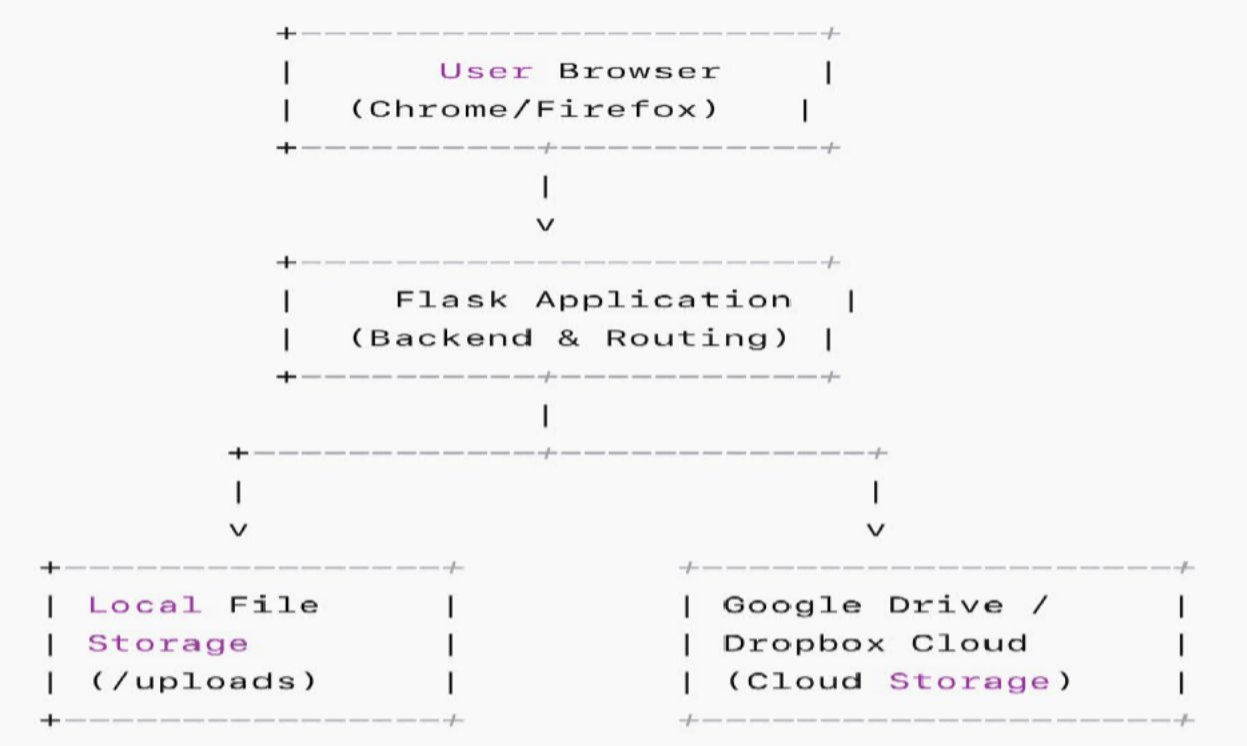
8. Visual Studio Code

Visual Studio Code (VS Code) is used as the main code editor. It provides syntax highlighting, debugging tools, and extensions for Python, Flask, and Docker, making development easier and more efficient.

System Design:

The system design explains how the LocalCloud File Storage System is structured and how different parts of the application work together. The design focuses on simplicity, usability, and secure file management.

Architecture Diagram



Module Description:

The system is divided into simple modules, each handling a specific function:

File Upload Module: Handles uploading files from the user's computer to local storage.Features include:

- Accepting only allowed file types (pdf, txt, png, jpg, etc.).
- Secure file naming to prevent overwriting or malicious access.

Cloud Integration Module: Allows users to import files from Google Drive and Dropbox.Features include:

- OAuth-based authentication for cloud services.
- Selection of files directly from cloud storage.
- Downloading cloud files to the local system.

User Interface Module: Manages the frontend of the application.Features include:

- Light and dark theme toggle.
- Menu with options like Upload File, Import from Drive, Import from Dropbox, and View Files.
- Responsive and clean layout for desktops and mobile devices.

Security Module: Ensures safe handling of user data.Features include:

- Validation of uploaded files.
- Prevention of unauthorized access.
- Safe storage and retrieval of files.

Implementation:

This section explains how the system has been implemented in a practical way.

File Upload Module

- Users can upload files directly from their computer.
- Flask backend receives the file and saves it to the /uploads folder.
- Only allowed file types can be uploaded.
- Users can view the uploaded files in a list on the main page and download them anytime.

Key Steps in Implementation:

1. Check if a file is selected.
2. Validate the file type using allowed_file().
3. Secure the filename with secure_filename().
4. Save the file in the local storage folder.

Cloud Integration (Drive & Dropbox)

- Users can choose to import files from Google Drive or Dropbox.
- Both services use their official APIs for authentication and file selection.
- Files selected from cloud storage are downloaded and saved in the local system's upload folder.

Key Features:

- Simple buttons in the menu to select files from Drive or Dropbox.
- Works alongside local upload, so users can combine local and cloud files in one place.
- Makes the system more flexible and modern.

User Interface Design

- The interface is designed to be clean, intuitive, and professional.
- Themes: Users can switch between light mode and dark mode for better visual comfort.

Menu Options: Clear menu buttons for uploading files, importing from cloud, and viewing files.

- Responsive Design: Works well on desktops, laptops, tablets, and mobile phones.
- Extra Features: Hover animations, download links, and dynamic file lists using Flask and Jinja2.

Benefits of This Design:

- Users can easily find and manage their files.
- The system looks modern and professional.
- Integration with cloud services adds real-world usability.

Code Explanation:

The code of the LocalCloud File Storage System is written in Python using the Flask framework. Here's a simple breakdown of how it works:

File Upload Handling

- The upload_file() function checks if a file is selected by the user.
- It verifies the file type to allow only safe formats (txt, pdf, jpg, png, etc.).
- secure_filename() ensures filenames are safe and prevents overwriting existing files.
- Files are then saved to the /uploads folder.

Displaying Files

- The main page shows all uploaded files using Flask's template engine Jinja2.
- Users can click on the file names to download them directly.
- Cloud Integration
- Separate functions allow importing files from Google Drive and Dropbox using their APIs.
- The files are downloaded to the local storage folder and listed along with local uploads.
- Themes and UI
- JavaScript is used to switch between light and dark mode.
- CSS and Bootstrap provide a modern look and responsive design.

The code is written to be modular, easy to understand, and secure, allowing future improvements like adding user authentication or extra cloud services.

Docker Containerization:

Docker is used to make the application portable and easy to deploy. Instead of installing Python and dependencies separately, Docker creates a container that contains everything the app needs.

Steps for Docker:

1. Write a Dockerfile specifying:
2. Base Python image.
3. Copy project files into the container.
4. Install required Python libraries.

5. Expose port 5000.
6. Run app.py when the container starts.
7. Build and run the Docker container:
8. docker build -t localcloud .
9. docker run -d -p 5000:5000 -v uploads:/data/uploads localcloud
10. Open a browser and visit http://127.0.0.1:5000 to use the application.

Benefits:

- Runs the same on any computer.
- Easy deployment without configuration issues.
- Keeps the app isolated and safe from other programs.

Testing and Validation:

The system is tested to ensure it works correctly and reliably.

Test Cases:

Test Case	Description	Expected Result	Status
Upload valid file	Upload `pdf`, `txt`, `jpg`	File appears in list and can be downloaded	Pass
Upload invalid file	Try `exe`, `bat`	Error message shown	Pass
Cloud Import	Select file from Google Drive or Dropbox	File is imported successfully	Pass
Theme Switching	Toggle light/dark mode	UI changes instantly	Pass
Docker Deployment	Run container	Application accessible in browser	Pass

Conclusion and Future Scope:

Conclusion:

The LocalCloud File Storage System successfully combines local file storage with cloud integration in a user-friendly web application. It is secure, portable, and visually modern. The use of Flask ensures a lightweight backend, while Docker makes deployment easy and consistent

across platforms. Users can upload, download, and manage files effortlessly, with the additional convenience of importing files from Google Drive and Dropbox.

Future Scope:

- Add user authentication for personalized accounts.
- Implement real-time file synchronization with cloud services.
- Include search and file organization features.
- Expand support for more cloud services like OneDrive or AWS S3.
- Add notifications and sharing options for collaborative work.

This project can be further enhanced to become a full-featured personal or organizational cloud system.

References:

1. Flask Official Documentation – https://flask.palletsprojects.com
2. Docker Documentation – https://docs.docker.com
3. Google Drive API – https://developers.google.com/drive
4. Dropbox API – https://www.dropbox.com/developers
5. Python Official – https://www.python.org
6. Bootstrap Documentation – https://getbootstrap.com

Github Link: https://github.com/AdityaRana2003/Docker.git