SkyVault Project Report

Priyanshu Kumar Sharma
BTech CTIS, Sem-5
Year-3 Sem-5
URN: 2022-B-17102004A
Information Security Applications
Course Code: MI300E
Prof Prachi Shukla

December 21, 2024

1. Introduction

SkyVault is a secure personal cloud storage solution that allows users to upload, manage, and access files privately. It prioritizes data security through encryption and authentication, giving users complete control over their data. Built with Flask and the `cryptography` library, SkyVault provides an accessible and secure alternative to third-party cloud services.

2. Objectives

The main objectives of the project are:

- Create a private cloud storage solution.
- Encrypt uploaded files for data confidentiality.
- Provide a user-friendly interface for file management.
- Implement robust error handling and security features.

3. Features

SkyVault includes:

- **User Authentication**: Secure login with hashed passwords.
- **File Encryption**: Files are encrypted using symmetric encryption.
- **File Management**: Upload, view, download, and delete files.
- **Error Handling**: Custom 404 and 500 error pages.
- **Secure Storage**: Encrypted files stored in a protected directory.

4. Project Structure

The project repository is available at: SkyVault GitHub Repository. The directory structure is:

```
project/
 app.py                   # Main application file
 uploads/                 # Original uploaded files
 encrypted_files/         # Encrypted files directory
 templates/               # HTML templates
    index.html           # Homepage for file uploads
    files.html           # View uploaded files
    login.html           # Login page
    404.html             # Custom 404 error page
    500.html             # Custom 500 error page
 static/                  # Static files (CSS, JS)
     css/
         styles.css       # Custom styles
```

5. Implementation Details

1. **File Upload and Encryption**: Uploaded files are encrypted using the `cryptography` library before being stored in the `encrypted_files/` directory.
2. **User Authentication**: Passwords are securely hashed using `bcrypt`, and sessions are managed for logged-in users.
3. **File Management**: Users can view, download, and delete their files through an intuitive interface.
4. **Error Handling**: Custom error pages provide meaningful feedback for common issues.

6. Usage

1. Clone the repository:

   ```
   git clone https://github.com/PriyanshuKSharma/SkyVault.git
   cd SkyVault
   ```

2. Install dependencies:

   ```
   pip install -r requirements.txt
   ```

3. Run the application:

   ```
   python app.py
   ```

4. Access the application at `http://127.0.0.1:5000/`.

7. Future Enhancements

- Multi-user support for separate storage spaces.
- File sharing capabilities.
- Two-factor authentication (2FA).
- Mobile app development for increased accessibility.

8. Conclusion

SkyVault offers a secure and private alternative to traditional cloud storage solutions. By combining encryption and user-friendly design, it ensures data confidentiality while providing an efficient file management system. For source code and further details, refer to the GitHub repository: SkyVault.

9. References

- Flask Documentation: `https://flask.palletsprojects.com`
- Cryptography Library: `https://cryptography.io`
- Bootstrap Framework: `https://getbootstrap.com`

10. Docker Integration

SkyVault can be containerized using Docker for seamless deployment and scalability. Follow the steps below to run SkyVault using Docker.

```
# Step 1: Build the Docker image
docker build -t skyvault .

# Step 2: Run the Docker container
docker run -d -p 5000:5000 --name skyvault-container skyvault

# Step 3: Stop the container (if needed)
docker stop skyvault-container

# Step 4: Restart the container
docker start skyvault-container

# Step 5: Remove the container (if no longer needed)
docker rm skyvault-container
```

Dockerfile: Below is the Dockerfile used for creating the Docker image for SkyVault.

```
# Use an official Python runtime as a parent image
FROM python:3.10-slim

# Set the working directory in the container
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install any needed packages specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Make port 5000 available to the world outside this container
EXPOSE 5000

# Define environment variable
ENV FLASK_APP=app.py

# Run the application
CMD ["flask", "run", "--host=0.0.0.0"]
```