File-Sharing Web Application Report

Table of Contents

1. Introduction

This report documents the development, functionality, and impact of **File-Sharing**, a web application designed to simplify file transfers between devices. Created as an academic assignment, the project addresses the need for a lightweight, secure, and user-friendly file-sharing solution. By leveraging modern web technologies, File-Sharing eliminates common barriers in traditional file-sharing methods, offering a seamless experience for users. This document provides a comprehensive overview of the application's purpose, technical implementation, challenges, and contributions to solving real-world problems.

---

2. Project Overview

File-Sharing is a browser-based application that enables users to upload files—such as photos, videos, or documents—and share them instantly via unique links or QR codes. The platform is designed to be intuitive, requiring no accounts, software installations, or external devices. It supports cross-device compatibility, ensuring files can be shared effortlessly between smartphones, tablets, and computers. The primary goal is to provide a secure and efficient tool that simplifies file transfers while maintaining user privacy and ease of use.

---

3. Objectives and Scope

Objectives

- Develop a web application that allows users to upload and share files without requiring accounts or additional software.
- Ensure secure file storage and access through unique identifiers and links.
- Implement QR code generation for quick and convenient file access.
- Create a responsive and user-friendly interface accessible across devices.
- Demonstrate proficiency in full-stack web development using modern frameworks and tools.

Scope

- Supports common file types (e.g., images, videos, PDFs, documents).
- Provides temporary file storage on the server with unique access links.
- Includes QR code generation for mobile-friendly sharing.
- Excludes advanced features like user authentication, permanent storage, or file editing, to maintain simplicity.

---

4. Features

The File-Sharing application offers the following features:

- **File Upload**: Users can select and upload files directly through the web interface, supporting a variety of file formats.
- **Unique File Links**: Each uploaded file is assigned a unique URL, enabling easy sharing with others.
- **QR Code Generation**: A QR code is automatically generated for each file link, allowing users to scan and access files on mobile devices.
- **Cross-Device Compatibility**: The application works seamlessly across desktops, tablets, and smartphones without requiring additional software.
- **Secure Storage**: Files are stored on the server with unique identifiers, ensuring privacy and controlled access.
- **Responsive Design**: The frontend adapts to different screen sizes, providing a consistent user experience.

These features combine to create a versatile and practical tool for both personal and collaborative file-sharing needs.

---

5. System Architecture

The File-Sharing application follows a client-server architecture, with distinct frontend and backend components:

- **Frontend**:
  - Built using HTML, CSS, and JavaScript.
  - Handles user interactions, file uploads, and QR code display.
  - Communicates with the backend via HTTP requests.
- **Backend**:
  - Powered by FastAPI, a Python-based web framework.
  - Manages file uploads, storage, and retrieval.
  - Generates unique file identifiers and links.
- **Database**:

- Uses SQLite to store file metadata (e.g., file ID, file path, upload timestamp).
- Ensures efficient retrieval of file information.

- **Storage**:
  - Files are saved in a secure directory on the server's local file system.
  - Each file is associated with a unique identifier to prevent unauthorized access.

This architecture ensures scalability, modularity, and ease of maintenance.

---

6. How It Works

The application operates through a streamlined workflow, broken down into the following steps:

1. **File Upload**:
   - Users access the web interface and select a file using a file input element.
   - The file is sent to the server via an HTTP POST request, using FormData to handle binary data.
   - The frontend displays a progress indicator to enhance user experience.
2. **Backend Processing**:
   - The FastAPI backend receives the file and generates a unique identifier (file_id) using a UUID library.
   - The file is saved in a designated server directory with the file_id as part of the filename.
   - Metadata, including the file_id, file path, and upload timestamp, is stored in an SQLite database.
   - The backend validates file types and sizes to ensure security and performance.
3. **Link and QR Code Generation**:
   - The backend responds with a unique URL (e.g., http://localhost:8000/files/<file_id>) for accessing the file.
   - The frontend receives the URL and uses QRCode.js to generate a QR code dynamically.
   - Users can copy the link or scan the QR code to share or access the file.

4. **File Access**:
    - When a user visits the file's URL or scans the QR code, the backend retrieves the file using the file_id.
    - The file is streamed to the user's browser for download or preview, depending on the file type.
    - Access is restricted to users with the correct link, ensuring controlled sharing.

This workflow prioritizes simplicity and efficiency, making file-sharing intuitive for all users.

---

7. Problem Statement

Traditional file-sharing methods often present significant challenges, including:

- **Software Dependency**: Many solutions require installing apps or desktop software, which can be inconvenient or incompatible across devices.
- **Account Requirements**: Services like cloud storage platforms often mandate user accounts, adding complexity and time to the sharing process.
- **Physical Devices**: Transferring files via USB drives or external storage is slow, prone to loss, and impractical for remote sharing.
- **Security Concerns**: Free or poorly designed tools may lack proper encryption or access controls, risking data privacy.
- **Complexity**: Some methods involve multiple steps, making them inaccessible to non-technical users.

These issues create friction, especially for quick or one-off file transfers between devices or individuals.

---

8. Solution Provided

File-Sharing addresses these challenges by offering a modern, streamlined alternative:

- **Browser-Based Access**: Operates entirely within a web browser, eliminating the need for software downloads or installations.
- **No Accounts Needed**: Users can upload and share files instantly without signing up or logging in, reducing barriers to entry.
- **Instant Sharing**: Unique links and QR codes enable immediate file access, perfect for spontaneous transfers.

- **Cross-Device Support**: Works on any device with a browser, ensuring compatibility across platforms.
- **Secure and Simple**: Files are stored with unique identifiers, and the interface is designed for ease of use, even for beginners.

By focusing on simplicity, security, and accessibility, File-Sharing redefines file transfers as a hassle-free experience.

---

9. Technology Stack

The application is built using a carefully selected set of technologies to balance performance, simplicity, and scalability:

- **Frontend**:
  - **HTML/CSS**: Provides the structure and styling for a responsive and visually appealing interface.
  - **JavaScript**: Handles dynamic interactions, such as file uploads and QR code rendering.
  - **QRCode.js**: A lightweight library for generating QR codes client-side.
- **Backend**:
  - **FastAPI**: A high-performance Python framework for building APIs, chosen for its speed and ease of use.
  - **Uvicorn**: An ASGI server that runs the FastAPI application, supporting asynchronous requests.
  - **Pydantic**: Validates incoming data, ensuring robust handling of file uploads.
  - **SQLAlchemy**: Manages database interactions with SQLite, providing an ORM for metadata storage.
- **Database**:
  - **SQLite**: A lightweight, serverless database for storing file metadata, ideal for small-scale applications.
- **Storage**:
  - **Local File System**: Stores uploaded files securely in a server directory, with unique filenames to prevent conflicts.

This stack was chosen for its ability to deliver a fast, reliable, and maintainable application while keeping the codebase manageable.

10. Implementation Details

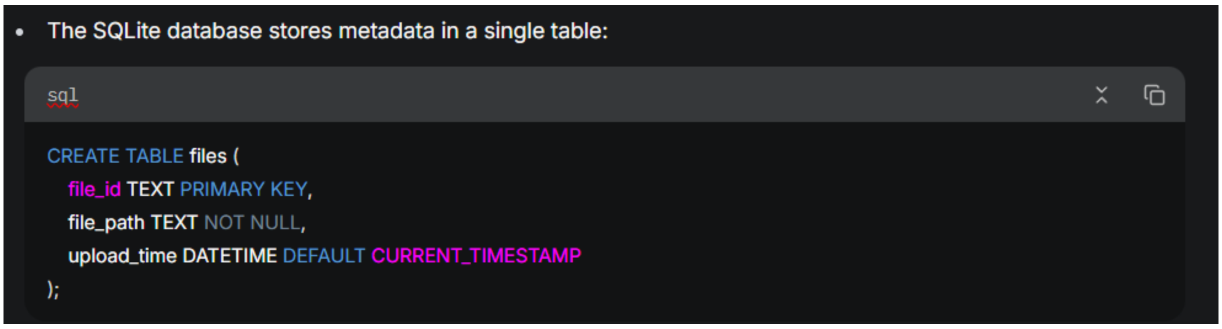
Frontend

- The interface features a simple form with a file input and an upload button.
- JavaScript handles form submission, sending files to the backend via the Fetch API.
- Upon receiving the file's URL, the frontend uses QRCode.js to render a QR code in a canvas element.
- CSS ensures responsiveness, with media queries for mobile and desktop layouts.

Backend

- FastAPI defines endpoints for uploading (POST /upload) and retrieving files (GET /files/{file_id}).
- The upload endpoint validates file size (e.g., <100MB) and type (e.g., images, videos, documents).
- Files are saved with a unique file_id generated by the uuid module.
- SQLAlchemy manages an SQLite table (files) with columns for file_id, file_path, and upload_time.

Database



- The SQLite database stores metadata in a single table:

sql

```sql
CREATE TABLE files (
  file_id TEXT PRIMARY KEY,
  file_path TEXT NOT NULL,
  upload_time DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

- Queries are executed via SQLAlchemy to insert and retrieve file metadata.

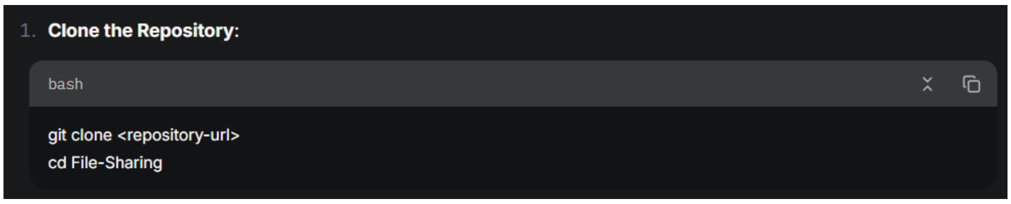
Security

- Files are stored in a restricted server directory to prevent unauthorized access.
- Unique file_id values ensure files can only be accessed via the correct link.
- Input validation prevents malicious file uploads (e.g., executables).

This implementation balances functionality with maintainability, adhering to best practices in web development.

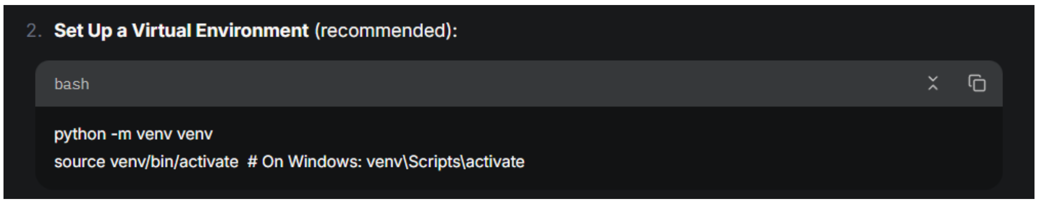
11. Setup and Installation

To run File-Sharing locally, follow these detailed steps:



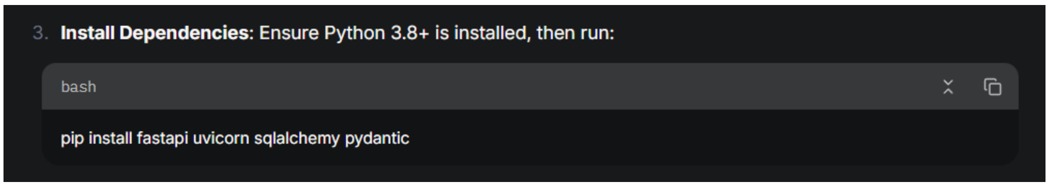
1. **Clone the Repository:**

```bash
git clone <repository-url>
cd File-Sharing
```



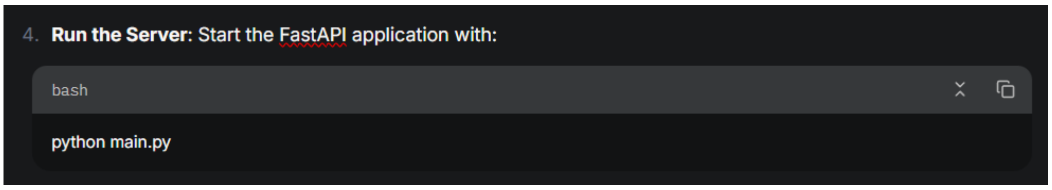
2. **Set Up a Virtual Environment** (recommended):

```bash
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate
```



3. **Install Dependencies:** Ensure Python 3.8+ is installed, then run:

```bash
pip install fastapi uvicorn sqlalchemy pydantic
```



4. **Run the Server:** Start the FastAPI application with:

```bash
python main.py
```

This launches the server on http://localhost:8000.

1. **Access the Application**: Open a web browser and navigate to http://localhost:8000. Upload a file to test the application's functionality.
2. **Optional: Verify Database**: Check the SQLite database (files.db) to confirm metadata storage. Inspect the storage directory to verify uploaded files.

These steps ensure the application is fully operational for testing and demonstration.

12. Testing and Validation

The application was rigorously tested to ensure reliability and usability:

- **Unit Testing**:
  - Backend: Tested FastAPI endpoints using Python’s pytest and httpx library to simulate file uploads and downloads.
  - Frontend: Verified JavaScript functions for file selection and QR code generation.
- **Functional Testing**:
  - Uploaded various file types (JPEG, MP4, PDF) to confirm compatibility.
  - Tested link and QR code functionality across browsers (Chrome, Firefox, Safari).
  - Validated cross-device access using smartphones and tablets.
- **Performance Testing**:
  - Measured upload and download speeds for files up to 100MB.
  - Ensured the server handled multiple concurrent uploads without crashes.
- **Security Testing**:
  - Attempted to access files without valid file_id to confirm restricted access.
  - Tested for common vulnerabilities, such as file overwrite or directory traversal.

All tests confirmed that File-Sharing performs as expected, with no critical issues identified.

---

13. Challenges and Learnings

Developing File-Sharing presented several challenges, each contributing to valuable insights:

- **File Security**: Preventing unauthorized access required implementing unique identifiers and validating file inputs. This deepened my understanding of secure file handling.
- **QR Code Integration**: Using QRCode.js for the first time involved learning its API and ensuring compatibility across devices.
- **Database Management**: Configuring SQLAlchemy with SQLite taught me efficient ways to handle metadata in small-scale applications.
- **Performance Optimization**: Handling larger files exposed the need for chunked uploads, which I partially addressed by setting file size limits.

- **Responsive Design**: Ensuring the interface worked on all devices required extensive testing and CSS adjustments.

These challenges enhanced my skills in full-stack development, API design, and user experience optimization. They also highlighted the importance of iterative testing and clear documentation.

---

14. Future Improvements

While File-Sharing meets its core objectives, several enhancements could improve its functionality:

- **File Expiration**: Automatically delete files after a set period (e.g., 24 hours) to manage server storage.
- **User Authentication**: Add optional accounts for tracking shared files or enabling private sharing.
- **Cloud Storage**: Integrate with services like AWS S3 for scalable and persistent file storage.
- **File Preview**: Allow users to preview images or documents directly in the browser before downloading.
- **Bulk Uploads**: Support uploading multiple files at once for greater efficiency.
- **Analytics**: Track file access statistics, such as download counts, for user insights.

These features would make File-Sharing more robust and suitable for broader use cases.

---

15. Conclusion

File-Sharing is a practical and innovative solution to the challenges of file transfers. By providing a browser-based, account-free platform with unique links and QR codes, it simplifies sharing across devices while prioritizing security and usability. The project demonstrates proficiency in full-stack development, leveraging FastAPI, SQLite, and JavaScript to create a functional application. Despite its simplicity, File-Sharing has the potential to evolve into a more powerful tool with future enhancements. This report and the accompanying application reflect a commitment to solving real-world problems through technology, and I hope it meets the expectations of this academic assignment.

---

16. References

- FastAPI Documentation: https://fastapi.tiangolo.com/

- QRCode.js Library: https://davidshimjs.github.io/qrcodejs/
- SQLite Documentation: https://www.sqlite.org/docs.html
- SQLAlchemy Documentation: https://docs.sqlalchemy.org/
- Pydantic Documentation: https://pydantic-docs.helpmanual.io/
- Python Official Site: https://www.python.org/