MyDrive

Google Drive-Like Image Upload Web Application

Language	Python 3.x
Framework	Flask
Database	SQLite
IDE	VS Code

1. Project Overview

MyDrive is a localhost web application built with Python and Flask that mimics core Google Drive functionality for image storage. Users can upload, preview, and delete images through a clean browser-based interface, with all metadata persisted in an SQLite database.

1.1 Objectives

- Build a fully functional image upload web application running on localhost.
- Accept JPG, JPEG, and PNG image formats.
- Store uploaded image files securely on the local server.
- Record image metadata (file name, upload time, file size, path) in SQLite.
- Provide a responsive gallery view with drag-and-drop support and lightbox preview.
- Allow users to delete images from both the server and the database.

1.2 Key Features

- Drag-and-drop upload zone with instant auto-submit.
- Google Drive-inspired UI with clean card-based image gallery.
- Lightbox preview on image click.

- File size display and upload timestamp for each image.
- Delete functionality with confirmation prompt.
- Flash messages for success and error feedback.

2. Project Folder Structure

The project uses the following directory layout:

Path / File	Type	Description
drive_app/	folder	Root project directory
app.py	Python file	Main Flask application — routes, DB logic, upload handling
requirements.txt	Text file	Python package dependencies
database.db	SQLite DB	Auto-created on first run; stores image metadata
templates/	folder	HTML template directory (Jinja2)
templates/index.html	HTML file	Main UI — upload form, gallery grid, lightbox
static/	folder	Static assets directory
static/uploads/	folder	Uploaded image files stored here

3. Module 1 — Web Application for Image Upload

3.1 How It Works

The web interface (index.html) presents a Google Drive-style upload card that accepts user image files. When a file is selected or dropped, a form POST is sent to Flask's /upload route, which validates the file extension, saves it to static/uploads/ with a timestamped filename, and records metadata in the database.

3.2 Supported File Formats

- JPG — Standard JPEG image
- JPEG — JPEG image (alternate extension)
- PNG — Portable Network Graphics

3.3 Upload Route (app.py)

The core upload logic in app.py:

```
@app.route('/upload', methods=['POST'])
def upload_file():
    file = request.files['file']
    if file and allowed_file(file.filename):
        filename = timestamp + secure_filename(file.filename)
        file.save(os.path.join(UPLOAD_FOLDER, filename))
        # Save metadata to SQLite
        flash('Uploaded successfully!', 'success')
```

3.4 UI Features

- Drag-and-drop upload zone with visual feedback on hover.
- Auto-submit when a file is selected via the file browser.
- Success / error flash messages displayed at the top of the page.
- Responsive image gallery with file name, upload date, and size.
- Click any image to open it in a full-screen lightbox preview.
- Press Escape or click outside to close the lightbox.
- Delete button with browser confirmation dialog on each image card.

4. Module 2 — Mini Database Setup

4.1 SQLite Overview

SQLite is a lightweight, serverless relational database that stores all data in a single .db file. No additional database server installation is required — Python includes SQLite support in its standard library via the sqlite3 module.

4.2 Database Schema

The images table stores the following fields:

Column	Type	Key	Description
id	INTEGER	PRIMARY KEY	Auto-incrementing unique identifier
filename	TEXT	NOT NULL	Timestamped, sanitized filename on disk
original_name	TEXT	NOT NULL	Original filename as uploaded by the user
file_path	TEXT	NOT NULL	Relative path to the file on the server

upload_time	TEXT	NOT NULL	Date and time of upload (YYYY-MM-DD HH:MM:SS)
file_size	INTEGER		File size in bytes

4.3 Database Initialization

The database and table are created automatically when app.py is first run via the init_db() function. No manual setup is needed.

```
def init_db():
    conn = sqlite3.connect('database.db')
    conn.execute(CREATE TABLE IF NOT EXISTS images (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        filename TEXT NOT NULL, ...
    ) -- end CREATE TABLE
    conn.commit()
```

5. Testing the Application

5.1 Testing Image Upload

1. Open a browser and go to http://127.0.0.1:5000.
2. Click the Choose Files button on the upload card.
3. Select a JPG, JPEG, or PNG file from your computer.
4. The form submits automatically — a green success banner appears.
5. The uploaded image appears as a card in the gallery below.

5.2 Testing Drag-and-Drop

6. Drag an image file from your file explorer.
7. Drop it onto the blue-bordered upload zone.
8. The image uploads and appears in the gallery.

5.3 Testing Image Preview

9. Click any image thumbnail in the gallery.
10. A full-screen lightbox overlay opens showing the image.
11. Click outside the image or press Escape to close.

5.4 Testing Invalid File Types

12. Try uploading a non-image file (e.g., a .pdf or .txt file).
13. A red error banner is displayed: 'Invalid file type.'
14. No file is saved and no database record is created.

5.5 Testing Delete

15. Click the Delete button on any image card.
16. A browser confirmation dialog appears.
17. Click OK to confirm. The image is removed from both the gallery and the database.

5.6 Verifying the Database

To inspect the SQLite database directly, run the following in the VS Code terminal:

```
python -c "import sqlite3; conn = sqlite3.connect('database.db');
[print(r) for r in conn.execute('SELECT id, original_name, upload_time,
file_size FROM images').fetchall()]"
```

This prints all stored image records to the terminal.

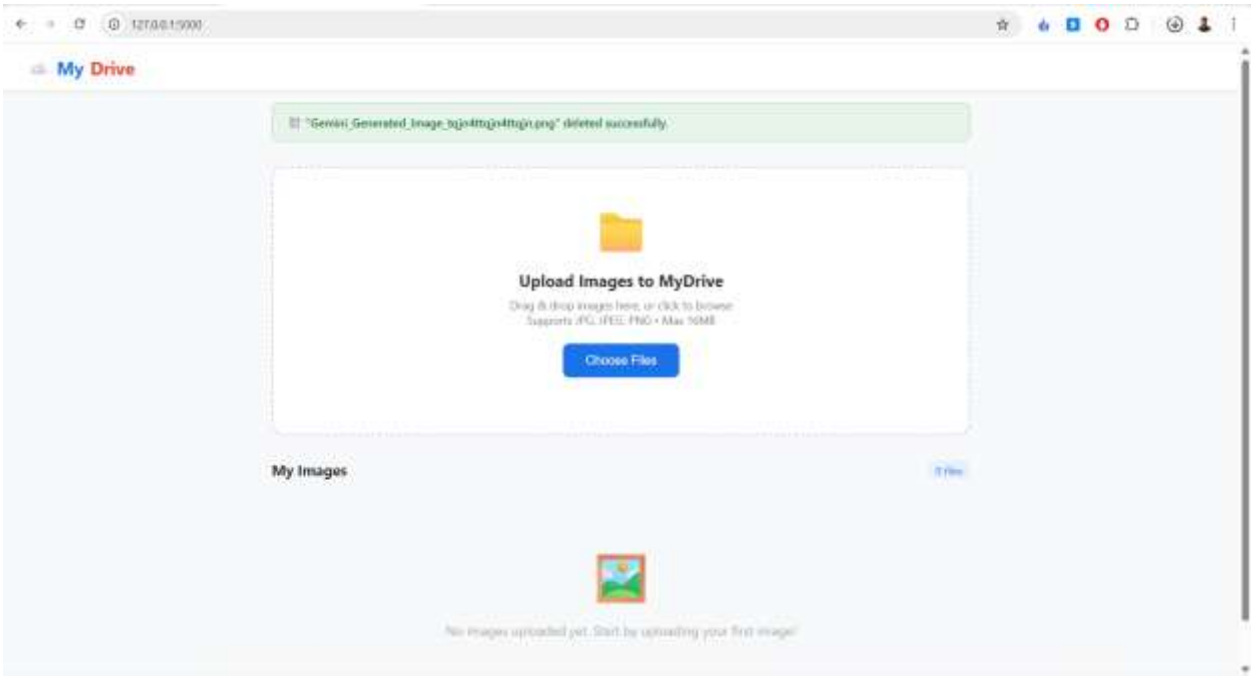

Fig. 1: Project Interface

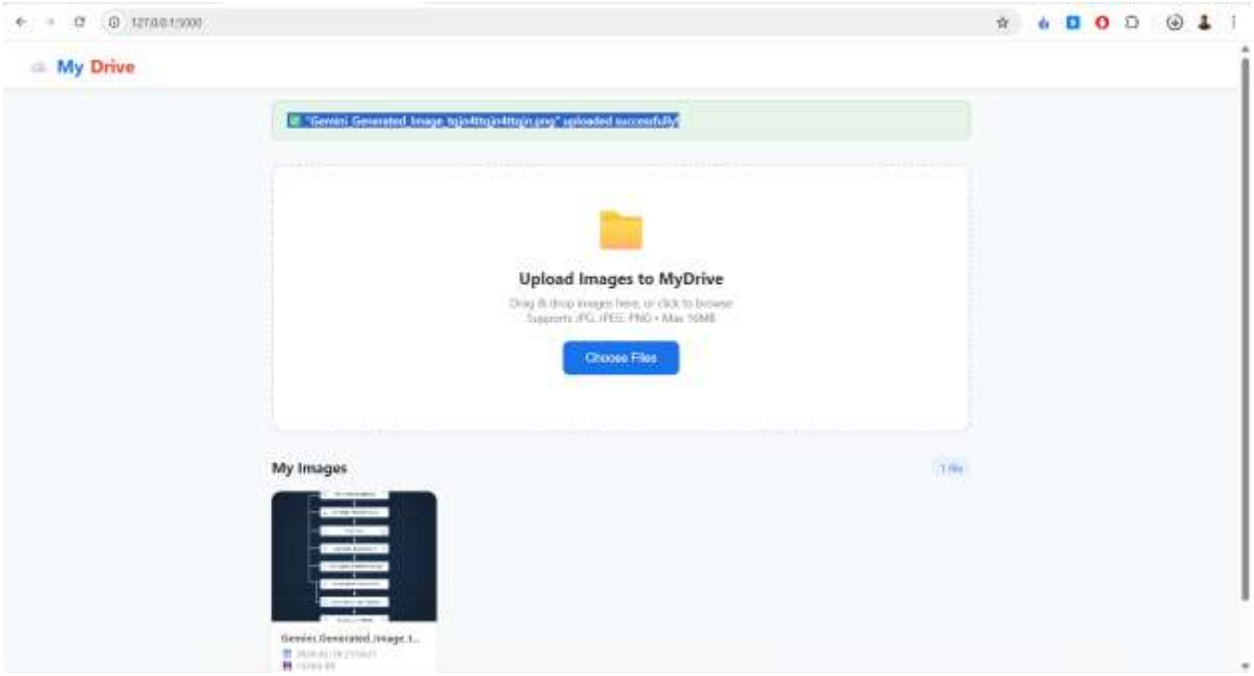

Fig. 2: The user successfully uploaded the image.

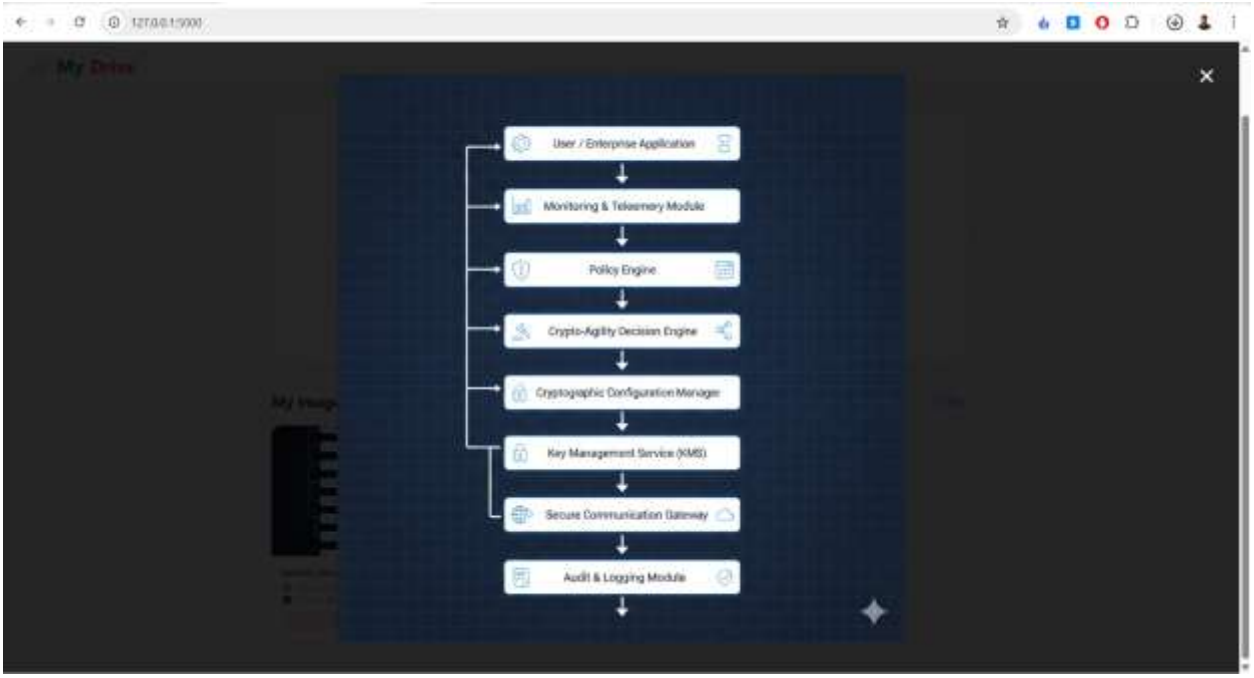

Fig. 3: The user can view and delete the uploaded image.

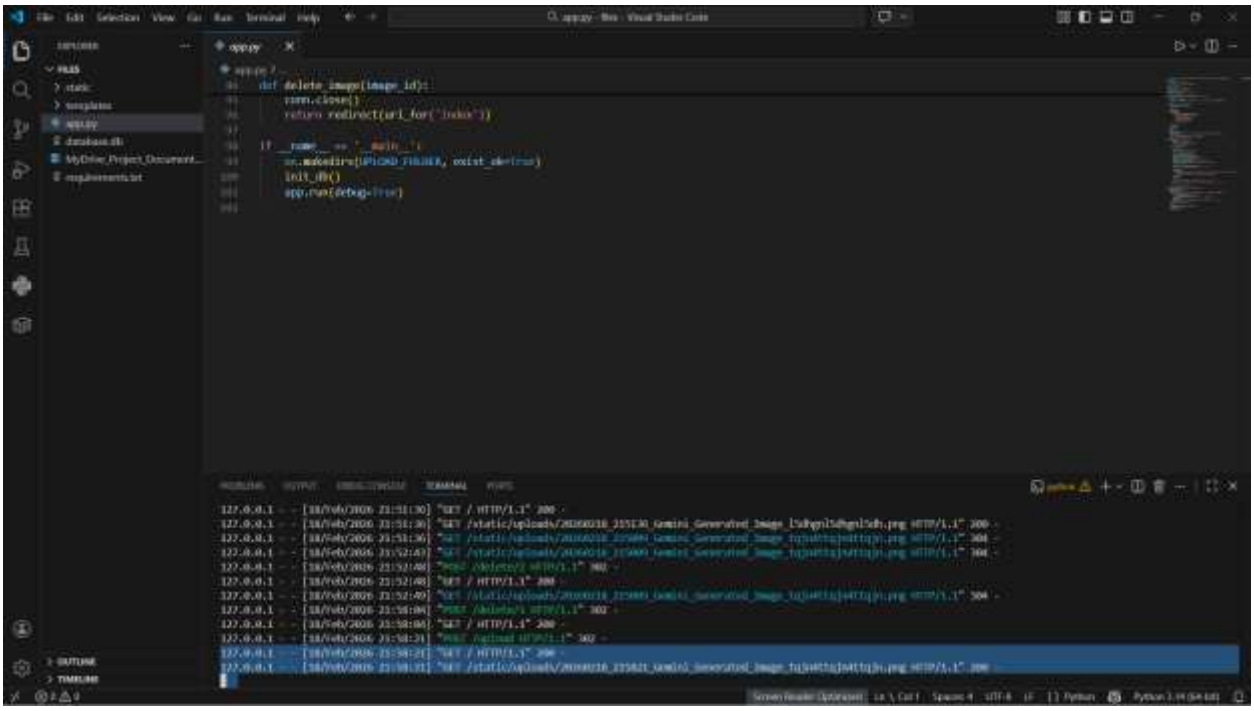

Fig. 4: Successful update request with 200 status code in the terminal.