Flask File Upload MVC with MySQL and Logging — Step-by-Step Documentation (for Students)

This guide walks you through creating a **Flask MVC application** that supports file uploads, saves file paths to **MySQL**, logs every action, and allows basic CRUD operations via **Postman testing**.

🧩Step 1: Project Setup

1. Create the project structure

```
flask_file_upload_mvc/
│
├── app.py                  # Entry point of the Flask app
├── config.py               # Configuration and .env loader
├── .env                    # Environment variables
├── requirements.txt        # Dependencies list
├── /models                 # Database models
│   └── uploaded_file.py
├── /controllers            # Flask routes / endpoints
│   └── file_controller.py
├── /services               # Business logic layer
│   └── file_service.py
├── /uploads                # Folder where uploaded files will be saved
└── /logs                   # Folder for logging output
```

🧩Step 2: Create and activate a virtual environment

```
python -m venv venv
venv\Scripts\activate   # For Windows
# source venv/bin/activate   # For Mac/Linux
```

Install required packages:

```
pip install flask flask_sqlalchemy python-dotenv pymysql werkzeug
```

Then freeze dependencies:

```
pip freeze > requirements.txt
```

🧩Step 3: Create the `.env` file

```
# Flask Environment
FLASK_ENV=development
UPLOAD_FOLDER=uploads

# Database Configuration
DB_USER=root
DB_PASSWORD=
DB_HOST=localhost
DB_PORT=3306
DB_NAME=flask_uploads_db
```

⚠️**Important:** Don't upload `.env` to GitHub. Add it to `.gitignore`.

🧩Step 4: Create `config.py`

```python
import os
from dotenv import load_dotenv

load_dotenv()  # Load .env variables

class Config:
    SQLALCHEMY_DATABASE_URI = (
        f"mysql+pymysql://{os.getenv('DB_USER')}:{os.getenv('DB_PASSWORD')}@"
        f"{os.getenv('DB_HOST')}:{os.getenv('DB_PORT')}/{os.getenv('DB_NAME')}"
    )
    SQLALCHEMY_TRACK_MODIFICATIONS = False
    UPLOAD_FOLDER = os.getenv('UPLOAD_FOLDER', 'uploads')
    LOG_FOLDER = 'logs'
    DEBUG = True
```

🧩Step 5: Create Model — `/models/uploaded_file.py`

```python
from app import db
```

```
# This model represents an uploaded file record in the database
class UploadedFile(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    filename = db.Column(db.String(255), nullable=False)
    file_path = db.Column(db.String(255), nullable=False)
```

🧩Step 6: Create Service — `/services/file_service.py`

```
import os
import uuid
from werkzeug.utils import secure_filename
from models.uploaded_file import UploadedFile
from app import db
import logging

ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'pdf', 'txt'}

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in
ALLOWED_EXTENSIONS

def save_file(file, base_folder):
    if not allowed_file(file.filename):
        raise ValueError('File type not allowed')

    os.makedirs(base_folder, exist_ok=True)
    filename = secure_filename(file.filename)
    unique_name = f"{uuid.uuid4().hex}_{filename}"
    save_path = os.path.join(base_folder, unique_name)
    file.save(save_path)

    new_file = UploadedFile(filename=unique_name, file_path=save_path)
    db.session.add(new_file)
    db.session.commit()
    logging.info(f"File saved: {save_path}")
    return new_file

def update_file(file_id, new_file, base_folder):
    existing = UploadedFile.query.get(file_id)
    if not existing:
        raise ValueError('File not found')

    if os.path.exists(existing.file_path):
        os.remove(existing.file_path)
    new_record = save_file(new_file, base_folder)
    existing.filename = new_record.filename
    existing.file_path = new_record.file_path
    db.session.commit()

    logging.info(f"File updated: ID={file_id}")
    return existing

def delete_file(file_id):
    file_record = UploadedFile.query.get(file_id)
    if not file_record:
        raise ValueError('File not found')

    if os.path.exists(file_record.file_path):
        os.remove(file_record.file_path)
        logging.info(f"File deleted from storage: {file_record.file_path}")

    db.session.delete(file_record)
    db.session.commit()
    logging.info(f"Record deleted: ID={file_id}")
```

🧩Step 7: Create Controller — `/controllers/file_controller.py`

```
from flask import Blueprint, request, jsonify, send_file
from services.file_service import save_file, update_file, delete_file
from models.uploaded_file import UploadedFile
from app import app
import logging

file_bp = Blueprint('file_bp', __name__)

# 🟢 POST – Upload file
@file_bp.route('/upload', methods=['POST'])
def upload_file():
    try:
        if 'file' not in request.files:
            return jsonify({'message': 'No file uploaded'}), 400

        file = request.files['file']
        folder = f"{app.config['UPLOAD_FOLDER']}/{uuid.uuid4().hex}"  # new
folder per request
        uploaded = save_file(file, folder)
        return jsonify({'message': 'Upload successful', 'id': uploaded.id,
'path': uploaded.file_path}), 201
    except Exception as e:
        logging.error(f"Upload error: {e}")
        return jsonify({'message': str(e)}), 400


# 🟡 GET — Retrieve file info or download
@file_bp.route('/file/<int:file_id>', methods=['GET'])
def get_file(file_id):
    try:
        file = UploadedFile.query.get(file_id)
        if not file:
            return jsonify({'message': 'File not found'}), 404

        # You can test using Postman → GET http://localhost:5000/file/<id>
        return send_file(file.file_path, as_attachment=True)
    except Exception as e:
        logging.error(f"Get error: {e}")
        return jsonify({'message': str(e)}), 400


# 🟠 PUT — Update file
@file_bp.route('/file/<int:file_id>', methods=['PUT'])
def update_existing_file(file_id):
    try:
        if 'file' not in request.files:
            return jsonify({'message': 'No file provided'}), 400

        file = request.files['file']
        updated = update_file(file_id, file, app.config['UPLOAD_FOLDER'])

        return jsonify({'message': 'File updated successfully', 'path':
updated.file_path}), 200
    except Exception as e:
        logging.error(f"Update error: {e}")
        return jsonify({'message': str(e)}), 400


# 🔴 DELETE — Delete file
@file_bp.route('/file/<int:file_id>', methods=['DELETE'])
def delete_existing_file(file_id):
    try:
        delete_file(file_id)
        return jsonify({'message': 'File deleted successfully'}), 200
    except Exception as e:
        logging.error(f"Delete error: {e}")
        return jsonify({'message': str(e)}), 400
```

🧩Step 8: Create `app.py`

```python
import os
import logging
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from config import Config

# Initialize Flask app
app = Flask(__name__)
app.config.from_object(Config)

# Initialize database
db = SQLAlchemy(app)

# Configure logging
os.makedirs(app.config['LOG_FOLDER'], exist_ok=True)
logging.basicConfig(
    filename=f"{app.config['LOG_FOLDER']}/app.log",
    level=logging.INFO,
    format='%(asctime)s [%(levelname)s] %(message)s'
)

# Register blueprints
from controllers.file_controller import file_bp
app.register_blueprint(file_bp)

# Create tables if not exist
with app.app_context():
    db.create_all()

if __name__ == '__main__':
    app.run(debug=True)
```

🧩Step 9: Testing with Postman

Instead of curl, we'll use **Postman** for all endpoints.

🏐POST `/upload`

- Method: **POST**
- Body: form-data → Key: `file`, Type: `File`, choose an image or text file.
- Expected Response: `{"message": "Upload successful", "id": 1, "path": "uploads/<uuid>/<filename>"}`

🏐GET `/file/<id>`

- Method: **GET**
- URL: `http://localhost:5000/file/1`
- Will download the file if found.

🏐PUT `/file/<id>`

- Method: **PUT**
- Body: form-data → Key: `file`, Type: `File`, choose a replacement file.
- Expected: `{"message": "File updated successfully", "path": "newpath"}`

🏐DELETE `/file/<id>`

- Method: **DELETE**
- URL: `http://localhost:5000/file/1`
- Expected: `{"message": "File deleted successfully"}`

🧩Step 10: Logging Output

Logs are stored in `/logs/app.log`. Each operation (upload, update, delete) will write to the log file, for example:

```
2025-10-13 10:32:15 [INFO] File saved: uploads/abcd1234/sample.png
2025-10-13 10:35:21 [INFO] File updated: ID=1
2025-10-13 10:40:12 [INFO] File deleted from storage: uploads/abcd1234/
sample.png
```

Summary

✅**MVC Architecture** — separation between controller, service, and model. ✅**Logging** — automatic file-based logs. ✅**Environment variables** — clean credentials. ✅**Postman testing** — simple and visual.

This structure is ideal for beginners learning Flask with MySQL and proper backend design.