Design Document for File Upload & Management System

# 1. Overview

The project is a **File Upload & Management System** using **microservices architecture**, allowing users to upload, view, and download files via a web interface.

**Technology Stack**:

* **FastAPI** for the backend.
* **Docker** for containerization.
* **Bootstrap** for frontend styling.
* **In-memory storage (file metadata)** (extendable to MongoDB or another database).

# 2. System Architecture

**Architecture Diagram:**

The system is divided into three microservices that handle specific tasks.



**Upload Service**: Manages file uploads and communicates with the metadata service to save metadata.

* **Metadata Service**: Stores metadata about the uploaded files.
* **Download Service**: Manages file download requests.

**Technology Overview**

1. **Frontend**: A simple HTML form styled with Bootstrap for uploading files, displaying them, and downloading them.
2. **Backend**: FastAPI services for handling file uploads, metadata storage, and downloads.
3. **Docker Compose**: Manages the services and ensures each service runs in its own container with a shared uploads directory.

# 3. Services and Components

**Frontend (HTML + JavaScript)**

The user interface provides:

* **File Upload Form**: Allows the user to upload files.
* **Uploaded Files Tab**: Displays the list of uploaded files with a search box for filtering.
* **Download Option**: Provides the ability to download files.

<div class="container">

<h2>File Upload & Management</h2>

<!-- Nav tabs -->

<ul class="nav nav-tabs" id="myTab" role="tablist">

<li class="nav-item" role="presentation">

<button

class="nav-link active"

id="upload-tab"

data-bs-toggle="tab"

data-bs-target="#upload"

type="button"

role="tab"

aria-controls="upload"

aria-selected="true"

>

Upload

</button>

</li>

<li class="nav-item" role="presentation">

<button

class="nav-link"

id="uploads-tab"

data-bs-toggle="tab"

data-bs-target="#uploads"

type="button"

role="tab"

aria-controls="uploads"

aria-selected="false"

>

Uploaded Files

</button>

</li>

</ul>

<!-- Tab content -->

<div class="tab-content" id="myTabContent">

<!-- Upload Form Tab -->

<div

class="tab-pane fade show active"

id="upload"

role="tabpanel"

aria-labelledby="upload-tab"

>

<form id="uploadForm" class="mt-4">

<div class="mb-3">

<label for="fileInput" class="form-label">Choose File</label>

<input

type="file"

class="form-control"

id="fileInput"

name="file"

/>

</div>

<button type="submit" class="btn btn-primary w-100">Upload</button>

</form>

<div id="result" class="mt-3"></div>

</div>

<!-- Uploaded Files Tab -->

<div

class="tab-pane fade"

id="uploads"

role="tabpanel"

aria-labelledby="uploads-tab"

>

<!-- Search box for filtering files -->

<input

type="text"

id="searchInput"

placeholder="Search files..."

onkeyup="filterFiles()"

/>

<div id="uploadedFiles" class="table-container table-responsive">

<table class="table table-striped table-bordered">

<thead class="table-dark">

<tr>

<th>#</th>

<th>File Name</th>

<th>Download</th>

</tr>

</thead>

<tbody id="fileTableBody">

<!-- File rows will be populated dynamically here -->

</tbody>

</table>

</div>

</div>

</div>

</div>

<script>

const uploadForm = document.getElementById("uploadForm");

const resultDiv = document.getElementById("result");

const fileTableBody = document.getElementById("fileTableBody");

const searchInput = document.getElementById("searchInput");

// Function to handle file upload

uploadForm.addEventListener("submit", async function (event) {

event.preventDefault();

const fileInput = document.getElementById("fileInput").files[0];

if (!fileInput) {

resultDiv.innerHTML =

'<p class="text-danger">Please choose a file to upload.</p>';

return;

}

const formData = new FormData();

formData.append("file", fileInput);

try {

const response = await fetch("http://localhost:8001/upload/", {

method: "POST",

body: formData,

});

const result = await response.json();

if (response.ok) {

resultDiv.innerHTML = `<p class="text-success">Upload successful: ${result.message}</p>`;

fetchUploadedFiles(); // Fetch and display uploaded files after successful upload

} else {

resultDiv.innerHTML = `<p class="text-danger">Error: ${result.detail}</p>`;

}

} catch (error) {

resultDiv.innerHTML = `<p class="text-danger">An error occurred while uploading the file: ${error.message}</p>`;

}

});

// Function to fetch and display uploaded files

async function fetchUploadedFiles() {

try {

const response = await fetch("http://localhost:8001/uploads");

const files = await response.json();

// Clear the current table body

fileTableBody.innerHTML = "";

// Assuming files is the list of filenames retrieved from the server

files.forEach((file, index) => {

const row = document.createElement("tr");

row.innerHTML = `

<td>${index + 1}</td>

<td>${file}</td>

<td><a href="http://localhost:8003/download/${file}" class="btn btn-success btn-sm">Download</a></td>

`;

fileTableBody.appendChild(row);

});

} catch (error) {

console.error("Error fetching uploaded files:", error);

}

}

// Function to filter files based on search input

function filterFiles() {

const filter = searchInput.value.toLowerCase();

const rows = fileTableBody.getElementsByTagName("tr");

Array.from(rows).forEach((row) => {

const fileName = row.getElementsByTagName("td")[1].textContent;

if (fileName.toLowerCase().includes(filter)) {

row.style.display = "";

} else {

row.style.display = "none";

}

});

}

// Fetch uploaded files on page load

fetchUploadedFiles();

</script>

**Upload Service**

* Receives files via POST /upload/.
* Saves the file to the shared uploads directory.
* Sends metadata to the metadata service.

**Code Snippet: Upload Service (upload\_service.py)**

from fastapi import FastAPI, File, UploadFile, HTTPException

from fastapi.middleware.cors import CORSMiddleware

import os

import requests

app = FastAPI()

app.add\_middleware(

CORSMiddleware,

allow\_origins=["\*"], # Allow requests from any ori`gin (use specific URLs in production)

allow\_credentials=True,

allow\_methods=["\*"],

allow\_headers=["\*"],

)

UPLOAD\_DIRECTORY = os.path.join(os.getcwd(), "uploads/")

# Create the uploads directory if it doesn't exist

if not os.path.exists(UPLOAD\_DIRECTORY):

os.makedirs(UPLOAD\_DIRECTORY)

@app.post("/upload/")

async def upload\_file(file: UploadFile = File(...)):

try:

file\_location = f"{UPLOAD\_DIRECTORY}{file.filename}"

print(f"Saving file to: {file\_location}")

# Ensure the directory exists

if not os.path.exists(UPLOAD\_DIRECTORY):

print(f"Directory {UPLOAD\_DIRECTORY} does not exist, creating it now.")

os.makedirs(UPLOAD\_DIRECTORY)

else:

print(f"Directory {UPLOAD\_DIRECTORY} already exists.")

# Save the file to disk

with open(file\_location, "wb") as buffer:

buffer.write(await file.read())

print(f"File saved successfully at {file\_location}")

# Check if file exists after saving

if os.path.exists(file\_location):

print(f"File {file\_location} exists after saving.")

else:

print(f"File {file\_location} does NOT exist after saving!")

# Metadata service request (assuming it's working)

metadata\_response = requests.post(

"http://metadata\_service:8002/save\_metadata/",

json={"filename": file.filename, "location": file\_location}

)

if metadata\_response.status\_code != 200:

raise HTTPException(status\_code=500, detail="Failed to save metadata")

return {"message": "File uploaded successfully", "metadata": metadata\_response.json()}

except Exception as e:

print(f"Error: {str(e)}")

raise HTTPException(status\_code=500, detail=f"An error occurred: {str(e)}")

@app.get("/uploads")

async def get\_uploaded\_files():

try:

files = os.listdir(UPLOAD\_DIRECTORY)

return files

except Exception as e:

raise HTTPException(status\_code=500, detail=f"Error occurred: {str(e)}")

@app.get("/download/{filename}")

async def download\_file(filename: str):

file\_path = os.path.join(UPLOAD\_DIRECTORY, filename)

if os.path.exists(file\_path):

return FileResponse(path=file\_path, filename=filename, media\_type='application/octet-stream')

else:

raise HTTPException(status\_code=404, detail="File not found")

**Metadata Service**

* Stores metadata (filename, location) using an in-memory dictionary.
* Can be extended to use a database.

**Code Snippet: Metadata Service (metadata\_service.py)**

# from fastapi import FastAPI

# from pymongo import MongoClient

# app = FastAPI()

# # MongoDB setup

# client = MongoClient("mongodb://localhost:27017/")

# db = client['file\_storage']

# metadata\_collection = db['metadata']

# @app.post("/save\_metadata/")

# def save\_metadata(file\_name: str, file\_location: str):

# metadata = {"file\_name": file\_name, "file\_location": file\_location}

# metadata\_collection.insert\_one(metadata)

# return {"message": "Metadata saved successfully"}

# @app.get("/get\_metadata/{file\_name}")

# def get\_metadata(file\_name: str):

# metadata = metadata\_collection.find\_one({"file\_name": file\_name})

# if metadata:

# return metadata

# return {"error": "File not found"}

from fastapi import FastAPI, HTTPException

from pydantic import BaseModel

app = FastAPI()

# In-memory metadata store (replace with a database in production)

file\_metadata = {}

class Metadata(BaseModel):

filename: str

location: str

@app.post("/save\_metadata/")

async def save\_metadata(metadata: Metadata):

if metadata.filename in file\_metadata:

raise HTTPException(status\_code=400, detail="File metadata already exists")

file\_metadata[metadata.filename] = metadata.location

return {"message": "Metadata saved successfully", "metadata": metadata.dict()}

# 4. Sequence of Operations

**Step-by-Step Flow:**

1. **File Upload**:
   * The user selects a file and clicks **Upload**.
   * The file is sent to the **Upload Service** (POST /upload/).
   * The file is saved to the shared uploads directory.
   * The **Metadata Service** is called to save the file metadata.
2. **Viewing Uploaded Files**:
   * The frontend fetches the list of uploaded files from the **Upload Service** (GET /uploads).
   * The uploaded files are displayed in a table with a download option.
3. **File Download**:
   * The user clicks the **Download** button.
   * The **Download Service** (GET /download/{filename}) retrieves the file and sends it to the user.

# 5. Error Handling

* **File Upload**: If the upload fails or the metadata can't be saved, an error message is returned to the frontend.
* **File Download**: If the requested file is not found, the download service returns a 404 error.

# **6.** Extensibility

* **Database Integration**: The current in-memory storage for metadata can be replaced with MongoDB or PostgreSQL.
* **Authentication**: User authentication and authorization can be added to secure file access.
* **File Management**: Features like file deletion and versioning can be integrated.

# 7. Deployment

**Steps to Deploy:**

1. **Install Docker and Docker Compose**.
2. **Clone the Repository**.
3. **Run the Application**:

**Access the Services**:

* + Upload Service: http://localhost:8001
  + Metadata Service: http://localhost:8002
  + Download Service: http://localhost:8003