Code Documentation

1. Upload\_service.py
2. from fastapi import FastAPI, File, UploadFile, HTTPException
3. from fastapi.middleware.cors import CORSMiddleware
4. import os
5. import requests
6. app = FastAPI()
7. app.add\_middleware(
8. CORSMiddleware,
9. allow\_origins=["\*"], # Allow requests from any ori`gin (use specific URLs in production)
10. allow\_credentials=True,
11. allow\_methods=["\*"],
12. allow\_headers=["\*"],
13. )
14. UPLOAD\_DIRECTORY = os.path.join(os.getcwd(), "uploads/")
15. # Create the uploads directory if it doesn't exist
16. if not os.path.exists(UPLOAD\_DIRECTORY):
17. os.makedirs(UPLOAD\_DIRECTORY)
18. @app.post("/upload/")
19. async def upload\_file(file: UploadFile = File(...)):
20. try:
21. file\_location = f"{UPLOAD\_DIRECTORY}{file.filename}"
22. print(f"Saving file to: {file\_location}")
24. # Ensure the directory exists
25. if not os.path.exists(UPLOAD\_DIRECTORY):
26. print(f"Directory {UPLOAD\_DIRECTORY} does not exist, creating it now.")
27. os.makedirs(UPLOAD\_DIRECTORY)
28. else:
29. print(f"Directory {UPLOAD\_DIRECTORY} already exists.")
31. # Save the file to disk
32. with open(file\_location, "wb") as buffer:
33. buffer.write(await file.read())
34. print(f"File saved successfully at {file\_location}")
35. # Check if file exists after saving
36. if os.path.exists(file\_location):
37. print(f"File {file\_location} exists after saving.")
38. else:
39. print(f"File {file\_location} does NOT exist after saving!")
40. # Metadata service request (assuming it's working)
41. metadata\_response = requests.post(
42. "http://metadata\_service:8002/save\_metadata/",
43. json={"filename": file.filename, "location": file\_location}
44. )
45. if metadata\_response.status\_code != 200:
46. raise HTTPException(status\_code=500, detail="Failed to save metadata")
47. return {"message": "File uploaded successfully", "metadata": metadata\_response.json()}
48. except Exception as e:
49. print(f"Error: {str(e)}")
50. raise HTTPException(status\_code=500, detail=f"An error occurred: {str(e)}")
51. @app.get("/uploads")
52. async def get\_uploaded\_files():
53. try:
54. files = os.listdir(UPLOAD\_DIRECTORY)
55. return files
56. except Exception as e:
57. raise HTTPException(status\_code=500, detail=f"Error occurred: {str(e)}")
58. @app.get("/download/{filename}")
59. async def download\_file(filename: str):
60. file\_path = os.path.join(UPLOAD\_DIRECTORY, filename)
61. if os.path.exists(file\_path):
62. return FileResponse(path=file\_path, filename=filename, media\_type='application/octet-stream')
63. else:
64. raise HTTPException(status\_code=404, detail="File not found")

**Documentation for upload\_service.py**

* **App Initialization**: The FastAPI() instance initializes the API, setting up routes.
* **CORS Middleware**: The CORS middleware is added to allow cross-origin requests. In production, you can restrict allowed origins.
* **Directory Setup**: A local directory uploads/ is created if it doesn't exist to store the uploaded files.
* **Upload Endpoint (/upload/)**:
  + Accepts a file upload using UploadFile.
  + Saves the file to the uploads directory.
  + Sends metadata about the uploaded file to the metadata\_service.
* **Get Uploaded Files (/uploads)**:
  + Lists all files in the uploads/ directory.
* **Download File (/download/{filename})**:
  + Returns a file as a download if it exists in the uploads/ directory.

**2. download\_service.py**

from fastapi import FastAPI, HTTPException

from fastapi.responses import FileResponse

import os

from urllib.parse import unquote

app = FastAPI()

# Ensure the upload directory is the same across services

UPLOAD\_DIRECTORY = "/app/uploads"

# Ensure the uploads directory exists

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

os.makedirs(UPLOAD\_DIRECTORY)

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

async def download\_file(filename: str):

# Decode the filename in case it contains special characters

decoded\_filename = unquote(filename)

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

# Check if the file exists

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

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

else:

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

**Documentation for download\_service.py**

* **App Initialization**: The FastAPI() instance is initialized.
* **Download Endpoint (/download/{filename})**:
  + Decodes the filename (to handle special characters).
  + Fetches the file from the uploads/ directory and returns it if found.

**3. 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()}

**Documentation for metadata\_service.py**

* **App Initialization**: The FastAPI() instance initializes the API.
* **In-memory Metadata Store**:
  + Metadata about uploaded files is stored in memory.
  + This can be extended to store metadata in a persistent database (e.g., MongoDB).
* **Save Metadata (/save\_metadata/)**:
  + Saves the filename and its location.
  + Throws an error if the metadata for the file already exists.

**4. docker-compose.yml**

version: "3"

services:

upload\_service:

build: ./upload\_service

ports:

- "8001:8001"

volumes:

- ./uploads:/app/uploads

metadata\_service:

build: ./metadata\_service

ports:

- "8002:8002"

volumes:

- ./metadata:/app/metadata

download\_service:

build: ./download\_service

ports:

- "8003:8003"

volumes:

- ./uploads:/app/uploads # Make sure this is consistent with upload service

**Documentation for docker-compose.yml**

* **Upload Service**:
  + Builds and runs the upload service on port 8001.
  + Mounts the uploads directory as a volume for file storage.
* **Metadata Service**:
  + Builds and runs the metadata service on port 8002.
* **Download Service**:
  + Builds and runs the download service on port 8003.
  + Shares the uploads/ directory for file access.

**5. index.html (Frontend)**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>File Upload</title>

<!-- Bootstrap CSS (for styling) -->

<link

href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"

rel="stylesheet"

/>

<style>

body {

background-color: #f4f7fc;

font-family: "Roboto", sans-serif;

}

.container {

max-width: 750px;

margin: 50px auto;

padding: 40px;

background: #ffffff;

border-radius: 12px;

box-shadow: 0 8px 30px rgba(0, 0, 0, 0.05);

}

h2 {

font-size: 2rem;

text-align: center;

margin-bottom: 30px;

color: #333;

font-weight: 600;

}

.form-label {

font-size: 1.1rem;

font-weight: 500;

color: #555;

}

.btn-primary {

background-color: #007bff;

border-color: #007bff;

transition: background-color 0.3s ease-in-out;

padding: 10px;

font-size: 1.1rem;

}

.btn-primary:hover {

background-color: #0056b3;

border-color: #004494;

}

.btn-secondary {

background-color: #6c757d;

border-color: #6c757d;

margin-top: 15px;

width: 100%;

padding: 10px;

font-size: 1.1rem;

}

.btn-secondary:hover {

background-color: #565e64;

}

.text-success {

font-weight: bold;

color: #28a745;

}

.text-danger {

font-weight: bold;

color: #dc3545;

}

#result {

margin-top: 20px;

}

.nav-tabs {

border-bottom: 2px solid #dee2e6;

}

.nav-link {

border: none;

border-radius: 0;

font-size: 1.1rem;

font-weight: 500;

color: #555;

}

.nav-link.active {

color: #007bff;

border-bottom: 3px solid #007bff;

background-color: transparent;

}

.table-container {

margin-top: 30px;

}

.table {

margin-top: 30px;

box-shadow: 0 4px 12px rgba(0, 0, 0, 0.05);

border-radius: 8px;

overflow: hidden;

}

.table th,

.table td {

vertical-align: middle;

padding: 15px;

font-size: 1rem;

}

.table-striped tbody tr:nth-of-type(odd) {

background-color: #f9f9f9;

}

.btn-sm {

padding: 8px 12px;

font-size: 0.9rem;

border-radius: 4px;

}

.table-dark {

background-color: #343a40;

color: #fff;

}

/\* Styles for the search input \*/

#searchInput {

margin-top: 20px;

margin-bottom: 10px;

padding: 10px;

border-radius: 5px;

width: 100%;

border: 1px solid #ced4da;

font-size: 1rem;

}

</style>

</head>

<body>

<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>

<!-- JavaScript for handling the file upload and fetching uploaded files -->

<script></script>

<!-- Bootstrap JS -->

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></script>

</body>

</html>

**Documentation for index.html**

* **HTML Structure**:
  + A simple interface with tabs for uploading and viewing files.
  + The uploaded files are displayed in a table format.
  + Bootstrap is used for styling and responsiveness.
* **JavaScript Functions**:
  + fetchUploadedFiles(): Fetches uploaded files from the server and displays them.
  + filterFiles(): Filters the displayed files based on a search query.

A screenshot of a computer program

Description automatically generated

# Design Diagram (Architecture)