**Document Management Application - DOCZY**

**Synopsis**

DOCZY is a document management and retrieval system designed to streamline document storage, secure uploads, and intelligent search functionalities. It is built using FastAPI and React and leverages AWS S3 for cloud storage and PostgreSQL for metadata management.

**Table of Contents**

1. Overview
2. Features
3. Architecture
4. Deployment
5. Technical Specifications
6. Code Snippets
7. API Documentation

**1. Overview**

DOCZY provides an integrated environment for document management, combining secure cloud storage with natural language query capabilities. It uses Docker for containerized deployment, making it easy to set up and scale.

**2. Features**

* **Secure Document Upload**: Documents are uploaded, stored in AWS S3, and indexed for searching.
* **NLP-Powered Search**: Allows natural language queries to retrieve documents and relevant sections.
* **User Authentication**: Supports secure access using OAuth2 and hashed passwords.
* **Database Management**: PostgreSQL for metadata and user data.
* **Responsive Frontend**: React-based UI compatible with various devices.

**3. Architecture**

**Frontend**

The frontend is a React-based single-page application with components for file upload, document querying, and responsive design.

**Backend**

The backend is a FastAPI application that handles API requests, processes file storage with AWS S3, performs NLP-based queries on documents, and manages metadata storage.

**Storage and Data Processing**

* **AWS S3** for document storage.
* **PostgreSQL** for metadata storage.
* **Haystack** for NLP-based document querying and retrieval.

**4. Deployment**

**Docker Compose**

DOCZY uses Docker for isolated services. The docker-compose.yml configures the frontend, backend, and PostgreSQL database.

**docker-compose.yml**

yaml

Copy code

services:

backend:

build: .

ports:

- "8000:8000"

environment:

DATABASE\_URL: "postgresql://user1:admin@db/doczy"

AWS\_ACCESS\_KEY\_ID: ${AWS\_ACCESS\_KEY\_ID}

AWS\_SECRET\_ACCESS\_KEY: ${AWS\_SECRET\_ACCESS\_KEY}

S3\_BUCKET\_NAME: ${S3\_BUCKET\_NAME}

depends\_on:

- db

frontend:

build: ./frontend

ports:

- "3000:3000"

db:

image: postgres:17

environment:

POSTGRES\_USER: user1

POSTGRES\_PASSWORD: admin

POSTGRES\_DB: doczy

volumes:

- postgres\_data:/var/lib/postgresql/data

volumes:

postgres\_data:

**5. Technical Specifications**

* **Frontend**: React with CSS for responsive UI.
* **Backend**: FastAPI with SQLAlchemy ORM, Boto3 for AWS S3 interaction.
* **Database**: PostgreSQL for user and document metadata.
* **NLP and Processing**: Haystack for document retrieval.

**6. Code Snippets**

**Frontend Code**

**App.js - Main Application Component**

javascript

Copy code

import React from "react";

import UploadForm from "./components/UploadForm";

import QueryComponent from "./components/QueryComponent";

import "./App.css";

function App() {

return (

<div className="app-container">

<header className="app-header">

<h1>DOCZY</h1>

<h3>A Document Management Inventory</h3>

<p className="app-description">

Effortlessly manage, upload, and retrieve documents with ease.

</p>

</header>

<div className="content-container">

<UploadForm />

<QueryComponent />

</div>

</div>

);

}

export default App;

**UploadForm.js - Document Upload Component**

javascript

Copy code

import React, { useState } from "react";

import axios from "axios";

function UploadForm() {

const [file, setFile] = useState(null);

const handleFileChange = (e) => setFile(e.target.files[0]);

const handleUpload = async () => {

const formData = new FormData();

formData.append("file", file);

await axios.post("/upload/", formData);

alert("File uploaded successfully");

};

return (

<div className="card">

<h2>Upload Document</h2>

<input type="file" onChange={handleFileChange} />

<button onClick={handleUpload}>Upload</button>

</div>

);

}

export default UploadForm;

**Backend Code**

**main.py - FastAPI Endpoints**

python

Copy code

from fastapi import FastAPI, Depends, HTTPException, UploadFile, File

from sqlalchemy.orm import Session

from database import SessionLocal, engine, Base

import file\_storage # Handles S3 uploads

import models

import nlp\_processing # NLP-based document querying

from typing import List

app = FastAPI()

# Create tables if they don't exist

Base.metadata.create\_all(bind=engine)

# Dependency to retrieve DB session

def get\_db():

db = SessionLocal()

try:

yield db

finally:

db.close()

@app.post("/upload/")

async def upload\_document(file: UploadFile, db: Session = Depends(get\_db)):

content = await file.read()

file\_url = file\_storage.upload\_to\_s3(content, file.filename)

doc = models.Document(file\_url=file\_url)

db.add(doc)

db.commit()

return {"file\_url": file\_url}

@app.post("/query/")

def query\_document(query: str, db: Session = Depends(get\_db)):

documents = db.query(models.Document).all()

answer = nlp\_processing.query\_with\_rag(query, documents)

return {"answer": answer}

@app.get("/documents/", response\_model=List[models.Document])

def get\_documents(db: Session = Depends(get\_db)):

return db.query(models.Document).all()

**file\_storage.py - File Upload to AWS S3**

python

Copy code

import boto3

import os

def upload\_to\_s3(file\_content, filename):

s3 = boto3.client(

"s3",

aws\_access\_key\_id=os.getenv("AWS\_ACCESS\_KEY\_ID"),

aws\_secret\_access\_key=os.getenv("AWS\_SECRET\_ACCESS\_KEY"),

region\_name=os.getenv("AWS\_REGION")

)

bucket\_name = os.getenv("S3\_BUCKET\_NAME")

s3.put\_object(Bucket=bucket\_name, Key=filename, Body=file\_content)

return f"https://{bucket\_name}.s3.amazonaws.com/{filename}"

**nlp\_processing.py - Document Querying**

python

Copy code

from haystack.document\_stores import InMemoryDocumentStore

from haystack.nodes import BM25Retriever

document\_store = InMemoryDocumentStore()

retriever = BM25Retriever(document\_store=document\_store)

def index\_documents(documents):

document\_store.write\_documents(documents)

def query\_with\_rag(query):

results = retriever.retrieve(query)

return [result.content for result in results]

**7. API Documentation**

**Endpoints**

1. **Upload Document**
   * **Endpoint**: /upload/
   * **Method**: POST
   * **Description**: Accepts file upload, stores it in S3, and saves metadata.
   * **Response**: JSON with file\_url.
2. **Query Document**
   * **Endpoint**: /query/
   * **Method**: POST
   * **Description**: Accepts a natural language query and returns matching content.
   * **Response**: JSON with answer.
3. **List Documents**
   * **Endpoint**: /documents/
   * **Method**: GET
   * **Description**: Retrieves all stored documents’ metadata.
   * **Response**: JSON array of documents.