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

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.