Documentation of the code

main.py file (code)

```
from fastapi import FastAPI, UploadFile, File, Request, Form, HTTPException
from fastapi.responses import HTMLResponse, RedirectResponse
from fastapi.templating import Jinja2Templates
from sqlalchemy import create_engine, Column, Integer, Text
from sqlalchemy.ext.declarative import declarative_base
from databases import Database
from typing import List
import uvicorn
import tempfile
import os
import logging
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from PyPDF2 import PdfReader
import nltk

# Download the required NLTK tokenizer resource
nltk.download('punkt')
from nltk.tokenize import sent_tokenize

# Set up logging configuration to display info-level logs
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Initialize FastAPI app
app = FastAPI()
# Set up template directory for HTML responses
templates = Jinja2Templates(directory="templates")

# Database configuration
DATABASE_URL = "sqlite:///./documents.db"
database = Database(DATABASE_URL)
Base = declarative_base()

# Define a Document model to store PDF content in the database
class Document(Base):
    __tablename__ = "documents"
    id = Column(Integer, primary_key=True, index=True)
    content = Column(Text)
```

```
# Create database engine and tables if they don’t exist
engine = create_engine(DATABASE_URL, connect_args={"check_same_thread": False})
Base.metadata.create_all(bind=engine)

# Dictionary to store document embeddings and vectorizers
document_embeddings = {}

# FastAPI event to handle database connection on app startup
@app.on_event("startup")
async def startup():
    await database.connect()
    logger.info("Database connected successfully")

# FastAPI event to handle database disconnection on app shutdown
@app.on_event("shutdown")
async def shutdown():
    await database.disconnect()
    logger.info("Database disconnected")

# Function to extract text from PDF files
def extract_text_from_pdf(file_path):
    try:
        reader = PdfReader(file_path)
        text = []
        # Extract text from each page in the PDF
        for page in reader.pages:
            page_text = page.extract_text()
            if page_text:
                text.append(page_text)
        # Raise error if no text is found in PDF
        if not text:
            raise ValueError("No text could be extracted from the PDF")
        return "\n".join(text)
    except Exception as e:
        logger.error(f"Error processing PDF: {str(e)}")
        raise HTTPException(status_code=400,
                          detail=f"Error processing PDF: {str(e)}")

# Split text into smaller chunks to process for embeddings
def create_text_chunks(text, max_chunk_size=500):
    sentences = sent_tokenize(text)
    chunks = []
    current_chunk = ''
    # Combine sentences into chunks up to a max length
    for sentence in sentences:
        if len(current_chunk) + len(sentence) <= max_chunk_size:
            current_chunk += ' ' + sentence
        else:
            chunks.append(current_chunk.strip())
            current_chunk = sentence
    # Append the last chunk if it exists
    if current_chunk:
        chunks.append(current_chunk.strip())
    return chunks

# Compute TF-IDF embeddings for text chunks
def compute_embeddings(text_chunks: List[str]):
    try:
        if not text_chunks:
            raise ValueError("No text chunks provided for embedding computation")
        vectorizer = TfidfVectorizer()
        embeddings = vectorizer.fit_transform(text_chunks)
        return embeddings, vectorizer
    except Exception as e:
        logger.error(f"Error computing embeddings: {str(e)}")
        raise HTTPException(status_code=500,
                        detail=f"Error computing embeddings: {str(e)}")

# Main page endpoint
@app.get("/", response_class=HTMLResponse)
async def read_root(request: Request):
    return templates.TemplateResponse("index.html", {"request": request})

# Endpoint to upload PDF files
@app.post("/upload_pdf/")
async def upload_pdf(request: Request, file: UploadFile = File(...)):
    try:
        # Check if file is a PDF
        if not file.filename.lower().endswith('.pdf'):
            raise HTTPException(status_code=400, detail="Only PDF files are allowed")

        # Save the PDF file temporarily
        with tempfile.NamedTemporaryFile(delete=False, suffix=".pdf") as tmp:
            content = await file.read()
            if not content:
                raise HTTPException(status_code=400, detail="Empty file uploaded")
            tmp.write(content)
            tmp_path = tmp.name

        logger.info(f"Processing PDF file: {file.filename}")

        # Extract text from the PDF
        text = extract_text_from_pdf(tmp_path)

        # Remove the temporary file after extraction
        os.unlink(tmp_path)

        # Check if text was successfully extracted
        if not text.strip():
            raise HTTPException(status_code=400, detail="No text could be extracted from the
PDF")

        # Split text into chunks
        text_chunks = create_text_chunks(text)
        logger.info(f"Created {len(text_chunks)} text chunks")

        # Compute embeddings and save the vectorizer for future queries
        embeddings, doc_vectorizer = compute_embeddings(text_chunks)

        # Save document content to database and store its embeddings
        async with database.transaction():
            query = Document.__table__.insert().values(content=text)
            document_id = await database.execute(query)
            document_embeddings[document_id] = {
                'text_chunks': text_chunks,
                'embeddings': embeddings,
                'vectorizer': doc_vectorizer
            }

        logger.info(f"Document saved with ID: {document_id}")

        # Redirect to main page and set a cookie with document ID
        response = RedirectResponse(url="/", status_code=303)
        response.set_cookie(key="document_id", value=str(document_id))
        return response

    except HTTPException as he:
        raise he
    except Exception as e:
        logger.error(f"Error uploading PDF: {str(e)}")
        raise HTTPException(status_code=500, detail=str(e))
```

```
# Endpoint to ask questions about the document content
@app.post("/ask_question/")
async def ask_question(request: Request, document_id: int = Form(...), question: str =
Form(...)):
    try:
        response = await process_question(document_id, question)
        logger.info(f"Processed question for document {document_id}")
        return templates.TemplateResponse("index.html", {
            "request": request,
            "result": response,
            "document_id": document_id,
            "question": question
        })
    except Exception as e:
        logger.error(f"Error processing question: {str(e)}")
        return templates.TemplateResponse("index.html", {
            "request": request,
            "error": str(e)
        })

# Function to process question and retrieve relevant text from document
async def process_question(document_id: int, question: str):
    if not question.strip():
        raise HTTPException(status_code=400, detail="Question cannot be empty")

    # Check if document exists in memory
    if document_id not in document_embeddings:
        raise HTTPException(status_code=404, detail="Document not found. Please upload a PDF
first.")

    try:
        # Retrieve document embeddings and vectorizer
        text_chunks = document_embeddings[document_id]['text_chunks']
        embeddings = document_embeddings[document_id]['embeddings']
        doc_vectorizer = document_embeddings[document_id]['vectorizer']

        # Transform the question to an embedding
        question_embedding = doc_vectorizer.transform([question])
        # Calculate cosine similarity between question and document embeddings
        similarities = cosine_similarity(question_embedding, embeddings)[0]

        # Get top 3 most similar chunks based on similarity scores
        top_indices = similarities.argsort()[-3:][::-1]
        # Combine top chunks to create a response
        response = "\n\n".join([text_chunks[i] for i in top_indices])

        return response

    except Exception as e:
        logger.error(f"Error in question processing: {str(e)}")
        raise HTTPException(status_code=500, detail=f"Error processing question: {str(e)}")

# Start the FastAPI app using Uvicorn server if this script is run directly
if __name__ == "__main__":
    uvicorn.run("main:app", host="0.0.0.0", port=8000)
```

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>PDF Question Answering System</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <!-- Tailwind CSS for styling -->
    <link href="https://cdn.jsdelivr.net/npm/tailwindcss@2.2.19/dist/tailwind.min.css"
rel="stylesheet">
    <!-- Alpine.js for handling reactive data binding -->
    <script src="https://cdn.jsdelivr.net/npm/alpinejs@2.8.2/dist/alpine.min.js" defer></script>
</head>
<body class="bg-gray-100 min-h-screen" x-data="appData()">
    <!-- Navigation Bar -->
    <nav class="bg-white shadow-md">
        <div class="container mx-auto px-4 py-4 flex justify-between items-center">
            <div class="text-2xl font-bold text-gray-800">PDF QA System</div>
            <div>
                <a href="#" class="text-gray-600 hover:text-gray-800 mr-4">Home</a>
                <a href="#" class="text-gray-600 hover:text-gray-800">About</a>
            </div>
        </div>
    </nav>

    <div class="container mx-auto px-4 py-8 max-w-4xl">
        <!-- Error Alert if there's an issue with uploading or processing -->
        <template x-if="showError">
```

```
          <div class="bg-red-100 border border-red-400 text-red-700 px-4 py-3 rounded relative
mb-4" role="alert">
             <strong class="font-bold">Error!</strong>
             <span class="block sm:inline" x-text="errorMessage"></span>
             <!-- Close button to dismiss the error message -->
             <button class="absolute top-0 right-0 px-4 py-3" @click="showError = false">
                <svg class="h-6 w-6 text-red-500" role="button"
xmlns="http://www.w3.org/2000/svg" viewBox="0 0 20 20">
                   <title>Close</title>
                   <path d="M14.348 14.849a1.2 1.2 0 0 1-1.697 0L10 11.819l-2.651 3.029a1.2 1.2
0 1 1-1.697-1.697l2.758-3.15-2.759-3.152a1.2 1.2 0 1 1 1.697-1.697L10 8.183l2.651-3.031a1.2
1.2 0 1 1 1.697 1.697l-2.758 3.152 2.758 3.15a1.2 1.2 0 0 1 0 1.698z"/>
                </svg>
             </button>
          </div>
       </template>

       <!-- Display document ID if a document is active -->
       <template x-if="documentId">
          <div class="bg-green-100 border border-green-400 text-green-700 px-4 py-3 rounded
relative mb-4">
             <span class="block sm:inline">Active Document ID: <strong
x-text="documentId"></strong></span>
          </div>
       </template>

       <!-- Upload Section for PDF Documents -->
       <div class="bg-white rounded-lg shadow-md p-6 mb-8">
          <h2 class="text-2xl font-semibold text-gray-700 mb-4">Upload PDF Document</h2>
          <form action="/upload_pdf/" method="post" enctype="multipart/form-data"
               @submit="isUploading = true"
               class="space-y-4">
             <div class="flex items-center justify-center w-full">
                <!-- File input with label for drag-and-drop style -->
                <label class="flex flex-col w-full h-32 border-4 border-dashed hover:bg-gray-100
hover:border-gray-300 cursor-pointer">
                   <div class="flex flex-col items-center justify-center pt-7">
                      <!-- Upload icon -->
                      <svg xmlns="http://www.w3.org/2000/svg" class="w-12 h-12 text-gray-400
group-hover:text-gray-600" viewBox="0 0 20 20" fill="currentColor">
                         <path fill-rule="evenodd" d="M4 4a2 2 0 012-2h4.586A2 2 0 0112
2.586L15.414 6A2 2 0 0116 7.414V16a2 2 0 01-2 2H6a2 2 0 01-2-2V4zm2 6a1 1 0 011-1h6a1 1
0 110 2H7a1 1 0 01-1-1zm1 3a1 1 0 100 2h6a1 1 0 100-2H7z" clip-rule="evenodd"/>
                      </svg>
```

```
                        <!-- Display the selected file name -->
                        <p class="pt-1 text-sm tracking-wider text-gray-400 group-hover:text-gray-600"
x-text="fileName || 'Select a PDF file'"></p>
                    </div>
                    <input type="file" name="file" accept=".pdf" required class="opacity-0"
                           @change="fileName = $event.target.files[0].name">
                </label>
            </div>
            <div class="flex justify-center">
                <!-- Submit button with loading animation during upload -->
                <button type="submit"
                        class="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded
focus:outline-none focus:shadow-outline disabled:opacity-50"
                        :disabled="isUploading"
                        x-html="isUploading ? '<span class=\'animate-pulse\'>Uploading...</span>' :
'Upload PDF'">
                </button>
            </div>
        </form>
    </div>

    <!-- Question Section visible when a document is active -->
    <template x-if="documentId">
        <div class="bg-white rounded-lg shadow-md p-6">
            <h2 class="text-2xl font-semibold text-gray-700 mb-4">Ask a Question</h2>
            <form action="/ask_question/" method="post" @submit="isAsking = true"
class="space-y-4">
                <input type="hidden" name="document_id" :value="documentId">

                <div class="space-y-2">
                    <!-- Textarea for entering a question -->
                    <label for="question" class="block text-sm font-medium text-gray-700">Your
Question</label>
                    <textarea name="question" id="question" required
                              class="w-full px-3 py-2 border border-gray-300 rounded-md
focus:outline-none focus:ring-2 focus:ring-blue-500"
                              rows="3"
                              placeholder="Type your question here...">{{ question if question else ''
}}</textarea>
                </div>

                <div class="flex justify-center">
                    <!-- Submit button with loading animation during processing -->
                    <button type="submit"
```

```
                                class="bg-green-500 hover:bg-green-700 text-white font-bold py-2 px-4
rounded focus:outline-none focus:shadow-outline disabled:opacity-50"
                                :disabled="isAsking"
                                x-html="isAsking ? '<span class=\'animate-pulse\'>Finding
Answer...</span>' : 'Ask Question'">
                        </button>
                    </div>
                </form>

                <!-- Display the answer after question processing -->
                {% if result %}
                <div class="mt-8 p-4 bg-gray-50 rounded-lg border border-gray-200">
                    <h3 class="text-xl font-semibold text-gray-700 mb-2">Answer</h3>
                    <div class="prose max-w-none">
                        <p class="text-gray-600 whitespace-pre-wrap">{{ result }}</p>
                    </div>
                </div>
                {% endif %}
            </div>
        </template>
    </div>

    <!-- Footer section with project info -->
    <footer class="bg-white mt-8">
        <div class="container mx-auto px-4 py-4 text-center text-gray-600">
            &copy; {{ current_year if current_year else '2024' }} Mustansir Project. All rights
reserved.
        </div>
    </footer>

    <!-- Alpine.js Component for reactive state management -->
    <script>
        function appData() {
            return {
                isUploading: false,  // Tracks if a file is being uploaded
                fileName: '',  // Holds the name of the uploaded file
                isAsking: false,  // Tracks if a question is being processed
                showError: {{ 'true' if error else 'false' }},  // Toggles error display
                errorMessage: '{{ error | safe if error else '' }}',  // Holds error message content
                documentId: '{{ request.cookies.get('document_id', '') }}',  // Stores active document ID
                question: '{{ question | safe if question else '' }}'  // Holds current question text
            }
        }
    </script>
```

```
</body>
</html>
```

Poetry.lock

```
# This file is automatically @generated by Poetry 1.5.4 and should not be changed by hand.
package = []

[metadata]
lock-version = "2.0"
python-versions = ">=3.10.0,<3.12"
content-hash = "c72ba1872668238cc9b2c02a1118c7b8f9049473b63bbad4ee2bd46365be7ce2"
```

Requirement.txt

```
fastapi==0.103.2
python-multipart==0.0.6
sqlalchemy==1.4.42
databases==0.8.0
scikit-learn==1.2.2
numpy==1.24.3
PyPDF2==3.0.1
nltk==3.8.1
uvicorn==0.22.0
jinja2==3.1.2
aiosqlite==0.19.0
pydantic==1.10.12
slowapi==0.1.4
```

.replit

```
entrypoint = "main.py"
modules = ["python-3.11"]

[nix]
channel = "stable-24_05"

[unitTest]
language = "python3"

[gitHubImport]
requiredFiles = [".replit", "replit.nix"]
```

```
[deployment]
run = ["python3", "main.py"]
deploymentTarget = "cloudrun"

[[ports]]
localPort = 8000
externalPort = 80
```

.gitignore

```
# Byte-compiled / optimized / DLL files
__pycache__/
*.py[cod]
*$py.class

# C extensions
*.so

# Distribution / packaging
.Python
build/
develop-eggs/
dist/
downloads/
eggs/
.eggs/
lib/
lib64/
parts/
sdist/
var/
wheels/
share/python-wheels/
*.egg-info/
.installed.cfg
*.egg
MANIFEST

# PyInstaller
#  Usually these files are written by a python script from a template
#  before PyInstaller builds the exe, so as to inject date/other infos into it.
*.manifest
```

```
*.spec

# Installer logs
pip-log.txt
pip-delete-this-directory.txt

# Unit test / coverage reports
htmlcov/
.tox/
.nox/
.coverage
.coverage.*
.cache
nosetests.xml
coverage.xml
*.cover
*.py,cover
.hypothesis/
.pytest_cache/
cover/

# Translations
*.mo
*.pot

# Django stuff:
*.log
local_settings.py
db.sqlite3
db.sqlite3-journal

# Flask stuff:
instance/
.webassets-cache

# Scrapy stuff:
.scrapy

# Sphinx documentation
docs/_build/

# PyBuilder
.pybuilder/
target/
```

```
# Jupyter Notebook
.ipynb_checkpoints

# IPython
profile_default/
ipython_config.py

# pyenv
#   For a library or package, you might want to ignore these files since the code is
#   intended to run in multiple environments; otherwise, check them in:
# .python-version

# pipenv
#   According to pypa/pipenv#598, it is recommended to include Pipfile.lock in version control.
#   However, in case of collaboration, if having platform-specific dependencies or dependencies
#   having no cross-platform support, pipenv may install dependencies that don't work, or not
#   install all needed dependencies.
#Pipfile.lock

# poetry
#   Similar to Pipfile.lock, it is generally recommended to include poetry.lock in version control.
#   This is especially recommended for binary packages to ensure reproducibility, and is more
#   commonly ignored for libraries.
#   https://python-poetry.org/docs/basic-usage/#commit-your-poetrylock-file-to-version-control
#poetry.lock

# pdm
#   Similar to Pipfile.lock, it is generally recommended to include pdm.lock in version control.
#pdm.lock
#   pdm stores project-wide configurations in .pdm.toml, but it is recommended to not include it
#   in version control.
#   https://pdm.fming.dev/#use-with-ide
.pdm.toml

# PEP 582; used by e.g. github.com/David-OConnor/pyflow and github.com/pdm-project/pdm
__pypackages__/

# Celery stuff
celerybeat-schedule
celerybeat.pid

# SageMath parsed files
*.sage.py
```

```
# Environments
.env
.venv
env/
venv/
ENV/
env.bak/
venv.bak/

# Spyder project settings
.spyderproject
.spyproject

# Rope project settings
.ropeproject

# mkdocs documentation
/site

# mypy
.mypy_cache/
.dmypy.json
dmypy.json

# Pyre type checker
.pyre/

# pytype static type analyzer
.pytype/

# Cython debug symbols
cython_debug/

# PyCharm
#  JetBrains specific template is maintained in a separate JetBrains.gitignore that can
#  be found at https://github.com/github/gitignore/blob/main/Global/JetBrains.gitignore
#  and can be added to the global gitignore or merged into this file.  For a more nuclear
#  option (not recommended) you can uncomment the following to ignore the entire idea folder.
#.idea/
```