```
import os
import time
import asyncio
import datetime
import hashlib
import json
import pdfplumber
import numpy as np
import streamlit as st
from pymongo import MongoClient
from functools import lru_cache
from concurrent.futures import ThreadPoolExecutor
from langchain_community.vectorstores import FAISS

from langchain_community.llms import Ollama
from langchain.prompts import ChatPromptTemplate
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_community.embeddings import OllamaEmbeddings
import re
from langchain_ollama import OllamaLLM, OllamaEmbeddings
import bcrypt
st.set_page_config(page_title="AI FORTRESS", page_icon=" ")
```

```
if "theme" not in st.session_state:
    st.session_state["theme"] = "light"
# Constants
PDF_STORAGE_PATH = "document_store/pdfs/"
FAISS_INDEX_PATH = "document_store/faiss_index"
MONGO_URI = "mongodb://localhost:27017/"
DATABASE_NAME = "pdf_database"  # Database name remains constant
# COLLECTION_NAME = {
#     "iot": "iot_pdf_database",
#     "infosec": "infosec_pdf_database"
# }
# USER_COLLECTION = "users"
# CHAT_COLLECTION_NAME = {
#     "iot": "iot_hist",
#     "infosec": "info_hist"
# }
```

```
def toggle_theme():
    st.session_state["theme"] = "dark" if st.session_state["theme"] == "light" else
"light"
import base64
def set_background(image_file):
    with open(image_file, "rb") as img_file:
        encoded_string = base64.b64encode(img_file.read()).decode()

    page_bg_img = f'''
    <style>
    .stApp {{
        background-image: url("{encoded_string}");
        background-size: cover;
        background-position: center;
        background-repeat: no-repeat;

    }}

    </style>
    '''
    st.markdown(page_bg_img, unsafe_allow_html=True)
```

```
# Apply theme based on session state

with st.sidebar:
```

```
    st.button("Toggle Dark/Light Mode", on_click=toggle_theme)
# st.sidebar.title("  Navigation")
# st.sidebar.markdown("### Use the sidebar to explore more!")
st.sidebar.text("Made by VIT ")
```

```
# Ensure the user made a selection on the Selection Page
if "selected_db" not in st.session_state:
    st.warning("You must select Subject First")
    if st.button("Go to Selection"):
        st.switch_page("pages/Selection.py")  # Adjust based on your login page path

    st.stop()  # Prevent further execution
```

```
# Initialize MongoDB
client = MongoClient(MONGO_URI)
selected_db_key = st.session_state.get("selected_db", "iot")  # Default to 'iot' if not
set
```

```
# Access the main database
db = client[DATABASE_NAME]
DOMAIN_LIST_COLLECTION = "domain_list"
domain_cursor = db[DOMAIN_LIST_COLLECTION].find({}, {"_id": 0, "name": 1})
domain_list = sorted([d["name"] for d in domain_cursor])  # Sort optional for UI
# Get the correct collection based on user selection
selected_domain = st.session_state.get("selected_db")
```

```
if not selected_domain or selected_domain not in domain_list:
    st.warning("Please select a valid subject domain to continue.")
    if st.button("Go to Subject Selection"):
        st.switch_page("pages/Selection.py")
    st.stop()
```

```
doc_collection_name = f"{selected_domain}_pdf_database"
chat_collection_name = f"{selected_domain}_hist"
# chat_collection = db[CHAT_COLLECTION_NAME]
def compute_file_hash(text):
    """Generate a unique hash for the document text."""
    return hashlib.sha256(text.encode()).hexdigest()
doc_collection = db[doc_collection_name]
chat_collection = db[chat_collection_name]
```

```
# Load AI models
EMBEDDING_MODEL = OllamaEmbeddings(model="deepseek-r1:7b")
LANGUAGE_MODEL = OllamaLLM(model="deepseek-r1:7b")
```

```
PROMPT_TEMPLATE = """
You are an expert research assistant. Answer the user's query concisely using the
provided document context.
If the answer is uncertain, explicitly state that you are unsure.
```

```
Query: {user_query}
Relevant Context:
```

```
{document_context}
```

```
Answer (cite sources if applicable):
"""
PROMPT = ChatPromptTemplate.from_template(PROMPT_TEMPLATE)
```

```
# Ensure directories exist
os.makedirs(PDF_STORAGE_PATH, exist_ok=True)
vectorstore = None  # Global FAISS Store
executor = ThreadPoolExecutor()
```

```
if "authenticated" not in st.session_state or not st.session_state.authenticated:

    st.warning("You must log in first!")

    # Provide a login button instead of looping
    if st.button("Go to Login"):
        st.switch_page("./Login.py")  # Adjust based on your login page path

    st.stop()  # Prevent further execution
```

```
def get_current_collections():
    selected_db_key = st.session_state.get("selected_db", "iot")  # fallback to 'iot'
```

```
    doc_collection_name = f"{selected_db_key}_pdf_database"
    chat_collection_name = f"{selected_db_key}_hist"
```

```
    doc_collection = db[doc_collection_name]
    chat_collection = db[chat_collection_name]
```

```
    return selected_db_key, doc_collection, chat_collection
```

```
# === Text Extraction ===
def extract_text_from_pdf(file_path):
    """Extract text from all pages of a PDF."""
    extracted_text = []
    try:
        with pdfplumber.open(file_path) as pdf:
            for page in pdf.pages:
                page_text = page.extract_text()
                if page_text:
                    extracted_text.append(page_text.strip())
    except Exception as e:
        print(f"⚠ Error extracting text from PDF: {e}")

    return "\n".join(extracted_text)
```

```
# === Document Indexing ===
def index_document(file_name, text):
    """Index a document into FAISS and MongoDB."""
    global vectorstore
    selected_db_key, doc_collection, _ = get_current_collections()
```

```
    file_hash = compute_file_hash(text)
    if doc_collection.find_one({"file_hash": file_hash}):
        print(f"⚠ Document {file_name} is already indexed.")
        return
```

```
    doc_collection.insert_one({
        "file_name": file_name,
        "file_hash": file_hash,
```

```
        "text_content": text
    })
```

```
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=1500, chunk_overlap=300)
    document_chunks = text_splitter.create_documents([text])
```

```
    if vectorstore is None:
        vectorstore = FAISS.from_documents(document_chunks, EMBEDDING_MODEL)
    else:
        vectorstore.add_documents(document_chunks)
```

```
    vectorstore.save_local(f"{FAISS_INDEX_PATH}_{selected_db_key}")
    print(f"✅ Document '{file_name}' indexed successfully!")
```

```
# === FAISS Initialization ===
def initialize_faiss():
    """Initialize FAISS index from existing files or rebuild from MongoDB."""
    global vectorstore
    selected_db_key, doc_collection, _ = get_current_collections()
    faiss_path = f"{FAISS_INDEX_PATH}_{selected_db_key}"
```

```
    if os.path.exists(faiss_path):
        try:
            vectorstore = FAISS.load_local(
                faiss_path, EMBEDDING_MODEL, allow_dangerous_deserialization=True
            )
            print(f"✅ FAISS index for '{selected_db_key}' loaded successfully!")
            return
        except Exception as e:
            print(f"⚠️ Error loading FAISS index: {e}")
            print("  Rebuilding FAISS index from MongoDB...")
```

```
    docs = list(doc_collection.find({}, {"_id": 0, "text_content": 1}))
    if not docs:
        print(f"⚠️ No documents found in MongoDB for '{selected_db_key}'. FAISS index
will not be created.")
        return
```

```
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=1500, chunk_overlap=300)
    document_chunks = text_splitter.create_documents([doc["text_content"] for doc in
docs])
```

```
    vectorstore = FAISS.from_documents(document_chunks, EMBEDDING_MODEL)
    vectorstore.save_local(faiss_path)
    print(f"✅ FAISS index for '{selected_db_key}' created and saved successfully!")
```

```
# === Chat Collection Access ===
def get_chat_collection():
    """Dynamically return the correct chat collection based on selected section."""
    _, _, chat_collection = get_current_collections()
    return chat_collection
```

```
# === Context Trimming ===
def trim_context(context, max_chars=10000):  # Roughly ~3000 tokens
    return context[:max_chars] if len(context) > max_chars else context
def main():
    """Streamlit UI for document indexing and AI chat."""
    st.title("Upload PDF For LLM ")
    st.markdown(f'<p class="domain-display">Domain: {selected_db_key}</p>',
unsafe_allow_html=True)
```

```
    st.markdown("---")
```

```
    # User Authentication
    if "username" in st.session_state:
        st.markdown(f"### Welcome, {st.session_state['username']}!!  ")
```

```
    # PDF Upload and Indexing
    uploaded_pdf = st.file_uploader("  Upload PDF", type="pdf")
    if uploaded_pdf:
        file_path = os.path.join(PDF_STORAGE_PATH, uploaded_pdf.name)
        with open(file_path, "wb") as f:
            f.write(uploaded_pdf.read())
```

```
        extracted_text = extract_text_from_pdf(file_path)
        if extracted_text:
            index_document(uploaded_pdf.name, extracted_text)
            st.success("✅ Document indexed successfully!")
```

```
    # Display Chat History
```

```
    # Logout Button


    # Clear Chat History Button

    if st.button("  Logout"):
        st.session_state.authenticated = False
        st.session_state.pop("username", None)
        st.success("Logged out successfully!")
        st.switch_page("Login.py")
        st.rerun()
if st.session_state.get("theme", "light") == "dark":
# Custom CSS for Styling
    set_background("black.jpeg")
    st.markdown("""
    <style>
                    p{  font-weight:bold;}
        h1, h2, h3 {
    color: white !important;
    }
    body {
            background: linear-gradient(135deg, #0D0D0D, #1A1A2E) !important;
        }
    .custom-button {
        padding: 10px 20px;
        font-size: 16px;
        font-weight: bold;
        border-radius: 8px;
        cursor: pointer;
        border: none;
        transition: 0.3s;
        text-align: center;
    }
    .login-btn {
        background: linear-gradient(90deg, #38BDF8, #3B82F6);
        color: white;
    }
    .login-btn:hover {
        background: linear-gradient(90deg, #3B82F6, #2563EB);
        transform: translateY(-2px);
```

```
    }
    .signup-btn {
        background: linear-gradient(90deg, #34D399, #10B981);
        color: white;
    }
    .signup-btn:hover {
        background: linear-gradient(90deg, #10B981, #059669);
        transform: translateY(-2px);
    }
    label {
        font-size: 25px !important;
        font-weight: bold !important;
        margin-bottom: 8px !important;
        color: white !important;
    }
    [data-testid="stSidebar"]::before {
        content: " Navigation";
        font-weight: bold;
        display: block;
        font-size: 25px;
        margin-top: 40px;
        text-align: center;
        color: white;
    }
      [data-testid="stSidebar"] {
        background-color: #211f26 !important;
        font-size: 18px;
        text-align: left !important;
        border-right:2px solid white;
    }


/* Chat Input Box Styling */
.stChatInput input {
    background-color: #1E1E1E !important;
    color: #FFFFFF !important;
    border: 1px solid #3A3A3A !important;
}
```

```
/* Styling for User & AI Messages */
.stChatMessage[data-testid="stChatMessage"]:nth-child(odd) {
    background-color: #1E1E1E !important;
    border: 1px solid #3A3A3A !important;
    color: #E0E0E0 !important;
    border-radius: 10px;
    padding: 15px;
    margin: 10px 0;
}
```

```
.stChatMessage[data-testid="stChatMessage"]:nth-child(even) {
    background-color: #2A2A2A !important;
    border: 1px solid #404040 !important;
    color: #F0F0F0 !important;
    border-radius: 10px;
    padding: 15px;
    margin: 10px 0;
}
```

```
/* File Uploader Styling */
.stFileUploader {
    background-color: #1E1E1E;
```

```
    border: 1px solid white;
    border-radius: 10px;
    padding: 15px;
}
```

```
.stFileUploader label {
    color: white !important;
    border-color: white !important;
}
```

```
/* Headings */
h1, h2, h3 {
    color: white !important;
}
```

```
/* User Question Styling */
.question {
    color: white !important; /* Neon Green Text */

    font-size: 18px; /* Readable Size */
    background-color: #1E1E1E !important; /* Dark Background */
    border: 1px solid #3A3A3A !important; /* Subtle Border */
    padding: 10px 20px; /* Balanced Padding */
    padding-let:20px;
    margin: 10px auto; /* Centers the Element */
    display: inline-block; /* Ensures it Wraps Around Content */
    max-width: 90%; /* Prevents Stretching */
    word-wrap: break-word; /* Prevents Overflow Issues */
    border-radius: 25px; /* Rounded Edges */
}
```

```
     [data-testid="stSidebar"] {
        background-color: #211f26 !important;
        font-size: 18px;
        text-align: left !important;
        border-right:2px solid white;
    }

        .ai-response {
        color: white;  /* Ensures black text */
        background-color: grey;  /* Light background */
        padding: 12px;
        border: 5px solid white;
        border-radius:10px; /* More rounded bubble */
        margin:10px;
        font-size: 16px;
    }
                            .manage-domains-header {
        background-color: black;
        color: white;
        padding: 10px;
        font-weight: bold;
        font-size: 20px;
        border-radius: 5px;
    }
    </style>
    """, unsafe_allow_html=True)
else:

    set_background("white.jpeg")
    st.markdown("""
```

```
<style>
                .stAlert {
        background-color: black !important;  /* Black background */
        color: white !important;  /* White text */
        border-radius: 8px;  /* Rounded corners */
        padding: 10px;  /* Add padding */
    }
      body {
        background: linear-gradient(135deg, #0D0D0D, #1A1A2E) !important;
    }
    h1, h2, h3 {
        color: #333333 !important;
    }
    .custom-button {
        padding: 10px 20px;
        font-size: 16px;
        font-weight: bold;
        border-radius: 8px;
        cursor: pointer;
        border: none;
        transition: 0.3s;
        text-align: center;
    }
    .question {
        color: #333333 !important;
        font-size: 18px;
        background-color: #F5F5F5 !important;
        border: 2px solid green !important;
        padding: 10px 20px;
        margin: 10px auto;
        margin-bottom:10px;
        display: inline-block;
        max-width: 90%;
        word-wrap: break-word;
        border-radius: 25px;
    }
      [data-testid="stSidebar"] {
    background-color: #211f26 !important;
    font-size: 18px;
    text-align: left !important;
    border-right:2px solid white;
}

.custom-button {
    padding: 10px 20px;
    font-size: 16px;
    font-weight: bold;
    border-radius: 8px;
    cursor: pointer;
    border: none;
    transition: 0.3s;
    text-align: center;
}
.login-btn {
    background: linear-gradient(90deg, #1E40AF, #1D4ED8);
    color: white;
}
.login-btn:hover {
    background: linear-gradient(90deg, #1D4ED8, #2563EB);
    transform: translateY(-2px);
}
```

```
.signup-btn {
    background: linear-gradient(90deg, #047857, #065F46);
    color: white;
}
.signup-btn:hover {
    background: linear-gradient(90deg, #065F46, #064E3B);
    transform: translateY(-2px);
}
label {
    font-size: 22px !important;
    font-weight: bold !important;
    margin-bottom: 8px !important;
    color: black !important;
}
   [data-testid="stSidebar"] {
    background-color: #211f26 !important;
    font-size: 18px;
    text-align: left !important;
    border-right:2px solid white;
}

[data-testid="stSidebar"]::before {
    content: " Navigation";
    font-weight: bold;
    display: block;
    font-size: 25px;
    margin-top: 40px;
    text-align: center;
    color: white;
}
            .ai-response {
    color: black;  /* Ensures black text */
    background-color: #f8f9fa;  /* Light background */
    padding: 12px;
    border: 5px solid black;
    border-radius:10px; /* More rounded bubble */
    margin:10px;
    font-size: 16px;
}
```

```
          .question {
    font-weight: bold;
    color: #007bff;
    font-size: 18px;
}
               h2, h3 {
    color: #4CAF50; /* Light green heading */
    font-size: 22px; /* Bigger heading */
    font-weight: bold;
}
                .stChatInput {
             margin-top:20px;
    border-radius: 12px !important; /* Rounded chat input */
}
```

```
          @media (prefers-color-scheme: dark) {
              .ai-response {
                  color: black;  /* Change text to white in dark mode */
                  background-color: white;  /* Dark background */
              }
```

```
            }
                  p{  font-weight:bold;}
               div.streamlit-expanderHeader {
        background-color: #000000 !important;
        color: white !important;
        font-weight: bold;
    }
```

```
    /* Optional: make the expander content area also dark */
    div.streamlit-expanderContent {
        background-color: #111111 !important;
        color: white !important;
    }
    .streamlit-expanderHeader {
        background-color: white;
        color: black; # Adjust this for expander header color
    }
    /* Optional: style inputs inside expander */
    .stTextInput > div > div > input {
        background-color: #222 !important;
        color: white !important;
    }
                  .manage-domains-header {
        background-color: black;
        color: white;
        padding: 10px;
        font-weight: bold;
        font-size: 20px;
        border-radius: 5px;
    }
                           .domain-display {
            font-size: 18px;
            font-weight: bold;
            color: black;
            margin-bottom: 10px;


    }
```

```
    </style>
    """, unsafe_allow_html=True)
```

```
if __name__ == "__main__":
    initialize_faiss()
    main()
```