```python
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=" ")
```

```python
if "theme" not in st.session_state:
    st.session_state["theme"] = "dark"
# 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": "infosec_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
if st.session_state.get("theme", "light") == "dark":
# Custom CSS for Styling
    set_background("black.jpeg")
    st.markdown("""
    <style>
        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: black !important;
        font-size: 18px;
        text-align: left !important;
    }

/* 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: black!important; /* Neon Green Text */

    font-size: 18px; /* Readable Size */
    background-color: #d3dceb !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;
```

```
    border-right:2px solid white;
    font-weight:bold;

}
                .ai-response {
        color: black;  /* Ensures black text */
        background-color: #21252b;  /* Light background */
        padding: 12px;
        border-radius: 12px; /* More rounded bubble */
        font-size: 16px;
        border: 2px solid white;
    }
```

```
             .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 */
        border:2px solid white!important;
    }
```

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

            font-weight:bold;}
    </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;
        }
    .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;
    }
    section[data-testid="stSidebar"] {
        # background-color: #6A0DAD !important;
        background-color: #211f26 !important;
```

```
        border-right: 2px solid white !important;
    }
    [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-radius: 12px; /* More rounded bubble */
        font-size: 16px;
        border: 2px solid black;
    }
    sub{
                color:black! important!;}
             .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 {

        border-radius: 12px !important;
                border:2px solid white!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;}
        .time{
            color:black ! important;
            font-weight:bold;}
          .stSpinner > div > div {
            color: black !important;
        }
    </style>
    """, unsafe_allow_html=True)

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")
```

```
# 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()
```

```
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
```

```python
executor = ThreadPoolExecutor()
```

```python
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
```

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

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

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

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

```python
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)
```

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

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

```python
    doc_collection.insert_one({
        "file_name": file_name,
        "file_hash": file_hash,
        "text_content": text
    })
```

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

```python
    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!")
```

```
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!")
```

```
def find_related_documents(user_query, k=5):
    """Retrieve top-k similar documents from FAISS for the selected category."""
    if vectorstore is None:
        raise ValueError(f"⚠ Vectorstore for '{selected_db_key}' is not initialized.
Run the indexing process first.")
```

```
    related_docs = vectorstore.similarity_search(user_query, k=k)
```

```
    # Ensure longer, more informative results are ranked higher
    related_docs.sort(key=lambda doc: len(doc.page_content), reverse=True)
```

```
    return related_docs
```

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

```
def save_chat_history(username, user_input, ai_response, response_time):
    """Save user-AI chat history with response time."""
    chat_collection = get_chat_collection()  # Get correct collection dynamically
    chat_collection.insert_one({
        "username": username,
        "user": user_input,
```

```
        "ai_response": ai_response.strip(),
        "response_time_seconds": response_time,
    })
```

```
def load_chat_history(username):
    """Retrieve user chat history from the correct collection, including response
time."""
    chat_collection = get_chat_collection()
    return list(chat_collection.find(
        {"username": username},
        {"_id": 0, "user": 1, "ai_response": 1, "response_time_seconds": 1}
    ))
```

```
# Changes from here
def clear_chat_history(username):
    """Delete chat history for the logged-in user from the correct collection."""
    chat_collection = get_chat_collection()  # Get correct collection dynamically
    chat_collection.delete_many({"username": username})
```

```
def trim_context(context, max_chars=10000):  # Roughly ~3000 tokens
    return context[:max_chars] if len(context) > max_chars else context
def main():
    st.title("  AI Fortress: Secure LLM")
    st.markdown("### Your AI Research Assistant")
    st.markdown("---")
```

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

```
    # --- Chat Input ---
    user_input = st.chat_input("  Ask a question...")
```

```
    if user_input:
        with st.spinner("  Searching for answer..."):
            relevant_docs = find_related_documents(user_input)
            document_context = trim_context("\n\n".join([doc.page_content for doc in
relevant_docs]))
```

```
            formatted_prompt = PROMPT.format(user_query=user_input,
document_context=document_context)
```

```
            start_time = time.time()
            ai_response = LANGUAGE_MODEL.invoke(formatted_prompt).strip()
            ai_response = re.sub(r"<think>.*?</think>", "", ai_response, flags=re.DOTALL)
            response_time = round(time.time() - start_time, 2)
```

```
            save_chat_history(st.session_state.username, user_input, ai_response,
response_time)
```

```
            st.markdown(f"<div class='question'>  {user_input}</div>",
unsafe_allow_html=True)
            st.markdown(f"<div class='ai-response'><strong>  AI Response:</strong>
{ai_response}</div>", unsafe_allow_html=True)
            st.markdown(f"<sub><div class='time'>  Response Time: {response_time}
seconds</div></sub>", unsafe_allow_html=True)
```

```
    # --- Chat History Display ---
    st.markdown("### Chat History:")
    chat_history = load_chat_history(st.session_state.username)
    if chat_history:
```

```
        for chat in chat_history:
            st.markdown(f"<div class='question'>  {chat.get('user', '')}</div>",
unsafe_allow_html=True)
```

```
            st.markdown(f"<div class='ai-response'><strong>  AI Response:</strong>
{chat['ai_response']}</div>", unsafe_allow_html=True)
            st.markdown(f"<sub><div class='time'>  Response Time:
{chat.get('response_time_seconds', '?')} seconds</div></sub>", unsafe_allow_html=True)
    else:
        st.info("No chat history found.")
```

```
    # --- Control Buttons (Bottom Row) ---
    col1, col2, col3 = st.columns(3)
```

```
    with col1:
        if st.button("  Clear Chat History"):
            clear_chat_history(st.session_state.username)
            st.rerun()
```

```
    with col2:
        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()
```

```
    with col3:
        if st.button("  Generate Questions"):
            st.session_state.show_input_popup = not
st.session_state.get("show_input_popup", False)
```

```
    # --- Generate Questions Form ---
    if st.session_state.get("show_input_popup", False):
        with st.expander("  Configure Question Generator", expanded=True):
            selected_module = st.selectbox("  Select a Module", [""] + [f"Module {i}"
for i in range(1, 8)], key="module_select")
```

```
            question_type = st.selectbox("  Select Question Type", ["1 mark", "3 marks",
"5 marks", "10 marks", "MCQ"], key="question_type_select")
            num_questions = st.number_input("  Number of Questions", min_value=1,
max_value=20, value=5, step=1, key="num_questions_input")
```

```
            if st.button("  Generate Now"):
                prompt = f"Generate {num_questions} ,{question_type} questions based on
{selected_module} and their answers."
                with st.spinner("  Generating questions..."):
                    relevant_docs = find_related_documents(prompt)
                    document_context = trim_context("\n\n".join([doc.page_content for
doc in relevant_docs]))
```

```
                    formatted_prompt = PROMPT.format(user_query=prompt,
document_context=document_context)
```

```
                    start_time = time.time()
                    ai_response = LANGUAGE_MODEL.invoke(formatted_prompt).strip()
                    ai_response = re.sub(r"<think>.*?</think>", "", ai_response,
flags=re.DOTALL)
                    response_time = round(time.time() - start_time, 2)
```

```
                    # Show immediately
```

```
                    st.markdown(f"<div class='question'>  {prompt}</div>",
unsafe_allow_html=True)
                    st.markdown(f"<div class='ai-response'><strong>  AI
Response:</strong><br>{ai_response}</div>", unsafe_allow_html=True)
                    st.markdown(f"<sub><div class='time'>  Response Time: {response_time}
seconds</div></sub>", unsafe_allow_html=True)
```

```
                    # ✅ Save to chat history
                    save_chat_history(st.session_state.username, prompt, ai_response,
response_time)
```

```
                # Close popup and rerun to show in chat history
                st.session_state.show_input_popup = False
                st.rerun()
```

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