**submission.py**

```python

# ================================================================================
# Step 1: Import essential tools
# ================================================================================

import os
import getpass
from flask import Flask, render_template_string, request, jsonify
from dotenv import load_dotenv,find_dotenv
from langchain_community.document_loaders import PyPDFLoader
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_community.vectorstores import FAISS
from langchain_openai import ChatOpenAI, OpenAIEmbeddings
from langchain.chains.combine_documents import create_stuff_documents_chain
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder
from langchain.chains import create_retrieval_chain
from langchain_core.messages import HumanMessage, AIMessage

# ================================================================================
# Step 2: Set up OPENAI API Key
# ================================================================================

load_dotenv(find_dotenv())

# ================================================================================
# Step 3: Load the HR policy PDF and split it into chunks
# ================================================================================

try:
    print("Loading and splitting the HR policy PDF...")
    loader = PyPDFLoader("./the_nestle_hr_policy_pdf_2012.pdf")
    docs = loader.load()
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
    documents = text_splitter.split_documents(docs)
    print(f"Successfully loaded and split the document into {len(documents)} chunks.")
except FileNotFoundError:
    print("Error: The PDF file 'the_nestle_hr_policy_pdf_2012.pdf' was not found.")
    print("Please ensure the file is in the same directory as this script.")
    exit()

# ================================================================================
# Step 4: Create vector representations for text chunks using FAISS and embeddings
# ================================================================================

print("Creating and saving the FAISS vector store...")
embeddings = OpenAIEmbeddings()

# vector_store = FAISS.from_documents(documents, embeddings)
# vector_store.save_local("faiss_index")
# print("FAISS index created and saved successfully.")

vector_store = FAISS.load_local("faiss_index", embeddings, allow_dangerous_dese-
rialization=True)

# ================================================================================
```

```
# Step 5: Build a question-answering system
# ========================================================================

llm = ChatOpenAI(model="gpt-4o-mini", temperature=0)

retriever = vector_store.as_retriever()

# ========================================================================
# Step 6: Create a prompt template to guide the chatbot
# ========================================================================
prompt = ChatPromptTemplate.from_messages(
    [
        (
            "system",
            "You are an AI-powered HR Assistant for Nestlé. Answer the user's questions strictly based on the provided context. If the answer is not in the context, politely state that you cannot answer. Maintain a professional and helpful tone. The context is:\n\n{context}"
        ),
        MessagesPlaceholder(variable_name="chat_history"),
        ("human", "{input}"),
    ]
)

# Combine the documents, prompt, and LLM to create a new, modern chain.
document_chain = create_stuff_documents_chain(llm, prompt)

retrieval_chain = create_retrieval_chain(retriever, document_chain)

chat_history = []

# ========================================================================
# Step 7: Chatbot interface
# ========================================================================

app = Flask(__name__)

html_template = """
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Nestlé HR Assistant</title>
    <script src="https://cdn.tailwindcss.com"></script>
    <style>
        @import url('https://fonts.googleapis.com/css2?family=Inter:wght@400;500;700&display=swap');
        body { font-family: 'Inter', sans-serif; }
    </style>
</head>
<body class="bg-gray-100 flex items-center justify-center min-h-screen p-4">
    <div class="bg-white shadow-xl rounded-2xl w-full max-w-2xl overflow-hidden flex flex-col h-[80vh]">
        <div class="bg-blue-600 text-white p-4 flex items-center justify-between shadow-md">
            <h1 class="text-xl font-bold">Nestlé HR Assistant</h1>
```

```
            <svg xmlns="http://www.w3.org/2000/svg" class="h-6 w-6" viewBox="0 0 24 24" fill="none" stroke="currentColor" stroke-width="2" stroke-linecap="round" stroke-linejoin="round">
              <path d="M12 2a10 10 0 1 0 10 10A10 10 0 0 0 12 2zM12 16v-4M12 8h.01"/>
            </svg>
        </div>
        <div id="chat-history" class="flex-1 p-4 overflow-y-auto space-y-4">
            <div class="flex justify-start">
                <div class="bg-gray-200 text-gray-800 p-3 rounded-xl max-w-sm">
                    <p>Hello! I'm your AI HR Assistant. How can I help you with the Nestlé HR policy today?</p>
                </div>
            </div>
        </div>
        <form id="chat-form" class="bg-gray-200 p-4 flex items-center">
            <input type="text" id="user-input" class="flex-1 p-3 rounded-full border border-gray-300 focus:outline-none focus:ring-2 focus:ring-blue-500" placeholder="Ask a question about the HR policy...">
            <button type="submit" class="ml-2 bg-blue-600 text-white p-3 rounded-full hover:bg-blue-700 transition duration-300">
                <svg xmlns="http://www.w3.org/2000/svg" class="h-6 w-6" fill="none" viewBox="0 0 24 24" stroke="currentColor" stroke-width="2">
                    <path stroke-linecap="round" stroke-linejoin="round" d="M14 5l7 7m0 0l-7 7m7-7H3" />
                </svg>
            </button>
        </form>
        <!-- Footer for the chatbot interface -->
        <div class="bg-gray-200 p-2 text-center text-gray-500 text-sm shadow-inner rounded-b-2xl">
            <p>Created by: Eric Michel</p>
            <a href="https://www.linkedin.com/in/ericmichelcv/" target="_blank" class="text-blue-600 hover:underline">Profile Page</a>
        </div>
    </div>
    <script>
        document.getElementById('chat-form').addEventListener('submit', async function(e) {
            e.preventDefault();
            const userInput = document.getElementById('user-input');
            const userMessage = userInput.value;
            if (userMessage.trim() === '') return;

            const chatHistory = document.getElementById('chat-history');

            // Display user message
            const userDiv = document.createElement('div');
            userDiv.className = 'flex justify-end';
            userDiv.innerHTML = `
                <div class="bg-blue-500 text-white p-3 rounded-xl max-w-sm">
                    <p>${userMessage}</p>
                </div>
            `;
            chatHistory.appendChild(userDiv);

            // Display loading indicator
            const loadingDiv = document.createElement('div');
            loadingDiv.className = 'flex justify-start';
            loadingDiv.innerHTML = `
                <div class="bg-gray-200 text-gray-800 p-3 rounded-xl max-w-sm">
```

```
                    <div class="flex space-x-2 animate-pulse">
                        <div class="w-2 h-2 bg-gray-400 rounded-full"></div>
                        <div class="w-2 h-2 bg-gray-400 rounded-full"></div>
                        <div class="w-2 h-2 bg-gray-400 rounded-full"></div>
                    </div>
                </div>
            `;
            chatHistory.appendChild(loadingDiv);
            chatHistory.scrollTop = chatHistory.scrollHeight;

            userInput.value = '';

            try {
                const response = await fetch('/chat', {
                    method: 'POST',
                    headers: {
                        'Content-Type': 'application/json',
                    },
                    body: JSON.stringify({ query: userMessage }),
                });

                const data = await response.json();

                // Remove loading indicator
                chatHistory.removeChild(loadingDiv);

                // Display AI response
                const assistantDiv = document.createElement('div');
                assistantDiv.className = 'flex justify-start';
                assistantDiv.innerHTML = `
                    <div class="bg-gray-200 text-gray-800 p-3 rounded-xl max-w-sm">
                        <p>${data.response}</p>
                    </div>
                `;
                chatHistory.appendChild(assistantDiv);
                chatHistory.scrollTop = chatHistory.scrollHeight;
            } catch (error) {
                console.error('Error:', error);
                chatHistory.removeChild(loadingDiv);
                const errorDiv = document.createElement('div');
                errorDiv.className = 'flex justify-start';
                errorDiv.innerHTML = `
                    <div class="bg-red-200 text-red-800 p-3 rounded-xl max-w-sm">
                        <p>An error occurred. Please try again.</p>
                    </div>
                `;
                chatHistory.appendChild(errorDiv);
                chatHistory.scrollTop = chatHistory.scrollHeight;
            }
        });
    </script>
</body>
</html>
"""

@app.route("/")
def home():
    """Renders the main chatbot interface."""
```

```
    return render_template_string(html_template)

@app.route("/chat", methods=["POST"])
def chat():
    """Endpoint to handle user queries and return chatbot responses."""
    global chat_history
    data = request.json
    user_query = data.get("query", "")

    if not user_query:
        return jsonify({"response": "Please enter a query."}), 400

    try:
        # Get the response from the retrieval chain
        response = retrieval_chain.invoke(
            {"input": user_query, "chat_history": chat_history}
        )

        # Add the new messages to the chat history for context in the next turn.
        chat_history.append(HumanMessage(content=user_query))
        chat_history.append(AIMessage(content=response["answer"]))

        return jsonify({"response": response["answer"]})
    except Exception as e:
        print(f"An error occurred: {e}")
        return jsonify({"response": "An error occurred while processing your request."}),
500

if __name__ == "__main__":
    app.run(debug=True)
```