**AI Intern Assignment**

**Objective:**

The goal of this selection process is to assess the candidate's hands-on experience in working with LLM-based tasks, their understanding of language models, and their ability to implement practical solutions using AI tools and frameworks.

**Duration:**

1.5 hours

**Problem Statement:**

Your company sells electronic gadgets, including smartphones, laptops, and accessories. Customers frequently contact your support team with questions about your products, orders, and company policies. Currently, these questions are being answered manually by customer support agents, which is time-consuming and inefficient.

You have been tasked with automating customer support to handle common inquiries, such as:

* Specifications of the latest smartphones or laptops.
* How customers can track their orders.
* Details about the company's return policy.
* Available payment methods for online purchases.
* Warranty information for products.

Your goal is to create an automated solution that can handle these types of inquiries effectively. The solution must provide accurate and contextually relevant answers to customers, and it should be able to carry on multi-turn conversations without losing track of context.

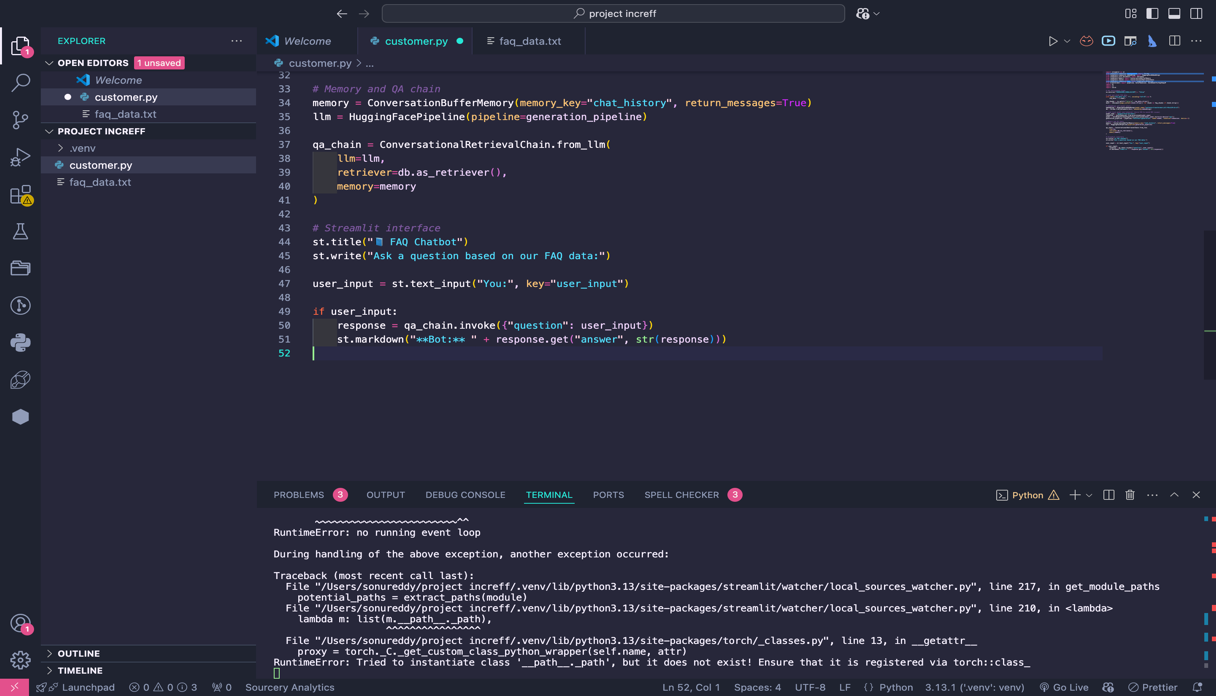
**Task:**

* Propose an approach to build a chatbot that can address these frequently asked questions.
* Use an LLM-based solution (e.g., GPT-3, GPT-4, or Hugging Face models) to implement the chatbot. You are free to use any available tools or frameworks that can help in developing the solution.
* Explain how you would structure the data (e.g., storing FAQ information) and ensure the chatbot can maintain context throughout a conversation.
* Discuss any challenges you anticipate in building this system, including edge cases such as ambiguous questions or handling multiple inquiries in one conversation.

**Deliverables:**

* Code that implements the chatbot.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

* Explanation of how you handled multi-turn conversations and context.

My chat boat uses:

memory = ConversationBufferMemory(memory\_key="chat\_history", return\_messages=True).

This component **stores the entire chat history**between the user and the bot. It allows the model to access **past questions and answers**, which is critical for multi-turn conversations.

When a user sends a new input:

**response = qa\_chain.invoke({"question": user\_input})**

The ConversationalRetrievalChain:

* Retrieves relevant documents from the vector store (Chroma)
* Passes those along with **chat history from memory**
* The language model (HuggingFacePipeline) generates a context-aware response
* Explanation of how you would implement and test the chatbot, including methods for handling context and user inputs.

**Setup and Preprocessing**

* Loaded faq\_data.txt, split it into chunks using regular expressions, and converted each chunk into langchain.Document objects.
* Used HuggingFaceEmbeddings (sentence-transformers/all-MiniLM-L6-v2) to convert text chunks into vector embeddings.

**2. Vector Store**

* Stored embeddings in **Chroma**, a local vector store that allows fast document retrieval using similarity search.

**3. Language Model**

* Loaded a **local language model** (google/flan-t5-small) using Hugging Face Transformers and restricted execution to **CPU** to avoid MPS errors on macOS.

**4. Context Management**

* Used ConversationBufferMemory to **maintain the full conversation** across multiple user questions.
* Integrated this memory into ConversationalRetrievalChain, ensuring context-aware answers across turns.

**5. Interface**

* Used **Streamlit** for a clean, simple web UI.
* Used st.text\_input() for user queries and st.markdown() to show bot responses.

**6. Testing**

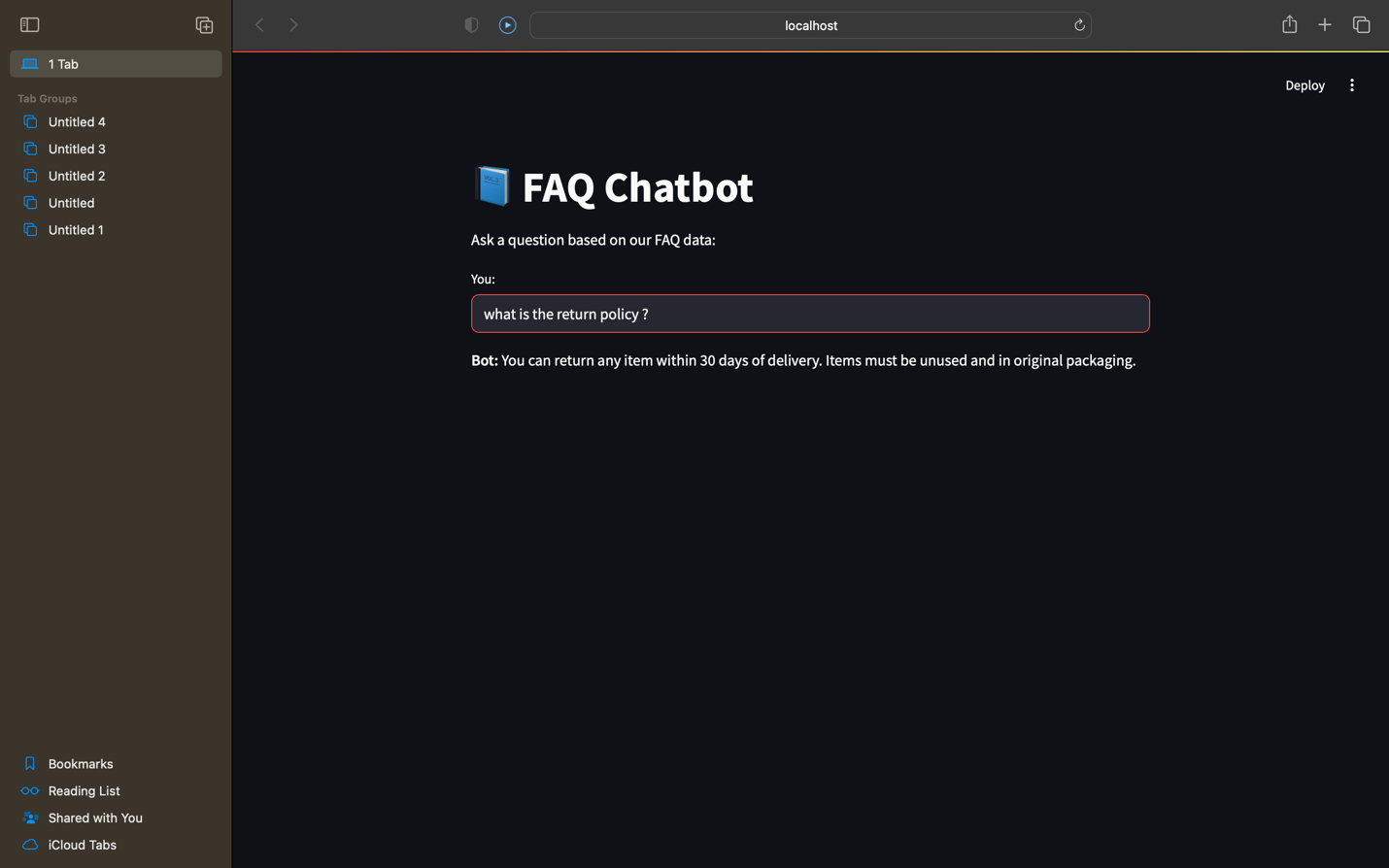
* Tested using:
  + Known FAQ questions (for correctness)
  + Follow-ups to prior questions (for context)
  + Unexpected or malformed queries (for robustness)
* Verified performance and response coherence in multi-turn scenarios.
* A list of potential edge cases you have considered (e.g., unanswered questions, ambiguous queries).

### **Edge Cases Considered**

| **Case** | **Handling** |
| --- | --- |
| **Unanswered Questions** | The LLM attempts to generate a fallback response if retrieval yields no useful context. Could enhance with "Sorry, I couldn't find that in our FAQ." |
| **Ambiguous Inputs** | Multi-turn memory helps clarify intent, but ambiguous single-turn queries might need user rephrasing. |
| **Mismatched Queries** | If a query doesn’t match FAQ content, Chroma returns irrelevant docs, which model may try to answer. Logging could be used to flag these. |
| **Non-FAQ/Out-of-Scope Questions** | The model still generates a response but may hallucinate. Consider adding scope validation or a fallback. |
| **Repetitive or Very Long Inputs** | Tokenizer truncation and generation length control in Hugging Face pipeline prevent crashes. |
| **Model Failure / Offline Mode** | Using local models avoids API quota errors and makes the chatbot fully offline and testable. |

* If time permits, add a simple web interface for the chatbot (optional, could use Flask/Django).

There is no time if it permits we can add definitely. Now I have done in streamlit.



**Evaluation Criteria:**

* Correctness of the responses based on the FAQ data.
* How well the chatbot handles multi-turn conversations.
* Efficiency of prompt engineering.
* Code readability and documentation.
* Creativity in handling edge cases (e.g., if the model doesn’t know the answer).