“TravelBot- ChatBot for all your travel needs”

Submitted as part of ESA Hackathon for the July 2024 Summer Course

Crafting Tomorrow’s Conversations: Large Language Models

Submitted by:

Prerana Sanjay Kulkarni	**PES1UG21CS451**
Chetana Mital	**PES1UG21CS155**
Amara Sai Anitej	**PES1UG21CS069**

Course Instructors:

Prof. Anantharaman Palacode Narayana Iyer
Prof. Shylaja S S
Prof. Preet Kanwal

Table of Contents

1. **Introduction**
    - Overview of the project
    - Objectives
2. **Implementation**
    - Description of the LLM model chosen
    - Explanation of the hyperparameters used
    - Overview of the Amadeus APIs integrated
    - Details on the front-end implementation using Streamlit
    - Use of ChatPromptTemplate
    - Code snippets and explanations
3. **Testing and Evaluation**
    - List the prompts used for testing
    - Results and analysis of the chatbot's responses
    - Validation and Evaluation of the results
4. **Challenges and Solutions**
    - Any challenges faced during the implementation
    - Solutions and workarounds applied
5. **Conclusion**
    - Summary of the project
    - Learning outcomes
6. **References**

1. INTRODUCTION

This ESA-project presents an advanced travel search application that utilizes natural language processing in conjunction with the Amadeus travel API. By allowing users to search for flight offers and hotels using conversational queries, it eases the travel planning experience, making it more intuitive and accessible to a wider audience. The application serves as a bridge between human language and complex travel data systems, simplifying the process of finding and booking travel arrangements.

A) Overview

This project:

1. Utilizes a tech stack comprising Streamlit for creating a responsive web-based user interface, Langchain for sophisticated natural language processing, and the Amadeus API for comprehensive travel data retrieval
2. Employs a language model to interpret user queries, extracting key information such as destinations, dates, and preferences to form structured search parameters
3. Implements a multi-step process that converts interpreted queries into appropriate Amadeus API calls, fetches relevant data, and then uses the language model again to present results in a clear, human-readable format

B) Objectives

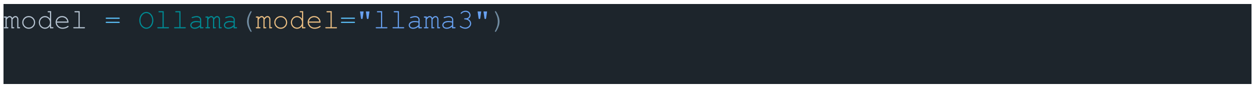
The primary objectives of this project are:

1. To develop a user-centric interface that simplifies the travel search process by allowing users to input queries in natural language, eliminating the need for complex form-filling or understanding of travel industry jargon
2. To showcase the practical application and benefits of integrating advanced language models with traditional API-based services, potentially paving the way for similar innovations in other industries
3. To enhance the accuracy and relevance of travel information provided to users by effectively translating their intent and preferences from conversational language into precise API queries, thereby improving the overall user experience in travel planning

2. IMPLEMENTATION

A) Description of the LLM model chosen

The project uses the Llama 3 model through Ollama:



```
model = Ollama(model="llama3")
```

Llama 3 is an advanced large language model developed as an open-source alternative to proprietary models. It's known for its strong performance across various natural language processing tasks, including text generation, summarization, and question-answering. The use of Ollama, a lightweight container for running large language models, suggests an emphasis on efficiency and ease of deployment.

B) Explanation of the hyperparameters used

While the code doesn't explicitly show hyperparameter tuning, it's using the default configuration for Llama 3 as provided by Ollama. This likely includes:

- A specific model size (e.g., 7B, 13B, or 70B parameters)
- Default temperature setting for text generation
- Default top-k and top-p settings for sampling
- Default context window size

Future improvements could involve fine-tuning these parameters for optimal performance in travel-related queries.

C) Overview of the Amadeus APIs integrated

The project integrates two main Amadeus API endpoints:

1. Flight Offers Search API

```
def get_flight_offers(self, origin, destination, departure_date, n_adults):
    response = self.amadeus.shopping.flight_offers_search.get(
        originLocationCode=origin,
        destinationLocationCode=destination,
        departureDate=departure_date,
        adults=n_adults
    )
    return response.data
```

This API allows searching for flight offers based on origin, destination, date, and number of passengers. It returns detailed flight information including prices, airlines, and itineraries.

2. Hotel Search API

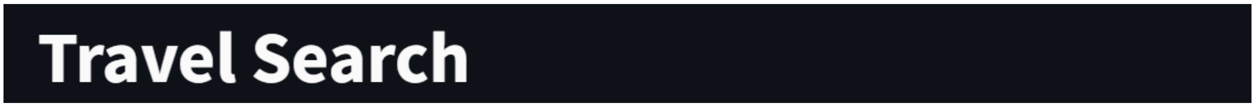
```
def get_hotel_by_city(self, city_code):
    response = self.amadeus.reference_data.locations.hotels.by_city.get(cityCode=city_code)
    return response.data
```

This API searches for hotels in a specific city, returning information about available accommodations. Both APIs are initialized with Amadeus credentials.

D) Details on the front-end implementation using Streamlit

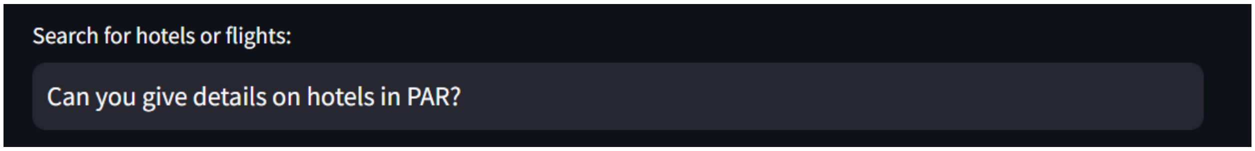
The front-end is implemented using Streamlit, a Python library for creating web apps with minimal code. The key components include:

1. App title



Travel Search

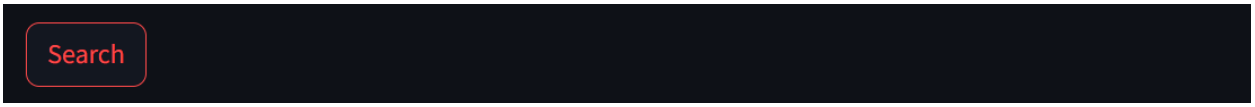
2. User input field



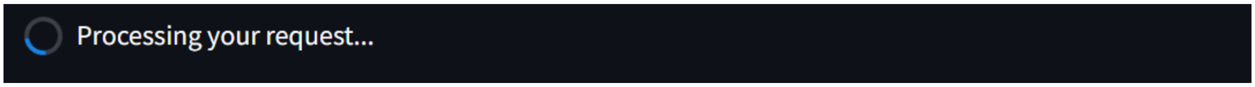
Search for hotels or flights:

Can you give details on hotels in PAR?

3. Search button

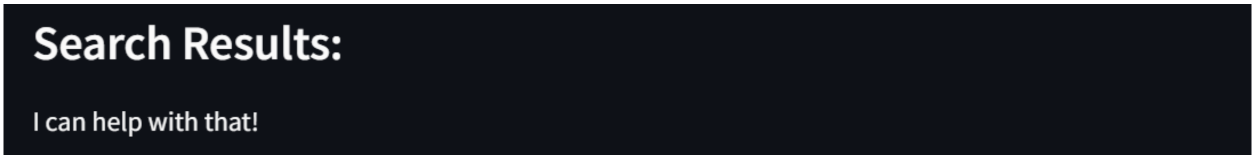


4. Loading spinner



Processing your request...

5. Results display



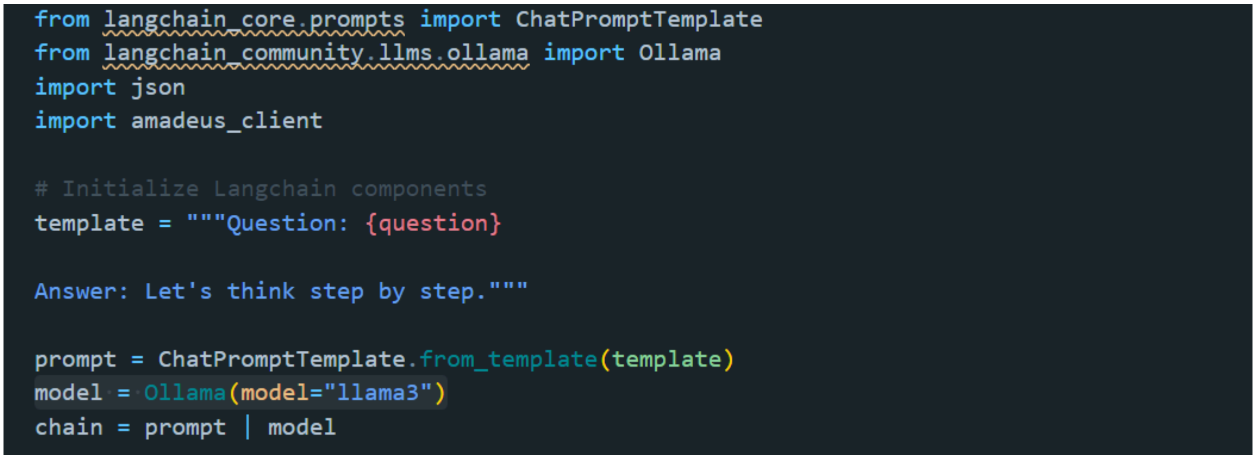
Search Results:

I can help with that!

Additionally, error-handling functions ensure the application runs smoothly and errors are handled gracefully.

E) Use of ChatPromptTemplate

The project uses ChatPromptTemplate from Langchain to structure prompts for the language model. This template encourages the model to approach the problem systematically. It's used in combination with more specific prompts tailored to each task (query interpretation and result formatting).



```
from langchain_core.prompts import ChatPromptTemplate
from langchain_community.llms.ollama import Ollama
import json
import amadeus_client

# Initialize Langchain components
template = """Question: {question}

Answer: Let's think step by step."""

prompt = ChatPromptTemplate.from_template(template)
model = Ollama(model="llama3")
chain = prompt | model
```

F) Code snippets and explanations

Here are some important code snippets and their explanations:

1. ***Query Processing***

```
def process_query(user_query):

    our_prompt = """

    i have this following class with two functions

    import amadeus



    def get_flight_offers(self, origin, destination, departure_date,
n_adults):

        response = self.amadeus.shopping.flight_offers_search.get(

            originLocationCode=origin,

            destinationLocationCode=destination,
            departureDate=departure_date,
            adults=n_adults
        )
        return response.data


    def get_hotel_by_city(self, city_code):
        response = self.amadeus.reference_data.locations.hotels.by_city.get(cityCode=city_code)
        return response.data


    (dates in YYYY-MM-DD format)
    based on the function definitions, can you pick the function definition that matches the user query that I am going to give, give the function call for it according to the information in the query? (do not give  ANY explanations, just the function call). You **must** give the response in JSON format where there are two different objects, one for hotels and the other for flights.
    """
    response = chain.invoke({"question": our_prompt + user_query})
    return json.loads(response)
```

This function:

- Combines a predefined prompt (explaining the available functions and desired output format) with the user's query
- Sends this combined prompt to the language model via the `chain.invoke()` method
- Parses the model's response as JSON

The prompt is designed to guide the model in interpreting the user's query and matching it to the appropriate API function, extracting relevant parameters in the process.

2. ***Result Retrieval***

```
def get_result(data):
    if data['task'] == 'get_flight_offers':
        origin = data['params']['origin']
        destination = data['params']['destination']
        departure_date = data['params']['departure_date']
        n_adults = data['params']['n_adults']
        flight_data = amadeus_client.getting_flight_offers(origin=origin, destination=destination, departure_date=departure_date, n_adults=n_adults)
        return flight_data
    else:
        city_code = data['params']['city_code']
        hotel_data = amadeus_client.getting_hotels(city_code=city_code)
        return hotel_data
```

This function:

- Takes the structured data produced by `process_query()`
- Determines whether to search for flights or hotels based on the 'task' field
- Extracts the relevant parameters
- Calls the appropriate Amadeus API function with these parameters
- Returns the raw API response

3. ***Results Formatting***

```python
def format_result(user_query, result):
    our_prompt2 = f"""I have provided to you with user question and the JSON answer. Can you present it in a human readable natural language format with only the important information
    : User question: {user_query}. Result: {result}. Can you give the answer in human understandable format?"""
    response2 = chain.invoke({"question": our_prompt2})
    return response2
```

This function:

- Takes the original user query and the raw API result
- Constructs a prompt asking the language model to summarize this information in a human-readable format
- Sends this prompt to the model and returns its response

4. ***Main Streamlit Loop***

```python
if st.button("Search"):
    if user_query:
        with st.spinner("Processing your request..."):
            data = process_query(user_query)
            result = get_result(data)
            formatted_result = format_result(user_query, result)
            st.subheader("Search Results:")
            st.write(formatted_result)
    else:

        st.warning("Please enter a search query.")
```

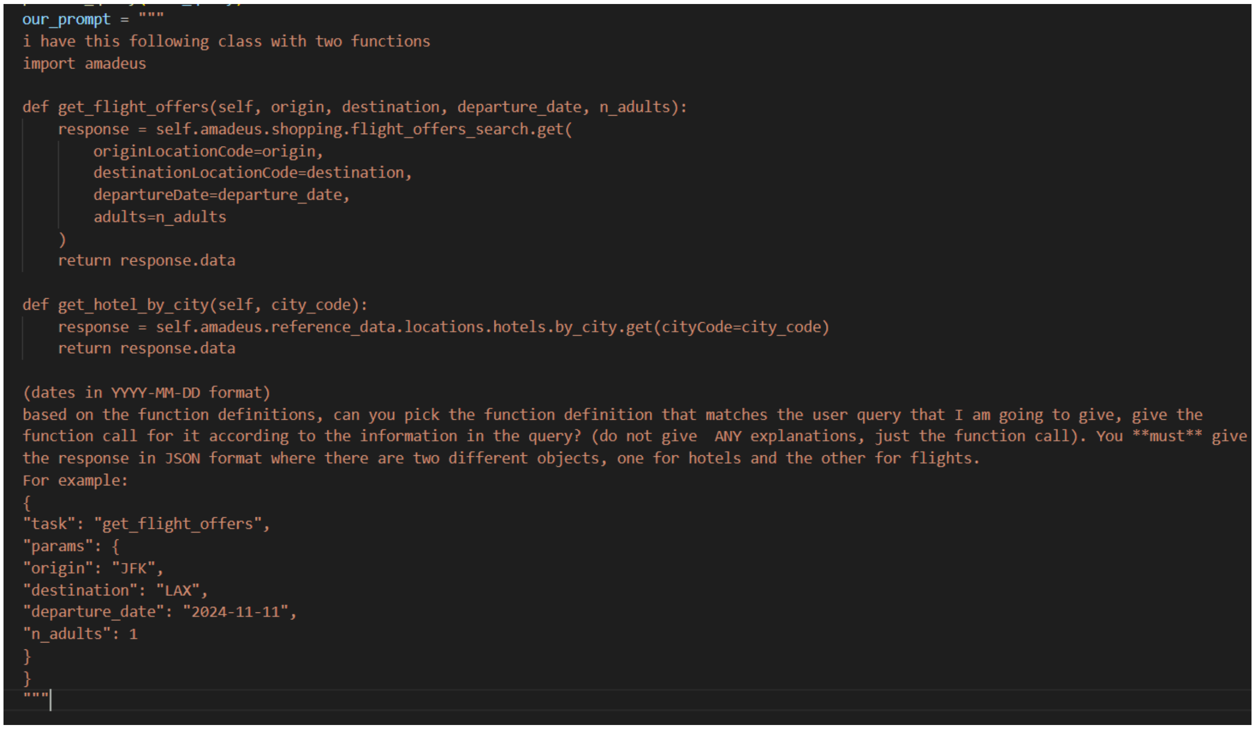
This code:

- Checks if the search button has been clicked
- Verifies that a query has been entered
- Processes the query through the entire pipeline: interpretation, API call, and result formatting
- Displays the formatted results or a warning if no query was entered

3. TESTING AND EVALUATION

Testing is done with the following prompts:

This prompt is a one-shot prompt (with a single example) on how to get the tool to be used after this step. This prompt directs to LLM to accurately choose between the two tasks, ‘Searching for Flights’ or ‘Searching for Hotels’ based on the user query. It then extracts the necessary parameters required to invoke the tool from the user query, into a JSON object.



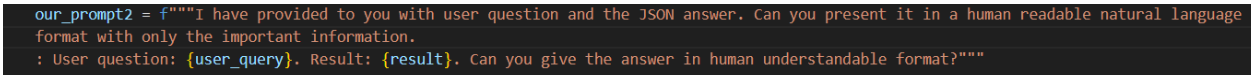
```
our_prompt = """
i have this following class with two functions
import amadeus

def get_flight_offers(self, origin, destination, departure_date, n_adults):
    response = self.amadeus.shopping.flight_offers_search.get(
        originLocationCode=origin,
        destinationLocationCode=destination,
        departureDate=departure_date,
        adults=n_adults
    )
    return response.data

def get_hotel_by_city(self, city_code):
    response = self.amadeus.reference_data.locations.hotels.by_city.get(cityCode=city_code)
    return response.data

(dates in YYYY-MM-DD format)
based on the function definitions, can you pick the function definition that matches the user query that I am going to give, give the
function call for it according to the information in the query? (do not give  ANY explanations, just the function call). You **must** give
the response in JSON format where there are two different objects, one for hotels and the other for flights.
For example:
{
"task": "get_flight_offers",
"params": {
"origin": "JFK",
"destination": "LAX",
"departure_date": "2024-11-11",
"n_adults": 1
}
}
"""
```

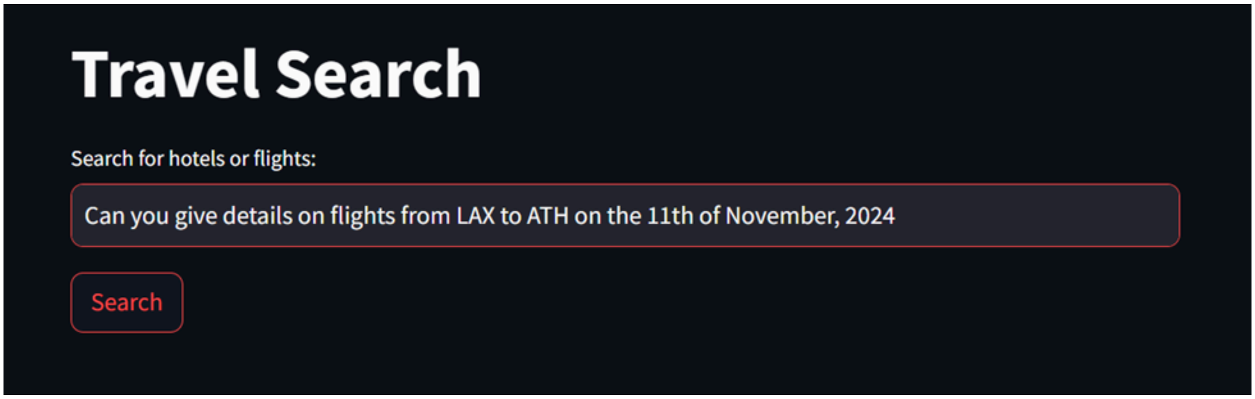
Below is a prompt that takes in the question and the result generated as the context, and creates a human-readable version of the result.



```
our_prompt2 = f"""I have provided to you with user question and the JSON answer. Can you present it in a human readable natural language
format with only the important information.
: User question: {user_query}. Result: {result}. Can you give the answer in human understandable format?"""
```

Results and analysis of the chatbot's responses

1. Searching for flights

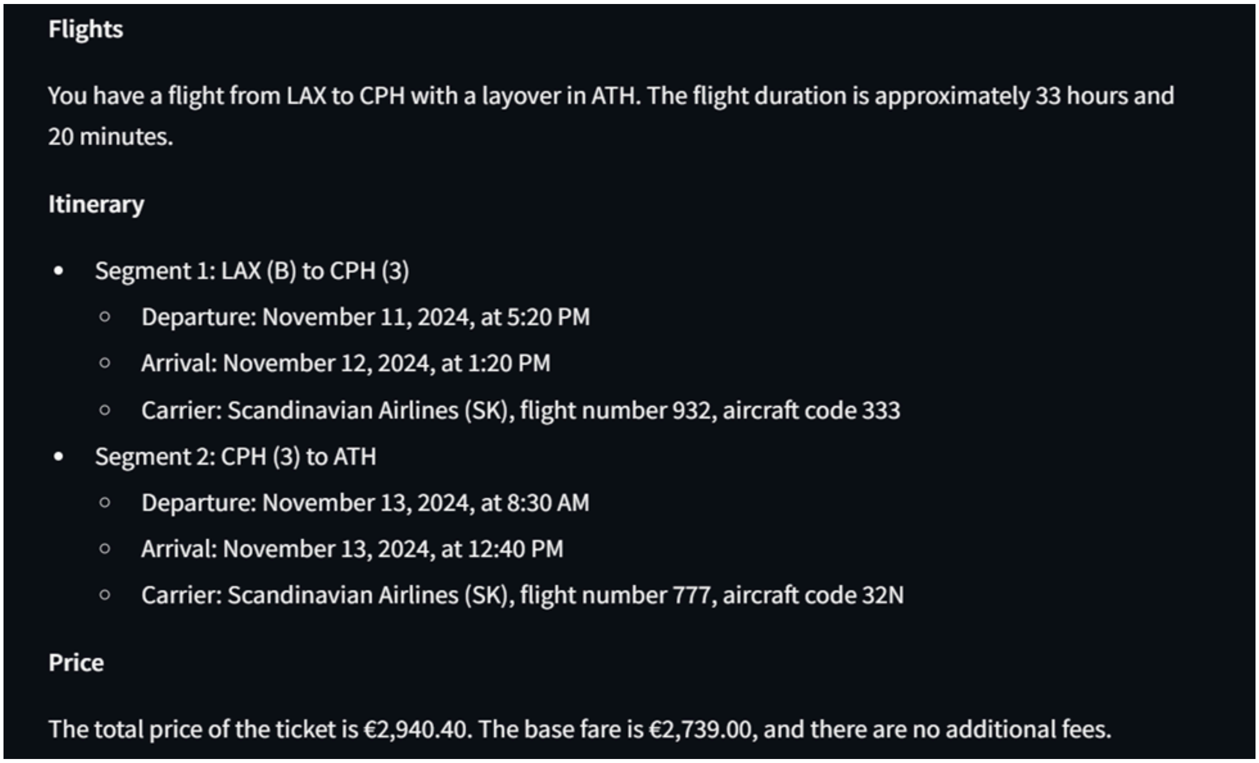


Travel Search

Search for hotels or flights:

Can you give details on flights from LAX to ATH on the 11th of November, 2024

2. Results generated about the asked query



Flights

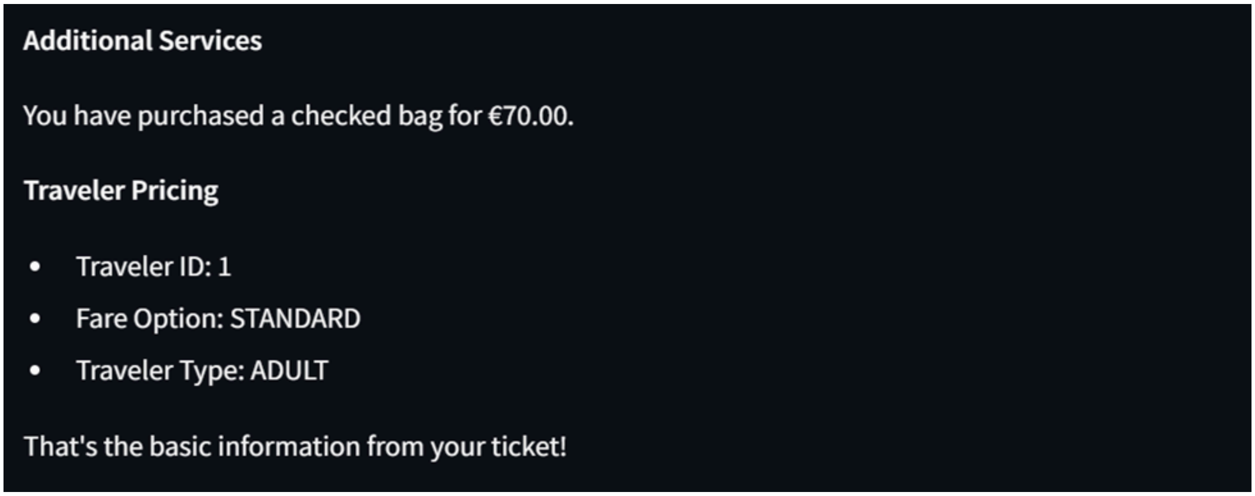
You have a flight from LAX to CPH with a layover in ATH. The flight duration is approximately 33 hours and 20 minutes.

Itinerary

- Segment 1: LAX (B) to CPH (3)
  - Departure: November 11, 2024, at 5:20 PM
  - Arrival: November 12, 2024, at 1:20 PM
  - Carrier: Scandinavian Airlines (SK), flight number 932, aircraft code 333
- Segment 2: CPH (3) to ATH
  - Departure: November 13, 2024, at 8:30 AM
  - Arrival: November 13, 2024, at 12:40 PM
  - Carrier: Scandinavian Airlines (SK), flight number 777, aircraft code 32N

Price

The total price of the ticket is €2,940.40. The base fare is €2,739.00, and there are no additional fees.



Additional Services

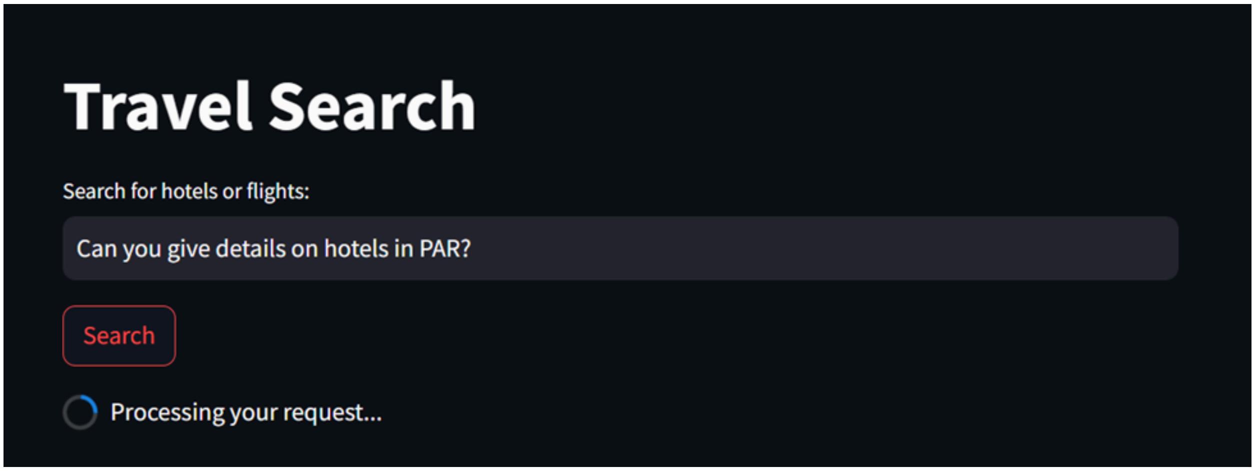
You have purchased a checked bag for €70.00.

Traveler Pricing

- Traveler ID: 1
- Fare Option: STANDARD
- Traveler Type: ADULT

That's the basic information from your ticket!

3. Searching about hotels



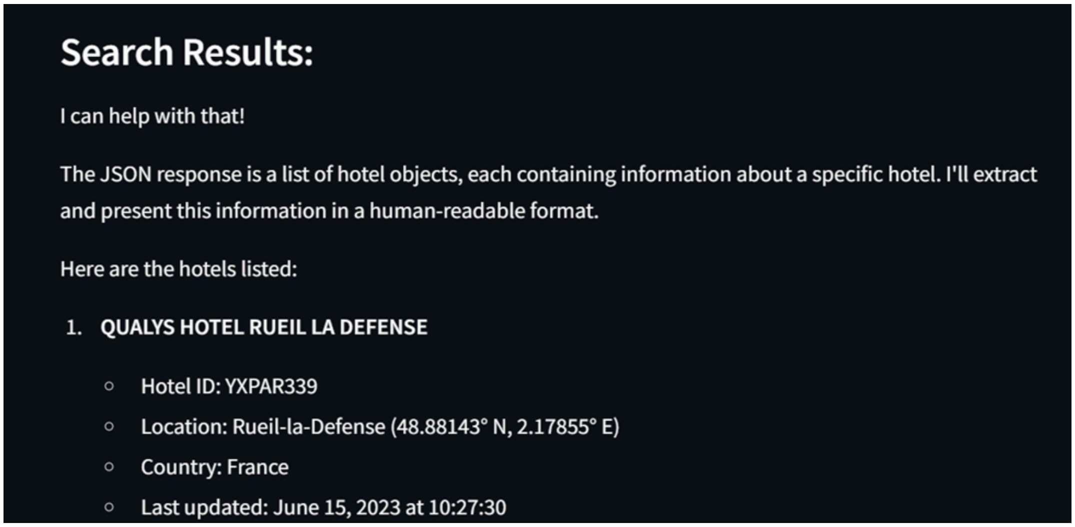
Travel Search

Search for hotels or flights:

Can you give details on hotels in PAR?

Processing your request...

4. Results generated about the asked query



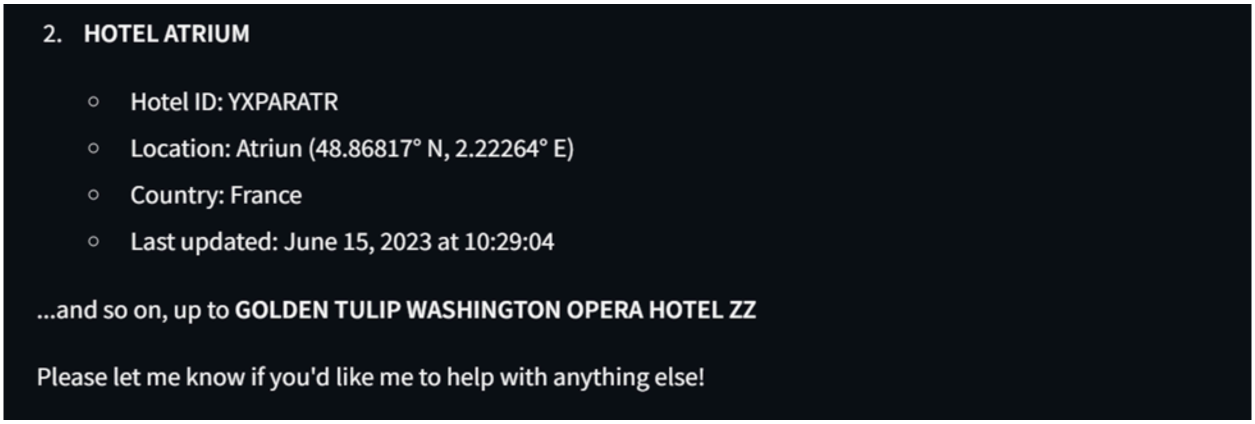
Search Results:

I can help with that!

The JSON response is a list of hotel objects, each containing information about a specific hotel. I'll extract and present this information in a human-readable format.

Here are the hotels listed:

1. **QUALYS HOTEL RUEIL LA DEFENSE**
    - Hotel ID: YXPAR339
    - Location: Rueil-la-Defense (48.88143° N, 2.17855° E)
    - Country: France
    - Last updated: June 15, 2023 at 10:27:30



2. **HOTEL ATRIUM**
   - Hotel ID: YXPARATR
   - Location: Atriun (48.86817° N, 2.22264° E)
   - Country: France
   - Last updated: June 15, 2023 at 10:29:04

...and so on, up to **GOLDEN TULIP WASHINGTON OPERA HOTEL ZZ**

Please let me know if you'd like me to help with anything else!

Validation and Evaluation of the results

1. **Response Time:** The LLM took about 5 Minutes to execute the query and provide the user with a response.
2. **Contextual Appropriateness and Manual Review:** Manual review of ChatBot's responses assured the quality, relevance and appropriateness with regards to the user's query.
3. **User Satisfaction:** Users were satisfied with the detailed response received for flight and hotel details as all the necessary information required to proceed further with booking tickets were provided.

4. Challenges and Solutions

Challenges faced during the implementation:

1. Long Download and Execution Times

The large size of LLMs meant that downloading and loading the models into memory took a considerable amount of time. This was particularly challenging in our development environment where network bandwidth and storage were limited.

Solution: To mitigate these issues, we implemented a caching mechanism that stored the models locally after the initial download. This way, subsequent runs could load the models from local storage, significantly reducing load times. We also explored model compression techniques to reduce the size of the models without compromising their performance.

2. Running Hugging Face Models Locally

Running Hugging Face models locally introduced challenges related to managing dependencies and ensuring compatibility between different versions of libraries and the models themselves.

Solution: We also used virtualenv to manage Python dependencies and avoid conflicts.

3. Lack of GPU Access

One of the primary challenges we faced during the implementation of our chatbot was the lack of access to GPU resources. Running models like LLaMA 3 typically requires significant computational power, which is best handled by GPUs. Without GPUs, the processing time for downloading, loading, and executing these models was significantly longer, leading to delays in development and testing.

Solution: To overcome this, we optimized our workflow by leveraging CPU resources efficiently. We used asynchronous programming to handle multiple tasks concurrently, reducing idle times. We also quantized models—converting them to lower precision formats to reduce their size and speed up inference.

5. Conclusion

A) Summary of the project

This project is a travel search application built using Streamlit, Langchain, and the Amadeus API. It allows users to search for flights and hotels through a simple text input interface. The application uses natural language processing to interpret user queries, convert them into API calls, and then present the results in a human-readable format. The core functionality is split between two main files: final.py, which handles the user interface and query processing, and amadeus_client.py, which manages the interaction with the Amadeus API.

Key points:

1. The application uses a language model to interpret user queries and generate appropriate API calls.
2. It leverages the Amadeus API to fetch real-time flight and hotel data.

B) Learning outcomes

The following are some of the learning outcomes of this project:

1. **Integration of multiple technologies:** We learnt how to combine different technologies like Streamlit for the frontend, Langchain for natural language processing, and external APIs (Amadeus) for data retrieval.
2. **Natural Language Processing (NLP) in practical applications:** The project demonstrates how to use NLP to interpret user queries and convert them into structured API calls, providing hands-on experience with language models and prompt engineering.
3. **API interaction and data handling:** We gained some experience in working with external APIs, managing API credentials securely, and processing the returned data.
4. **User interface design with Streamlit:** The project showcases how to create a simple yet effective user interface for a web application using Streamlit, teaching important concepts in frontend development.

6. References

https://github.com/amadeus4dev/amadeus-python

https://github.com/amadeus4dev/amadeus-python

https://developers.amadeus.com/self-service/category/flights/api-doc/flight-offers-search

https://www.langchain.com/

https://huggingface.co/docs/transformers/en/agents#what-is-an-agent