TravelAssist Chatbot

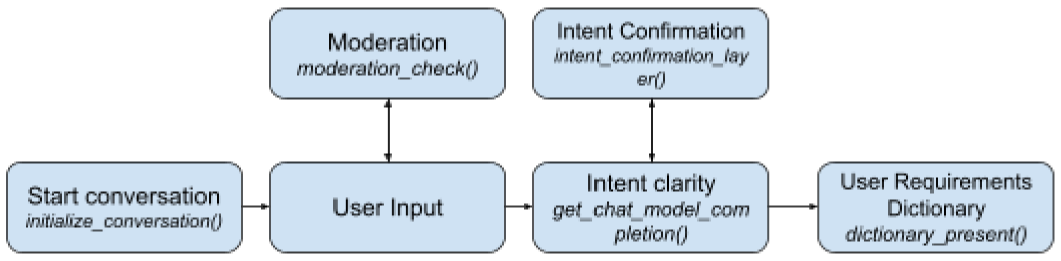
Objective:

The TravelAssist is an assistant flight trip planner that helps in suggesting round trip flight combinations to a particular destination from a reference flight corpus. Few important inputs are required in order to give the recommendation such as - destination city, source city, max budget, and number of days for holiday.
Using the information, the chatbot should be able to suggest round-trip flights for a particular destination. The chatbot should be able to do any moderation checks to avoid any harmful conversations.

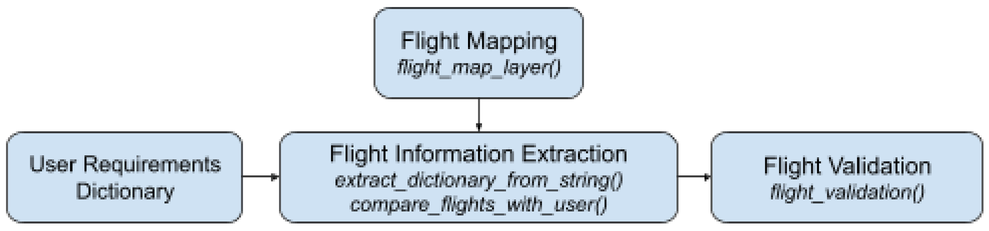
Design:

Following is the flow diagram of the TravelAssist Chatbot.

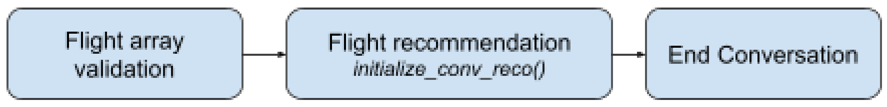
A. Intent Clarity and Confirmation Layers



B. Flight Mapping and Information Extraction


Flight Mapping
flight_map_layer()
User Requirements Dictionary
Flight Information Extraction
extract_dictionary_from_string()
compare_flights_with_user()
Flight Validation
flight_validation()

C. Flight Recommendations



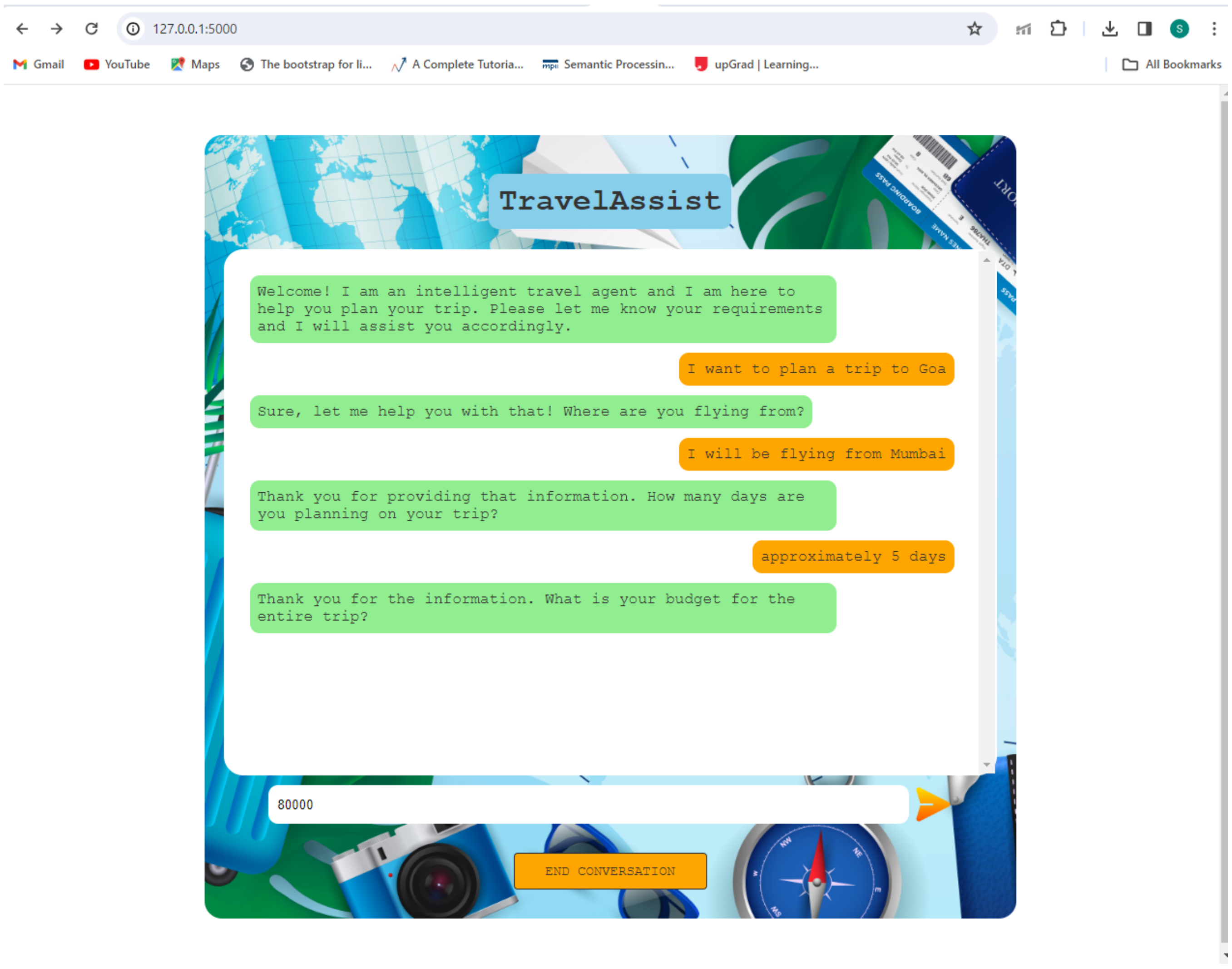
Implementation:

Tech Stack

- UI -> HTML, CSS, Javascript
- Backend -> Python Flask framework
- Environment -> Python virtual environment

UI/UX

- TravelAssist chatbot is made using HTML, CSS, and Javascript
- index_invite.html file contains the structure of the UI with inline JS code
- styles.css file contains the styling of each component to give it a very vibrant look to the bot
- Screenshots of the bot are given below:



Welcome! I am an intelligent travel agent and I am here to help you plan your trip. Please let me know your requirements and I will assist you accordingly.

I want to plan a trip to Goa

Sure, let me help you with that! Where are you flying from?

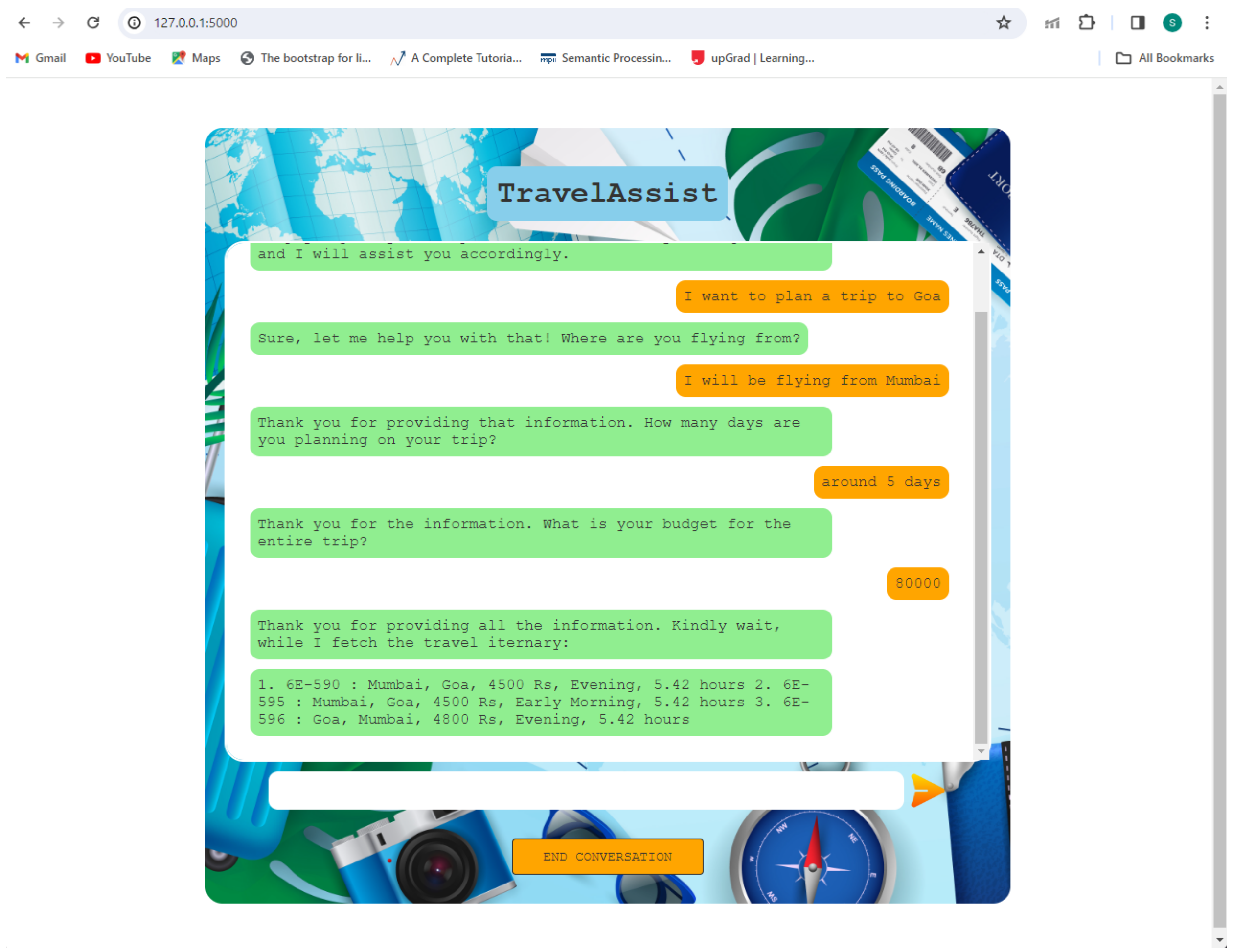
I will be flying from Mumbai

Thank you for providing that information. How many days are you planning on your trip?

approximately 5 days

Thank you for the information. What is your budget for the entire trip?

80000



and I will assist you accordingly.

I want to plan a trip to Goa

Sure, let me help you with that! Where are you flying from?

I will be flying from Mumbai

Thank you for providing that information. How many days are you planning on your trip?

around 5 days

Thank you for the information. What is your budget for the entire trip?

80000

Thank you for providing all the information. Kindly wait, while I fetch the travel iternary:

1. 6E-590 : Mumbai, Goa, 4500 Rs, Evening, 5.42 hours 2. 6E-595 : Mumbai, Goa, 4500 Rs, Early Morning, 5.42 hours 3. 6E-596 : Goa, Mumbai, 4800 Rs, Evening, 5.42 hours

Architecture

- Initial prompt of instructions are given to openAI as follows:

```
def initialize_conversation():
    '''
    Returns a list [{"role": "system", "content": system_message}]
    '''

    delimiter = "####"
    example_user_req = {'Origin':'Delhi', 'Destination':'Manali', 'Point of interest':'Hadimba Temple', 'Hotel': 'Baragarh Regency', 'Budget': '150000'}

    system_message = f"""
    You are an intelligent travel agent and your goal is to create
iternary to users.
    You need to ask relevant questions and understand the user profile by
analysing the user's responses.
    You final objective is to fill the values for the different keys
('Source','Destination','Duration','Budget') in the python dictionary and
be confident of the values.
    These key value pairs define the user's profile.
    The python dictionary looks like this {{'Source':
'values','Destination': 'values','Duration': 'values','Budget': 'values'}}
    The values for keys 'Source' and 'Destination' should be any city in
India. The values for 'Duration' should be in number between 2 to 12. The
value for 'Budget', should be in number. All the values will come from the
user.
    The values currently in the dictionary are only representative values.

    {delimiter}Here are some instructions around the values for the
different keys. If you do not follow this, you'll be heavily penalised.
    - The values for keys 'Source' and 'Destination' should be any city in
India extracted from user's response.
    - The values for 'Duration' should be in number between 2 to 12
extracted from user's response.
    - The value for 'Budget', should be a numerical value extracted from
user's response.
    - 'Budget' value needs to be greater than or equal to 4000 INR. If the
user says less than that, please mention that traveling will require a
larger budget.
    - Do not randomly assign values to any of the keys. The values need to
be inferred from the user's response.
    {delimiter}

    To fill the dictionary, you need to have the following chain of
thoughts:
    {delimiter} Thought 1: Ask a question to understand the user's profile
and requirements. \n
    You are trying to fill the values of all the keys
('Source','Destination','Duration','Budget') in the python dictionary by
understanding the user requirements.
```

```
    Identify the keys for which you can fill the values confidently using
the understanding. \n
    Remember the instructions around the values for the different keys.
    Answer "Yes" or "No" to indicate if you understand the requirements
and have updated the values for the relevant keys. \n
    If yes, proceed to the next step. Otherwise, rephrase the question to
capture their profile. \n{delimiter}

    {delimiter}Thought 2: Now, you are trying to fill the values for the
rest of the keys which you couldn't in the previous step.
    Remember the instructions around the values for the different keys.
Ask questions you might have for all the keys to strengthen your
understanding of the user's profile.
    Answer "Yes" or "No" to indicate if you understood all the values for
the keys and are confident about the same.
    If yes, move to the next Thought. If no, ask question on the keys
whose values you are unsure of. \n
    It is a good practice to ask question with a sound logic as opposed to
directly citing the key you want to understand value for.{delimiter}

    {delimiter}Thought 3: Check if you have correctly updated the values
for the different keys in the python dictionary.
    If you are not confident about any of the values, ask clarifying
questions. {delimiter}

    Follow the above chain of thoughts and only output the final updated
python dictionary. \n


    {delimiter} Here is a sample conversation between the user and
assistant:
    User: "Hi, I want to plan a trip to Delhi."
    Assistant: "Sure, let me help you with that! Where are you flying
from?"
    User: "I will be flying from Mumbai."
    Assistant: "Thank you for providing that information. How many days
are you planning on your trip?"
    User: "I am looking at 4 days."
    Assistant: "Thank you for the information. What is your budget for the
entire trip?"
```

```
    User: "my max budget is 30000 inr"
    Assistant: "{example_user_req}"
    {delimiter}

    Start with a short welcome message and encourage the user to share
their requirements. Do not start with Assistant: "
    """
    conversation = [{"role": "system", "content": system_message}]
    return conversation
```

Here we are trying to give instructions as a prompt in such a way that the bot is able to ask the user all the relevant questions to get the result as a python dictionary of format: {{'Source': 'values','Destination': 'values','Duration': 'values','Budget': 'values'}}

Here we are using a combination of chain of thought and few shot prompting technique to define the prompts.

When the conversation starts, the user's inputs first undergo moderation check and then they are appended in the conversation array and send to openAI chatCompletion api to get back the response to show to the user after moderation check.

This process continues until our intent_confirmation_layer is able to confirm that all the user's inputs are filled in as per the format given. {{'Source': 'values','Destination': 'values','Duration': 'values','Budget': 'values'}}

```
def intent_confirmation_layer(response_assistant):
    delimiter = "####"
    prompt = f"""
    You are a senior evaluator who has an eye for detail.
    You are provided an input. You need to evaluate if the input has the
following keys: 'Source','Destination','Duration','Budget'
    Next you need to evaluate if the keys have the the values filled
correctly.
    The values for keys 'Source' and 'Destination' should be any city in
India extracted from user's response. The values for 'Duration' should be
in number between 2 to 12  extracted from user's response.
    The value for 'Budget', should be a numerical value in currency INR
extracted from user's response. Only extract the numerical value from
user's response.
```

```
    Output a string 'Yes' if the input contains the dictionary with the
values correctly filled for all keys.
    Otherwise out the string 'No'.

    Here is the input: {response_assistant}
    Only output a one-word string - Yes/No.
    """


    confirmation = openai.Completion.create(
                                    model="text-davinci-003",
                                    prompt = prompt,
                                    temperature=0)


    return confirmation["choices"][0]["text"]
```

The intent confirmation prompt also contains specific instructions for the completion api to ensure that expected dictionary format is achieved.

Once the intent confirmation layer is passed, the result is sent to dictionary_present() function that converts the result into python dictionary using extract_dictionary_from_string() function.

```
def dictionary_present(response):
    delimiter = "####"
    user_req = {'Source': 'Delhi','Destination': 'Mumbai','Duration':
'3','Budget': '20000'}
    prompt = f"""You are a python expert. You are provided an input.
            You have to check if there is a python dictionary present in
the string.
            It will have the following format {user_req}.
            Your task is to just extract and return only the python
dictionary from the input.
            The output should match the format as {user_req}.
            The output should contain the exact keys and values as present
in the input.

            Here are some sample input output pairs for better
understanding:
```

```
            {delimiter}
            input: - 'Source': 'Delhi','Destination': 'Mumbai','Duration':
'3','Budget': '20000'
            output: {{'Source': 'Delhi','Destination':
'Mumbai','Duration': '3','Budget': '20000'}}

            input: - 'Source': 'Mumbai','Destination': 'Pune','Duration':
'5','Budget': '50000'
            output: {{'Source': 'Mumbai','Destination': 'Pune','Duration':
'5','Budget': '50000'}}

            input: - 'Source': 'Bangalore','Destination':
'Mumbai','Duration': '3','Budget': '10000'
            output: {{'Source': 'Bangalore','Destination':
'Mumbai','Duration': '3','Budget': '10000'}}

            {delimiter}

            Here is the input {response}

            """
    response = openai.Completion.create(
        model="text-davinci-003",
        prompt=prompt,
        max_tokens = 2000
        # temperature=0.3,
        # top_p=0.4
    )
    return response["choices"][0]["text"]
```

Once the dictionary is extracted, it is sent to fetch_travel_iternary() function that compares the dictionary ({{'Source': 'Mumbai','Destination': 'Pune','Duration': '5','Budget': '50000'}}) with ***fllight_data.csv*** file contents.
First the source, destinations, and budget are matched to get the outbound flights. Secondly the locations are matched in a reverse manner and budgets are matched to get the inbound flights.

```
def fetch_travel_iternary(user_req_string):
    flight_data= pd.read_csv('flight_data.csv')
    print('user_req_string',user_req_string)
    user_requirements = extract_dictionary_from_string(user_req_string)
    print(user_requirements)

    budget = int(user_requirements.get('budget', '0').replace(',',
'').split()[0])
    print('budget', budget)
    #This line retrieves the value associated with the key 'budget' from
the user_requirements dictionary.

    filtered_flight_data_source = flight_data.copy()
    filtered_flight_data_source =
filtered_flight_data_source[filtered_flight_data_source['source_city'].str
.lower() == user_requirements['source']]
    filtered_flight_data_source =
filtered_flight_data_source[filtered_flight_data_source['destination_city'
].str.lower() == user_requirements['destination']]
    filtered_flight_data_source =
filtered_flight_data_source[filtered_flight_data_source['price'] <=
budget].copy()
    #These lines create a copy of the flight_data dataframe and assign it
to filtered_flight_data.
    filtered_flight_data_destination = flight_data.copy()
    filtered_flight_data_destination =
filtered_flight_data_destination[filtered_flight_data_destination['source_
city'].str.lower() == user_requirements['destination']]
    filtered_flight_data_destination =
filtered_flight_data_destination[filtered_flight_data_destination['destina
tion_city'].str.lower() == user_requirements['source']]
    filtered_flight_data_destination =
filtered_flight_data_destination[filtered_flight_data_destination['price']
<= budget].copy()

    print('filtered_flight_data_source', filtered_flight_data_source)
    print('filtered_flight_data_destination',
filtered_flight_data_destination)

    df_merged = pd.concat([filtered_flight_data_source,
filtered_flight_data_destination], ignore_index=True, sort=False)
```

```
    return df_merged.to_json(orient='records')
```

Challenges:

- While working on this task there were a lot of challenges in setting up the virtual environment and running the code
- Initial prompt instructions had to be tweaked properly to ensure that only Indian cities are considered as source and destination.
- Format of the dictionary had to be specified properly and values had to be converted to lower case to allow comparisons between the user input and the flight corpus.
- It was challenging to make the assistant stick to the scope of the application. It sometimes led to hallucinations when the conversation was a little different.

Lessons learned:

- Prompts need to be very informative with chain of thought and few shot techniques
- The expected format of the output needs to be specified with few shot techniques are well to avoid issues with processing
- Chatbot may lead to hallucinations when different kind of inputs are given
- It was sometimes difficult to control the bot to stick to the context of the application.