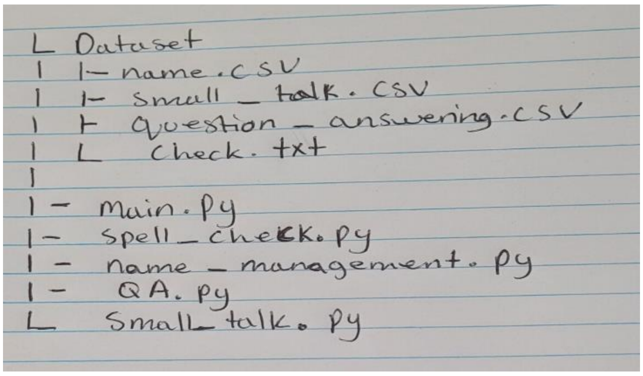
COMP3074-HAI – Coursework

This is the COMP3074 Coursework Human-AI Interaction coursework on building an interactive NLP-based system. The chatbot created is a flight booking system prototype implementing features such as intent matching, Identity management, transactional dialogue, information retrieval and question answering and small talk along the way. All of which will be described in this report along with an evaluation of the system and how I feel it could be improved in the future.

Below is an example of the structure of the files. Question answering, small talk and name are part of the core architecture of the chatbot therefore I felt developing each functionality in its own individual file was the best way to go. The 'main.py' file hold the is where each file is called and contains the chatbots core logic. Some of the main reasons I decided to spit the files was to get a cleaner and clearer structure making it easier to understand for future developers and it also makes it much easier when it comes to debugging and unit testing.



```
L Dataset
|  |- name.csv
|  |- small_talk.csv
|  |- question_answering.csv
|  L  check.txt
|
|- main.py
|- spell_check.py
|- name_management.py
|-  QA.py
L  small_talk.py
```

Question answering

Question answering involves interpreting the meaning of the question, searching through relevant information or knowledge sources, and generating an accurate and concise response. I tried to incorporate this through retrieval-based question answering so the user could ask a question and the system would use the predefined 'question_answering.csv' dataset to find an exact or closely related match to answer question. Text vector was first pre-processed via stemming as it takes less time to achieve text form compared to lemmatization, then TF-IDF was applied to the 'question_answering.csv' data, increasing the weight of less used words while decreasing the weight of high-frequency words.

$$\text{similarity}(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|}$$

$$TF = \frac{\text{Number of times a word "X" appears in a Document}}{\text{Number of words present in a Document}}$$

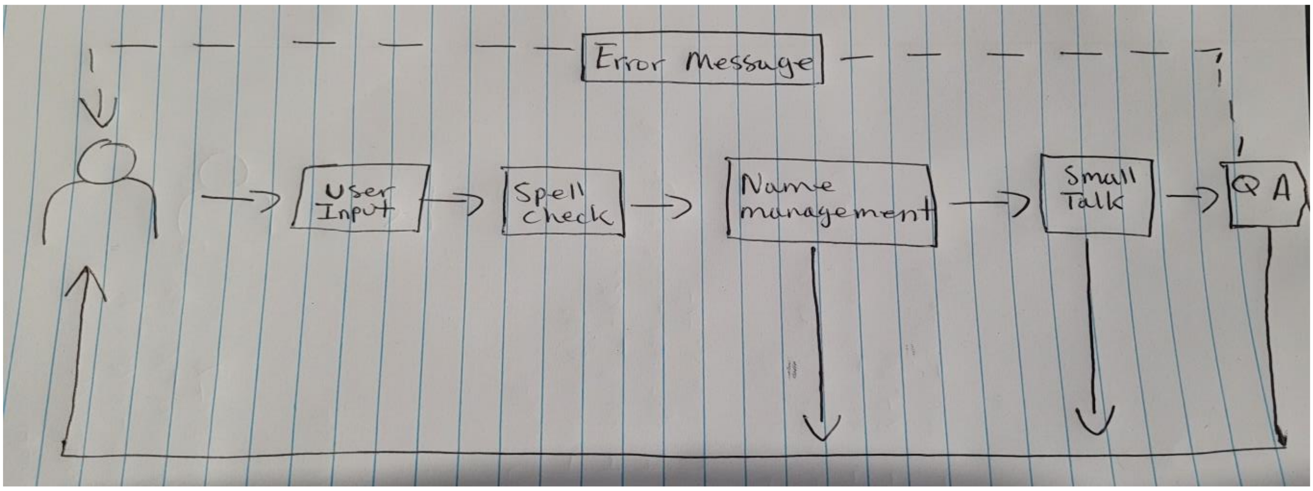
$$IDF = \log\left(\frac{\text{Number of Documents present in a Corpus}}{\text{Number of Documents where word "X" has appeared}}\right)$$

$$TF\,IDF = TF * IDF$$

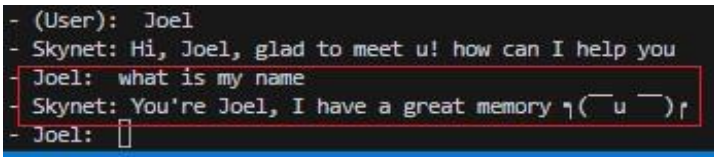
Cosine similarity function was also applied in order to compare the users question with the dataset, with a threshold set to decide if question matches with 'question_answering.csv' dataset. If it doesn't an error message is sent

Intent matching

The chatbot has been designed in a way that there are set priorities checking the users input and analysing them. As shown in the graph below the system checks, name management, small talk then question answering, with the order being how long the task should take.

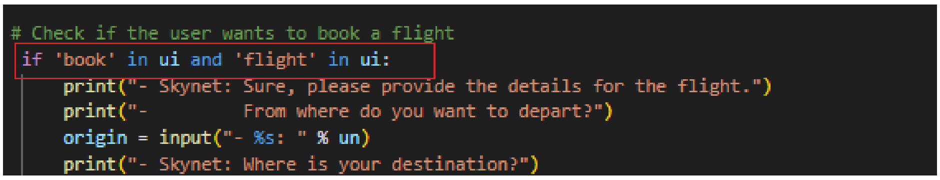


Name related question are handled by name management and question answering has a fairly large dataset it has to go through therefore it's not put in first as it may cause an unnecessary delay when the user simply asked what their name is. Also this alleviates biases by putting question answering at the end as less data is involved in name management with the tasks being cantered around one clear task therefore being less data.



```
- (User):  Joel
- Skynet: Hi, Joel, glad to meet u! how can I help you
- Joel:  what is my name
- Skynet: You're Joel, I have a great memory ┐(￣u ￣)┌
- Joel:  
```

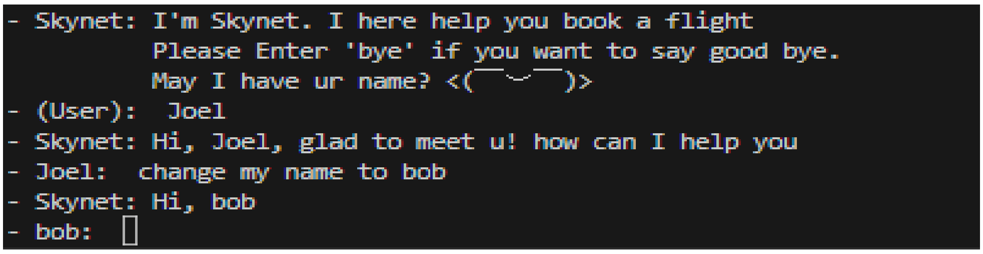
One of the main reasons I did not add an intent classifier in my design was because I felt especially for QA stop words should be cleaned in order to find the keywords, and it's not efficient applying three pre-processing techniques to 3 separate tasks within the same classifier. I felt that that name management tasks especially would be negatively affected with whole sentences such as 'what is my name' potentially being removed. Another reason intent classifier was not applied was because of the uneven distribution of the dataset, therefore it's better to use the intent tasks differently instead of having the same classifier, this ties into data dependency as although I felt I had sufficient data it may end up being more biased due to the discrepancy in size of the name management data compared to the question answering data which may have resulted in misclassification or limited recognition of diverse intents. Simple text matching was also used in this system. For example to begin the transaction. Below is a snippet of code of text matching:



```
# Check if the user wants to book a flight
if 'book' in ui and 'flight' in ui:
    print("- Skynet: Sure, please provide the details for the flight.")
    print("-          From where do you want to depart?")
    origin = input("- %s: " % un)
    print("- Skynet: Where is your destination?")
```

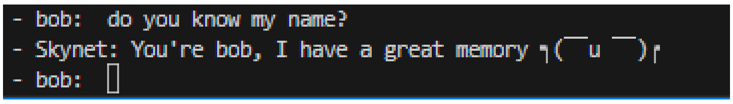
Name management

Using a person’s name creates a more personalised experience, making the ser more engaged as the interaction is more human like, therefore allowing the user to build a rapport with the chatbot. In the chat bot the first thing that is asked of the user is their name, and in the example below the user is able to change their at any time. This was achieved through the storage of several relevant words such as ‘rename’ and ‘switch’ and if the user was to use one of these words this would indicate to the system user may want to change their name. The new name would then be taken from the user’s input and through the filtration of the stop words and the ones on the checking list.



```
- Skynet: I'm Skynet. I here help you book a flight
          Please Enter 'bye' if you want to say good bye.
          May I have ur name? <(￣︶￣)>
- (User):  Joel
- Skynet: Hi, Joel, glad to meet u! how can I help you
- Joel:  change my name to bob
- Skynet: Hi, bob
- bob:  ▯
```

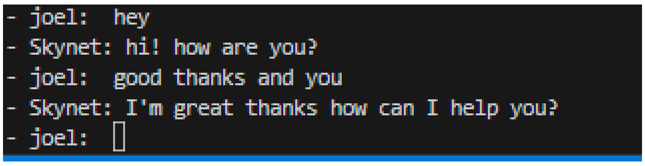
I created a matching dataset with questions that are comparable to make sure the algorithm could output the username in certain situations, such as "Do you know my name?" Small conversation and query retrieval use the same methodology.



```
- bob:  do you know my name?
- Skynet: You're bob, I have a great memory ╮(￣u ￣)╭
- bob:  ▯
```

Small talk

Small talk contributes to a more natural flow of conversation, making the users interaction with the system feel less scripted and robotic. The small talk data set was built for exactly this purpose. All text data associated with the query will be vectorized into TF-IDF format for searching purposes. To assess the Cosine similarity, a threshold is established. Here, the data is neither lemmatized or stemmed; otherwise, the entire sentence may be eliminated. One way of ensuring natural conversation was having different answers added to the dataset for the same query, enabling the algorithm to select one at random. Additionally, for a few greeting expressions, they are found in separate QA pairs' "question" and "answer" data columns. Rather than constantly needing the user to inquire, the system will ask a few questions, answer them, and then respond. The dataset may therefore help the conversation flow naturally. An example of this is show below:

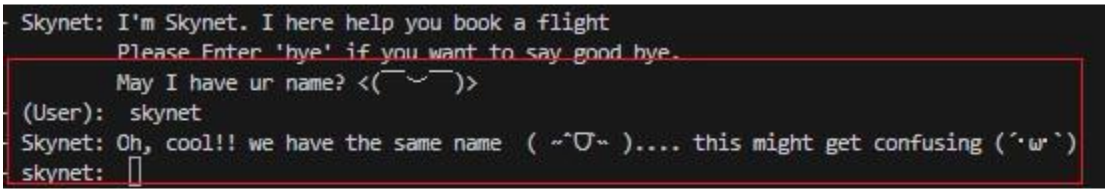


```
- joel:  hey
- Skynet: hi! how are you?
- joel:  good thanks and you
- Skynet: I'm great thanks how can I help you?
- joel:  ▯
```

Performance and Additional design features

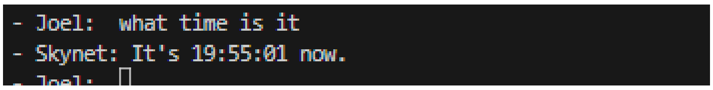
The system is able to perform some tasks such as tell the time correct typo's and be able tell if user has the same name as system.

The system is able to recognise if user has same name as it as shown in the example below. I decided to include this feature as I felt it made the chatbot more personable with emojis added to convey emotion.



```
Skynet: I'm Skynet. I here help you book a flight
        Please Enter 'bye' if you want to say good bye.
        May I have ur name? <(￣︶￣)>
(User):  skynet
Skynet: Oh, cool!! we have the same name  ( ~ˆ▽ˆ~ ).... this might get confusing (´·ω`)
skynet:
```

System is also able to tell the time as shown in the example below

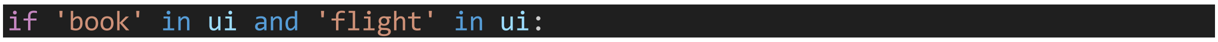


```
- Joel:  what time is it
- Skynet: It's 19:55:01 now.
```

Various forms of user input have been used to test the chatbot. The outcomes show that the chatbot functions reasonably well. The entire interaction is carried out in a loop, making it possible to classify various intents each time. Regarding name management, the user can modify their name on the system, and book a flight as shown in the example below. Additionally, the algorithm effectively manages casual conversation with the help of my datasets.

When it comes to quality assurance, the system can respond to the majority of queries, but it is limited in its ability to fully comprehend the user's true question because terms like "are" and "what" are regarded as stop words. Although the system is far from perfectly was at least able to correctly match the keywords given by the user cross referencing it with what is available in its data. Below is an example of booking a flight in the system.

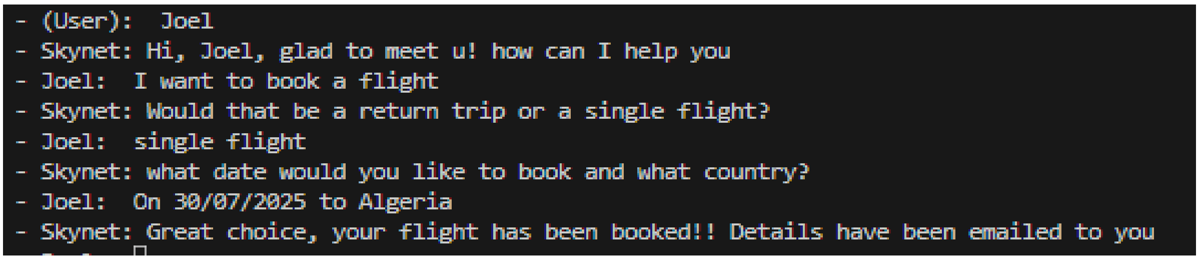
As you can see in both images below, the user is at first asks to book a flight, which triggers the if statement and begins the process. The statement through the use of the keywords book and flight:



```python
if 'book' in ui and 'flight' in ui:
```

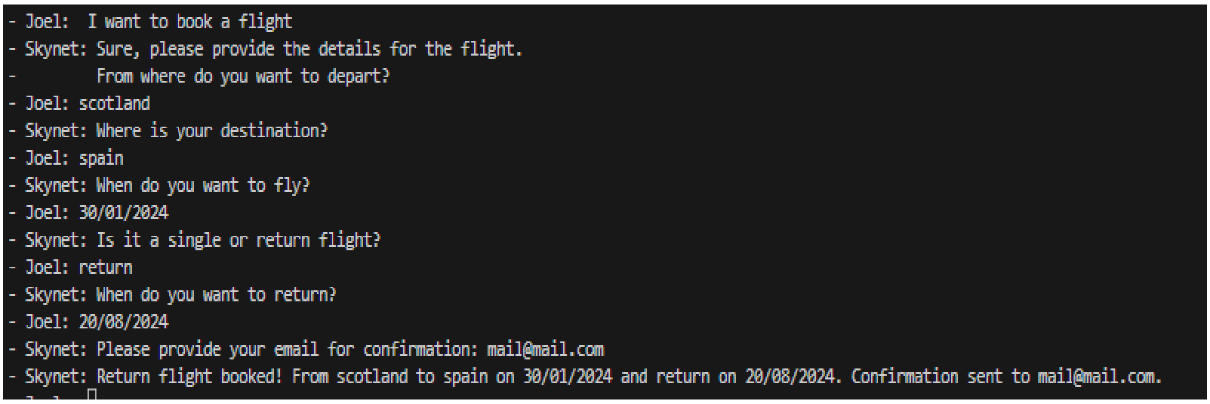
The user is then taken through the classic booking a flight steps with an important crossroad when user is asked if flight is a single or return. A is an example of the user choosing a single flight with b the return flight example.

a)



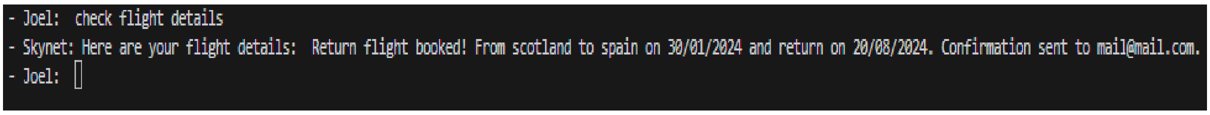
```
- (User):  Joel
- Skynet: Hi, Joel, glad to meet u! how can I help you
- Joel:  I want to book a flight
- Skynet: Would that be a return trip or a single flight?
- Joel:  single flight
- Skynet: what date would you like to book and what country?
- Joel:  On 30/07/2025 to Algeria
- Skynet: Great choice, your flight has been booked!! Details have been emailed to you
```

b)



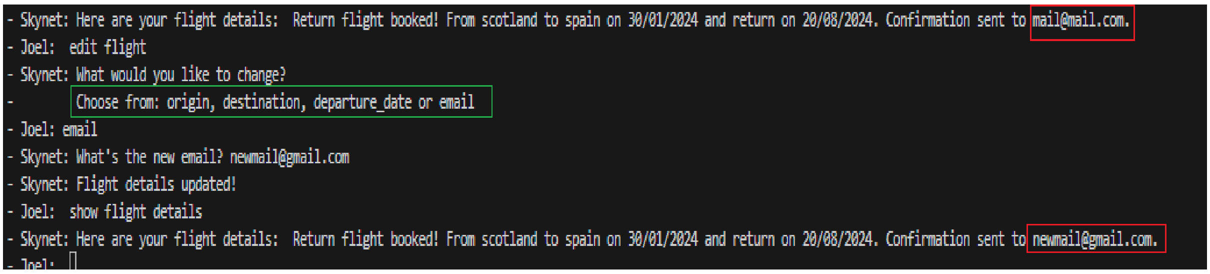
```
- Joel:  I want to book a flight
- Skynet: Sure, please provide the details for the flight.
-         From where do you want to depart?
- Joel: scotland
- Skynet: Where is your destination?
- Joel: spain
- Skynet: When do you want to fly?
- Joel: 30/01/2024
- Skynet: Is it a single or return flight?
- Joel: return
- Skynet: When do you want to return?
- Joel: 20/08/2024
- Skynet: Please provide your email for confirmation: mail@mail.com
- Skynet: Return flight booked! From scotland to spain on 30/01/2024 and return on 20/08/2024. Confirmation sent to mail@mail.com.
```

Once a booking has been made user is able to check flight details as showcased below:



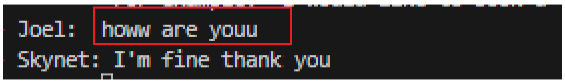
```
- Joel:  check flight details
- Skynet: Here are your flight details:  Return flight booked! From scotland to spain on 30/01/2024 and return on 20/08/2024. Confirmation sent to mail@mail.com.
- Joel:
```

User is also able to edit flight details. They are first asked what part of the flight details they would like to change with the options then listed as you can see highlighted in green. In the example below user chooses to change email with the red highlighted the old email which was ‘mail@mail.com’ and the new one being ‘newmail@gmail.com’.



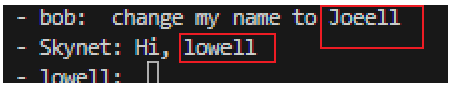
```
- Skynet: Here are your flight details:  Return flight booked! From scotland to spain on 30/01/2024 and return on 20/08/2024. Confirmation sent to mail@mail.com.
- Joel:  edit flight
- Skynet: What would you like to change?
-         Choose from: origin, destination, departure_date or email
- Joel: email
- Skynet: What's the new email? newmail@gmail.com
- Skynet: Flight details updated!
- Joel:  show flight details
- Skynet: Here are your flight details:  Return flight booked! From scotland to spain on 30/01/2024 and return on 20/08/2024. Confirmation sent to newmail@gmail.com.
- Joel:
```

There’s also a spell check element of this chatbot which I felt was necessary I feel it helps in clarify the user's intent, reducing the chances of miscommunication and ensuring that the system provides relevant and accurate responses.



```
Joel:  howw are youu
Skynet: I'm fine thank you
```

One con of this is that when it comes to changing names the system may think certain names are Typos and change them name as seen in the example below where I wanted the name set as ‘Joeell’ but it thought it was a typo and changed it to ‘lowell’:



```
- bob:  change my name to Joeell
- Skynet: Hi, lowell
- lowell:
```

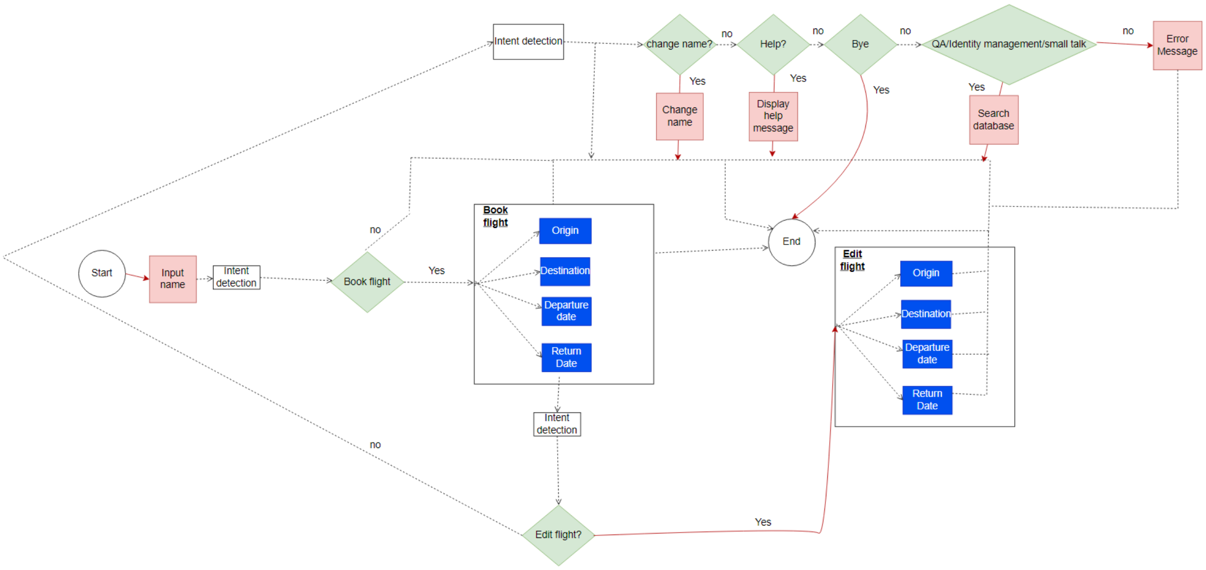
My chatbot can typically carry out a natural conversation and perform the task it is built for which is to book a flight. In order to give the user a variety of responses answers to the small talk chat are varied and randomly selected, and have added emojis to some fixed responses such as the name management section to convey emotion

Conversational design

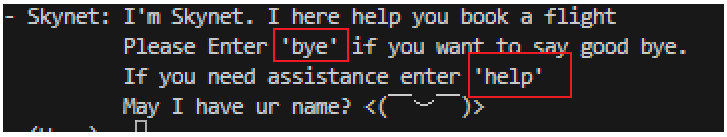
The chart below shows the workflow of the flight booking system with the solid red lines representing the procedures that have to be taken due to users triggering an event of some sort, and the dashed lines represent optional steps from which the user can choose from.

One example of this is when the user starts the program they are asked to input their name, the event they are triggering is them running the program. Another example is when the intent detection is triggered, with the kite shapes in green representing both similarity based and machine learning based approaches. As shown in the workflow the system goes through check, first if user is asking for name change, it will go down the red arrow and change the name, if asked for something else it will keep going through it's check which include: 'Help', system checks if user is asking for help. 'Bye', system check if user wants to shut down application, and lastly it will check if QA, small-talk, OR identity management will solve users query. If not an error message is ran.

When it comes to editing a flight, the intent detection is triggered and looking for the key words 'edit' and 'flight'. If the intent matches then the editing flight process can begin, with the flight editing options highlighted in blue in the 'edit flight' section of the graph. The lines when it comes to editing are dashed and not strait lined meaning it's optional for the user to edit even once event has been triggered, unlike the change name section which requires the user to change once they've chosen that option.

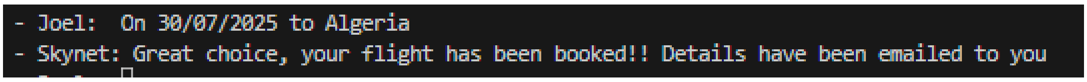


1.) Prompt design
Key phrases and conversational markers are used throughout the design of the chatbot. I tried to build in a way that effectively engages users, encourage interactions, and facilitate the achievement of the chatbot's objectives. One example of this is the first thing the user is shown when starting the chatbot is the chatbot's purpose, which I felt was necessary as it straight away let's the user know the type of questions it can answer. User is also from the beginning made aware, and given instructions on how to end chatbot and how to get help.



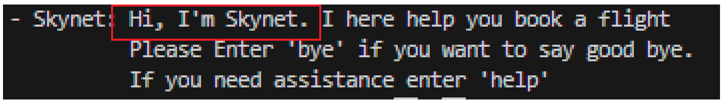
```
- Skynet: I'm Skynet. I here help you book a flight
          Please Enter 'bye' if you want to say good bye.
          If you need assistance enter 'help'
          May I have ur name? <(￣︶￣)>
```

I tried to incorporate positive feedback as shown when a flight has been booked through the use of positive words such as ‘Great’.



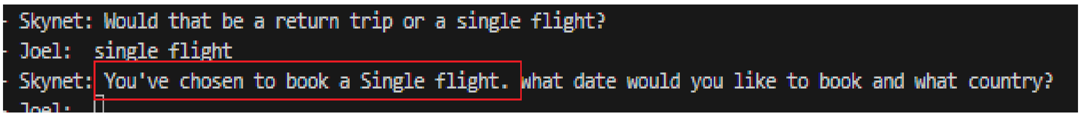
```
- Joel:  On 30/07/2025 to Algeria
- Skynet: Great choice, your flight has been booked!! Details have been emailed to you
```

Example of Greeting in chatbot.



```
- Skynet: Hi, I'm Skynet. I here help you book a flight
          Please Enter 'bye' if you want to say good bye.
          If you need assistance enter 'help'
```

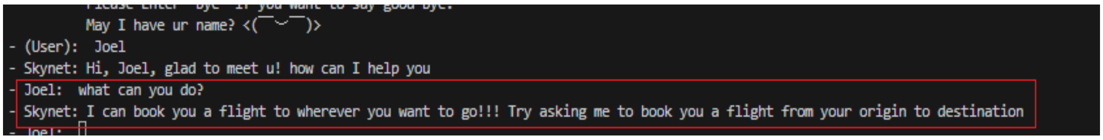
Also acknowledgement which can be showcased through confirmation of the type of flight user is choosing to book.



```
- Skynet: Would that be a return trip or a single flight?
- Joel:  single flight
- Skynet: You've chosen to book a Single flight. what date would you like to book and what country?
```

2.) Discoverability

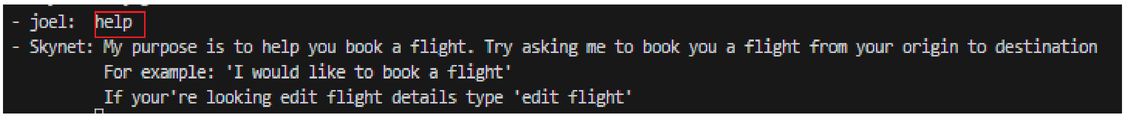
As shown in the example above user are straight away made aware of the chatbot’s purpose, and once the system has their name, it as them what it can do for them, the user can then go ahead and ask to book a flight or even go ahead or maybe ask a question such as ‘what can you do’ and system will explain it’s purpose in more detail, as shown in the example below.



```
          May I have ur name? <(￣︶￣)>
- (User):  Joel
- Skynet: Hi, Joel, glad to meet u! how can I help you
- Joel:  what can you do?
- Skynet: I can book you a flight to wherever you want to go!!! Try asking me to book you a flight from your origin to destination
```

Automatic strategy was also used through the use of intent matching and also making a conscience effort to make all the questions system asks relevant to the overall purpose of the design.

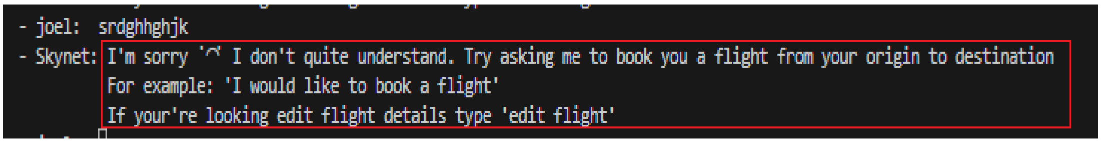
Response design is taken into consideration, for example if the user was to type help they would get a response explaining what the user can try to do to help.



```
- joel:  help
- Skynet: My purpose is to help you book a flight. Try asking me to book you a flight from your origin to destination
          For example: 'I would like to book a flight'
          If your're looking edit flight details type 'edit flight'
```

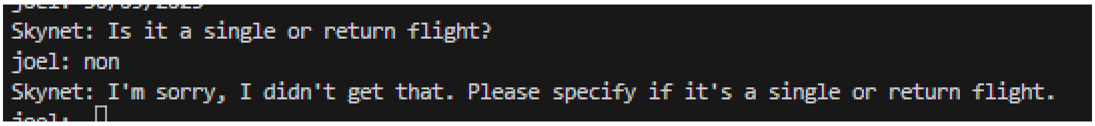
3.) Error handling

Error handling in chatbots is crucial for maintaining smooth and effective conversations. The ways I tried to handle this is by having clear error message and offering a suggestion. I felt offering the suggestion was crucial when it can to error handling as it allows the user to understand the route they have to potentially follow to get back on track with they want to get out of the system. Below is an example from the system:



```
- joel:  srdghhghjk
- Skynet: I'm sorry '^' I don't quite understand. Try asking me to book you a flight from your origin to destination
          For example: 'I would like to book a flight'
          If your're looking edit flight details type 'edit flight'
```

Another error handling example is when user does not specify if they want to book a single or return flight, the system responds by saying ‘I'm sorry, I didn't get that. Please specify if it's a single or return flight’:

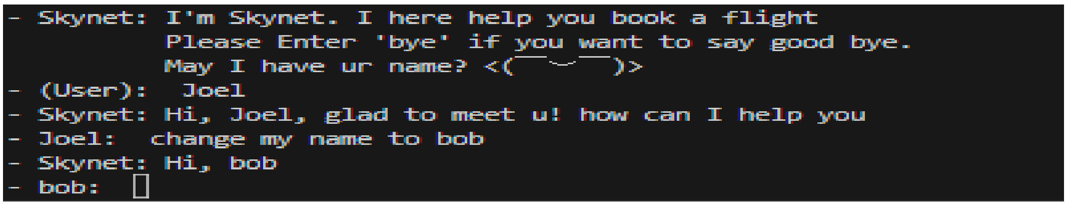


```
Skynet: Is it a single or return flight?
joel: non
Skynet: I'm sorry, I didn't get that. Please specify if it's a single or return flight.
```

4.) Personalisation

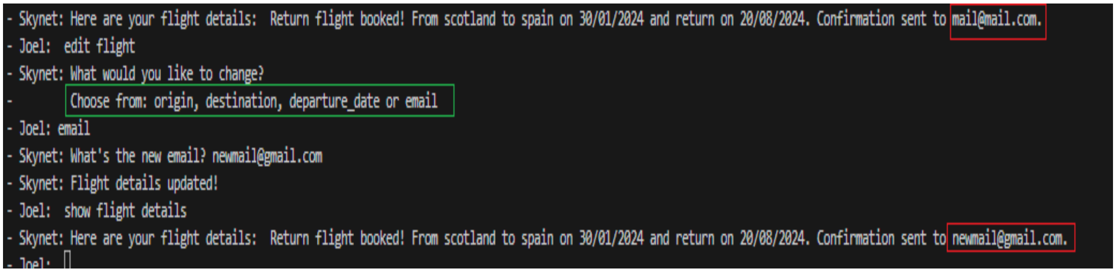
Personalisation plays an important role in enhancing the user experience and one way I included this in the chatbot was by allowing the user to not only add their name but change it if they wanted to. I feel this enhances the users experience making the interaction more engaging and increasing the likelihood of them continuing the conversation.

Change name



```
- Skynet: I'm Skynet. I here help you book a flight
          Please Enter 'bye' if you want to say good bye.
          May I have ur name? <(￣︶￣)>
- (User):  Joel
- Skynet: Hi, Joel, glad to meet u! how can I help you
- Joel:  change my name to bob
- Skynet: Hi, bob
- bob:
```

Change flight details

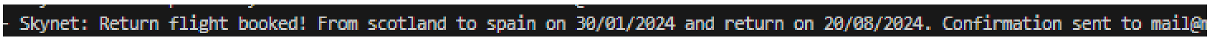


```
- Skynet: Here are your flight details:  Return flight booked! From scotland to spain on 30/01/2024 and return on 20/08/2024. Confirmation sent to mail@mail.com.
- Joel:  edit flight
- Skynet: What would you like to change?
-         Choose from: origin, destination, departure_date or email
- Joel: email
- Skynet: What's the new email? newmail@gmail.com
- Skynet: Flight details updated!
- Joel:  show flight details
- Skynet: Here are your flight details:  Return flight booked! From scotland to spain on 30/01/2024 and return on 20/08/2024. Confirmation sent to newmail@gmail.com.
- Joel:
```

5.) Confirmation

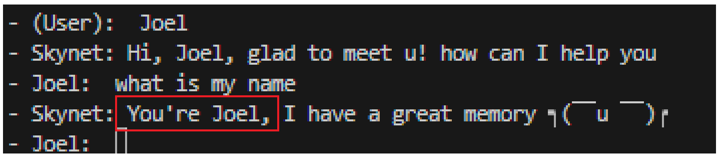
I use both explicit and explicit confirmation throughout the system and fell they are effective as it clearly get the message it’s trying to convey to the user. Below is an example of both.

Explicit confirmation



```
- Skynet: Return flight booked! From scotland to spain on 30/01/2024 and return on 20/08/2024. Confirmation sent to mail@
```

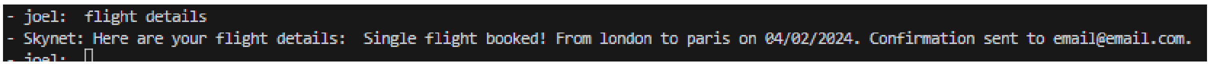
Implicit confirmation



```
- (User):  Joel
- Skynet: Hi, Joel, glad to meet u! how can I help you
- Joel:  what is my name
- Skynet: You're Joel, I have a great memory ╮(￣u ￣)╭
- Joel:
```

6.) Context tracking

Context tracking is crucial when developing a chatbot as it enables the bot to maintain an understanding of ongoing conversations, ensuring coherent and relevant interactions. An example of this is system recalling flight details upon request from the user:



```
- joel:  flight details
- Skynet: Here are your flight details:  Single flight booked! From london to paris on 04/02/2024. Confirmation sent to email@email.com.
- joel:
```

Evaluation

During the development of the prototype unit tests where performed to ensure individual blocks of code work as expected such as if/else statements, and intent activation.

Integration testing was also a big part as I had to combine the units to make sure they where compatible and working as intended. Each unit was tested along as weather or not it was performing the specific task it’s intended for such as user management etc.

System testing began once the integration was done. I the system multiple times fixing problems that can along the way, with the aim of making the system as streamlined as possible.

The next step was usability testing. I asked a few friends to test the entire application to make sure it was user-friendly, and I made changes depending on their input. Generally, the user-system interaction assisted me in reviewing the features to ensure that they are simple to use.

I’ll be the first to make it clear the system is no perfect below is my evaluation on things that could be improved on in the future:

The system is not really smart enough and is only useful during certain scenarios therefore cannot answer a multitude of questions. Various kinds of datasets could be added. It is possible to train deep neural networks for the system design. The memory-based discussion could be handled by structures like RNN and LSTM.

By enlarging the small talk dataset I could handle more expressions about name management question answering and small talk.

The spell check can also only really fix misspelled words, to better this I would introduce semantics into it, allowing the system to guess what the user really wants to convey.

Something that could definitely be improved is the chatbots accessibility. Making technology accessible is an important RRI principle and I feel in future, taking more into account users with disabilities would massively improve it. For example, including the capability for text speech.

The RRI framework was always at the forefront through the design of the system.

Anticipate- I tried to consider how the system might influence travel patterns and although I could potentially promote eco-friendly travel options the creation of this system, it would also definitely have environmental implications such as the rise of carbon footprints, with this being a flight booking system.

Reflect- consideration of how the system handles user data is something I sent a while reflecting on, with the idea of providing a fair and transparent service for users. In the current system the only bit of personal information the user has to give is name and email address. In future developing a payment system behind this would be ideal, and would make a conscious effort to make sure users payment details are not stored one payment has gone through as it's unnecessary.

Engage- I tried to make the system as engaging as possible one way was by initiating interaction as described throughout the report. I felt this was important as it gets the user to get a feeling of the sense of what the system can do, through further interaction.

Act- I was sure to act upon the insights gained to design and implement the system ethically, and feel upon further development it would be important to monitor the systems potential impact on society no matter how small.

By embracing each aspect of the AREA framework, I felt confident anticipating potential challenges, and took into consideration ethical implications of the system, therefore believe the system aligns itself with societal values.