Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

Enhancing CUI Designs
Using a Chatbot Prototyping Tool

Swagat Dash

August 19, 2022

A dissertation submitted in partial fulfilment
of the requirements for the degree of
M.Sc. in Computer Science (Data Science)

Declaration

I hereby declare that this dissertation is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at `http://www.tcd.ie/calendar`.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at `http://tcd-ie.libguides.com/plagiarism/ready-steady-write`.

Signed: ______________________ Date: ______________________

Abstract

Many large companies have shown tremendous interest in chatbot development and its use lately. With most of the tech giants providing their own platforms of development, many developers have jumped into developing a plethora of chatbots. However, studies have reported that most of the chatbots have caused displeasure to many survey participants. The design procedure of chatbots or CUIs in general thus still remains a bit unclear and thus is an area which needs significant research. To enhance such a research space which studies how design characters impact user experience, we need good tools which let researchers prototype chatbots quickly. Existing chatbot prototyping tools get too bulky with loads of features and thus are difficult to adopt. Also, most of them don't facilitate easy integration to other platforms and thus negatively affect the overall user experience. The scope of this research was to study the requirements and expectations which chatbot users have had. This in turn helped us explore the design of an NLP enabled chatbot prototyping tool. The research outcomes helped us develop a chatbot prototyping tool that helps researchers prototype chatbots with ease. The prototypes created can be integrated to any experiment workflow. The easy integration will let users interact with the chatbots in the same survey setup (e.g. Qualtrics) without having to move to another platform. We evaluated the tool developed for various scenarios and domains which have majorly seen the use of chatbots and were able to successfully create sample conversation in these domains.

Acknowledgements

I would like to extend my sincere gratitude to my dissertation supervisor, Dr. Gavin Doherty for his valuable and constructive feedback throughout my research work. His guidance helped me proceed in the right path.

I would also like to thank Mr. Rob Bowman for his consistent guidance. His domain knowledge expertise helped me get the right literature. His constructive and diligent feedback throughout the course of this research helped me identify and address bottlenecks quite early in the process.

I would like to extend my gratitude to my second reader as well. Ms. Gaye Stephens gave me feedback on the demo which did help me further refine the work I had done by then.

I am also thankful my Department. The Computer Science and Statistics Department of Trinity College Dublin provided me with the environment to succeed in my work.

Finally, I would like to extend my gratitude to my family and friends for all the motivation and support during this wonderful journey.

Contents

List of Figures

1 Introduction

1.1 Background

Since the development of world's first chatbot (ELIZA) in 1966 (6), there was very little development in the years that followed. With advancement in the fields of NLP and AI in general, chatbots finally gained limelight again after around 2016. There were huge numbers of chatbots developed using different platforms. Facebook reported 300,000 chatbots developed by CUI developers in its platform only in the year 2018 (7). However, soon after that, the world witnessed a shift to Voice User Interface (VUI) with the emergence of Intelligent Personal Assistants (IPAs) like Siri, Google Assistant etc. Due to this shift, chatbot – which is described as a software program using which a human computer interaction can be carried out in Natural Language (mostly using texts) – still remains an area which is very underexplored. In (8), Dave Feldman, the then Design Manger of Facebook Messenger Platform suggested that lack of documentation and best practices knowledge transfer by major tech giants who created chatbot development tools could be a reason, which led to bad chatbot designs being made. There is a lot of research evidence which insists that the CUI design process is quite different than that of GUIs (Graphical User Interfaces) and is complicated as well. Hence, to formulate standards in the CUI design space, researchers need to have right set of tools, which would help them quickly and easily prototype different chatbots with fairly different design characteristics, so that user experience could be evaluated for each of them. The presence of such tools is very scarce today. The ones which are present have got a steep learning curve and also, with the plethora of 'good-to-have' features they provide, using them feels like developing a chatbot more than prototyping one. Thus, the existence of such a tool which facilitates rapid prototyping of chatbots design will speed up the research process and will add efficiency to the process.

1.2 Motivation

We have already explained in the sections above how the craze for chatbots development has increased over the years and how chatbots have become a huge part of how we interact with different platforms today. Designing conversation is indeed a complicated interpretation task, as mentioned in (9). Unlike GUI, we don't explain users how to use this or how to interact with the interface. The ask here is rather to interpret what the intent of the user is and then adapt to it on the go. To get the basics of such designs clear, there is a lot of continuous user feedback needed. People have tried to solve this problem by chatbot prototyping tools (10, 11). However, the existing tools have a whole lot of features which make the platform very complicated. This implies that there is a steep learning for any developer to be able to develop chatbot prototypes from these platforms. The sophisticated features do add value to the final product, but the added complicacy defies the very reason of their existence. It sometimes feels like actually developing a chatbot rather than prototyping one. The best way the prototypes can be used, is by getting embedded into other platforms where the research survey is being carried out. However, existing tools lack this capability. Mostly due to their 'X-Frame-Options' response header set to 'SAMEORIGIN', they prevent their webpages to be embedded onto other platforms. Generally, X-Frame-Options (Now obsolete and replaced by 'Content-Security-Policy' HTTP header's 'frame-ancestors' directive) is used to decide whether or not a response page should be allowed to be rendered inside an <iframe> or <embed> tag. When set to 'SAMEORIGIN', the response page can be embedded onto other pages which have been hosted in the same domain. This particular directive of the existing prototyping tools prohibits the chatbots to be embedded into any experiment platform. As a result of which, the users may need to navigate to other platforms to interact with the chatbot and then navigate back to the experiment flow to be able to answer the survey questions. This proves to be a bit of a hassle for the users. Thus, the research outcome will help us design a tool that will:

- Facilitate seamless integration
- Provide Ease of Development
- Have very limited design effort

1.3 Problem Statement

As mentioned above, while designing a chatbot, there are a lot of design decision that a developer needs to take. This includes (but not limited to) deciding what could be the appropriate response to certain text, what would be the domain of the chatbot, whether the chatbot will store the conversation context or not, whether or not to add personification to a chatbot etc. These design decisions combined together make the whole chatbot design process way more complicated than it initially is anticipated to be. Along with this, while doing research on chatbots, researchers do need users to interact with different chatbot designs. The feedback so received helps them come up with CUI design principles and helps better understand how design characteristics of such interfaces can impact the user experience. For them to be able to study implications of different designs, they need to quickly design multiple such prototypes with ease, which the users can later interact with. Currently, the availability of such tools is limited and the ones that are present do not suffice the needs of researchers with respect to points mentioned earlier in this report. This is where the need of such a tool is felt, which would let researchers quickly protype multiple such designs having different design characteristics. Having the ease of use is one of the main factors that users would look for in such a tool. This tool in turn will help them get research outcomes that will derive standard design principles for such Human-to-Computer interactions and overcome some of the major design challenges as mentioned below (7):

1. Difficulties and complications in tasks – When chatbots are designed for specific tasks, it gets really difficult to be able to address all the possible use cases. Once the development starts, the task may turn out to be much more complicated than it was expected to be (e.g: Booking an appointment for a patient with a dentist). The requirement gathering of such developments is thus not as straight forward as it may be for a GUI development. The ideal way of doing it should be an iterative process.

2. Difficulties and complications in Conversations – The flow of a conversation can never be just one defined way. Since here we are talking about human-like conversations, we must acknowledge that such conversations will hardly ever be uni-directional. They may go back and forth, in circular loops, ask for clarifications or many other ways.

3. Natural Language is complicated – Even though the advancement in NLP, natural language understanding still remains a tough task and gets tricky to match user expectations. When the chatbots try to have the conversation using Natural language (mainly through text), the key difficulty is to understand the user's intent in the absence of gestures, tones and expressions.

4. Too many platforms to design chatbots – Having many platforms to develop chatbots can sometime lead to confusion. It becomes very crucial to study the capabilities and lacunas of all such platforms to find one that matches the requirements. Thus, having the requirements properly defined in turn becomes vital. Also, having the best practices drafted for such platforms is very important for easier adoption of such platforms by developers. The absence of which, may result in bad designs being developed and in turn poor performance of chatbots.

5. Whether to add a personality or not? – Personification of chat bot adds dimensions to the interaction and makes it much more interesting and human-like. However, as soon as it is done, the expectation of end-user increases. Users would expect to get more human-like sensible answers to questions. The scope of having errors in interpretation decreases significantly in such cases. Also, if the personality of the user doesn't match

the personality of the chatbot, it can cause severe displeasure (3, 12).

Rest of the flow of this document will start with a bit of Literature Review. We will go through existing studies to set a base for our research. This will help us understand the evolution of chatbots, user-expectations, and challenges associated with them. We will then use the knowledge gained along with a bit of reverse engineering to derive the requirements of our tool. The Design Process will demonstrate how the design of the tool was finalized in multiple iterations. It will talk about some of the major design decisions taken during the course of this research and development. This will be followed by the engineering choices made for the tool. This part will elaborate on how such a tool, when publicly published can ensure scalability and availabilty. After which, we will evaluate the tool by prototyping multiple chatbots from different domains discussed in the document. We will see if such a tool can fit into different domains efficiently by having sample chats from such common domains like healthcare, entertainment etc. implemented. The report will finally have the Discussion and Conclusion sections which will re-iterate on the findings, implication, evaluation results, limitations, and potential future work.

2 Literature Review

What is a Chatbot?

A Chatbot is a software solution which tries to mimic a conversational agent. The agent generally exchanges dialogs with the users in a certain context (e.g.: Travel, Shopping, News, Entertainment, Leisure etc.) using natural language (1). Generally, conversation with chatbots is done through text-based interactions. However, the domain may further expand intelligent personal assistants like Apple's Siri, Amazon Alexa etc., where the mode of conversation is usually verbal.

Designing Chatbots

As mentioned in (9), designing chatbots is more of an interpretational task rather than an explanatory one like in the case of GUIs. A chatbot is supposed to serve the user whatever he/she wishes to know/do, in contrast to developing the interface and explaining the user how to use it, as in the case of GUIs. It is very sad to realize that there is not much resource available today to learn Conversational Designs. Even though we have seen a significant rise in the number of chatbots being developed over the years, research on chatbot designing in particular has had very less visitors. Other than some of the tech giants publishing their guides to design Chatbots using their platforms like Google (https://developers.google.com/actions/design/), Apple (https://developer.apple.com/design/human-interface-guidelines/technologies/siri/introduction/), IBM (http://conversational-ux.mybluemix.net/design/conversational-ux/), Amazon (https://developer.amazon.com/designing-for-voice/) etc. there are very limited resources that establish a standard way of designing and developing such interactions. It is because of this reason that chatbots users have shown displeasure more often than not.

Classification of Chatbots

In the absence of a standard way of classifying chatbots, people through their experiments have tried to classify chatbots with respect to different perspectives. The most basic way of classifying chatbots is based on the domain they have been built for. The most prevalent ones include consumer goods, health, finance, media, food and beverage, travel, social, general utilities etc. Along with this, (1) classifies chatbots based on 2 major dimensions. The first one being the 'Locus of Control' and the second one being the 'Duration of Relation'.

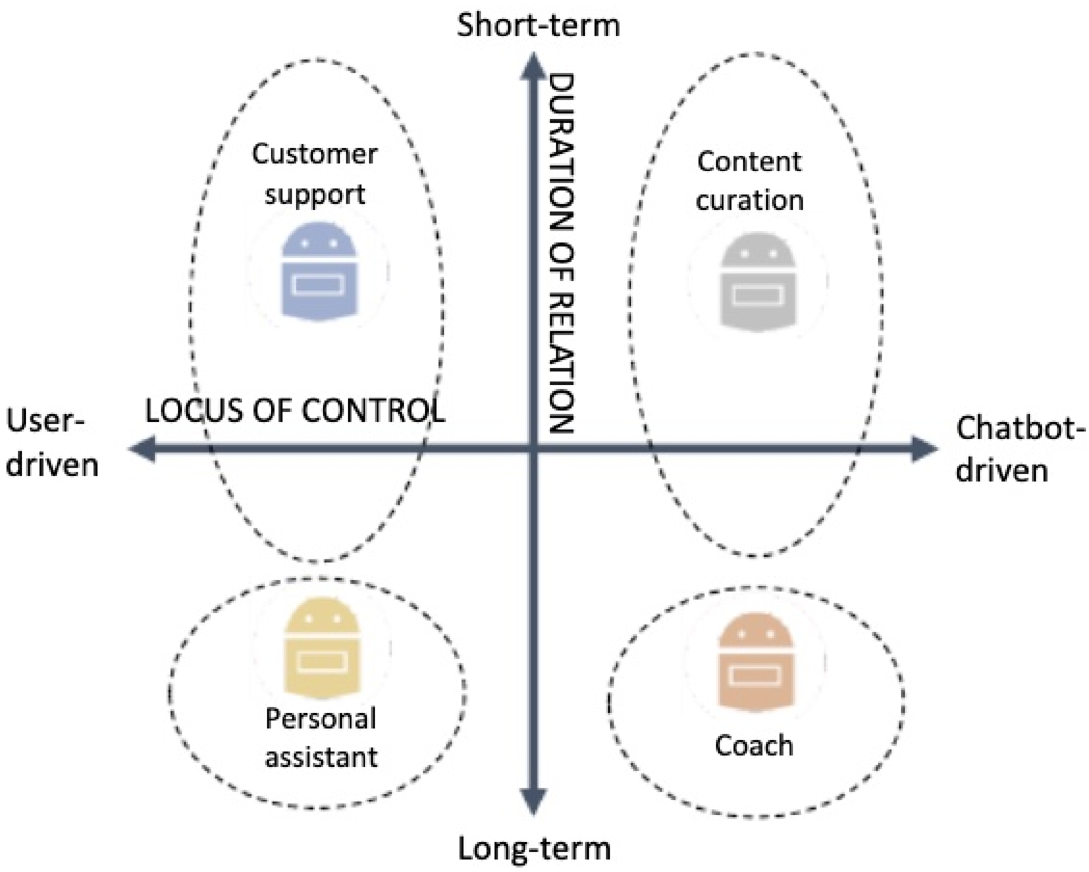


Figure 2.1: Classification of Chatbots (1)

It says, unlike human-human conversation where both the participants generally contribute equally to the conversation, interaction with chatbots generally involves one party taking the control and thus deciding the flow of the conversation. Based on this fact, dialogs with chatbots can be classified into 'Chatbot-Driven Dialog' and 'User-Driven Dialog'. The chatbot-driven dialogs are generally presented by scripted chatbots where there are limited branches and thus the ways of interactions and in turn, ways in which the conversation can flow are pre-defined. On the contrary, when the user is given a higher flexibility and the conversation is kept open ended, the dialog can be termed as a user-driven dialog. This is traditionally more difficult to achieve than that of the chatbot-driven one mostly because of the fact that the variety of user input becomes very high and thus correctly identifying user's intent becomes challenging. Based on the second dimension on classification, a conversational dialog can be classified as 'Short-Term Relation' and 'Long-Term Relation'. In the case of a short-term relation, the aim is not to preserve the minutes of the interaction and is generally intended to be a sole task-based one. Whereas, for long-term relations, the chatbots try to save context of conversations along with other user information which can then be used to personalize the interaction.

Chatbots' History and Progression

Ever since the emergence of ELIZA – a text messaging based conversational agent (6) in 1966, the idea of having a Conversational Agent which can mimic a human and talk to other humans was already set. However, improvements in the Graphical User Interfaces (GUI) space made itself a comfortable and hence the chosen way of interacting with computers. This made both users and researchers stick to the GUI space and thus there was not much development seen in the CUI space. Not until the years of 2015-2016 and thereabouts did Conversational User Interfaces (CUI) receive significant attention. With the emergence of conversational assistants (like Siri, Alexa, etc.) and them being embedded into personal smart devices, the need to interact with Computers which were more 'human-like' was felt. Since then, the research and development in CUIs has been tremendous. To add a figure to it, in the year 2018, there were 300,000 chatbots developed by CUI developers using Microsoft and Facebook's platforms and the older Pandorabot platform hosted over 200,000 chatbots (7). The idea of having chatbots handle task-based conversations like customer queries, managing orders etc. did look like a feasible way of letting computers do the redundant work and let the intelligent workforce engage in doing other more important tasks. From an end-user perspective, this new way of asking queries or placing orders was fancy and at times much more interesting than having to talk to an Interactive Voice Response (IVR) system. As mentioned in (7), below are some of the Gartner predictions:

- "By 2020, 80% of new enterprise applications will use Chatbots."
- "By 2021, most enterprises will treat Chatbots as the most important platform paradigm; and 'Chatbots First', will replace the meme 'Cloud First, Mobile First'."
- "By 2021, more than 50% of enterprises will spend more per annum on bots and chatbot creation than traditional mobile app development. Individual apps are out. Bots are in. In the 'post-app era', chatbots will become the face of AI...."
- "By 2022, 85% of customer service interactions will be powered by chatbots."

However, after the boom in chatbots development, there was a countertrend that was seen. In

2017, Facebook reported that 70% of the chatbots developed in its platform were not able to fulfill the users' needs (13) and thus were not found useful. In 2017, a survey was conducted (14) which asked 54 people to interact with a chatbot for a week. The chatbots used in the survey included Apple's Siri amongst other task-based chatbots developed to accomplish various needs like ordering food etc. This study suggested that the users were not satisfied with the chatbots' performance and thus their expectations were not matched. The downsides reported by them included undesirable notifications, decreased/similar efficiency, slow replies among others. Many such studies during that phase contributed to the idea that there was something wrong in the way the chatbots were being designed and it did clearly indicate that there was still a lot of scope for research in this particular area. Later in 2018, Dave Feldman who was himself associated with Facebook chatbot platform published an article (8) which was titled 'Chatbots – What Happened?'. In this article he did an analysis of what exactly could be the possible causes of the displeasure that chatbots caused. In his article he quoted the statement of Digit's founder and CEO, Ethan Bloch - "I'm not even sure if we can say 'chatbots are dead,' because I don't even know if they were ever alive.". Following this, he concluded that some of the major reasons could be Complicated Platforms, Difficulties in conducting conversations. He clearly mentioned about the lack of design knowledge among the chatbot developers and rather insisted that tech giants who launched chatbot developing apps should also have hired mentors to educate developers on how to use their platform and how to design such an interface. Fast-forward few years to 2022 and we see that modern assistants are being designed to support not just task-based use cases but also a varied range of other conversations. Developments in the field of AI and NLP in particular have made these assistants better in the way they interpret input's semantics, give an impression of an artificial pleasant persona, infer the user's intent etc. However, this has also led to increased user expectations and thus adding complications to the already complicated process of designing such interfaces.

Existing Tools

Lately, the need for having good tools has been felt and there have been many proposals of such prototyping tools both in the academia as well as in the industry. However, there is very little validation for most of them. As also highlighted in (15), there are very few which have been adopted in the industry. Botmock (10) is one such product which helps in designing conversational flows. Founded in 2016, Botmock started its journey pretty much during the time when conversational interfaces were gaining popularity. With lack of resources for developers to rapidly build and test conversational designs, CUI developers were directly using heavy tools like IBM Watson or DialogFlow, as mentioned in (16). It has a drag-and-drop interface which lets its users prototype CUIs. It was recently acquired by Walmart, which intends to use it "conversational commerce". The next tool we can look at is RASA (17). It is an open-source framework which lets its users build AI powered text and voice-based chatbots. It supports Level-3 AI, which means the chatbots developed using RASA can understand context and process any change in them. All the features that it provides, makes it a very powerful tool where anyone can develop a chatbot. However, these tools have a lot of scenarios to satisfy, and this results in an overloaded list of features that comes along with them. This long list of features makes them bulky and thus it is not too easy for someone to adopt these tools. Albeit there are tutorials published (18) to support its users, learning the use of such tools and then adopting it does demand considerable amount of time to be invested, which, for the specific scenario that we are trying to address, may make the researchers' lives a bit difficult. This is where a tool like the one we are proposing comes handy. With availability of features which are just enough, but not more, researchers will be able to quickly adopt the tool and right away start prototyping designs that they feel are worth studying.

3 Requirements Analysis

Like any other project, requirements gathering was vital for this project as well. Having gone through the existing research works and then drafting all the challenges that researchers generally face while studying design characteristics, I realized that it was very important to take note of all the basic features that must be implemented in the tool. During my literature review, I tried to search for research papers where surveys were carried out on certain groups of users. There were multiple such studies where users were asked to interact with different chatbot designs having distinct design characteristics with respect to chatbot's personality, domain, duration of engagement, etc. Going through the outcomes of these surveys did help me understand the general user reviews, perception, and expectations from such chatbots. These did help me derive the requirements for the tool and thus in turn would help provide a better user experience to the users of the tool. Taking notes from these surveys, I found out some of the most common user expectations from chatbots in general and tried to incorporate those into the tool. Some of the notable ones are as follows:

- Adding a human-like touch to chatbots will help sustain human-agent interaction in the long run (3, 12), as it will give the end-users an impression to some extent that they are talking to a human and thus in turn will help create a sustained relation of trust.
- As per the studies, it was found out that user engagement with agents has generally been for limited task-based conversation (12, 19). This might as well primarily be the case because of the high failure rate of chatbots in conducting an open-ended conversation.
- As fascinating and tempting as it may appear but personalization of chatbots can be a clear displeasure if not done properly; a default nature of chatbots is thus a good and safe starting point (3, 12).
- The papers did list down some of the significant domains where chatbots are currently being widely used instead of dedicated workforce. Most prominent of them are News, Travel, Shopping, Social, Game, Utility, Chit-chat, Entertainment (3).
- While there are multiple ways a user response can be fed to the system during an interaction like pictures, dropdowns, radio buttons etc., reportedly Buttons and Text are the more common ways of interacting with Bots (3, 5).

- The efficiency of a chatbot does depend on the domain it is being designed for to a greater extent. Although the design process does need the domain to be considered and well thought through, there are certain domains, which are less suited for chatbots (e.g.: shopping. where the options that the user can choose from are many) as compared to others which require specific answers (3). Some survey participants chose GUI as the preferred way of interacting with such domains.

- Purpose of chatbots should better be clearly defined in the description (3). Not every chatbot user is well aware of all the functionalities that the chatbots might support. During the survey conducted in (3), the participants didn't have any prior interaction experience with chatbots and thus it helped conclude that in such case, the users tend to expect much more from a chatbot and ultimately getting dissatisfied as it fails to deliver. Hence, having the expectations set by divulging the capabilities at the very beginning of the conversation is ideal for the user as well as the chatbot's overall performance.

- Some intelligence in the bot's response to user's text input is expected (3). During some surveys participants reported that they would be more interested in having a conversation with the chatbot if there was some sort of intelligence in the way the chatbot interprets user input and delivers a response to that.

- Having the failure cases handled makes the conversation easy (3). Some users reported that, as it is normal for a chatbot to fail in some scenarios, it is better to either admit that it failed to understand or process the user input or probably make up for it by replying with a witty response like the one below as quoted in (3):

  User's Question: "among the US 2016 presidential candidate, who is more popular?"

  Bot's Response: "The one who has the greatest number of fans and friends."

- Having conversational context retained gives a pleasing experience to the end-user (3). Sometimes for the user to engage in conversation to build long term relation, it is vital that the chatbot should retain some of the conversational context. Like the one below (3):

  Context – The user told the bot that X was his friend and Y was his friend X's wife.

  User's Question : "Who is Y ?"

  Bot's Response : "X's Wife!"

3.1 Deriving Requirements from Samples

As mentioned earlier, one of the major user personae which could potentially benefit from such a tool is a HCI researcher, who needs to create different chatbot designs and iteratively, keeps improving the designs by getting user feedbacks while establishing design principles. Let's look at a hypothetical situation, where a researcher named 'X' is working on a study where he is trying to see how different users are reacting to certain design of a chatbot in the Travel domain. For the tool to be of use for him, he must be able to prototype a sample conversation. So, if we do a bit of reverse-engineering here and have a sample conversation derived in the travel space, we will be able to get the requirements from that. We can in-detail go through the conversation to see how the responses are being used, what could be their potential types, if there is a pattern in the way the conversation panes out, what are the ways in which such a conversation may start, how would a bot want to end the conversation, what would a traveler using the travel enquiry chatbot expect. Having answers to these questions and many more will definitely get us closer to getting the requirements drafted. If we are able to do this drill for multiple domains, we might just establish a commonality between that, which in turn will help us derive a general set of requirements. In the anticipation of the above process to yield results and help us get the requirements gathered, we fetched some sample conversations from the Internet, some from the papers, and for some domains, we even tried to draft a sample conversation ourselves. This way, it was easy to understand the key features that any conversation could potentially require. The way the conversation would flow, the way the responses were drafted and the way the repair paths were executed would help me understand the runtime scenarios of any conversation better and thus in turn would help me design them better. Let's go through some of the sample conversations:

Speaker	**Response**
Diarybot	*Hi, I'm Diarybot. Your writing session will begin. Shall we continue?* [buttons to select: 'Later' or 'Continue']
User	[*button selected*] Continue
Diarybot	*Good to see you! I'm ready to begin. I'll keep all your secrets, so no worries. :)* *You can write about your deepest feelings and thoughts on the most traumatic experience of your life so far.* *You might relate it to your past, present and future, or tie it with your relationships with family and social circle. For anything, please write more than 100 words!*
User	When I was little, I was terrified of water. Mum always tried to take me to the pool to teach me how to swim, but I utterly hated it. I tried to say no many times, but my mum took it as a joke and lured me with toys or told me to toughen up. Growing up, I never told mum how I felt back then. I guess mum must have forgotten all about it. I hardly recall the specifics of the times, either. But whenever the memory surfaces, a feeling of distant fear makes me shiver. I don't know if it is the water, or the fear itself, or my mum.
Diarybot	*Thank you for writing today. Your session is finished. See you again soon!*

Figure 3.1: Example conversation for the Basic Chatbot (2)

Figure 3.1 is an example conversation present in (2). Here, the publisher has created a chatbot name 'Diarybot' on one of the most popular messenger apps in Korea called KakaoTalk. The idea was to have a chatbot help users in expressive writing in mental health. They implemented 2 different versions of the chatbot. One would have a basic chat implemented whereas, the other will have a responsive way of chatting with the users. It would follow-up on the user's response with open-ended questions. A sample conversation of the responsive chatbot is depicted in figure 3.2.

Speaker	Response
Diarybot	*Hi, I'm Diarybot. Your writing session will begin. Shall we continue?* [buttons to select: 'Later' or 'Continue']
User	[*button selected*] Continue
Diarybot	*Good to see you! I'm ready to begin. I'll keep all your secrets, so no worries. :)* *You can write about your deepest feelings and thoughts on the most traumatic experience of your life so far.* *You might relate it to your past, present and future, or tie it with your relationships with family and social circle. For anything, please write more than 100 words!*
User	When I was little, I was terrified of water. Mum always tried to take me to the pool to teach me how to swim, but I utterly hated it. I tried to say no many times, but my mum took it as a joke and lured me with toys or told me to toughen up. Growing up, I never told mum how I felt back then. I guess mum must have forgotten all about it. I hardly recall the specifics of the times, either. But whenever the memory surfaces, a feeling of distant fear makes me shiver. I don't know if it is the water, or the fear itself, or my mum.
Diarybot	*In your writing today, feelings of "terrified," "hate," and "fear" were found. What impact have they had on your body and mind?*
User	I feel weak, as in I feel slightly under the weather? I feel insecure.
Diarybot	*Now let's think more about "mum" in your writing. What have you received from your relationship with mum, if anything?*
User	Well, she's done a lot of things for me. She has been a supportive, dedicated mother, though there were times that her philosophies didn't really click with me.
Diarybot	*What about the things you tried for mum? You can write the things you've done so far.*
User	I was a good child, for sure. I tried to fulfill all of her wishes for me all along.
Diarybot	*Let's move on. What difficulties or trouble have you caused in your relationship with mum? Take your time and think about it.*
User	Erm... I know she made many sacrifices for me. We were never that well-to do, for sure. There were times that I snapped and yelled at her especially in high school years.
Diarybot	*I hope you muse on these things a while. Before we wrap up our writing today, what would you like to say to yourself?*
User	I guess it may not be as bad as I thought it to be, but I think I will have to work on my feelings for a while first.
Diarybot	*Thank you for writing today. Your session is finished. See you again soon!*

Figure 3.2: Example conversation for the Responsive Chatbot (2)

The next one has been referenced from (3). Here, the presenters have tried to present how a sample interaction with a chatbot might look like when used in the context of a Hotel Booking, which is a popular domain where chatbots have been used.

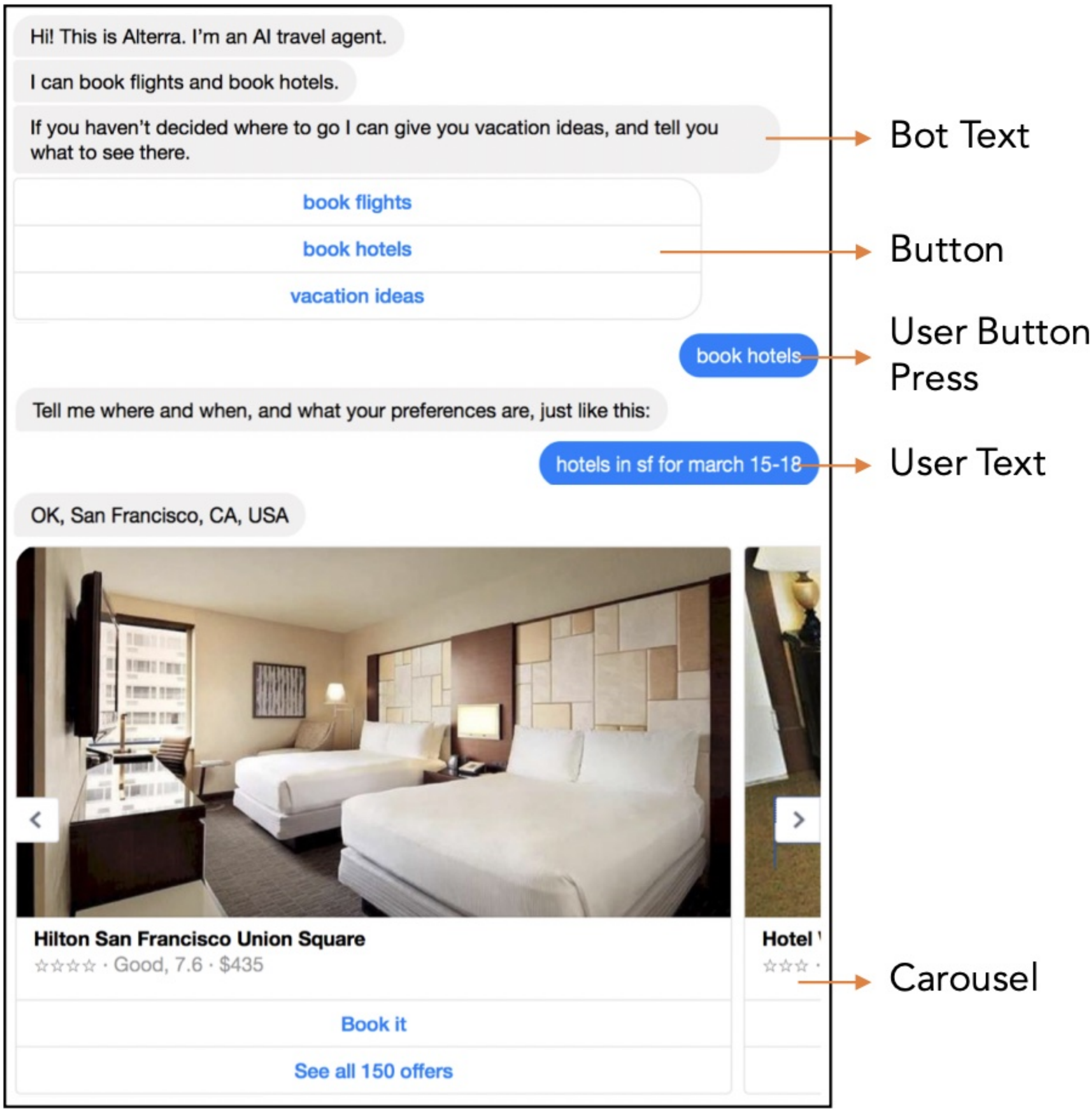


Figure 3.3: Sample conversation showing all the interactive elements of a chatbot (3)

In figure 3.3, we can see that there were multiple ways in which interaction with the chatbot was done like Text, Button and Carousel.

The next sample has been taken from (4), which is of a chatbot that has been designed to be an exercise advisor for adults. The study later concluded that the use of a chatbot to keep track of the physical activities the user was involved in and advise the users on their exercise routine did help the users and was liked and accepted in most cases. Figure 3.4 demonstrates a partial sample conversation of the agent with the user.

1. A: Hello, Bertha. Great to see you.
2. U: Hi Laura.
3. A: How are you feeling today?
4. U: I'm not feeling so great.
5. A: I am sorry to hear that.
6. A: Tell me more about how you are feeling.
7. U: I'm tired.
8. A: How tired are you feeling?
9. U: I'm pretty beat.
10. A: Good grief, I hope you're not getting sick. Can you get some rest soon?
11. U: I think so.
12. A: That is great. I hope you come back feeling refreshed.
13. U: Me too!
14. A: So, are you from the East Coast originally?

•••

24. A: So, is there anything I can do to help make this a more productive working relationship?
25. U: Nope, everything is OK.
26. A: That's wonderful.
27. U: Yeah.
28. A: So, let us talk about your pedometer readings.

•••

40. A: Looks like mission accomplished on the exercise.
41. A: We're doing some great work together.

•••

Figure 3.4: Sample conversation of the agent with a user from (4)

From figure 3.4, we can see that most of the user responses were of the form of text input, and it would be difficult to fit these scenarios using buttons or pictures. Also in this conversation, we can see that the chatbot saved the conversation context and the same was used throughout the course of the conversation.

Next, we will look into a scenario where the researchers tried to check the impact of humor in persuasive messages. They believed that adding humor to certain conversations would increase the effectiveness of motivational counselling done by virtual agents. A sample script from the same study (20) has been presented in figure 3.5

DS1 **Agent:** So, how is the weather out there today?
User: It's fine
Agent: I see, well that's good.

DS2 **Agent:** You know what they say about the weather in Boston? If you don't like it, just wait five minutes.
User: Haha

DS3 **Agent:** Alright, about how many half cup servings do you eat on a typical day?
User: 2
Agent: Okay, I am glad to hear that you are eating some fruits and vegetables. I would like to help change your diet, so that you can reach the weekly recommendation of five servings per day, five days per week.
User: Okay

DS4 **Agent:** One time, my friend suggested I go on a diet where I would only eat almonds. But I thought, that's just nuts.
User: Haha

Agent: All kidding aside, on a scale of 1 to 10, where 1 means a healthy diet is not at all important, and 10 means health eating is very important, how important is healthy eating to you?
User: 4
Agent: Alright, that's good. You could have chosen a lower number, but you didn't. You chose a 4 and not a lower number. Why is it important for you to eat fruits and vegetables?
User: To be healthy and live longer
Agent: Alright, so I hear that you would eat more fruit and vegetables to be healthy and live longer. That's a really good reason.

DS5 **Agent:** You know, there are a lot of little things people can do to get more fruits and vegetables into their life. For example, making sure you always walk through the produce section when you go grocery shopping.
User: I might give that a try

DS6 **Agent:** Speaking of diets, I know people who are on the seafood diet. Every time they see food, they eat it.
User: Haha

DS7 **Agent:** I think that's all the time we have for now. I appreciate you sharing your thoughts with me today. Have a great day.
User: Goodbye

Figure 3.5: A sample script from a humorous virtual agent's interaction with a user for motivational counselling [8]

The next one is taken from (5) where a chatbot called 'Vincent' was implemented. The study aimed at analyzing user behavior towards a care receiving chatbot as well as a care-giving chatbot and thus get an idea of how users themselves wanted to be treated based on the theory of 'Reverse-Psychology'. Illustrated in figure 3.6 is a sample conversation of the care-receiving Vincent with a user.

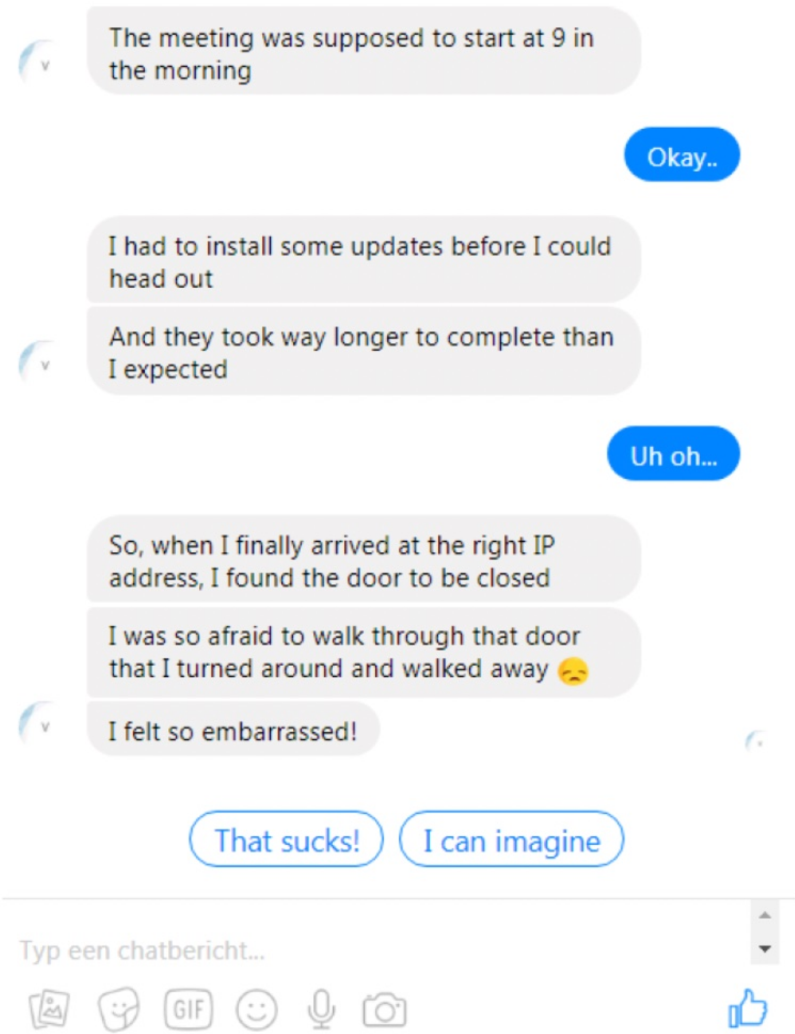


Figure 3.6: Sample conversation of care-receiving chatbot trying to express a situation where it faced embarrassment to the user (5)

Finally, in figure 3.7, I would also like to illustrate another sample conversation which I tried to derive myself in the context of flight booking, which again is a popular domain which has seen chatbots being deployed. Here, we have tried to illustrate how a conversation can flow into 2 different directions and thus have 2 completely different user intent.

In the figure 3.7, we can see that the customer who had booked a flight ticket wants to get a refund of his ticket as the flight ticket was cancelled for some reason. We see there are

Figure 3.7: Sample conversation with a chatbot in a Flight Booking context

multiple response types like Buttons and Text. Also, we can see that there were repair paths in case the validation rules fail. When the Booking ID entered by the user didn't validate to an authentic booking id, the chatbot responded with a message and routed back to the same question again using a repair path.

In figure 3.8, we show a flow of the same conversation where the user wants to re-schedule the booked flight. Here the user intent is completely different from the 1st one. Here, we see date entries as an additional user input type. Also, the conversation shows use of multiple follow-up questions to be able to accomplish the task asked by the user.

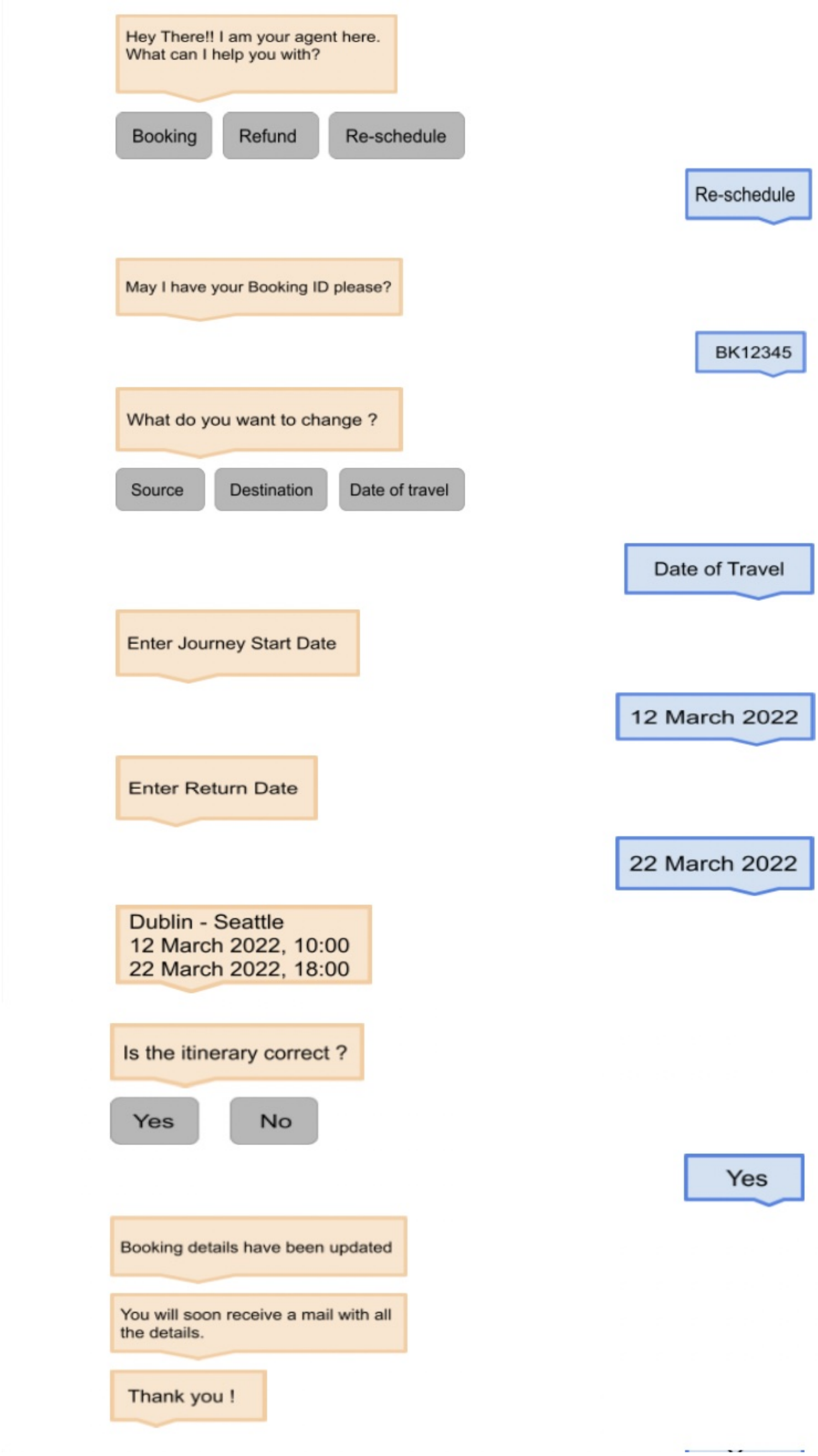


Figure 3.8: Sample conversation with a chatbot in a Flight Booking context

Having gone through all the above-mentioned literature and the sample dialogues and multiple iterative feasibility analysis, I could finalize the below mentioned functional requirements for the tool. Since the tool has 2 major phases of chatbot creation, 'Design Phase' and 'Interaction Phase', I have defined requirements for each of the 2 phases separately.

3.2 NLP Requirements

In some of the research work that I went through, I found out that most of the survey participants wanted to have text inputs so that they have added flexibility in the way they interact with the chatbot and also, the chatbots to have some sort of intelligence rather than just being a scripted bot when it comes to processing the text inputs. This would give a better user experience and to some extent, it will also provide an ease of access to the users. After a lot of brainstorming around the feasibility, adaptability, and integration of many Natural Language Processing capabilities, I could come up with 2 different approaches which would add intelligence to the chatbots.

1. **Semantic Matching**: When scripted bots are made, in the runtime paths are chosen based on user input's keyword match or the whole text match. The engine will generally look for some keywords in the user's input and if found, it will execute the path accordingly. If no match is found, it will execute the repair path. When implementation is done this way, the chances of keywords match is much less as the end-user is oblivious about the configured keywords and since natural language has multiples ways of conveying the same intent, it adds to the difficulties. Hence, the repair path will get executed more often or in some cases, a bit too many times for the end user's liking. To tackle this scenario, I found Semantic Similarity could be used. Semantic similarity doesn't just look for a keyword match, it compares 2 sentences to check if they have similar meaning rather than just looking for same keywords. The tool can provide flexibility to the user to decide whether or not he/she wants to use semantic similarity and thus it can be configured during the design time. If configured to 'Enable', in run-time, user inputs to those nodes would be passed to the back-end server to check if the semantics of the input matches any other sentences configured during the design time and accordingly take the further path. The sentences can be converted to Embeddings using Sentence Transformers. A word embedding is basically a vector representation of any word. Thus, the input text's vector can be matched to the configured texts and based on their cosine similarities, the path ahead can be decided.

   The illustration in figure 3.9 will provide more clarity on how this technique can be used in integration to the tool:

   In figure 3.9, we can see after the first chatbot response 'How can I help you?', the

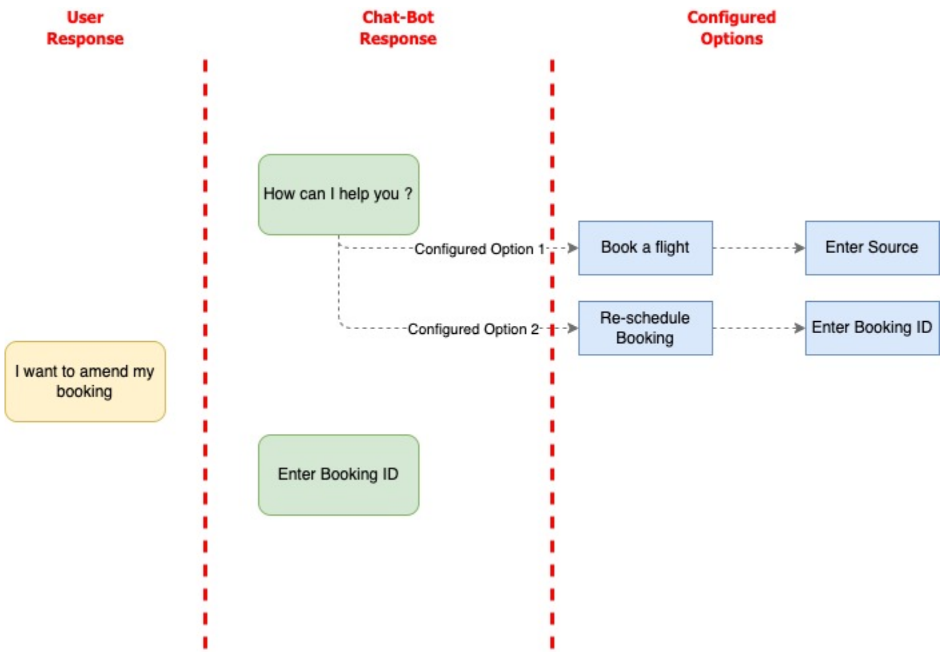


Figure 3.9: Illustration of Semantic Matching

designer configured 2 different options 'Book a Flight' and 'Reschedule Booking' and their respective paths thereafter. In the run-time, when user enters 'I want to amend my booking' which wouldn't directly match either of the configured options, however still path-2 would be executed as the user semantically matches the 2nd option 'Reschedule Booking'. In this case, even though the user was unaware of the exact configured text, he was still able to go ahead in the happy path of the conversation. Another complexity in this case is whether either of the configured options is a good match of the user's response. If neither of it is a good match, the tool needs to execute a repair path and ask the same question again, anticipating a more informative response. To implement this, the tool can have a configurable THRESHOLD value. This can be set by the designer while designing the chatbots. If the cosine similarity of any of the configured options matches at least by the THRESHOLD value, then only it will be considered good enough a match of the user input. Anything less than that would trigger the repair path.

2. **Entity Extraction**: In quite a few surveys, we did see that having the context preserved did give better user experience more often than not. This adds mor human-like feels to the whole conversation. With an intention to do that, I researched on using entity extraction as a part of the conversation. Extracting key entities from the user's response and being able to use them in the bot's question/response that follows will give an impression of a short-term context re-use. For e.g.: in the context of a food ordering chatbot, if the user says, 'I want to order a pizza for lunch', the chatbot can extract the key entity 'pizza' in this case and re-use that in the follow-up question as 'Which

pizza would you like to have?'.

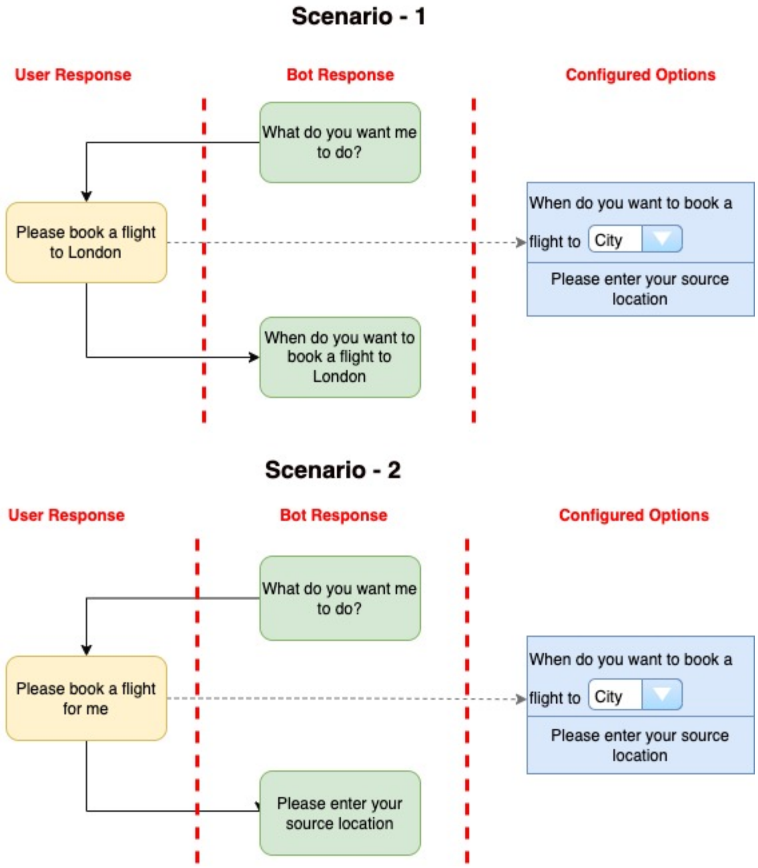


Figure 3.10: Illustration of Entity Recognition

For the integration of Entity Extraction with the tool, we can see in figure 3.10 that after the first Bot Response 'What do you want me to do?', the user configured an option with entity extraction along with its fallback text. In this case, the user input is will be sent to the backend server to extract all the entities based on their weights using the TFIDF vectorizer. And match all the extracted words ('Book', 'Flight' and 'London' in this scenario-1 and just 'Book', 'Flight' in scenario-2) to the type of entity set during the configuration ('City' in this case). When there is a match found, the same entity gets used in the follow-up question (like in scenario - 1). If no such match is found, the fallback text gets printed (like in scenario - 2) and accordingly the path follows.

3.3 Drafting Requirements

All the samples and example conversation flows discussed above did give me a lot of clarity with respect to the expectation from the tool. The insights learnt from the literature review, the sample conversations illustrated and the self-drafted conversations mentioned above helped me finally draft the requirements for the tool.

3.3.1 Design Phase Requirements

- Enable the user to create a conversation ladder having multiple paths in the design time.
- The Design time should be an easy drag and drop interface where user can add different types of User/Bot response nodes.
- The Response nodes can then be connected to each other by dragging from the source node and dropping to the destination.
- The link creation must have validations in place so that there are no erroneous links created.
- User must be able to configure the NLP functionality for the user nodes during the design time.
- User must be able to save the design created as a JSON, which later gets translated into a chatbot.
- The default user response types that the tool must support are Text, Buttons and Yes/No type.

3.3.2 Interaction Phase Requirements

- During the runtime, the saved JSON is parsed, and the conversation tree is created.
- Chat window is then rendered with the chatbot's, and the user's responses aligned on either side of the window.
- Small delays are added to the Chatbot's responses to add human-like touch to the conversation.
- Each of the user response type has its own visualization based on the type of response.
- Text type responses must be checked for semantic similarity if the same is configured during the design time.
- Semantic similarity failure scenarios to be handled and the appropriate repair path must be executed.

Having drafted the basic requirements, I tried to create a conversational flow which would mimic my design-time tree. The same has been illustrated in figure 3.11

In figure 3.11, we can see that the conversational tree above has all the requirements incorporated as mentioned earlier in the 'Design Phase Requirements' sub-section of this section. We can see implementation of different response types like Buttons, Yes/No, Text. We can

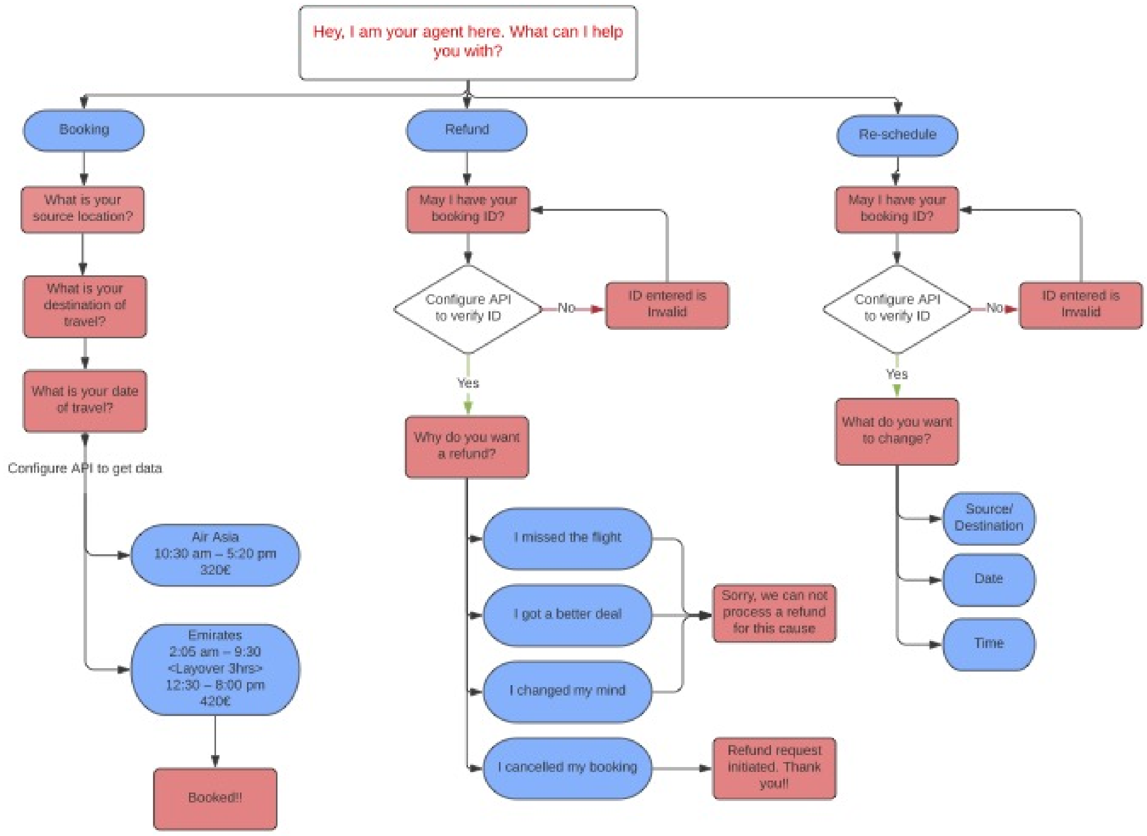


Figure 3.11: Sample conversation flow in the Design Time

see intelligence in the form of semantic similarity. We can also see exception handling being illustrated in the form of repair path.

4 Design Process

The design process of the tool can mainly be divided into 2 parts. One is the Interface design and the other one being Engineering design. In this section, I will try to explain the design choices made throughout the process and the reasons for the choices made.

4.1 Interface Design Process

The aim of the interface design process is to come up with an interactive design that can incorporate the basic requirements listed above. The final design will also try to have scope of further extension with minimum design changes. First of all, I started the design process by drafting a rough sketch which had all the basic features in it. The sketch looked as shown in figure 4.1

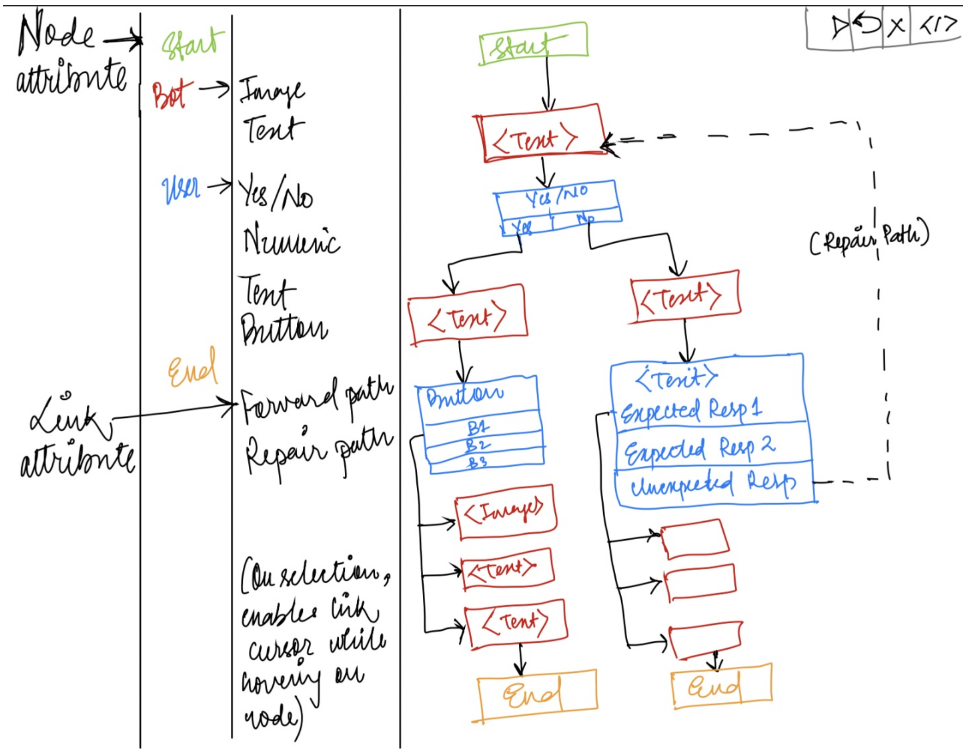


Figure 4.1: Rough sketch of Design-time Interface

The leftmost pane in the sketch is a navigation menu which would basically have 2 different menu items. One would be the ‘Node Attributes’ this (when expanded) would basically have all the different types of nodes that the tool would support. The other one would be the ‘Link Attributes’ item. This would be used to create links between any two nodes. I planned to have a ‘Start Node’ which would indicate the start of any conversational flow and an ‘End Node’ would denote the end of every branch in the conversation tree. For the User Nodes, the idea was to have Yes/No, Text, Numeric and Button type response nodes. In the conversation tree, we can see that the user text response node can have some pre-defined texts and a path for each of them respectively. The same is the case with a button type node. There would be different button labels configured in a button type node and each of them would have a path attached to it. In the design, for the Link attribute menu item, we can see that there are 2 types of link attributes that any use could select. The first one be a forward path or as may also be called ‘the happy path’ and the other one being the Repair path which would get executed in case of any error or mismatch at that particular node. Figure 4.2 shows the integration of the design-time interface with the run-time interface:

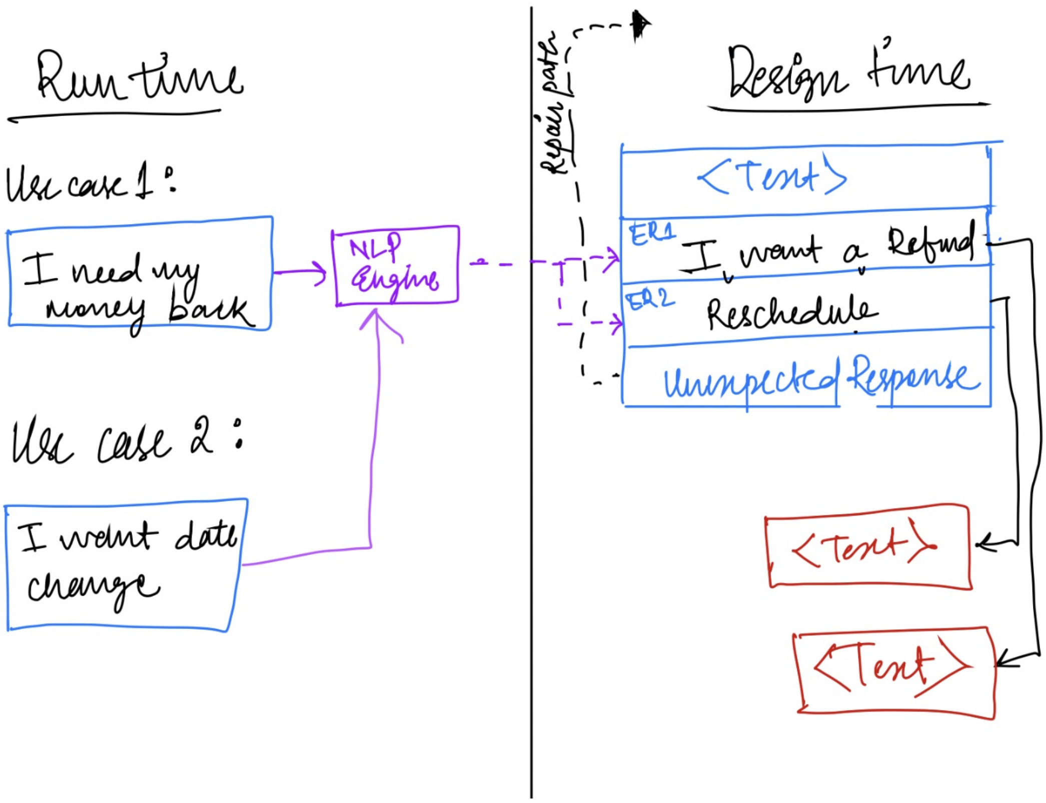


Figure 4.2: Design-time and Run-time Interface Integration

Figure 4.2 depicts how the design time tree translates to the runtime flow. It shows a part of the tree where a text type user node has been shown. The node has 2 different options configured (1. I want a Refund. 2. Re-schedule) along with another default faulty/unexpected response. Each of the expected responses configured have a path attached to it and the faulty response has a repair path attached to it. At this stage in the runtime, if the user enters ‘I need my money back' or 'I want a date change', the user input will be sent to the NLP engine to be converted into vectors and the cosine similarity of each of them would be calculated with the pre-defined texts (In this case, 'I want a Refund' and 're-schedule'). If any of the comparison has similarity more than the configured Threshold, its path gets executed otherwise the repair path gets executed. Taking notes from this stage, I then drafted a more presentable and a closer representation of what the actual User Interface (UI) may look like using a Figma design. [Note: Figma (21) is a powerful graphics design tool which lets users create UI designs of Web-apps, Native Apps, Logos etc. It has a plethora of plug-ins available which lets users create beautiful designs and have the UI design draft ready much before the implementation starts.] Presented below are the Figma designs of what the UI may look like:

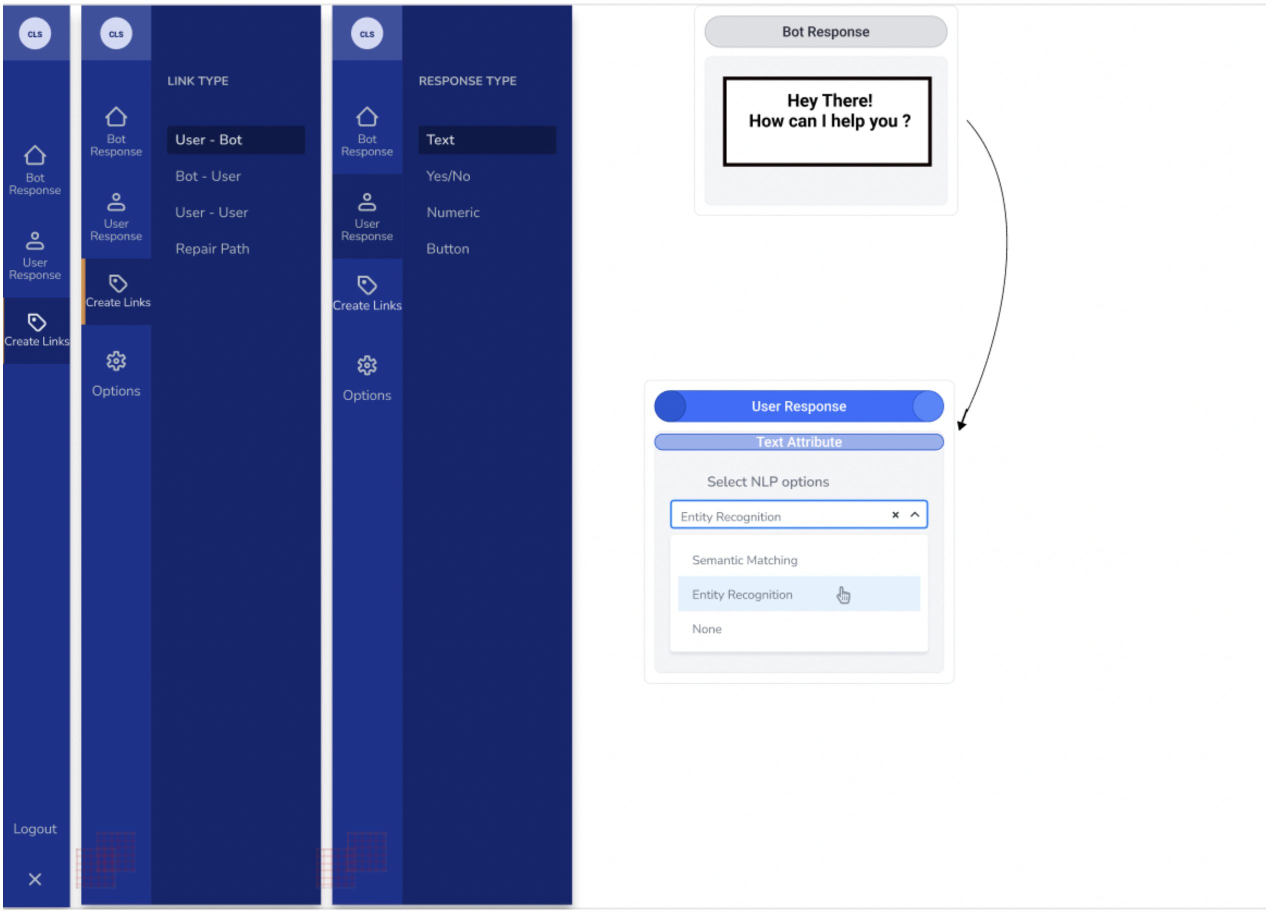


Figure 4.3: Design-time Interface Prototype

In figure 4.3, we can see that the rough sketch has translated well into a more practical design. The collapsible navigation bar has got all the attributes illustrated along with a logout button (In case, user management is implemented). Figure 4.4 will demonstrate and explain how the NLP configuration integration happens with this design.

Figure 4.4 shows the Text type user node in detail. This is where the NLP configuration is supposed to be done. On the left-hand side of the diagram, we can see a basic user text response node. It has a drop down which says 'Select NLP Options'. This will have 'Entity

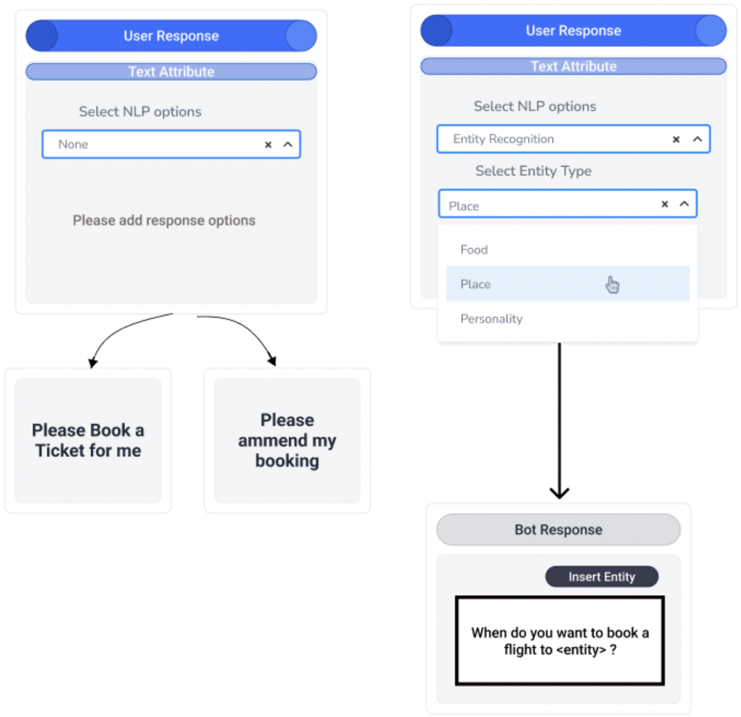


Figure 4.4: Design-time NLP Integration Prototype

Recognition', 'Semantic Matching' and 'None' as option values. By default, its value will be 'None'. In which case, the user can add multiple text options and their respective paths to the node and an exact match of these options will be checked in the run-time.

However, as shown to the right of the picture, if the user selects 'Entity Recognition' as the type, then dynamically a new dropdown will appear which will let the user select the 'Corpus Type', which basically will be the type of entity that particular node is concerned about. The next bot responses attached to such a node can use the extracted entity as a part of the response by using the 'Insert Entity' button.

After this stage, I did a bit of re-engineering to discover that there are better ways to implement the link connection. Rather than having that as an attribute, we can directly hover over the involved nodes and drag either of them to create a link. Figure 4.5 shows the screenshots of the actual tool taken from the design time and the run time.

In figure 4.5, we can clearly see that the design prototyped earlier has translated nicely into the live product. Along with all the features mentioned in the earlier prototypes, the actual design of the product has a new 'Save Flow' button, as can be seen in the navigation bar. The functionality of this button is to save the design-time conversation tree in the form of

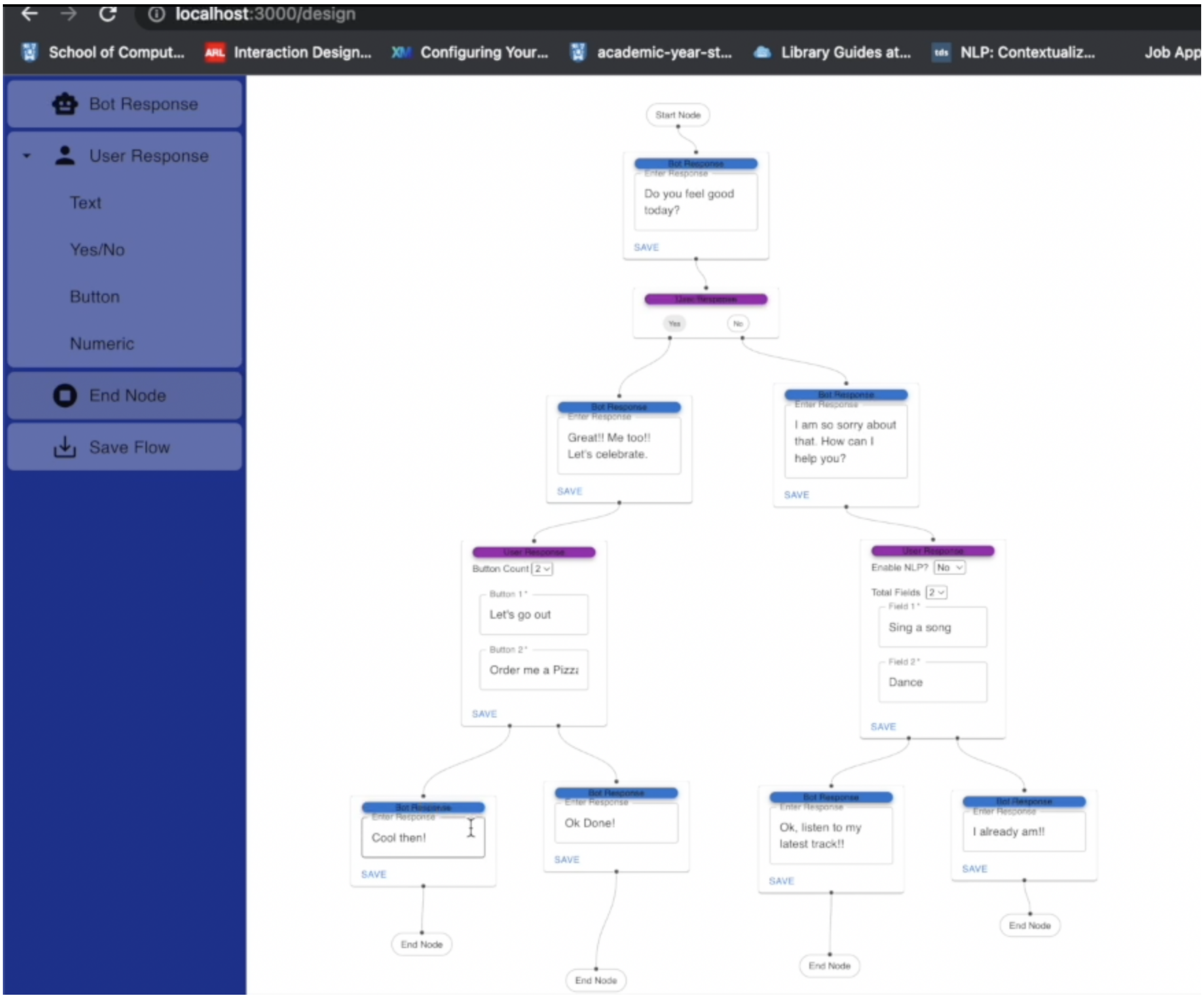


Figure 4.5: Design-time UI Mock-up of the Final Product

a JSON to the local system, so that the same can later be read and parsed to create the chatbot interaction. Let's have a look at each of the node types in detail.

4.1.1 Design-time Node Designs

Bot Response Node

With the Bot Response Node (Figure 4.6, there will always be a single textarea type input field, where the response of the Bot for that particular stage can be saved. Additionally, we can see to the bottom right that there is a prompt which says, 'Please Save'. This prompt appears every time there is an unsaved change in the input field of the node. The prompt indicates that the unsaved changes must be saved using the 'SAVE' button so that the updated text gets stored in the state.

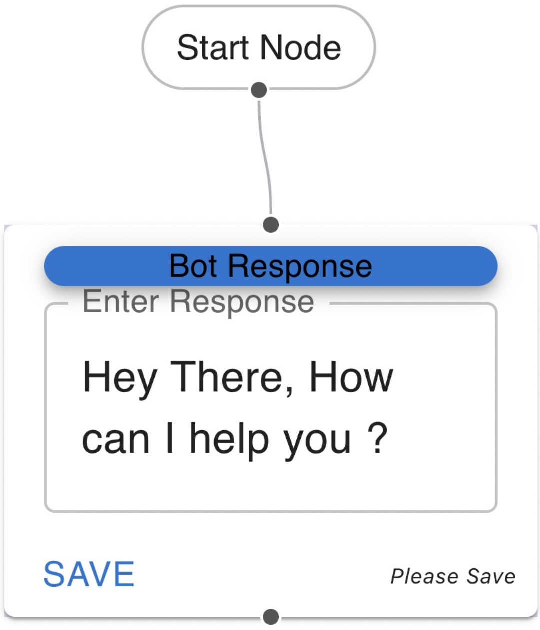


Figure 4.6: Bot Response Node

Yes/No Type User Response

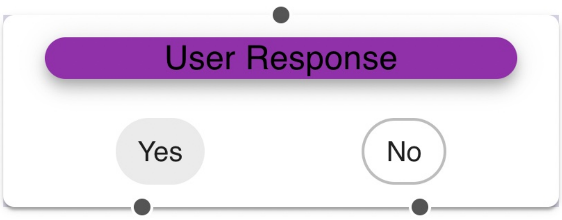
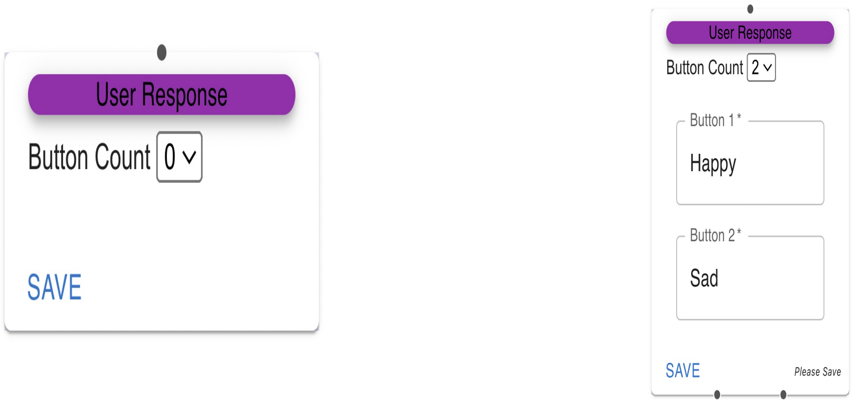


Figure 4.7: Yes/No Type User Response Node

A Yes/No type of user response (Figure 4.7) is one of the simplest nodes of this tool. It has 3 handles, 1 input handle and 2 output handles. The input handle will connect any Bot Response node to the Yes/No node. Whereas the output handles create paths for 'Yes' and 'No' options respectively.

Button Type User Response

For a Button Type User Response, the designer has the flexibility to select the number of

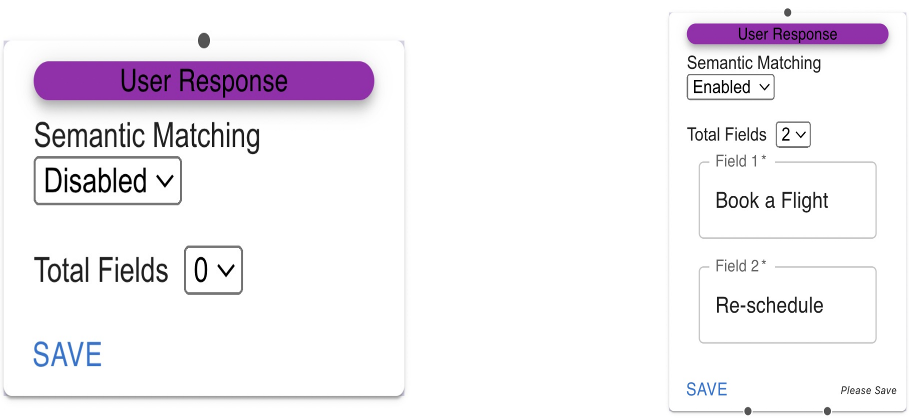


(a) Basic Button Node (b) Adding Button labels

Figure 4.8: Button Type User Response

buttons that particular node is going to have. The same can be seen in figure 4.8. To the left side (Figure 4.8a), we can see that in the initial stage, the Button Type user response just has the 'Button Count' dropdown along with one incoming handle and the 'SAVE' button. As soon as the designer selects the number of buttons to be configured, the textbox type user inputs and an equal number of output handles (one for each of the buttons) get dynamically added to the node (Figure 4.8b). These textboxes can be used to add labels of the configured buttons. Finally, similar to the Bot Response Type Nodes, we must save the text added to the text boxes using the 'SAVE' button.

Text Type User Response



(a) Basic Text Node (b) Configuring Text for different paths

Figure 4.9: Text Type User Response

The Text type user response node is quite similar to the Button type user response node with a very few changes. Here (Figure 4.9), we additionally have a dropdown labeled 'Semantic

Matching', whose value by default is 'Disabled' and can later be changed to 'Enabled' by the designer. If set to 'Enabled', user input to this particular node in the runtime will be sent to the backend server for semantic matching. If it is set as 'Disabled', the runtime engine will perform an exact text match rather. Other functionalities are similar to Button type nodes besides the 'Button Count' label as in the case of the button type node gets labelled as 'Total Fields' in the case of Text type user response node (Figure 4.9b).

4.1.2 Validation Rules for connecting Nodes

Along with the design decisions taken for the nodes, there were other aspects as well in the design time to maintain consistency and integrity. To make sure that the designers by mistake don't make designs that don't work, I had to place validation rules in place. This would basically put checks to negative scenarios. The validations with link creations are as follow:

1. The 'Start Node' must always be connected to a Bot Response Node. Whenever a designer tries to create a link between a start node and a User Response Node, the tool won't let this link be created. This is for the very basic rule that a chatbot interaction will always start with the Bot greeting the user and describing its capabilities and not the other way round.
2. A user node (be it of any type) can never be connected to another user node. This validation was put in place because of the fact that a user response cannot follow another user response without the chatbot intervening in-between.

This concludes the interface design process for the design-time phase of the tool. Now let's look at the run-time phase of the tool. This is the interaction part, where the user actually interacts with the chatbot created using the tool's designer. A sample conversation in the run-time looks like as shown in figure 4.10:

The responses are contained in rounded rectangle containers. The user responses are aligned to the right side of the window while the Bot's responses are aligned to the left side of the window. Different background colors make a clear visual differentiation between the types of the nodes. The color contrasts of the background and the text are maintained at 13.36:1 and 5.31:1 for the Bot's and the User's responses, making it WCAG level AA accessible w.r.t color contrast.

Let's see how each of the user response types translate to run-time designs.

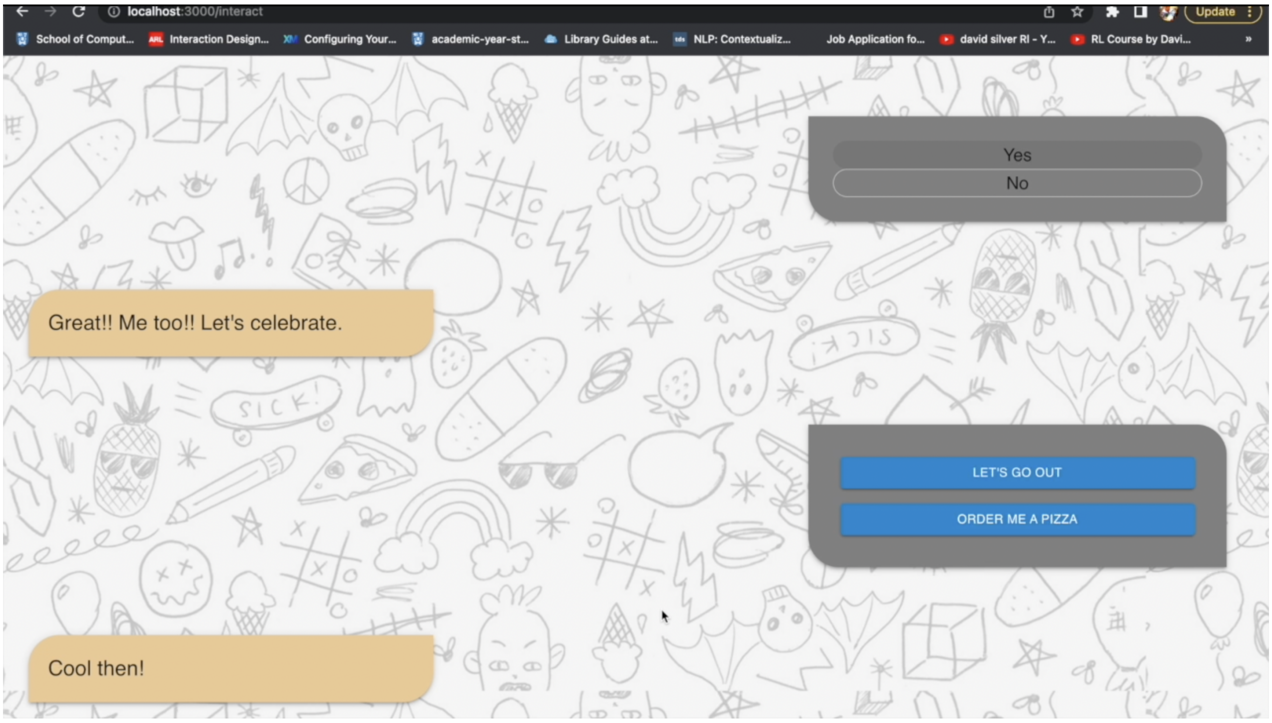


Figure 4.10: Final Interaction Interface of the Product

4.1.3 Interaction-time Node Designs

Button Type Response

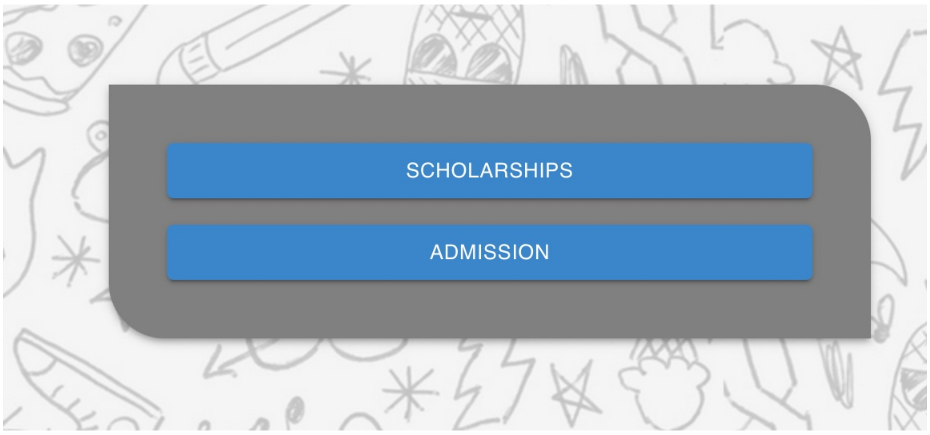


Figure 4.11: Button Type User Response

The Button type response will typically have the configured number of buttons with their labels as configured during the design time (Figure 4.11).

Text Type Response

The Text Type Response has a text area type user input field. This lets user input multiline responses and then finally send it using the 'SEND' button at the end (Figure 4.12.

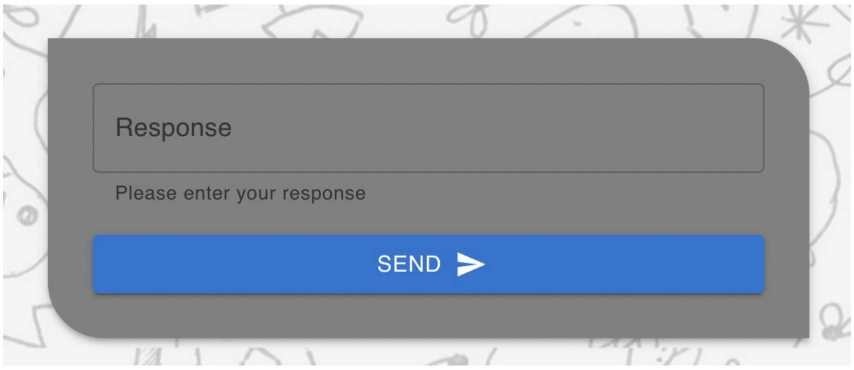


Figure 4.12: Text Type User Response

Yes/No Type Response

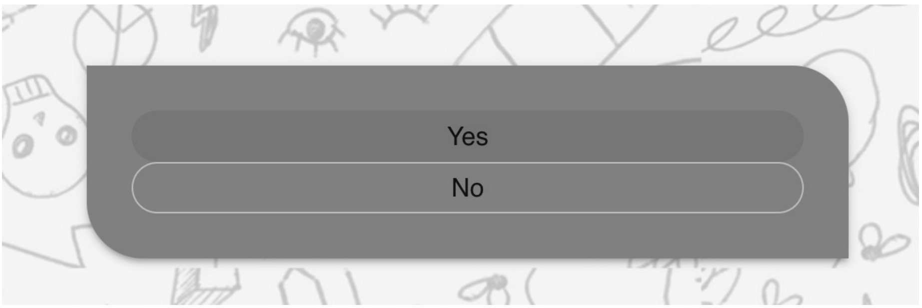


Figure 4.13: Yes/No Type User Response

The Yes/No Response Type basically has 2 chips one for Yes and the other for No. User can click on either of them to execute its mapped flow.

Along with these, there was a decision taken to delay the Bot's response by some time every time after the user's input. This gives the impression as if the Bot is thinking before replying and thus adds a humanlike touch to the conversation. During this delay, the chat window will display an ellipsis visual which typically denotes 'loading'. This is what it looks like in the runtime (Figure 4.14):

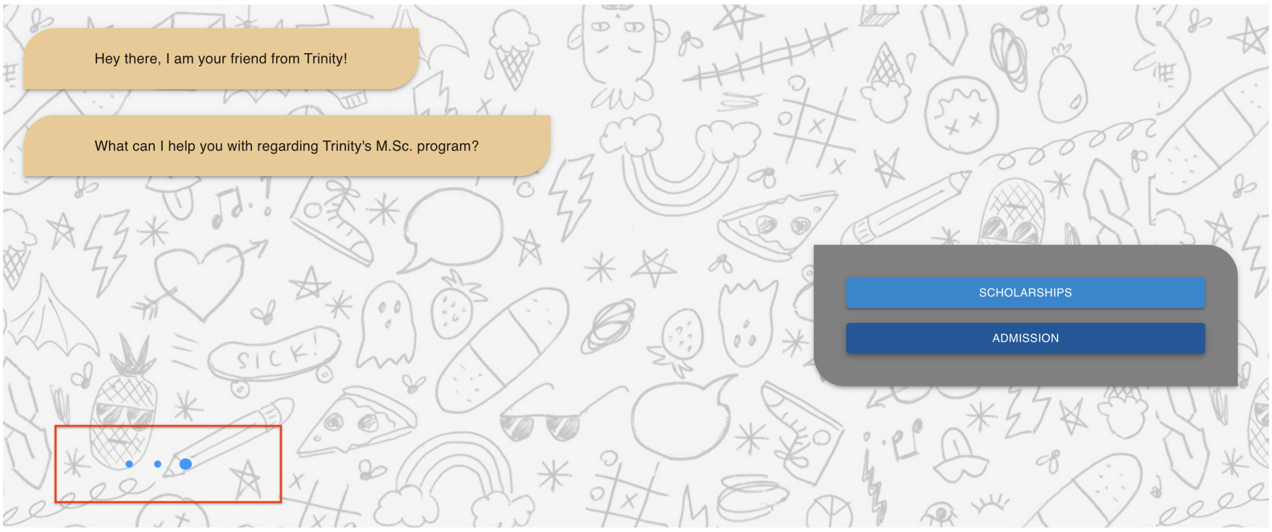


Figure 4.14: Bot's Response Loading

The next prominent Interface design choice made was to make the whole design responsive. This would make the whole chat window presentable even on smaller spaces as chatbots generally be. By making this design choice, I ensured that the chatbots so created can be effectively embedded on to any webpage and at any size. When embedded, the chat window looks like as shown in figure 4.15:

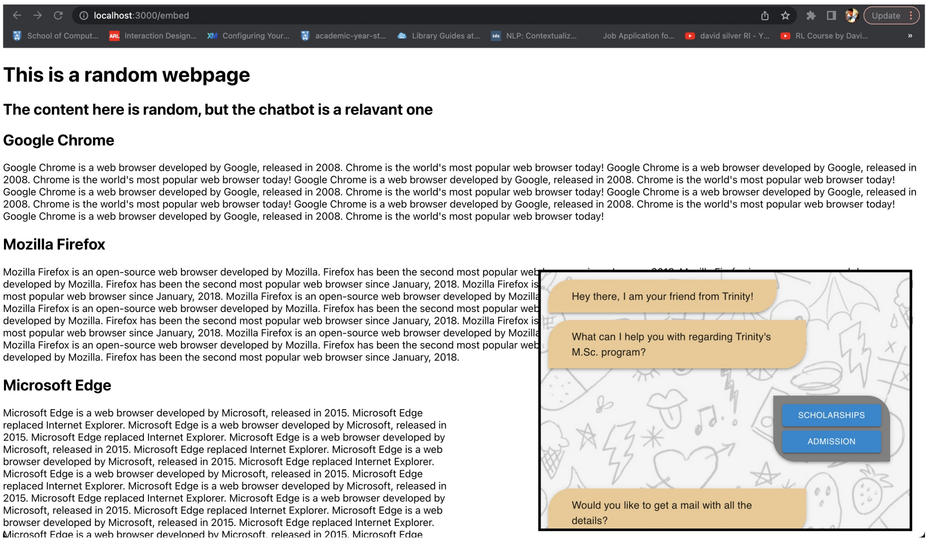


Figure 4.15: Chat Window getting embedded to a random Web Page

In the picture (Figure 4.15), we can see that the chatbot has been embedded into a random page using iframe. Since the chat window is responsive, the elements have scaled down accordingly.

The next prominent design choice that I had to make during the course of the design process is the configuration of the cosine similarity threshold. As described earlier in this document, for any user response to be semantically matched with any configured option, their cosine similarity will at least have to be equal to the threshold value configured by the user. This configuration as of now resides in the server-side code base as below (Figure 4.16):

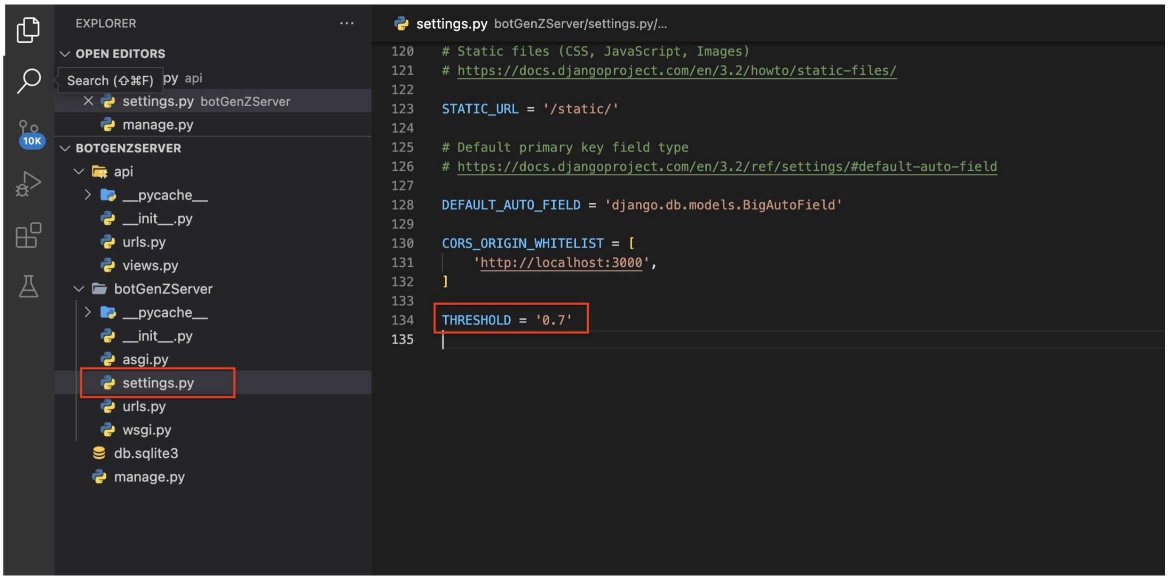


Figure 4.16: Server-Side 'Threshold' Setting

The server-side code has a settings.py file in the root directory. This file has a 'THRESHOLD' variable, whose value by default is set to 0.7, which can be changed by the designer for the interaction to allow a lesser cosine similarity for a semantic match.

5 Engineering

5.1 Engineering Design Choices

5.1.1 Technologies to Use

In this section of the document, we will look into the decisions taken for the engineering design. In talks in detail about the choice of technologies made, the high-level architecture of the system and the reasons those choices were made.

Before starting the development process, first of all it was important to figure out what is the technical scope of this project. It was important to choose the right technologies to build the project keeping in mind scalability, integration, and future extension of the tool. Below are the decisions with respect to technologies taken for the tool:

1. **React**:

    - The best part about React is the concept of Virtual-DOM which is a copy of the actual DOM which react keeps, to make selective DOM updates and thus making the rendering process significantly faster than any other framework / library.
    - Since React works on top of Node, the dependency management using npm (Node Package Manager) is handled really nicely.
    - The component style development of the UI facilitates modularity as well as reusability.
    - It has a vast community support.

2. **Material UI**:

    - Material UI is a React design library based on Google Material Design. It facilitates faster Web Development.
    - Material UI components support component level styling and just only the required styles are injected to the component where they are being used.

- It supports customization to a greater extent. Hence, any design changes to the out-of-the-box designs can easily be created.
- Similar to React, this also has a huge support community and thus the library keeps getting regular updates.

3. **Django**:
   - Django has the support for REST framework and thus developing Rest APIs happens much quickly as compared to other frameworks.
   - Django provides scalability to the application as it is based on loosely coupled architecture. Anticipating further enhancements of this project (which we will discuss in the 'Future Work' section of this document) Django seemed like a good fit.
   - Out-of-the-box support for SQLite database makes it easy to dump any data in a persistent storage. This would be helpful when the current local storage of conversation tree is changed to a persistent storage.
   - It has been adopted by many companies and thus it is a tried and tested solution. This also implies that it has a huge community support as well.
4. **React-flow**:
   - For the draggable flow-chat kind of visualization, React-flow looked like a good fit to me as it satisfied all the requirements that I had for this project
   - Although the learning for all its alternatives including react-flow itself was there to a large extent, however when compared to other options like React-dnd, react-beautiful-dnd, react-draggable , and gojs the ease of development using React-flow seemed better to me.
5. **AWS**:

   The plan was to use AWS (Amazon Web Services) to enable communication between multiple components of the solution. However, since my priority was to come up with a design and check the feasibility of such a tool, the deployment was not done due to time limitations. Having said that, when expanded, AWS would be the perfect choice for multiple parts (described below in the tech architecture part) of this project. The reason being:
   - AWS has multiple services which would suffice the cloud communication requirements of the tool.
   - Integration between different cloud services within the same AWS ecosystem is seamless.

- AWS will provide scalability to the project.
- With Auto-scaler components like AWS Lambda, it will ensure high availability of the project.

5.1.2 Development Task Break-up

To start with any technical project, it becomes very important to have the tasks along with their priorities listed, so that phase wise development can be done. Same was the case with this project. I started by listing down the development tasks breakup alongside the priorities of each of them, which looked like this (Figure 5.1):

Task	Component Type	Priority
Design Time Tasks		
Nav Bar Designing	Front End	P1
Entity Design (All Entity types)	Front End	P1
Entity Addition	Front End	P1
Entity Relation	Front End	P1
NLP technique selection	Front End	P2
Save Button functionality	Front End	P2
DB Setup	Back End	P2
Save & Deploy feature implementation	Back End	P2
NLP Engine Deployment	NLP	P3
Corpus selection API	Back End / NLP	P3
Run Time Tasks		
Entity Design (All Entity types)	Front End	P1
Conversation Flow Design	Front End	P1
NLP resolution APIs	Back End	P2
AWS SQS Setup	NLP	P3

Figure 5.1: Development Tasks break-up

5.1.3 NLP Integration

The NLP piece of this project was not the main areas of research. Due to the fact that most of the survey results in the literature review iterated that the participants expected the chatbots to have some sort of intelligence in processing the user text, the NLP integration was thought of. It would be a 'good-to-have' kind of a touch to the tool.

Semantic Matching

As mentioned in the 'Requirement Analysis' section of this document, we will look into how 'Semantic Matching' was integrated into the tool.

One way of carrying out semantic matching was to build a NLP classification model which when given 2 sentences, can classify whether they are similar or not. For this purpose, I could find 2 public datasets which could be used.

- SNLI Dataset (22) – The Stanford Natural Language Corpus is a public dataset which has 570K English sentence pairs which are manually labelled for balanced classification (23).
- Quora Question Answer Dataset (24) – This is a publicly available, manually labelled dataset released by Quora, which helps researchers experiment on question answer type challenges. The dataset contains 400K lines of pairs of potential duplicate questions. Each pair has been labelled with 0 or 1 whether the questions are duplicate of each other or not.

The above-mentioned datasets could be used to build a NLP based classification model which would require time being invested into Data fetching, Exploratory Data Analysis, Data Clean-up, Modelling, Training and then finally Prediction. This would take substantial amount of time and effort being put into just the NLP model creation part and then the NLP integration with the tool is anyways going to take its own time as well. Considering the demands w.r.t time and efforts of this approach, I thought of pivoting to using pre-trained model as the accuracy of the prediction was not my area of research interest in this case. The benefit of having NLP integrated would rather be having a way established, in which intelligence could be brought into the tool. At a later stage, when adoption of the tool happens and/or any better performing model has been developed, one can anyways go ahead and replace the existing model to get better prediction.

Word Embeddings to rescue

Embedding of any form basically means representing any point present in a high dimensional vector space into a comparatively low dimensional vector space. Similarly, according to (25) word embeddings are low level vector representation of Natural Language words having the semantics preserved. Once we translate the semantic importance of words into a mathematical representation, it becomes easy to perform mathematical operations on it.

I could again find 2 ways to derive word embeddings:

- **Word2Vec**:

  Word2Vec is a classic method developed by Tomas Mikolov's team at Google in 2013. This method translated words into their vector representation using 2 different learning models namely Continuous Bag-of-Words (CBOW) model and Continuous skip-gram model.

- **BERT Embedding**:

  BERT (Bidirection Encoder Representations and Transformers) is a machine learning technique developed by Google in 2018. It has a pre-trained model for Natural Language Processing which has been trained on a huge corpus of 2,500 million words (i.e. the entire Wikipedia). It being bidirectional, BERT derives information about a word's context both from the start as well as the end of a sentence. This basically results in more contextual information being saved. Using BERT embeddings thus will give better vectors representation of words. For e.g.: the word 'apple' in the sentences 'I want to eat an apple' and 'Apple is going to release its iphone 14 early September this year' is going to have different vector representation. Hence, the embedding is respecting the context in which, the words are being used. It is because of this fact, that I decided to go ahead with BERT embeddings in my project.

Similarity Calculation using Cosine Similarity

To calculate the similarity between 2 vectors, I considered using cosine similarity. Two points would align close to each other in the vector space if the cosine of the angle between them is equal to 1. Which means the angle between the vectors representing them is 0. Similarly, 2 words would be similar to each other if the cosine of angle between the vector representation of the words is closer to 1. For example, the word 'happy' and 'delighted' would occupy places closer to each other in the vector space. Hence, once the vectorization of the user texts is done, all we need to do is check the cosine similarity between them. If it is greater than the threshold set by the designer (by default, 1), the sentences can be deemed semantically similar to each other. The other way of calculating similarity between 2 sentences was to calculate Euclidean Distance between them. Once the words have been converted to vectors, we can calculate the Euclidean distance and then check the words similar to each other will appear close by to each other and thus the Euclidean distance between them will be smaller when compared to the words which are less similar.

5.1.4 Persisting Chat Bot Designs in a JSON file

With the current scope of this project, having the complete tool deployed was not a priority as it didn't hold much of research importance. Hence, I thought of adopting a fairly simple way of storing the conversation tree that the user would create during the design phase. The idea was to store the metadata in the form of a JSON in the react global state. Mentioned below is how the JSON structure for a simple conversation tree with 3 nodes would look like (Figure 5.2):

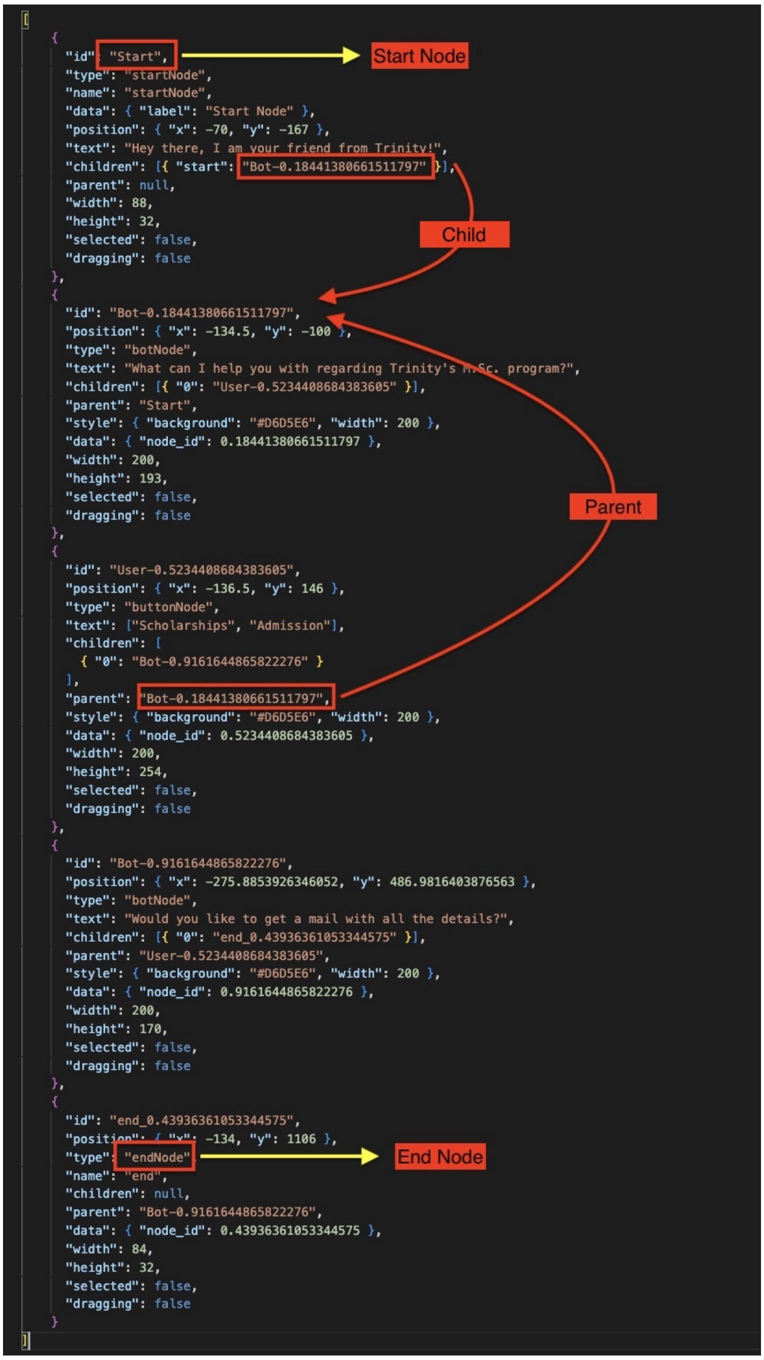


Figure 5.2: Sample Structure of the saved JSON file

We can see that the JSON is in turn a list of JSON objects. As shown in the picture above, the Start Node with node id 'Start' denotes the start of the conversation and the End Node with type as "endNode" and id "end_<random_id>" denotes the end of one branch of the flow. Other than that, the JSON has maintained a bi-directional relationship. Wherein, a parent node contains node ids of all its child nodes, and any child node contains the node id of its parent node as well. This way, resolving the forward path and back-tracking in case of runtime text mismatch. Along with that, each object in the list has other attributes like id, position, text, type etc. Additionally, the text type nodes will have an attribute 'isNLP' to determine whether semantic matching has been enabled for that node or it.

This JSON in the React state during the design phase gets updated every time:

- A new node is added during the design phase
- There is a new link created between 2 nodes: The respective nodes' 'children' and 'parent' attributes get updated with correct values
- The save button of any node is clicked: When user clicks any node's save button, the JSON gets updated with the text entered in that node.

5.1.5 The CORS Issue

CORS (Cross-Origin Resource Sharing) is a way of letting a server choose origins that are allowed to make request other than the ones which originate from its own domain. This is done using through a HTTP header. By default, a Django server doesn't allow cross origin server and thus we were not able to make requests to the server from our UI client which was running at a different port of localhost. Hence, we used CORS middleware which lets a user add whitelisted origins to the Django settings file. This way, all the requests from these origins, which are added to the CORS whitelist are allowed to make calls and are responded to by the server.

5.1.6 Chat Rendering Libraries in React

For easier implementation, I looked into some chat rendering libraries available for React like pubnub (26), Chat UI Kit from chatscope (27) etc. However, I found that the amount of customization offered by these was very limited. Thus, these would not suffice me in designing all the different types of user response and bot response components. Hence, I refrained myself from using these and rather built the components myself using Material UI components.

5.1.7 NLP Integration Complexities

NLP integration with the front-end was a bit complex not only in the design phase but also in the interaction phase. To overcome this, we decided to add the 'isNLP' attribute (also described earlier) to the constructed JSON so that in the runtime, this can be used to decide whether the node being executed has to be checked for semantic similarity or not. For the case where it is set to true, a REST call is made to the server and since this is an async call, it was important to handle the sequence of execution. We decided to handle this case using async/await block in the front-end,

5.2 Technical Architecture

5.2.1 High-Level Architecture

Having a proper idea of the detailed technical aspects associated with the project, I then designed the high-level architecture of the project. The tool at its current stage has not been deployed. Due to time limitation, another local JSON based approach was taken to accomplish saving and loading the chatbots during design phase and the interaction phase respectively. However, after its full-scale deployment, the high-level architecture would look something like figure 5.3:

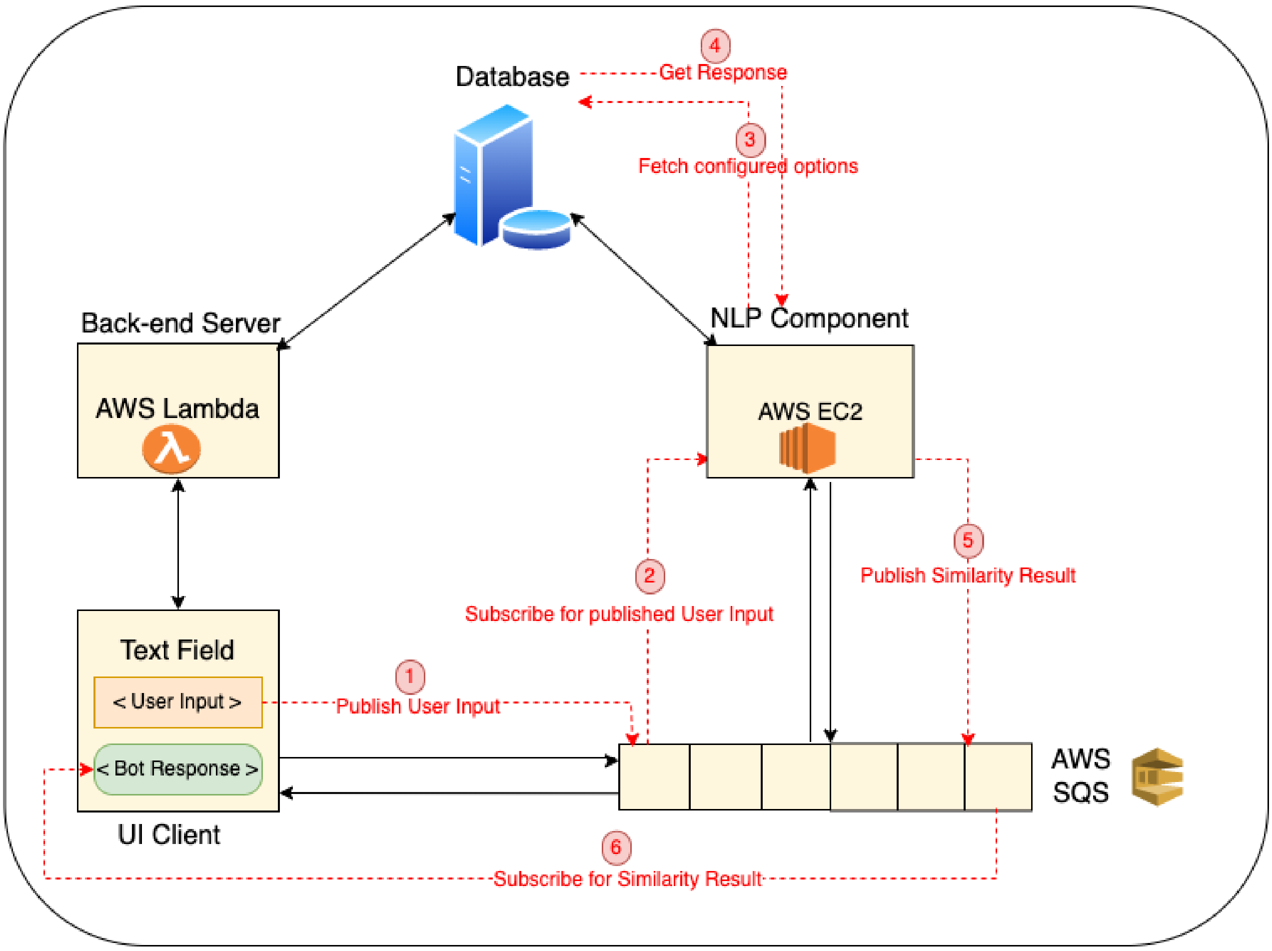


Figure 5.3: Technical Architecture

Let's look in detail at all the components mentioned in the architecture and then we will look into how each of them will connect with other components for the overall functionality.

Back-End Server deployment to AWS Lambda

AWS Lambda is a serverless computing platform which comes as a part of the Amazon Web Services. It is event-driven, which means based on events, a Lambda can run its code. It manages the resources required for computing automatically. The back-end server can be deployed to a serverless lambda. AWS Lambda being an auto-scaler will ensure that the server scales horizontally and spins new instances in the case of high demand or shuts additional instances down in case the number of hits are less. This also ensures that the server isn't up and running even if there is no traffic. The server sleeps when there is no traffic and wakes-up only when there is traffic encountered.

The Lambda will additionally have an API Gateway attached to it through which, interaction with the lambda can be facilitated.

UI deployment to Heroku

Heroku is a Cloud Platform-as-a-service (PaaS). It allows cloud hosting and also provides SSL certified domains. It also allows domain forwarding to any custom domain the user has purchased. With Heroku, continuous deployment (CD) is possible using GitHub.

NLP component deployment to an AWS EC2

AWS EC2 (Elastic Compute Cloud) is a service provided by Amazon as a part of Amazon Web Services which lets user rent virtual machines in the cloud.

The NLP component can independently be deployed to an AWS EC2 machine. This ensures that it works as a separate service and remains segregated from the back-end server. The pre-trained NLP models being used for sentence transformation can be saved in the EC2 as well.

AWS SQS as an intermediate messaging queue

AWS SQS (Simple Queue Service) is a distributed messaging queue service provided by Amazon. Having an AWS SQS queue as an intermediate service will give us the flexibility to have one or limited instances of the EC2 instances running at any point of time. This can have the semantic matching requests queued thus providing serial execution of request. We will look more into its working in the following paragraphs.

ArangoDB as Database

ArangoDB is a multi-modal open-source database. It enables us to store data as a graph. The data being stored for our tool is of a tree format and the ideal way of storing this would be to a graph DB. For that very reason, I would consider using ArangoDB. We can have different collection types defined for each of the node types. On deployment of the design-time draft, we can accordingly create instances of the collections and link them in the database, which means we can easily query them in the runtime.

5.2.2 Runtime Execution Flow

In the architecture diagram (figure 5.3), we can see the red dotted lines which depict the runtime execution flow in case of a NLP enabled text-node. Let's look into it in detail:

1. When user enters text into the text field, the UI client publishes the text into the SQS queue. When multiple instances of UI client are operative, they will accordingly queue their requests in the queue in sequence.

2. Since the NLP component has subscribed to the SQS queue, it will get the message as soon as it is published to the queue. It will take the message and extract the user input along with other details like stage_id (denoting current stage of the conversation flow) etc.

3. The NLP component sends a request to the database to get the configured options for that particular stage.

4. The database is queried for the configured options based on the 'stage_id' and the retrieved result is sent back to the NLP component.

5. Once it receives that, it transforms all the received texts to vectors, calculates semantic similarity, and finds the path that needs to be executed next. The output of the NLP component gets published to the SQS queue, so that the UI client can receive the response.

6. The UI client then takes the response from the Queue and displays the next Bot response accordingly in the screen.

The above-mentioned flow was the case when the run-time engine finds a user text node which had 'Semantic Matching' set to 'Enabled'. For every other case however, the UI client makes a direct call to the Back-end server which then queries the Database and returns the path to be followed. In this case, there is no interaction required with the NLP component or with the SQS.

6 Evaluation

In the book **'Interaction Design'** (28), Evaluation has been described as *'the process of systematically collecting data that informs us about what it is like for a particular user or group of users to use a product for a particular task in a certain type of environment'*.

To emphasis on the importance of evaluation, the book quotes Nielsen Norman Group (a usability consultancy company) 's lines, which goes as follow:

"User experience encompasses all aspects of the end-user's interaction . . . the first requirement for an exemplary user experience is to meet the exact needs of the customer, without fuss or bother. Next comes simplicity and elegance that produce products that are a joy to own, a joy to use."

For the evaluation of the tool, we would have loved to have some actual users interact with the tool and provide their feedbacks. However, to avoid getting into the ethics clearance, which might take some time, we chose not to take that route. Also, finding the right set of users who would have prior working experience in the HCI space and would have interacted with such tools is difficult as well. Sharing the technical know-how of the tool, including the setup, the functionalities, the validations in place etc would also have taken significant amount of time. Thus, we decided to do Scenario-based Evaluation. The idea is to use our tool to develop conversations for scenarios that we have mentioned earlier during this document and some other scenarios as well. The conversation situation would be chosen to fit in the basic scenarios and domains where chatbots have generally been used. Based on whether or not the tool was able to create conversations for all such different scenarios, we would be able to establish an impression of the tool.

Let's first get back to the example of our researcher "Mr. X" mentioned during the requirement analysis. He was looking to have a conversation drafted that would fit into his Travel domain's research work. One of the basic flows of such a tool would be to have a conversation to have a booking done or to ask for a refund or to re-schedule a booking. For that purpose, we did design the conversation mentioned in the figure below (Figure 6.1):

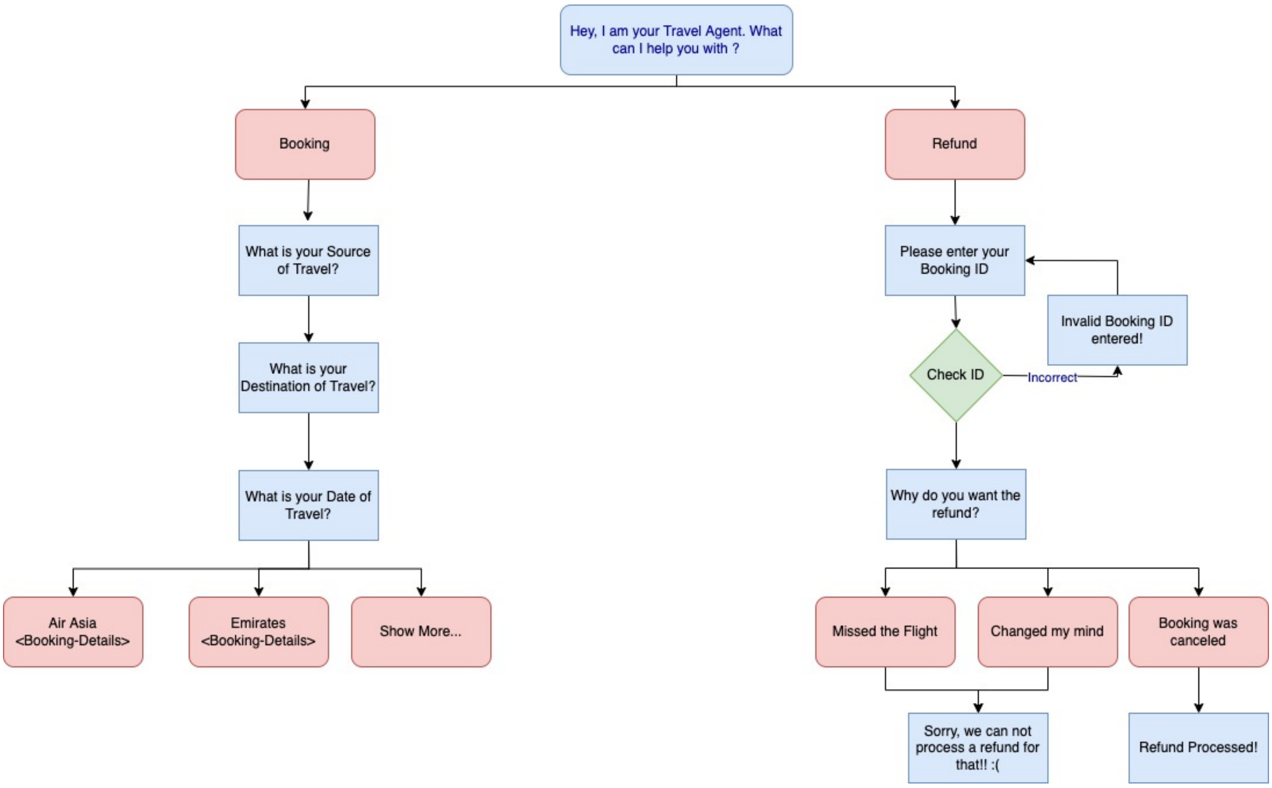


Figure 6.1: Sample conversation in a Travel Domain

The conversation above encompasses different types of user responses. The chatbot also shows the use of a repair path, in which case, it tracks back to the same question again. It was very quick and convenient to design the above flow and even in the runtime, the chat flows seamlessly.

The next scenario I considered was of an admission process background. Many institutes use chatbots embedded with their web apps, which incoming or current students can interact with. The FAQs that students ask in such a setup are generally similar in such cases. Thus, having a chatbot generally expedites the query resolving process and doesn't necessarily have to have a human-in-the-loop involved to take these questions, unless it is very specific. For such a scenario, we tried to prototype a chatbot that will help incoming students with information regarding admission and scholarships. Illustrated below is how its flow goes (Figure 6.2):

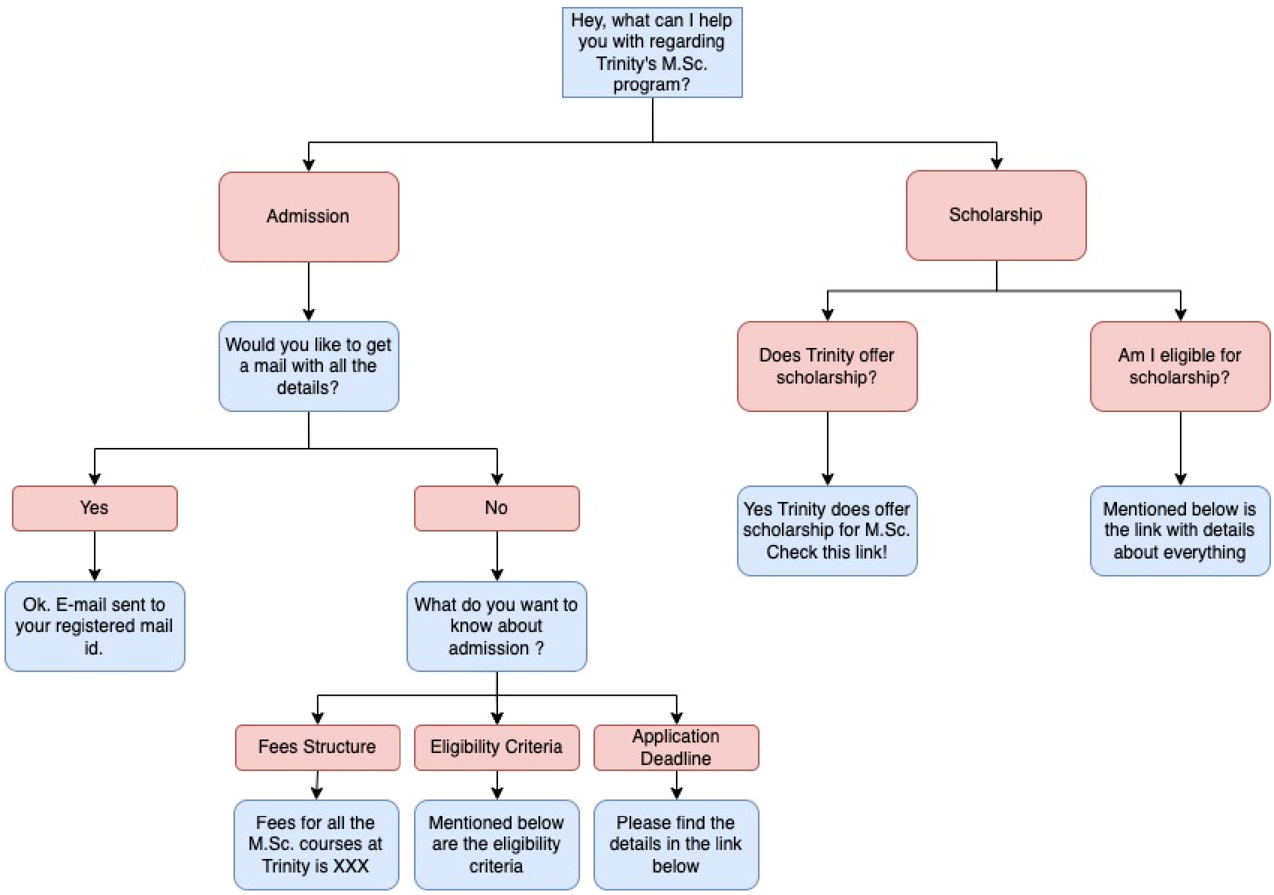


Figure 6.2: Sample Conversation for Admission Queries

In this flow, we could implement 2 basic flows of a conversation form any admission related query. The chatbot addressed questions about admission eligibility, fee structure and deadlines. It also helped its users route to relevant documents for information regarding scholarships. In this case as well, the tool was able to address some of the use cases that we thought would be relevant for this scenario.

The next scenario we looked to implement was more of a casual chat. This as well is a domain that chatbots are being built for in huge numbers. Although an intelligent chatbot which tries to preserve the context and is able to guess the user's intention and mental states in this scenario seems very interesting. This is still an active area of research. We have seen multiple papers which have tried to study how chatbots can build a camaraderie with the user and thus operate in spaces like mental health counselling (2, 4, 5, 20, 29). In this quest, having prototypes in this space which can help researchers study user behavioral patterns becomes very crucial. We thus tried to prototype such a basic conversation as below (Figure 6.3):

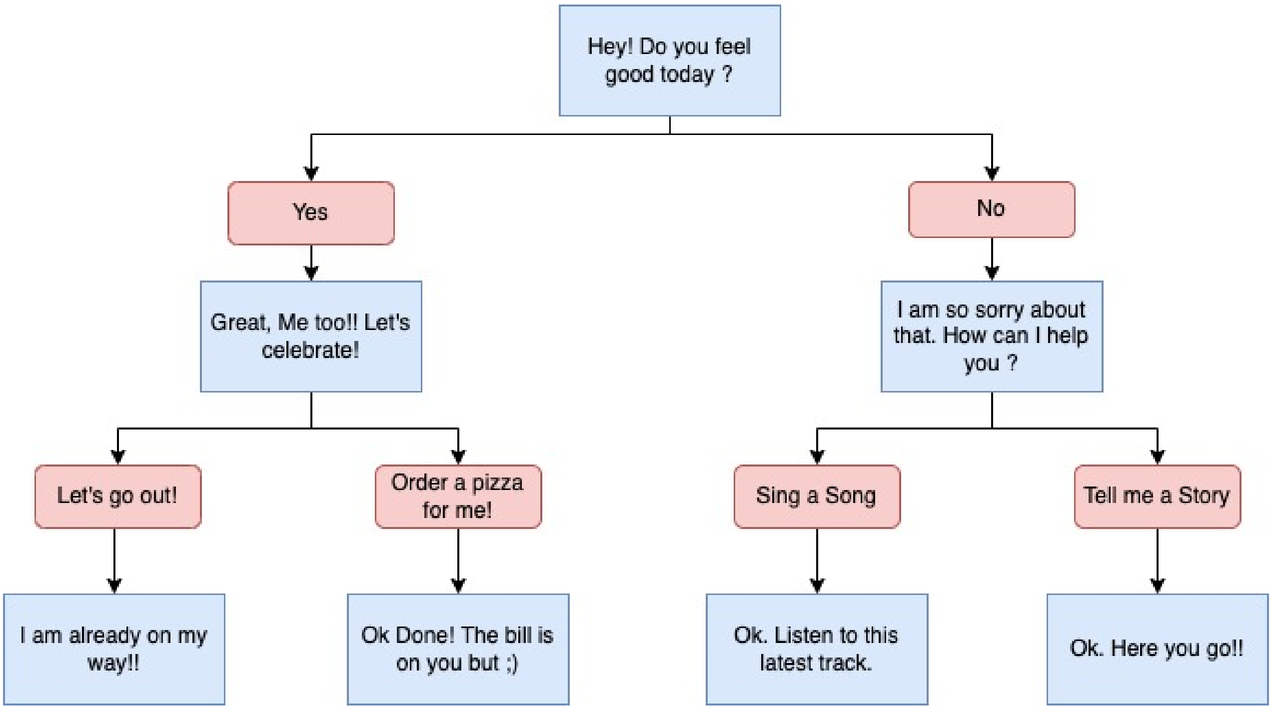


Figure 6.3: Sample Casual Conversation

For the scenario mentioned above, we could implement a very basic conversation a chatbot would have with a user in leisure. Such prototypes can help add different human behaviors like humor, care, compassion etc. and see how users react to it.

From the examples implemented above, we saw that there were different domains that we could use the tool for. With the kind of response type, validations and integration the tool supports, we feel that the tool can definitely be used for a lot more different domains as well as scenarios.

7 Discussion

7.1 Implications

The research helped us:

- Find the basic user expectations from a CUI.
- Get insights on challenges in designing chatbots.
- Develop a prototyping tool which can help researchers prototype chatbots for the most common domains.
- Come-up with an easy-to-use tool which has the basic required functionalities.
- Create chatbots in responsive chat windows and thus provide easy integration.

7.2 Limitations

- Current design doesn't support offline interactions. Once the front-end and back-end are deployed to the cloud, the user will need to be online to be able to access the tool.
- Not every application supports the integration of CUIs.
- To certain tasks, a bot can add much more complexity. They are best done through GUIs. As mentioned in (7), chatbots can not be a substitute of GUI in every domain. Selecting the right domain to have a chatbot deployed is very important.
- The tool is currently not deployed and thus the front-end and back-end servers run locally.
- A permanent store integration (like ArangoDB) is missing and thus users can't have centrally stored chatbot prototypes.
- Implementation of Entity Extraction will add another layer of human-like touch to the chatbots. Although the design of Entity Extraction was finalised, the integration of the same could not be done during the course of the development.

- The tool doesn't support User Management yet. Thus having protytype drafts saved is not feasible at this moment.
- The tool currently doesn't support a repair path. In case of a mismatch, the tool replays the same message again. However a repair path will ideally let the user handle this in multiple ways.

7.3 Future Work

The tool can be extended to any level to add features to it. Some of the major updates that can be added to the tool are listed below:

- Implementing Redux for state management.
- Using ArangoDB as the conversation metadata store.
- Integrating MongoDB to store user data.
- NLP Engine deployment to an EC2 machine.
- Using AWS SQS for NLP Integration.
- Implement and integrate the Entity Extraction technique with the tool.
- Deploy the Front-end and Back-end servers to a Cloud so that the chatbots embedding can be efficiently done to different platforms.
- Implement User Management (may be through services like AWS Cognito) so that users can save their drafts and designs to their accounts.
- Implement long term conversational context preservation.
- This tool can be extended to have voice support as well. That will add another modality to the ways of interaction. (30)

8 Conclusion

This study did extended research on chatbots' history, recent development, characteristics, user expectations, design challenges and limitations. I further studied how certain design characteristics have caused displeasure amongst chatbot users. This paper gathers support for having proper design awareness to be able to design better chatbots or any CUI for that matter. Implications of having bad design process mostly resulting in bad designs could also be seen throughout the study.

The study then also looked at the challenges researchers face while studying impact of design characteristics in chatbots. To facilitate researchers in prototyping chatbots quickly in order to formulate better design standards as well as to help chatbot developers develop better chatbot designs for long term sustainability of chatbots, this study finally proposed a chatbot prototyping tool which could incorporate all the basic features required in a chatbot yet, keeping the chatbot designing process fairly easy.

Evaluation of the tool was done by creating chatbots for different domains where they are widely being used. Along with that, it was also made sure that the chatbots could be integrated to any webpage so that experiments could be performed without the participants having to leave the experiment setup.

Bibliography

[1] Asbjørn Følstad, Marita Skjuve, and Petter Bae Brandtzaeg. Different chatbots for different purposes: towards a typology of chatbots to understand interaction design. In *International Conference on Internet Science*, pages 145–156. Springer, 2018.

[2] SoHyun Park, Anja Thieme, Jeongyun Han, Sungwoo Lee, Wonjong Rhee, and Bongwon Suh. "i wrote as if i were telling a story to someone i knew.": Designing chatbot interactions for expressive writing in mental health. In *Designing Interactive Systems Conference 2021*, pages 926–941, 2021.

[3] Mohit Jain, Pratyush Kumar, Ramachandra Kota, and Shwetak N Patel. Evaluating and informing the design of chatbots. In *Proceedings of the 2018 designing interactive systems conference*, pages 895–906, 2018.

[4] Timothy W Bickmore, Lisa Caruso, Kerri Clough-Gorr, and Tim Heeren. 'it's just like you talk to a friend'relational agents for older adults. *Interacting with Computers*, 17(6): 711–735, 2005.

[5] Minha Lee, Sander Ackermans, Nena Van As, Hanwen Chang, Enzo Lucas, and Wijnand IJsselsteijn. Caring for vincent: a chatbot for self-compassion. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2019.

[6] Joseph Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966.

[7] Jonathan Grudin and Richard Jacques. Chatbots, humbots, and the quest for artificial general intelligence. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–11, 2019.

[8] Dave Feldman. Chatbots: What happened?, Apr 2018. URL `https://chatbotslife.com/chatbots-what-happened-dcc3f91a512c`.

[9] Følstad Asbjørn and PB Brandtzæg. Chatbots and the new world of hci. *Interactions*, 24(4):38–42, 2017.

[10] Welcome back!, . URL https://app.botmock.com/.

[11] Your conversational design suite, . URL https://botsociety.io/.

[12] Leigh Clark, Nadia Pantidi, Orla Cooney, Philip Doyle, Diego Garaialde, Justin Edwards, Brendan Spillane, Emer Gilmartin, Christine Murad, Cosmin Munteanu, et al. What makes a good conversation? challenges in designing truly conversational agents. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2019.

[13] Facebook chatbots hit 70% failure rate as consumers warm up to the tech. URL https://www.mediapost.com/publications/article/295718/facebook-chatbots-hit-70-failure-rate-as-consumer.html.

[14] Jennifer Zamora. I'm sorry, dave, i'm afraid i can't do that: Chatbot perception and expectations. In *Proceedings of the 5th international conference on human agent interaction*, pages 253–260, 2017.

[15] Christine Murad and Cosmin Munteanu. "voice-first interfaces in a gui-first design world": Barriers and opportunities to supporting vui designers on-the-job. 2022.

[16] Walmart acquires design tool botmock as its invests in shopping by voice and text. URL https://tcrn.ch/3k1Us81.

[17] Open source conversational ai, . URL https://rasa.com/.

[18] Tutorials, . URL https://rasa.com/blog/category/tutorials/.

[19] Ewa Luger and Abigail Sellen. " like having a really bad pa" the gulf between user expectation and experience of conversational agents. In *Proceedings of the 2016 CHI conference on human factors in computing systems*, pages 5286–5297, 2016.

[20] Alaa A Abd-Alrazaq, Mohannad Alajlani, Ali Abdallah Alalwan, Bridgette M Bewick, Peter Gardner, and Mowafa Househ. An overview of the features of chatbots in mental health: A scoping review. *International Journal of Medical Informatics*, 132:103978, 2019.

[21] the collaborative interface design tool. URL https://www.figma.com/.

[22] The stanford nlp group. URL https://nlp.stanford.edu/projects/snli/.

[23] Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*, 2015.

[24] First quora dataset release: Question pairs. URL https://www.kaggle.com/datasets/sambit7/first-quora-dataset.

[25] Nlp: Contextualized word embeddings from bert. URL https://towardsdatascience.com/nlp-extract-contextualized-word-embeddings-from-bert-keras-tf-67ef29f60a7b.

[26] Chat ui components to build chat applications with pubnub on react with typescript support. URL https://github.com/pubnub/react-chat-components.

[27] @chatscope/chat-ui-kit-react. URL https://www.npmjs.com/package/@chatscope/chat-ui-kit-react.

[28] Jenny Preece, Yvonne Rogers, and Helen Sharp. *Interaction Design*. John Wiley & Sons, Inc., USA, 1st edition, 2002. ISBN 0471492787.

[29] Stefan Olafsson, Teresa K O'Leary, and Timothy W Bickmore. Motivating health behavior change with humorous virtual agents. In *Proceedings of the 20th ACM international conference on intelligent virtual agents*, pages 1–8, 2020.

[30] The current state of chatbots in 2017, Apr 2017. URL https://getvoip.com/blog/2017/04/21/the-current-state-of-chatbots-in-2017/.