MASTER OF TECHNOLOGY

(COGNITIVE SYSTEMS)

PROJECT REPORT

Smart Singapore Zoo Chatbot

GROUP 9 TEAM MEMBER
MA WEIZHONG
XU KAIXIN
YU XIAOXI
ZHANG ZHILIN

CONTENTS

1 EXECUTIVE SUMMARY 3

2 PROBLEM OVERVIEW 3

2.1 Problem Statement 3

2.2 Project Objectives 4

2.3 Product Overview 4

2.4 Requirement & Constraint 5

3 TECHNICAL DISCUSSION 5

3.1 System Architecture 5

3.2 Google Dialogflow 8

3.3 TF-IDF 9

3.4 Cosine Similarity 10

4 TECHNICAL SOLUTIONS 11

4.1 Data Scraping & Collection 11

4.2 Intent and Entity Design 13

4.3 Algorithm Implementation 14

4.4 Google Assistant 16

5 CONCLUSION 16

5.1 Limitations 16

5.2 Improvement 16

5.3 Conclusion 16

1 EXECUTIVE SUMMARY

AI chatbots are becoming a more common form of service technology, with applications in many industries, including banking, customer service, logistics and education. Traditional computer interfaces require structured and predictable input to work properly, which makes their use neither natural nor friendly. If users cannot easily understand their structured input, it is difficult for them to figure out how to query using such a system. An ideal interface would be able to infer what users want from the natural language they use, whether it's text input format or presentation. The project of the Singapore Zoo, proposes and implements a chat robot of artificial intelligence system, allows the user to enter a text-based query, system will be response to relevant information. The purpose of the chatbot system is to allow users to search for more public information often referred to as frequently asked questions in a more efficient and responsive way that adapts to different syntax challenges related to syntax but still returns a response based on keyword queries through natural language processing (NLP). The effectiveness of such systems is evaluated on the basis of efficiency of performance and effectiveness of function. On this basis, some suggestions are put forward to improve the robustness of the system.

2 PROBLEM OVERVIEW

2.1 Problem Statement

Currently, there are no chat bots embedded in existing Singapore Zoo websites. Singapore Zoo has five main part, Wildlife Reserves, Jurong Bird Park, River Safari, Night Safari and Singapore Zoo. In each part, it has many kinds of zones, activities and animals. There is a large range of information only in Night Safari part. Browsing through the many webpages of these activities to find out and compare various pieces of information would be a tedious and cumbersome process.

2.2 Project Objectives

This project for part of the Singapore Zoo website
(https://www.wrs.com.sg/en/night-safari/animals-and-zones.html), proposes and implements a chat robot of artificial intelligence system , which allows users to input text based query, system will be response to relevant information. The intent behind this chatbot system is to allow users to search for more common information often referred to as frequently asked questions in a more efficient and responsive way by providing query syntax for different grammars but still be able to return a relevant response based on keyword queries through natural language processing (NLP).
The chatbot, developed by our team, can answer visitors' questions about the night safari section of Singapore zoo and provide specific answers to each activity. By interacting with the chatbot, users can get answers to specific questions such as time and place, as well as a general answers about animal knowledge. Our goal is to provide a quick, effective and engaging experience for visitors.

2.3 Product Overview

Our processing of user input is mainly divided into two modules. First, the query matches the user's input with the intent defined in Google Dialogflow. If the corresponding intent matches, it will be processed by the intent and entity in Dialogflow. If the corresponding intent does not match the user's input, it will be transmitted to python by Dialogflow, processed by TF-IDF method and Cosine Similarity, and the answer to the user's input question will be returned.
This application is built using Google Dialogflow technology and integrates Python to provide dynamic response. Dialogflow provides predefined entities that can be used to categorize the extracted parameters, while allowing the creation of custom entities. The agent matches the user's utterance to intent, extracts identified parameters, and returns a response, either static or dynamic. The purpose of this particular agent was to answer questions about the curriculum of the administrative education programme.
This chatbot basically takes a user's query and provides an appropriate answer based on the information it crawls from the Singapore zoo website. The design and deployment of chatbot system is carried out in DialogFlow hosted by Google.

DialogFlow is a service that lets users create a chatbot based on artificial intelligence (AI) and learn about it for a while using machine learning technology. In the system, there is an implementation engine that allows dynamic responses to be generated based on queries and matching the information. This chatbot is a retrieval based system designed to quickly guide users through a streamlined information channel as efficiently as possible. As a means of customer service, it is designed to make user surveys and other processes more efficient and user-friendly than traditional methods by simply using a search function that will return relevant information based on keyword tags.

2.4 Requirement & Constraint

1) Intents pertaining to the activities and animals
2) Customized entities
3) Python scripts for responses
4) Organized data (scraped from Singapore Zoo website)

3 TECHNICAL DISCUSSION

3.1 System Architecture

Self-learning bots are the ones that use some Machine Learning-based approaches and are definitely more efficient than rule-based bots. These bots can be of further two types: Retrieval Based or Generative

1) In retrieval-based models, a chatbot uses some heuristic to select a response from a library of predefined responses. The chatbot uses the message and context of the conversation for selecting the best response from a predefined list of bot messages. The context can include a current position in the dialogue tree, all previous messages in the conversation, previously saved variables (e.g. username). Heuristics for selecting a response can be engineered in many different ways, from rule-based if-else conditional logic to machine learning classifiers.

2) Generative bots can generate the answers and not always replies with one of the answers from a set of answers. This makes them more intelligent as they take word by word from the query and generates the answers.

The anatomy of a chatbot is showed below.

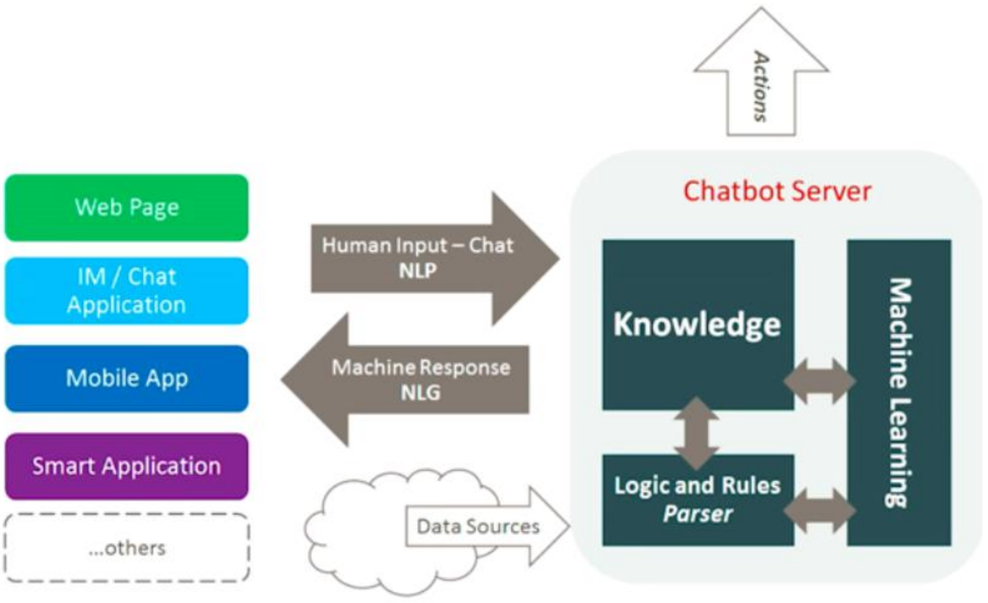

Figure 1 Anatomy of A Chatbot

Our chatbot is a kind of retrieval-based chatbot. The most important parts of our chatbot is building Dialogflow and achieving algorithm by python. In the python scripts, we use TF-IDF and Cosine Similarity to analyze user input.

We trained one intent on Dialogflow platform, which includes dozens of parameters that allows our customized fulfillment script to retrieve information from our knowledge base based on the parameters. One of the advantages of our fulfillment implementation compared to others is that, we implemented an ambiguous approach in the parameter-matching steps, making our product more robust to dynamic user utterances. Beyond that, we also implemented a fallback mechanism in the matching step, allowing a separate FAQ database taking place to generate response. This whole integrate process have great flexibility gluing the dynamic user's query and complex knowledge base structure together.

The core modules of our chatbot system are made out of the parts and corresponding functions in Table below.

No.	Module	Function
1	User Input Interface	To serve as a platform for users to input their queries
2	Natural language processing	To transform unstructured text input into structured and logic form so that the information can be processed and make sense of by the DialogFlow agent (intent and slot/entity detection)
3	Fulfillment	Intent and entity detected will be matched to the business logic configured and once the matching is successful, a dynamic response can be constructed based on the matched intent and its corresponding information.
4	Python Scripts	If the intents do not match the user input, the input will be send here to be processed using TF-IDF and Cosine Similarity.
5	Database	The information that crawled from Singapore Zoo to provide the answer to user input.

Table 1 Core Modules of System

3.2 Google Dialogflow

The flow of conversation within DialogFlow takes place as illustrated in the diagram below.

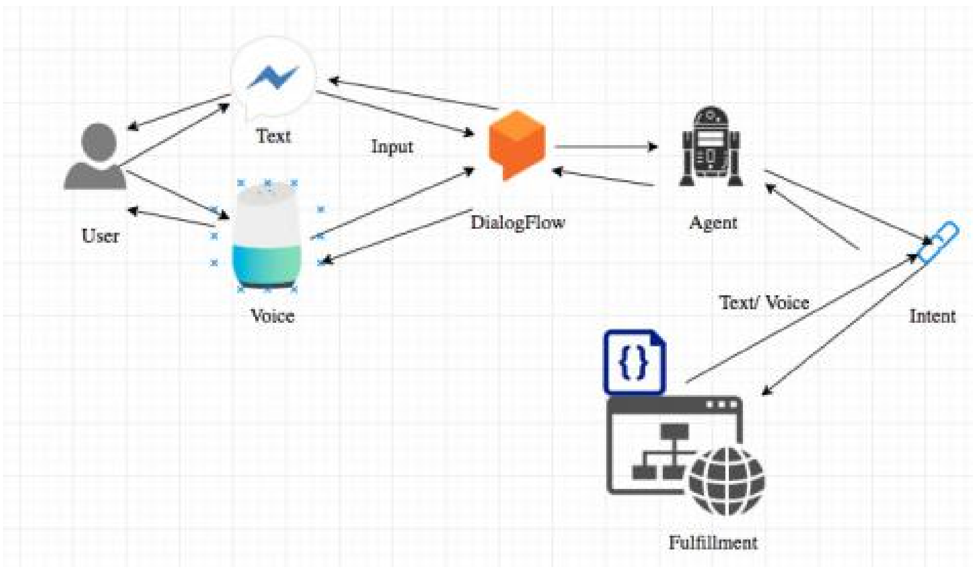

Figure 2 Conversation Flow within DialogFlow

The chatbot is implemented using the DialogFlow platform and python scripts to provide dynamic response through implementation. Figure 3 is the architecture diagram of the system, which demonstrates how an application in the front-end application or device interacts with the user, transmits the discourse to DialogFlow to determine the correct intention, and then returns the corresponding response through fulfillment component.

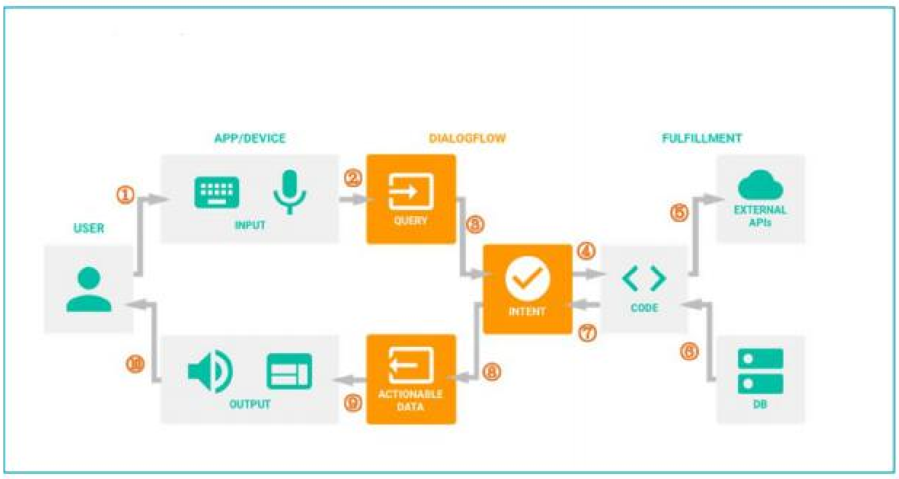

Figure 3: Typical system architecture of the deployed DialogFlow chatbot

3.3 TF-IDF

The main idea of TFIDF is that if a certain word or phrase appears in a high frequency TF in one article and rarely in other articles, it is considered that this word or phrase has good classification ability and is suitable for classification. TFIDF is actually: TF-IDF, TF Term Frequency, IDF Inverse Document Frequency. TF represents the frequency with which entries appear in document d. The main idea of IDF is that if fewer documents contain t, i.e. smaller n and larger IDF, then t has good categorization ability. If the number of documents containing t in a certain category of document C is m, while the total number of documents containing t in other classes is k, obviously the number of documents containing t is n=m+k. When m is large, n is also large, and the value of IDF obtained according to the IDF formula will be small, indicating that the discrimination ability of t category is not strong. But in practice, if an entry appears frequently in a class's documentation, it is a good representation of the text of that class, and such entries should be given a high weight and chosen as a feature word to distinguish them from other class documents. In a given document, term frequency (TF) refers to the frequency of occurrence of a given word in the document. This number is a normalization of term count to prevent it from leaning toward long files. (the same word in a long file may have a higher word count than in a short file, regardless of its importance.) For a word in a particular document, its importance can be expressed as:

$$\mathrm{tf}_{\mathrm{i,j}} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

In the formula above, the numerator is the number of occurrences of the word in the file, while the denominator is the sum of the number of occurrences of all the words in the file.Inverse document frequency (IDF) is a measure of the universal importance of a word. The IDF of a specific word can be obtained by dividing the total number of files by the number of files containing the word, and then taking the logarithm base 10 of the obtained quotient to get:

$$\mathrm{idf}_{\mathrm{i}} = \lg \frac{|D|}{|\{j : t_i \in d_j\}|}$$

Among them

- **|D|** total number of files in the corpus
- **:** the number of files containing the word (that is, the number of files). If the word is not in the corpus, it will cause the denominator to be zero, so it is generally used as the denominator.

Then the product of TF and IDF is calculated.

$$\mathrm{tfidf}_{i,j} = \mathrm{tf}_{i,j} \times \mathrm{idf}_i$$

The high frequency of words in a particular file, and the low frequency of words in the whole file set, can produce tf-idf with high weight. Therefore, tf-idf tends to filter out common words and retain important words.

3.4 Cosine Similarity

TF-IDF is a transformation applied to texts to get two real-valued vectors in vector space. We can then obtain the Cosine similarity of any pair of vectors by taking their dot product and dividing that by the product of their norms. That yields the cosine of the angle between the vectors. The cosine value between two vectors can be obtained by using Euclidean dot product formula:

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \, \|\mathbf{b}\| \cos\theta.$$

Given two attribute vectors, A and B, the cosine similarity is given by dot product and vector length, as follows:

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}.$$

These Ai Bi represent the components of vectors A and B.The similarity given ranges from -1 to 1: -1 means that two vectors point in exactly opposite directions, 1 means that they point in exactly the same direction, 0 usually means that they are independent of each other, and the value between means that they are similar or opposite in the middle.For text matching, attribute vectors A and B are usually word frequency vectors in the document. Cosine similarity can be considered as a method to normalize file length in comparison.In the case of information retrieval, since the frequency (TF-IDF weight) of a word cannot be negative, the cosine similarity of the two documents ranges from 0 to 1. Moreover, the Angle between the frequency vectors of two words cannot be greater than 90°.

4 TECHNICAL SOLUTIONS

4.1 Data Scraping & Collection

The data source for information retrieval is the website of the Singapore Zoo. We extract data before-hand and use as a repository for web-hook to retrive from. This method can improve system reliability. If we run information retrieval in real time, the agent will be exposed to any potential website problems and may return errors. In addition, Dialog flow has a timeout of 5 seconds, which can be used by dynamic retrieval to increase the possibility of timeout. To reduce the risk, extract the data before you choose to run the spatula, and run it regularly every week to update the database.

For web-scraping, we choose Beautiful Soup.

Beautiful Soup uses the structure and properties of web pages to analyze web pages. With it, we do not have to write some complex regular expressions, just need a few simple statements, can complete the extraction of an element in the web page.

In a nutshell, Beautiful Soup is Python's parsing library for HTML or XML to easily extract data from web pages. Beautiful Soup provides some simple, python functions for handling navigation, searching, modifying analysis trees, and more. It's a toolkit that parses documents to provide users with data to grab, and because it's simple, it doesn't take much code to write a full application. Beautiful Soup automatically converts the input document to Unicode encoding and the output document to UTF-8 encoding. You don't need to worry about the encoding, unless the document does not specify a encoding, in which case you just need to specify the original encoding. Beautiful Soup has become as good a Python interpreter as LXML or html6lib, giving users the flexibility to offer different parsing strategies or strong speeds.Therefore, it can save a lot of tedious extraction work and improve the analytical efficiency.

Beautiful Soup actually relies on the parser for parsing, and in addition to supporting HTML parsers in the Python standard library, it also supports some third-party parsers (such as LXML).

Night Safari KB			
“description”: str			
“time”: dict			
	“Opening Hours”: str		
	“Closing Hours”: str		
	“time Slot”: str		
“ticket”: dict			
	“html”: str		
	“Admission ticket prices”: dict		
		“Adult”: str	
		“Child”: str	
		“Child below 3: str	
	“Online discount”: str		
	“tips”: str		
“events”: dict			
	“event 1”: dict		
		“name”: str	
		“data”: str	
		“time”: str	
		“position”: str	
		“content”: str	
	“event 2”: dict		
		“name”: str	
		……	
“animals”: dict			
	“name”: str		
	“life span”: str		
	“diet”: str		
	“Habitat”: str		
	“Range”:str		
	“info”: list		
		“info1”: dict	
			“name”: str
			“content”: str
		“info2”: dict	
			“name”: str
			“content”: str
		…	
…			

Table 2 Knowledge Base Structure

In addition, we divide the crawling data into two modules. The first part is in the form of Jason, which is used to answer the questions matched by Dialog flow. The second part is a TXT file, when Dialog flow does not match the user input, we will use the algorithm to analyze the user input, and query the corresponding answer from the TXT document back to the user.
The structure of knowledge base is showed in table 2 above.

4.2 Intent and Entity Design

A total of 2 different intents and 17 entities were identified, and table 3 shows an example. This set of frequently asked questions will be used as natural language expressions or training phrases to train machine learning (ML) models in Dialogflow. Also, custom entities are annotated during statement compilation. Based on this set of defined data, DialogFlow will expand to create an intent model, and the ML model will later be able to decide which intent to match the user's query to generate the appropriate response. The process is shown in figure 4

Utterance	Entity	Intent
how to get to the night safari?	@NivigateKeyword @Admission	Navigate
where can i buy souvenir?	@QueryKeywords @DineandShop	Night Safari Intent Solver
could you tell me the timetable of night safari?	@QueryKeywords @TimeBoundary @Admission	Night Safari Intent Solver

Table 3 Example of utterance with matching entity and corresponding Intent

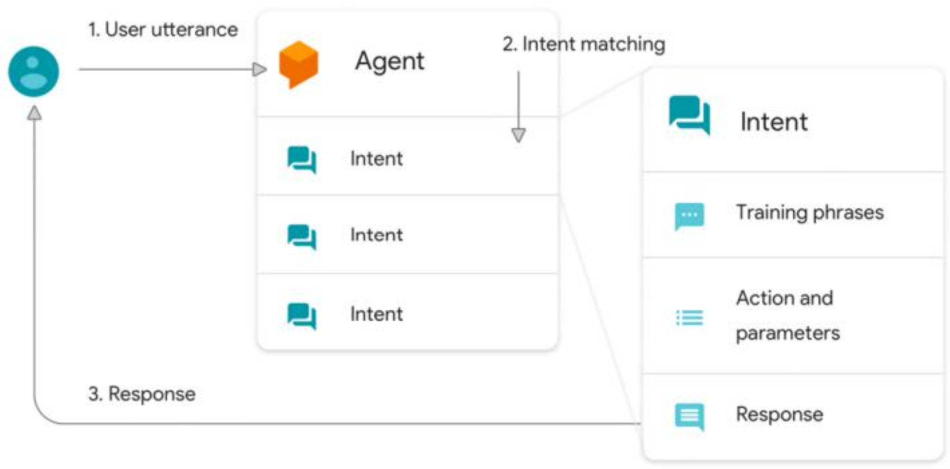

Figure 4 Intent matching to Utterance

4.3 Algorithm Implementation

In the project, we use NLTK to achieve the main function. NLTK(Natural Language Toolkit) is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries.

After reading in data, the first thing we need to do is converting the entire corpus into a list of sentences and a list of words for further pre-processing. Then the function called LemTokens and LemNormalize are defined to normalize tokens.

```
def LemTokens(self, tokens):
    lemmer = nltk.stem.WordNetLemmatizer()
    return [lemmer.lemmatize(token) for token in tokens]

def LemNormalize(self, text):
    remove_punct_dict = dict((ord(punct), None) for punct in string.punctuation)
    return self.LemTokens(
        nltk.word_tokenize(text.lower().translate(remove_punct_dict))   )
```

To generate responses to input questions from our robot, we will use the concept of document similarity. From scikit learn library, import the TFidf vectorizer to convert a collection of raw documents to a matrix of TF-IDF features. Then import cosine similarity module to find the similarity words entered by the user and the words in the corpus.

We define a function "response" that searches for one or more known keywords in a user's utterances and returns one of several possible responses. If it does not find input that matches any keywords, it will return a response: "I am sorry! I don't understand you."

```
    def response(self,user_response):
        robo_response = ''
        self.sent_tokens.append(user_response)
        TfidfVec = TfidfVectorizer(tokenizer=self.LemNormalize,
stop_words='english')
        tfidf = TfidfVec.fit_transform(self.sent_tokens)
        vals = cosine_similarity(tfidf[-1], tfidf)
        idx = vals.argsort()[0][-2]
        flat = vals.flatten()
        flat.sort()
        req_tfidf = flat[-2]
        if (req_tfidf == 0):
            robo_response = robo_response + "I am sorry! I don't understand
you"
            print(robo_response)
            return robo_response
        else:
            robo_response = robo_response + self.sent_tokens[idx]
            print(robo_response)
            return robo_response
```

In addition, we treat user input differently. In the case of a request from Dialogflow, we match it to the defined intent and entities. If it's a request from the Google assistant, we use algorithm showed above to process it and return a relative response.
When both string are single words, we implemented ambiguous matching using NLTK in our project.

```
def __eq__(self, other):
        if not self.s and not other.s:
            return True
        if USE_SYNONYMS:
            return self.s in other.synonym or other.s in self.synonym
        else:
            return self.s == other.s
```

4.4 Google Assistant

We also used Google assistants to help us improve our system. We connect to the Google assistant by calling the API in Dialogflow, so that we can directly test the system on the phone.

5 CONCLUSION

5.1 Limitations

First, our database only contains data from the night safari section of the Singapore zoo, which means we can only answer users' questions about this section.
In addition, in order to effectively process the natural language input from the user, it is necessary to train the chatbot with big data set. In this proof of concept, our chatbot lacks such a data set and is limited to a number of training phrases identified arbitrarily by our team members. This means that if the input is different from our limited training phrases, the chatbot may not be able to recognize the correct user intent.

5.2 Improvement

1) Expand the chatbot's answering capabilities to cover all parts of Singapore Zoo.

2) Add more intents and entities to improve the matching of Dialogflow to user input.

3) Add more training phase and improve the algorithm to provide users with more accurate answers

4) Periodic automated scraping of the Singapore Zoo website to maintain an up-to-date knowledge base.

5.3 Conclusion

This project designs and develops a search-based chatbot system through Dialogflow, which helps Singapore Zoo to handle general queries about its animals and activities. First, the chatbot was trained with many utterances, all with intent annotations and entity markers. Then, when text input is received, DialogFlow detects its respective intent and entity, and the performance engine retrieves and returns relevant information and response to user. Information retrieval is performed through python. With such a system, people's reactions are standardized and there are fewer mistakes. For such a specific closed domain, it can work well to increase customer satisfaction and attention. However, this system shows limited intelligence because it is primarily configured with hard-coded rules. Therefore, no new text of any kind can be generated when the response is selected from a predefined dataset. Nevertheless, the chatbot system designed in this project is easy to develop and implement, because it does not need high data for training, but still can achieve high precision in response generation. Therefore, the chatbot can be used as an effective customer service tool and valuable human resources can be reallocated to handle more complex tasks.