FLIGHTBOT

Knowledge Based Systems

Alex Almansa Casanovas, Marc Llort Maulion

La Salle

2019-2020

index

The problem

The problem that is intended to be solved with this chatbot is basically to make life easier for people who travel a lot, hence its name, Flightbot. Choosing a domain for a chatbot is not easy, because when you think about it, it is easy to find many possible applications for which it could be useful to have a chatbot. Despite these initial doubts, and after evaluating different options, a clear need was seen in the world of travel, and is that most travelers do not have the whole trip planned 100% and there are many things to decide during the same trip. What is intended with this chatbot, is to help you make these last minute decisions that can make the difference between a good trip and a great trip.

It is clear that there is much more information on the internet than a chatbot can offer, which will surely be connected to a single database and will not have as much information as can be found on the internet, but this can also be a point positive for the traveler, because when you are on the streets of a big city, you do not need a lot of information to make decisions, rather you need little information, from a reliable place, and that it offers you information quickly you need what you need at that moment.

For all the points set out above, we have seen the possibility of creating a chatbot, which can be useful for many people, and which can improve the quality of travel of many people from those who use it only for those decisions. less relevant, such as where to have lunch or dinner, as for those who trust it more and leave things to chance, such as what will be the next city to visit or where to sleep the next night.

Goal

To become really useful, this chatbot should, first of all, be made for mobile devices, as it is what you always carry around while traveling. Secondly, you need a large database with reliable ratings of the different sites, to help users decide correctly, which a priori will decide only on the basis of previous ratings as the most normal thing is that nothing is achieved. which surrounds them when they are traveling to a new city.

Apart from all this, we also thought about the possibility of adding a functionality to search for flights to different places, for those more daring travelers, who leave the destination of their trip completely random. For this functionality, a system of recommending cities within a country has been thought of. First, the user will ask the chatbot for one of the major cities in a country. Once the user makes this decision, the chatbot will answer all kinds of questions about things like where to eat, where to sleep, what to visit, and so on.

Another important point to this chatbot, will be that apart from recommending places, give enough information to decide if you want to go or not, so apart from recommending, you must say if the site you are recommending is open, if you have good ratings, and your address, to see if it is very far away. Apart from this, it has been thought to add a functionality to search for interesting places near a place, so that if for example the chatbot recommends a hotel, the user can consult, at the time, what interesting things are near 'this hotel, be it bars, restaurants, or tourist sites.

Knowledge

As for knowledge, it was useful in order to give us an idea of how knowledge databases are created, get in touch with the protégé and make a design of how our knowledge database could be. chatbot.

However, we have finally not put into practice this part of knowledge design as all the information needed to answer and address all questions from the user, is done through Google's apps, so that only we have to worry about making the corresponding request to get the desired information.

However, it must be said that we had designed an ontology that we believe would have been perfectly valid, it may even have gone better than Google, but because in everything that concerns the amount of information and data that there are available, we had nothing to do against the google api, we chose to adapt our chatbot to it, so we did not implement the ontology we had designed.

Reasoning

The main problem when dealing with a recommending system is the information available to the end user. There is always the possibility of making use of existing datasets, or making random recommendations until you can show a more personalized suggestions for the user.

In our case, not much user information is needed, with a continent where you may be interested is enough to make our algorithm work.

It also offers the possibility, in case of not having the continent, to offer a random recommendation.

Randomness causes a new problem, showing the same result to the user in different queries. This fact can make the user think that it is a chatbot still in a very initial state, so we have decided to implement a history, which helps us not only to avoid showing a repeated result, but also adds the possibility of offer the user to show this history, in order to look at previous searches / suggestions that may be of interest to him.

As a result of these two problems, we can extract the following algorithms:

- User information (s):

  It provides the available information of the requested user, so that after the user recommendation functionality a decision can be made based on the information collected by this function.

- updateUser (s):

  It allows, based on the different requests made by the user, to store the information to later make more accurate recommendations.

- User recommendation (s):

  With the information collected and updated by the two previous functions, and making use of the history, it provides a recommendation. This will be more or less accurate depending on the information available to the user.

- historicUser (s):

  Algorithm in charge of informing of which the previous consultations of the user have been, thus to be able to avoid the repetitions of a same recommendation.

System

From the beginning, the intention has been to develop a recommendation system focused on tourism. At first, in the first implementation of the chatbot, it was more focused on flights, and the cities you could visit from your current location.

The first iteration, very simple, was developed in Java because, despite being aware that it is not the best language because it has not many libraries specializing in natural language processing and is quite slow, it is the language with the that we are more comfortable and have more experience, so we could focus more on how to focus our chatbot and understand what is behind it, implementing it from 0 us without the use of any auxiliary library.

This first implementation, helped us realize that implementing a chatbot manually is a lot of work and even more so if you want it to have a good recognition of natural language. It also helped us realize that our domain needed to be expanded, as if we focused only on places to travel / fly, it was a very limited area.

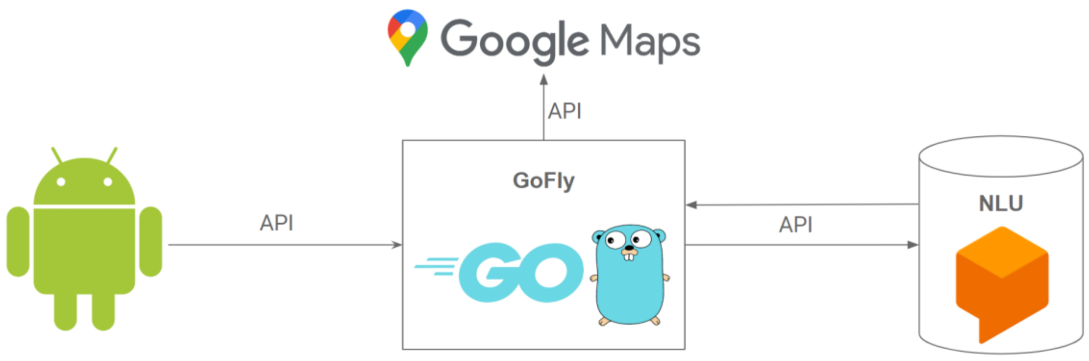
Once it was decided to extend the bot's focus to the different activities that can be done in the recommended city, it was necessary to decide which interface would be used for the user to use the chatbot and how to process the natural language.

Initially, a trial version was implemented with RASA X, a system that already gave an interface and an API with which to communicate with the chatbot, and a natural language system that recognized phrases quite well. It also has a system of calls to different APIs to be able to collect the information.

Although it seemed like an ideal solution, it was decided to opt for DialogFlow, as it allowed the models to be trained faster, had a more understandable interface and a lot of online support.

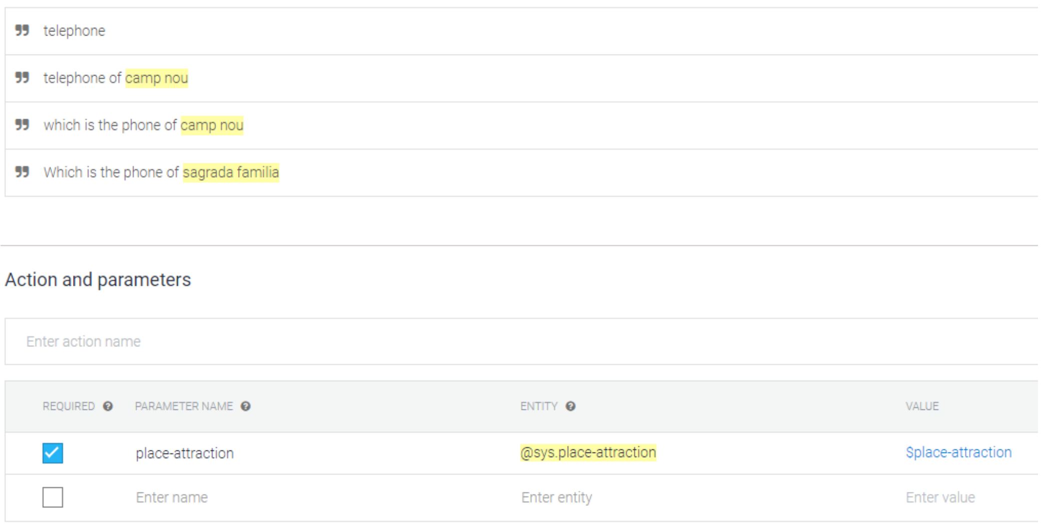
It also offers API calls, although to make use of this feature you need to add a payment system. It is for this reason that a server was created, which would receive the user's calls, redirect them to DialogFlow, and the DialogFlow response would return to the server, which would return the response to the user.

This system design allows that in case the user's message, or DialogFlow's response, has some keywords, such as "hotels", call the Google API [8] to find the hotels in the place requested.



Going more specifically into the design, it should be noted that the interception made by the server, although it can be from Android to Server or from DialogFlow to Server, the second case is always used.

This is because, as you can see in the following image, DialogFlow is still used to recognize the phrase, and once DialogFlow finds that it needs to be answered, it returns the place from which you want to search for the information.



telephone

telephone of camp nou

which is the phone of camp nou

Which is the phone of sagrada familia

Action and parameters

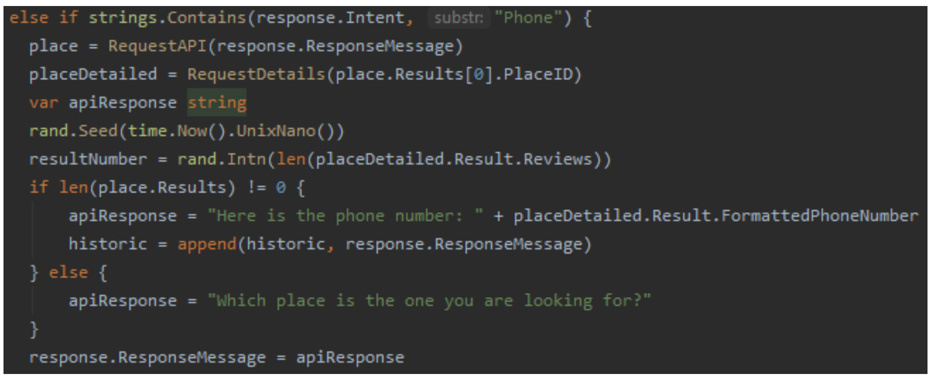
Enter action name

REQUIRED	PARAMETER NAME	ENTITY	VALUE
✓	place-attraction	@sys.place-attraction	$place-attraction
	Enter name	Enter entity	Enter value

Once you know what information to look for (in the example above you could return to the “new field” server), the server needs to know what information to return. To find out what needs to be returned, the server (which uses the DialogFlow [4] api for GoLang [7]) knows that the response returned by DialogFlow is formatted:

```
type NLPResponse struct {
    Intent              string          `json:" intent "`
    Confidence          float32         `json:" confidence "`
    ResponseMessage string              `json:" fulfillmentText "`
    Entities            map [ string ] string `json:" entities "`
}
```

Therefore, you know that you have entered the "Phone" attempt and therefore must return the phone from the specified location to "ResponseMessage".



```
else if strings.Contains(response.Intent,  substr: "Phone") {
  place = RequestAPI(response.ResponseMessage)
  placeDetailed = RequestDetails(place.Results[0].PlaceID)
  var apiResponse string
  rand.Seed(time.Now().UnixNano())
  resultNumber = rand.Intn(len(placeDetailed.Result.Reviews))
  if len(place.Results) != 0 {
      apiResponse = "Here is the phone number: " + placeDetailed.Result.FormattedPhoneNumber
      historic = append(historic, response.ResponseMessage)
  } else {
      apiResponse = "Which place is the one you are looking for?"
  }
  response.ResponseMessage = apiResponse
```

On the other hand, the Android application is responsible for making use of the API provided by the server, where you simply need to make a POST as follows:

```
{
        "User": "Marc",
        "Message": "information of sagrada familia"}
```

The User field, although not currently used, would serve in the future to save the suggestions that have been made to a specific user so as not to repeat them or show results similar to those I had previously requested.

The answer to this POST is as follows:

```
{
     "attempt" : "Info" ,
     "confidence" : 0.64802855 ,
     "fulfillmentText" : "The Holy Family info -
-   Address: Carrer de Mallorca, 401, 08013 Barcelona, Spain - Rating: 4.7 " ,
     "entities" : {
          "place-attraction" : "The Holy Family"
     }}
```

It is always indicated in which Attempt the message sent by the user has entered. The attempt is the classification made by DialogFlow, to find out the correct answer to return. This field, as already explained above, is used to identify what information to look for in the place you are looking for (eg city, tourist attraction ...).

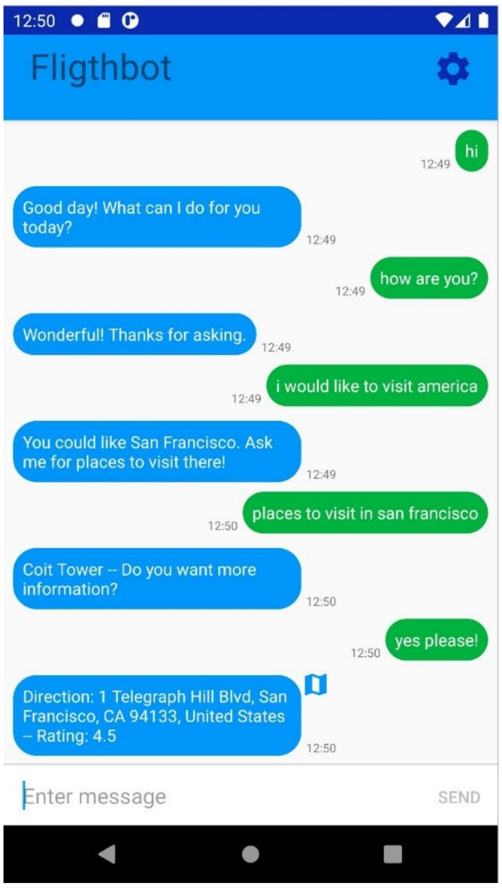
Depending on the similarity to the phrases with which DialogFlow has been trained, you will have greater or lesser security. This fact can be seen reflected in the field "confidence". It is currently simply used for debugging, this way you can know if we need to improve the training according to which phrases are tested.

Fullfilment text is the answer that will appear on the screen to the user, therefore it has the information collected by the different APIs used.

Finally, the entities field returns the place, city, country ... that DialogFlow has recognized from the message sent. It is useful when searching for information about the place that the user wants, for example, if you want information about the Sagrada Familia, in this field the API has the string with which to search inside to external bees.

As it is an API, Postman has been used during development to test its proper functioning easily. When it comes to being used by the public, an Android app that calls the API has been developed. This application provides a configuration screen where the server ip can be easily configured.

It also provides a visual interface that interacts with any address returned by the API, drawing a small map next to the message that when clicked, the address will be shown on the maps and if asked how we can get there will calculate the fastest route from current location.


12:50

Fligthbot

hi 12:49

Good day! What can I do for you today? 12:49

how are you? 12:49

Wonderful! Thanks for asking. 12:49

i would like to visit america 12:49

You could like San Francisco. Ask me for places to visit there! 12:49

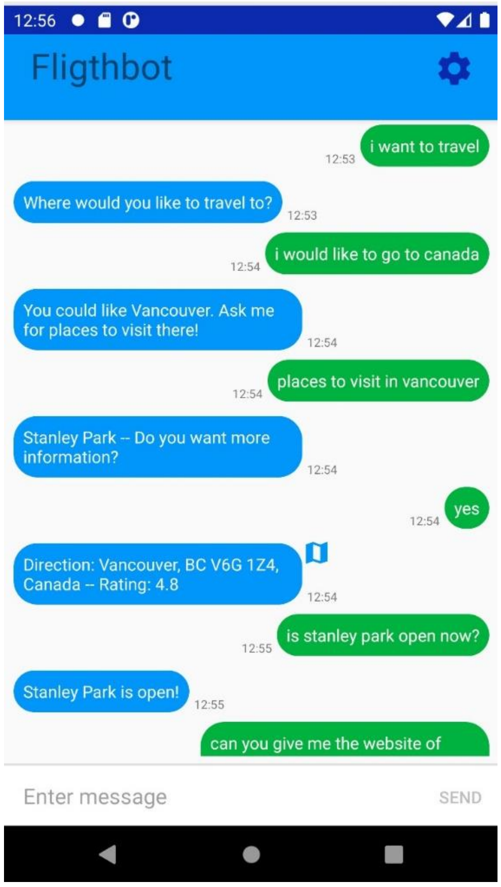
places to visit in san francisco 12:50

Coit Tower -- Do you want more information? 12:50

yes please! 12:50

Direction: 1 Telegraph Hill Blvd, San Francisco, CA 94133, United States -- Rating: 4.5 12:50

Enter message SEND


12:56

Fligthbot

i want to travel 12:53

Where would you like to travel to? 12:53

i would like to go to canada 12:54

You could like Vancouver. Ask me for places to visit there! 12:54

places to visit in vancouver 12:54

Stanley Park -- Do you want more information? 12:54

yes 12:54

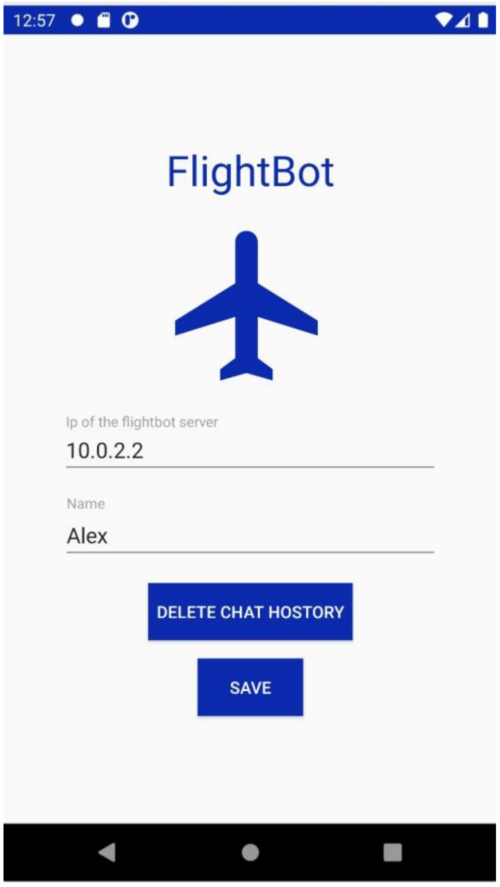
Direction: Vancouver, BC V6G 1Z4, Canada -- Rating: 4.8 12:54

is stanley park open now? 12:55

Stanley Park is open! 12:55

can you give me the website of

Enter message SEND


12:57

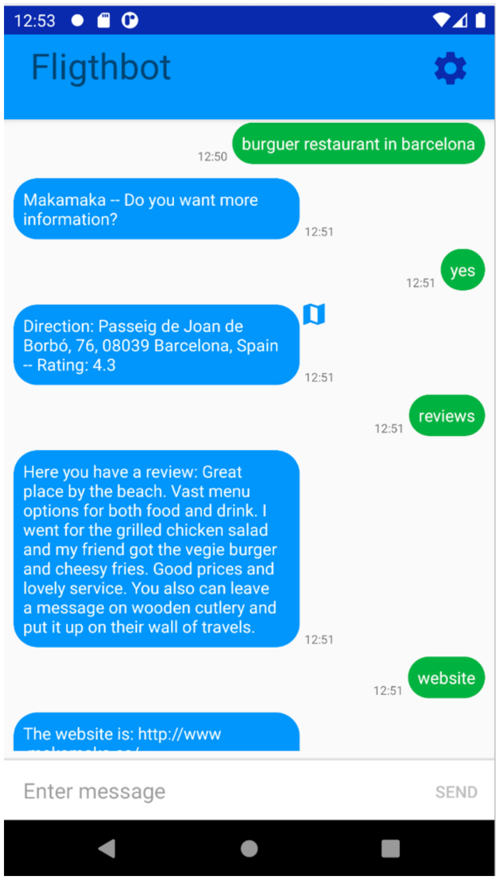
FlightBot

Ip of the flightbot server
10.0.2.2

Name
Alex

DELETE CHAT HOSTORY

SAVE


12:53

Fligthbot

burguer restaurant in barcelona 12:50

Makamaka -- Do you want more information? 12:51

yes 12:51

Direction: Passeig de Joan de Borbó, 76, 08039 Barcelona, Spain -- Rating: 4.3 12:51

reviews 12:51

Here you have a review: Great place by the beach. Vast menu options for both food and drink. I went for the grilled chicken salad and my friend got the vegie burger and cheesy fries. Good prices and lovely service. You also can leave a message on wooden cutlery and put it up on their wall of travels. 12:51

website 12:51

The website is: http://www

Enter message SEND

Experimental design

The chatbot is implemented with the idea that it can be used globally, so it was decided to use English. It also has the advantage of having greater support from dialogflow, so you will better understand requests.

A tour of the chatbot, to check that all features work could be as follows:

- Cities to visit in CONTINENT
- Cities to visit in COUNTRY

- Hotels in X city
- Attractions in X city
- Bars in X city
- Restaurants .... in X city

- More info?
  - o Yes → info
  - o No → other suggestion

- Is X place open? (sometimes, Google doesn't have the information,) "is sagrada familia open?" Works

- Reviews of sagrada familia
- Holy Family Website
- Phone holy family
- Address sagrada familia

- “Show me my historic” will show the places you asked for more information

Final assessment

The flightbot has ended up performing the vast majority of the functionality we had set out to perform.

In the more technical section, its operation is very stable and fast thanks to the backend written in Golang combined with the API of DialogFlow.

Regarding access to the bot service, it is also very easy to use, because by implementing a system where the frontend is completely disconnected from the backend (android - server), it allows us to easily change the frontend of it, and connect any application (such as the one we developed), web or even other services such as Telegram, Slack, Discord or Google Assistant. This facility is due to the fact that our server exposes simple API endpoints, with which any system can connect.

As negative points, the first could be the memory of our system. At first, we had the idea of trying to store each user to thus have a history of the suggestions that the bot had made, and the interests we had detected that this person had. Our server API is ready to receive a username, although it does not use it.

On the other hand, we intended that, once the user showed interest in a recommendation, apart from being able to offer more information about it, we wanted to offer the possibility to show the directions to follow to get to the location. In case of being in another city, we would show the prices and possibilities of flights, and in case of being within a range of distance, the public transport options.

Although we started implementing this functionality, we quickly realized that we did not have enough time to implement it correctly, because to show the flights and directions we had to make use of different APIs, and the overall operation of the program became quite complicated.

Conclusions and future lines

At first, we found that we weren’t very clear about what functionality we wanted our bot to solve for us, until we realized that a place recommender to visit could be a good idea when it comes to finding places. new to travel or visit. This was the initial idea, to focus on recommending cities, towns ... based on some interests you could show or a budget.

As the idea evolved, we realized that we could go a step further to make a more useful version of the bot.

We believe that although some of the features we had in mind, explained in the previous point, we have not been able to develop due to not having enough time to research the necessary APIs and at the same time implement them in our system, the result has been a very solid bot that can solve many doubts, and at all times has a useful answer, thanks to the Google maps API.

As main future lines, it would be to try to complete the functionality of being able to recommend how to get to the location you are interested in, having a basic support of both public transport and aircraft.

Another feature also discussed above, would be to store the information of different users who make use of our application. This functionality would also require carrying out some authentication system to have security. Therefore we should also modify our API, so that you use an API key to make the different requests.

Our last idea of the future line would be to try to implement the bot directly in different services, so that our own application is not necessary. While not very complicated to perform, it takes a lot of configuration to make it work properly with our API that acts as an intermediary between DialogFlow and the service to which we connect it.

In general we believe that with this practice we have learned quite specifically the operation of bots and knowledge systems, first making a very primitive version and programming it ourselves by hand, and then evaluating the different services that currently exist, and that they offer in a very simple way many powerful functionalities.

Bibliography

[1] 4 Reasons for Enterprise Chatbot Failure and How to Overcome Them Using a Multi-Bot Approach. (2020, January 17). Retrieved from https://servisbot.com/four-reasons-for-enterprise-chatbot-failure /

[2] Anand, A. (2017, May 1). The problem with chatbots - how to make them more human? Retrieved from https://chatbotsmagazine.com/the-problem-with-chatbots-how-to-make-them-more-human-d7a24c22f51e

[3] Chatbot: What is a Chatbot? Why are Chatbots Important? (2020, May 7). Retrieved from https://expertsystem.com/chatbot/

[4] Dialogflow. (nd). Retrieved from https://dialogflow.com/

[5] gk_. (2020, January 8). Text Classification using Algorithms. Retrieved from https://chatbotslife.com/text-classification-using-algorithms-e4d50dcba45

[6] Go client library | Dialogflow Documentation | Google Cloud. (nd). Retrieved from https://cloud.google.com/dialogflow/docs/reference/libraries/go

[7] Golang Documentation. (nd). Retrieved from https://golang.org/doc/

[8] Google Maps Documentation. (nd). Retrieved from https://developers.google.com/maps/documentation

[9] John, A. (2020, March 12). Why knowledge bases and chatbots are the future of tech support? Retrieved from https://chatbotsjournal.com/why-knowledge-bases-and-chatbots-are-the-future-of-tech-support-790e238295cc

[10] Pelk, H. (2017, September 21). Machine Learning, Neural Networks and Algorithms. Retrieved from https://chatbotsmagazine.com/machine-learning-neural-networks-and-algorithms-5c0711eb8f9a

[11] reader, ABSPA avid. (2020, May 4). What is Chatbot? Why are Chatbots Important? Retrieved from https://www.revechat.com/blog/what-is-a-chatbot/

[12] Techlabs, M. (2018, December 24). What Are The Inner Workings of a Chatbot? Retrieved from https://chatbotsmagazine.com/what-is-the-working-of-a-chatbot-e99e6996f51c

[13] Tutorials & samples | Dialogflow Documentation | Google Cloud. (nd). Retrieved from https://cloud.google.com/dialogflow/docs/tutorials

[14] What is a Chatbot Knowledge Base ?: Phenompeople. (nd). Retrieved from https://www.phenom.com/blog/what-is-a-chatbot-knowledge-base

[15] What is a Chatbot? (nd). Retrieved from https://www.oracle.com/solutions/chatbots/what-is-a-chatbot/