Intelligence Reasoning Systems, Cognitive Systems

# IRS-CGS-SGEventsFinderChatbot

Singapore Events Finder Chatbot

credit: image from Google Doc template

Authors: team IRS-CGS-SGEventsFinderChatbot

Ng Siew Pheng (A0198525R)

Tarun Rajkumar (A0198522X)

Tea Lee Seng (A0198538J)

Yang Xiaoyan (A0056720L)

## Executive Summary

Singapore is one of the most popular tourist destinations in Asia. Most tourists are exposed to the typical tourist attractions like Sentosa and Zoo. Event Finda provides a unique range of events happening in Singapore, from exhibition for kids to dine & wine events for adults. However, their website is not the easiest to navigate and to find the events that are of interest to visitors.

This chatbot agent is created to introduce these unique events to tourists in a more interactive way. Tourists can ask the chatbot to find them event based on keywords. Based on the keywords provided by the tourists, our chatbot is also capable of recommending other events that the tourists might be interested in, based on their search so far with the chatbot. Even locals can ask our chatbot for event recommendation.

Creation of the dialog intent with a good list of training phrases, has been a never ending task as we test proof the agent. We found that if we keep the training phrase too simple, the DialogFlow is unable to clearly identify the correct intent, often end up triggering the wrong intent. If we added too many entities in a single phrase, Dialogflow can't match the intent and end up in the fallback intent.

The next biggest challenge is the integration to Google assistant. Documentation from both DialogFlow and Actions on Google is not detailed enough. And the lack of documentation for DialogFlow V2 API is also causing us a lot of time in debugging the code, instead of training our chatbot.

Eventually, we managed to get the basics working and able to successfully deploy a working chatbot agent. There are still test scenarios where the chatbot is unable to detect the intent. With more time, we can enhance the agent.

# Introduction

## Objective

The objective of our project is to build a chatbot for Event Finder Singapore, https://www.eventfinda.sg/. The chatbot will answer queries regarding various events happening in Singapore.

## Scope

The queries addressed by the chatbot will be limited to the EventFinda API and WeatherForecast API capability

The chatbot would be integrated to Google Assistant and accessible from any device with Google Assistant. We can talk to it anytime by saying “Talk to Singapore Events Finder”.

## Solution

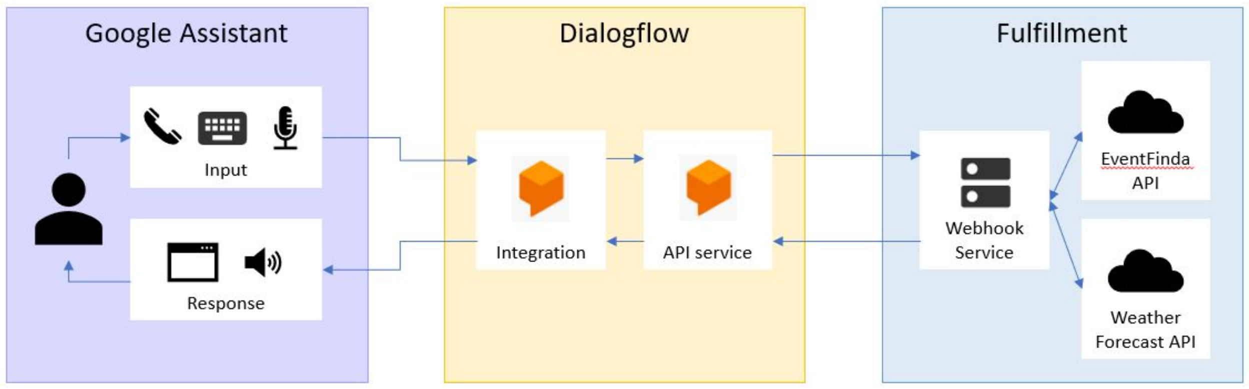
The chatbot will be using the data provided by EventFinda Singapore, through their API. Thus, we have chosen retrieval-based chatbot where it will contain a repository of responses that it uses to solve the queries.

### Overview

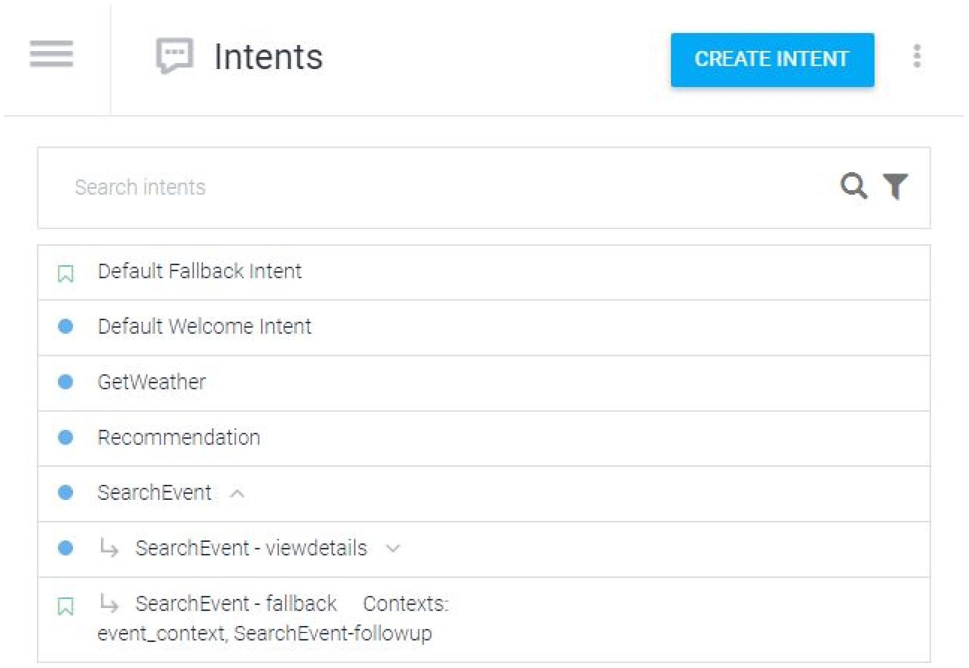
The chatbot accepts input from Google Assistant accessible from various devices, such as smartphones and laptops.

The user inputs (utterances) are processed by Dialogflow engine and matched against defined Intents. Each Intent has Entities which capture the parameters to be processed further.

### Flow Diagram



## Intents



Default Fallback Intent
Default Welcome Intent
GetWeather
Recommendation
SearchEvent
SearchEvent - viewdetails
SearchEvent - fallback Contexts: event_context, SearchEvent-followup

### Default Welcome Intent

The Default Welcome Intent is called whenever the user begins a conversation with the chatbot agent via supported one-click integrations. The Default Welcome Intent can also be matched through its ***training phrases***, which are pre-populated with common greetings like:

a. Hi
b. Hello
c. hey

When the Default Welcome Intent is matched, the agent will respond with one of the pre-populated text responses in the Responses section randomly. There are few responses for welcome messages:

a. Yo! Looking for any event?
b. Hello! Looking for something to do in Singapore? Give me some hints on what you are looking for.
c. Welcome to Event Finder Singapore! How can I help you today?

d. Welcome to Event Finder Singapore! Are you looking for events?

For each subsequent matching, the agent selects a different response until all responses have been used; then, it begins again choosing responses randomly.

## Default Fallback Intent

The Default Fallback Intent is automatically configured by Dialogflow with a variety of static text responses when creating the chatbot agent.  It's a kind of catch-all for any unrecognized user input. This intent is matched when user's input does not match any of the intents (**SearchEvent, SearchEvent – viewdetails, SearchEvent-fallback, Recommendation, GetWeather**) in the chatbot agent.

There are some pre-defined text responses in the Default Fallback Intent. Below are the samples for the fallback responses.

a. Sorry, I didn't get that. At Event Finder Singapore, we are here to help you find the event you are looking for. Are you looking for concert?
b. I missed what you said. At Event Finder Singapore, we are here to help you find the event you are looking for. Are you looking for family friendly event?
c. Sorry, could you say that again?

Just as with the Default Welcome Intent, the agent randomly selects a response variant to send to the user when the Default Fallback Intent is matched, and selects a different response for each subsequent matching until all the responses have been used. Then, the agent will begin again selecting responses at random.

Example question, user ask: what is the GST rate here?
The Default Fallback Intent will respond to one of the pre-defined text responses: "Sorry, can you say that again?"
The user should input something related to weather or events training phrase, then the agent will give the matched intent's response accordingly.

## Search Event Intent

This is main event search intent of the chatbot and is configured with a variety of training phrases to allow the system to match the user input to this intent.

1. Training phrases

   Following are some of the training phrases

   a. looking for concert
   b. attend a talk
   c. would like to watch a concert
   d. any exhibition at Marina Square
   e. I want to go to museum, any event there
   f. what will happen in this week

2. Action and parameters

   In the training phrases, words are tagged as parameter to entity (system or developer) that is defined in this intent.

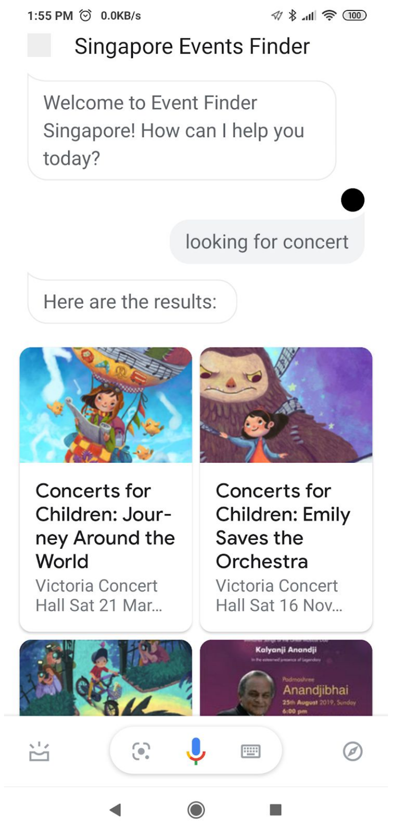
   concert is tagged to a developer defined entity (@sys.any) and named **queryText**.

   this week is tagged to a system duration entity (@sys.duration) and named **duration**.

3. Fulfillment

   Webhook call is enabled for this intent. When this intent is detected, it will make a webhook call to our webhook service, which will use the parameters and call EventFinda Singapore API.

   The list of events from the API call will be loaded into google actions message object (Carousel) and presented to the user, as shown below.


1:55 PM 0.0KB/s

Singapore Events Finder

Welcome to Event Finder Singapore! How can I help you today?

looking for concert

Here are the results:

Concerts for Children: Journey Around the World
Victoria Concert Hall Sat 21 Mar...

Concerts for Children: Emily Saves the Orchestra
Victoria Concert Hall Sat 16 Nov...

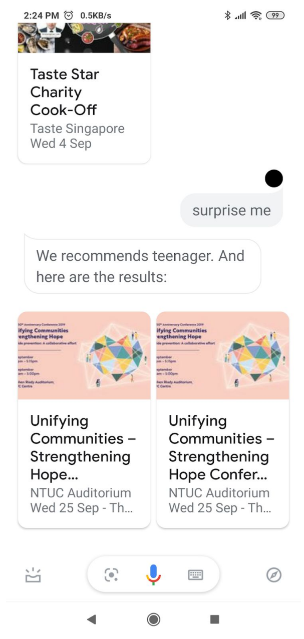
## Follow-up Intent - View Detail Intent

This is a follow-up intent from Search Event intent. It is trigger when user respond with:

- ➔ Clicking one of the cards (google assistant option event)
- ➔ Typing “view details of <event name>” (training phrase “view details for”)

Based on the event (option) selected, the webhook will response with a google basic card message object, containing details of the selected event include the weather forecast on the event date as well, as shown below.

## Recommendation Intent


Taste Star Charity Cook-Off
Taste Singapore
Wed 4 Sep

surprise me

We recommends teenager. And here are the results:

Unifying Communities – Strengthening Hope...
NTUC Auditorium
Wed 25 Sep - Th...

Unifying Communities – Strengthening Hope Confer...
NTUC Auditorium
Wed 25 Sep - Th...

We love surprises. It can be boring to go to events that we kind of expected. Yet, we still don't like surprises that drastically different from our taste. With that in mind, we build a recommendation feature for user A to surprise user B if they are of similar taste.

Let's find out how we do that.

1. We log all user queries along with their chat session id in during all event search queries.

```
insertData(FILE_SESSION_DATA_CSV, session_id, DATATYPE_QUERY_TEXT, queryText)
```

2. We trigger recommendation intent by saying Training phrase like "surprise me"

   List of training phrases:

   a. show me anything
   b. show me something
   c. surprise me
   d. recommends me anything
   e. please recommends

3. The recommendation API will build TF-IDF sparse matrix of all queries under scikit-learns library.

```
def recommends(filename, searchQuery):
    querysDF = pd.read_csv(filename)
    # loading queryText dataType historical data.
    querysDF = querysDF[querysDF['dataType'] == 'queryText']
    querysDF['rawContent'] = querysDF['rawContent'].str.lower()
    all_queries = querysDF.groupby('session_id')['rawContent'].apply(lambda s: "%s" %
','.join(s))

    # build tfidf recommendation sparse matrix.
    tfidf = TfidfVectorizer()
    queries_tfidf = tfidf.fit_transform(all_queries.values)
```

4. The TF-IDF will transform current user queries into another TF-IDF sparse matrix.

```
    df = pd.read_csv(FILE_SESSION_DATA_CSV)
    df_session = df[df['session_id'] == session_id]
    searchQuery = df_session.groupby('session_id')['rawContent'].apply(lambda s: "%s"
% ','.join(s)).values[0].lower()
    qr_sm = tfidf.transform([searchQuery.lower()])
```

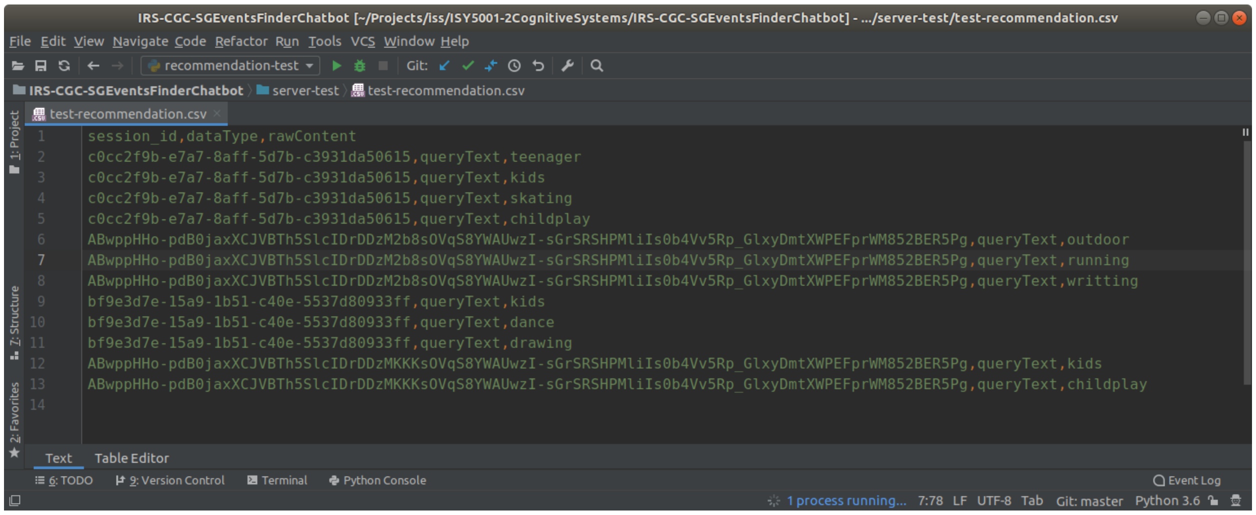
5. We compute cosine similarity and having a list of similarity. Based on the similarity, we picked the more similar queries 1 by 1. Current session queries is always the most similar query, therefore we always get empty set from set.difference(). After excluding queries we searched before, we return 1 unseen query for recommendation.

```
    vals = cosine_similarity(qr_sm, queries_tfidf).argsort()[0][::-1]
    # loop until no empty set from set.difference()
    for i in vals:
        similar_queries = all_queries.values[i]

        # excluded past search keywords.
        possible_recommendations_set =
set(similar_queries.split(',')).difference(set(searchQuery.split(',')))
```

```
        if len(possible_recommendations_set) > 0:
            break

    recommendations = list(possible_recommendations_set)
    # returns 1 recommendations randomly.
    result = random.choice(recommendations)
    return result
```

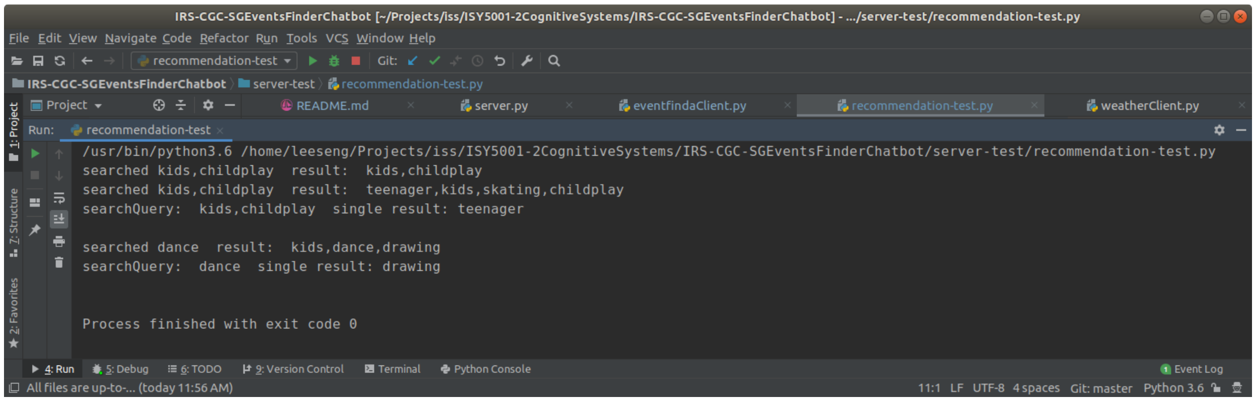
## Recommendation test results:

From the following dataset,



```
session_id,dataType,rawContent
c0cc2f9b-e7a7-8aff-5d7b-c3931da50615,queryText,teenager
c0cc2f9b-e7a7-8aff-5d7b-c3931da50615,queryText,kids
c0cc2f9b-e7a7-8aff-5d7b-c3931da50615,queryText,skating
c0cc2f9b-e7a7-8aff-5d7b-c3931da50615,queryText,childplay
ABwppHHo-pdB0jaxXCJVBTh5SlcIDrDDzM2b8sOVqS8YWAUwzI-sGrSRSHPMliIs0b4Vv5Rp_GlxyDmtXWPEFprWM852BER5Pg,queryText,outdoor
ABwppHHo-pdB0jaxXCJVBTh5SlcIDrDDzM2b8sOVqS8YWAUwzI-sGrSRSHPMliIs0b4Vv5Rp_GlxyDmtXWPEFprWM852BER5Pg,queryText,running
ABwppHHo-pdB0jaxXCJVBTh5SlcIDrDDzM2b8sOVqS8YWAUwzI-sGrSRSHPMliIs0b4Vv5Rp_GlxyDmtXWPEFprWM852BER5Pg,queryText,writting
bf9e3d7e-15a9-1b51-c40e-5537d80933ff,queryText,kids
bf9e3d7e-15a9-1b51-c40e-5537d80933ff,queryText,dance
bf9e3d7e-15a9-1b51-c40e-5537d80933ff,queryText,drawing
ABwppHHo-pdB0jaxXCJVBTh5SlcIDrDDzMKKKsOVqS8YWAUwzI-sGrSRSHPMliIs0b4Vv5Rp_GlxyDmtXWPEFprWM852BER5Pg,queryText,kids
ABwppHHo-pdB0jaxXCJVBTh5SlcIDrDDzMKKKsOVqS8YWAUwzI-sGrSRSHPMliIs0b4Vv5Rp_GlxyDmtXWPEFprWM852BER5Pg,queryText,childplay
```

The recommendation API is able to recommend query from similar tastes based on past searched queries.



```
/usr/bin/python3.6 /home/leeseng/Projects/iss/ISY5001-2CognitiveSystems/IRS-CGC-SGEventsFinderChatbot/server-test/recommendation-test.py
searched kids,childplay  result:  kids,childplay
searched kids,childplay  result:  teenager,kids,skating,childplay
searchQuery:  kids,childplay  single result: teenager

searched dance  result:  kids,dance,drawing
searchQuery:  dance  single result: drawing


Process finished with exit code 0
```

## GetWeather Intent

We want to know the weather before going to any event, or want or know the weather on the event date to do some preparation like bring an umbrella / sunglasses etc  before outing.

GetWeather Intent is configured to give user weather forecast information in Singapore when user wants to know the weather on a specific date. This is to help the user to know the weather before outing.

1. API
   Two APIs are used for the weather forecast which are provided by **Data.gov.sg**
   24 hours weather forecast API for today:
   https://api.data.gov.sg/v1/environment/24-hour-weather-forecast
   4days weather forecast API for the coming 4 days:
   https://api.data.gov.sg/v1/environment/4-day-weather-forecast

2. Training phrase for this intent
   To recognize that user want to know the weather forecast on which day, we give list of training phrases:
   a. how is the temperature on Sunday
   b. how is the weather on Sun 25 Aug
   c. is it cold tomorrow
   d. how is the weather today
   e. is it hot tomorrow
   f. will be rain today
   g. what is the temperature now

3. Action and parameters
   There are two system entities are used in the training phrases provided by Dialogflow, one is @sys.any to match any text related to weather like “weather”, “cold”, “temperature” etc.
   Another one entity is @sys.date matches common date references such as “Sunday”, “today”, “tomorrow” etc.

4. Responses

Webhook call is enabled for this intent. it will make a webhook call to our webhook service when this intent is detected.

We have a simple text response to user which is coded in **weatherClient.py**.

- Get the date from query and compare with today to determine the API to be called, get the json data from API.

```
def getWeatherMessage(weatherdate):
   today = datetime.datetime.today().date()
   deltaDate = weatherdate - today
   if deltaDate.days == 0:
       API_ENDPOINT =
f"https://api.data.gov.sg/v1/environment/24-hour-weather-forecast?date={today}
"
   else:
       API_ENDPOINT =
f"https://api.data.gov.sg/v1/environment/4-day-weather-forecast?date={today}"
   resp = requests.get(API_ENDPOINT)
   fulfillmentMessages = []
   weatherJson = resp.json()
```

- Generate the response message from the json data.

```
   if deltaDate.days == 0:
       deltadays = deltaDate.days
       forecast = weatherJson["items"][deltadays]["general"]
   else:
       deltadays = deltaDate.days - 1
       if deltadays > 3:
           weatherText = f"Sorry there is no weather forecast for
{weatherdate}"
           return fulfillmentMessages, weatherText
       forecast = weatherJson["items"][0]["forecasts"][deltadays]
   temperature = str(forecast["temperature"]["low"]) + "°C~" +
str(forecast["temperature"]["high"]) + "°C"
   wind_speed = str(forecast["wind"]["speed"]["low"]) + "km/h~" +
str(forecast["wind"]["speed"]["high"]) + "km/h"
   message = f"The weather on {weatherdate} is: "
   weatherText = message + forecast["forecast"] + "\n Temperature: " +
temperature + ", Wind speed: " + wind_speed
   return fulfillmentMessages, weatherText
```

When user ask some questions about the weather:

*How is the weather on Sunday?*

The response includes the weather, temperature and wind speed.

The weather on 2019-08-25 is: Thundery Showers Temperature: 26°C~34°C, Wind speed: 10km/h~20km/h

If user ask date is one week later:

*What is the weather on 25 Dec?*

The message will be shown to user:

Sorry there is no weather forecast for 2019-12-25

### Integration with Google Assistant

We faced a lot of challenges during implementing responses in Dialogflow and Google Assistant. Documentation is great but multiple copies of information are confusing. We also encountered in situation like proper responses already trigger zero errors, yet Dialogflow test console output not showing anything, sometimes also no output in Google assistant. The complicated different responses for different integrations, is also quite laborious to work with, as we expect the same card display output should be handled by Dialogflow integration with other messaging client.

Nonetheless, Dialogflow is still a very powerful natural language processing platform. We like the user friendly interface on capturing training phrases and entity slotting capability. Google Assistant UI response is pleasure to interact with, as soon as we can get it working.

## Conclusion

We love how little coding is required to create a chat bot. And a basic set up is enough to trigger the right intent, all by providing training phases to the agent.

Built-in system entities from DialogFlow is very useful, for example, date/duration entity.

However, the documentation on DialogFlow and actions on google (for Google Assistant integration) are not well documented. Entity extraction captures punctuation symbols.

System entities, like location, is not complete for Singapore context.

Eventually, we managed to get the basics working and able to successfully deploy a working chatbot agent. There are still test scenarios where the chatbot is unable to detect the intent. With more time, we can enhance the agent.