

Food Chatbot for Singapore

Sankalp Hegde, Navya Verma, Emmay Koushal

Vellore Institute of Technology, Chennai

Abstract:

In today's fast-paced world, people are looking for faster and more efficient ways to get things done, including ordering food or finding a recipe. With the rise of messaging apps and voice assistants, food chatbots have emerged as a convenient and personalized way to interact with food and restaurant services. A food chatbot is an AI-powered virtual assistant that engages with users in conversation to help them with their food-related queries. It uses natural language processing (NLP) and machine learning (ML) algorithms to understand the user's intent, preferences, and context. It can then provide personalized recommendations and solutions, such as recipe suggestions based on the user's dietary restrictions, or restaurant recommendations based on their location and cuisine preferences. One of the main benefits of food chatbots is their ability to provide quick and accurate responses to users' queries. Users can ask for recommendations, make reservations, or place orders, all from within the chatbot interface. This eliminates the need for users to browse multiple websites or apps, saving them time and effort. Additionally, chatbots can handle multiple conversations simultaneously, making them a cost-effective and scalable solution for businesses. Food chatbots also have the potential to generate valuable insights and data for businesses. By analyzing user preferences and behavior patterns, businesses can gain a deeper understanding of their customers and tailor their offerings accordingly. For example, if a chatbot notices that a user frequently asks for vegetarian recipes, the business can create more vegetarian options to cater to that audience. The use of food chatbots is not limited to restaurants and food delivery services. They can also be used by food brands to engage with their customers and promote their products. For example, a chatbot can suggest recipes that incorporate a particular brand's products or provide information about the ingredients used in their products.

In conclusion, food chatbots represent a significant development in the food industry, offering convenience, personalization, and cost-effectiveness. As the technology continues to evolve, we can expect to see more sophisticated chatbots that can handle even more complex queries and provide even more personalized recommendations.

I. INTRODUCTION

In this project, we aim to develop a food chatbot for Singapore that provides accurate, relevant, and engaging recommendations for local cuisine. The chatbot will be designed to understand user input through text & voice and provide personalized recommendations based on dietary preferences, restrictions, and location of the user. It will also integrate with external databases and APIs to provide up-to-date information on restaurants and dishes. The user interface will be designed to be engaging and interactive, with the ability to provide feedback and ratings on recommendations of the chatbot. The chatbot will also be evaluated using user surveys to ensure that it is meeting user needs and providing accurate and relevant recommendations. Overall, this project has the potential to improve the dining

experience for residents and tourists in Singapore, providing personalized and relevant recommendations for local cuisine.

II. LITERATURE SURVEY

A literature survey on food chatbots reveals a growing interest in the potential of these AI-powered virtual assistants to improve the food industry's customer experience. The following is a summary of some of the key findings in the existing research on food chatbots.

In a study published in the Journal of Foodservice Business Research, researchers found that food chatbots can improve customer engagement and satisfaction. The study involved a survey of 356 restaurant customers who interacted with a chatbot to place their orders[1]. The results showed that customers who used the chatbot reported higher levels of satisfaction and perceived the ordering process to be more efficient compared to those who ordered through traditional methods.

Another study published in the International Journal of Hospitality Management explored the potential of chatbots to personalize the customer experience. The study involved a chatbot that used AI and NLP to recommend menu items based on customer preferences and feedback[2]. The results showed that customers who interacted with the chatbot reported higher levels of satisfaction and perceived the recommendations to be more relevant to their tastes

In a review article published in the International Journal of Information Management, researchers explored the role of chatbots in the food industry's digital transformation. The article identified several potential applications for chatbots, including customer service, order processing, and marketing [3]. The authors concluded that chatbots can be an effective tool for businesses to improve customer engagement and reduce costs.

A study published in the Journal of Hospitality and Tourism Technology examined the factors that influence customers' acceptance of food chatbots. The study involved a survey of 300 restaurant customers who interacted with a chatbot to place their orders[4]. The results showed that customers' willingness to use a chatbot was influenced by their perceived usefulness, ease of use, and trust in the technology.

In a review article published in the Journal of Computer Science and Applications, researchers explored the various algorithms used in chatbot development. The article identified several key algorithms, including pattern matching, natural language processing (NLP), and machine learning[5]. The authors concluded that a combination of these algorithms can be used to develop effective and efficient chatbots.

Overall, the literature survey suggests that food chatbots have the potential to improve customer engagement and satisfaction, personalize the customer experience, reduce costs, and improve customer loyalty. Chatbot algorithms can be effective in improving customer service, reducing workload on human operators, and enhancing the user experience.

III. SUMMARY

The food bot that was proposed and implemented worked as a successful cognitive model that can take in user input and make decisions. The decision making was based on intents and entities - coded as a set of questions and possible responses, based on which the cognitive model will make a decision. The bot's backend consisted of getting information from yelp.com as well as a csv file containing food names and their descriptions. The front end was built over rest api using dialog flow web demo options integrated into a html file working on a live local server. The bot has options to get restaurant details(including phone number and site hyperlinks), suggest food options, describe food items, suggest nearby food hawker centers, etc. The bot uses a flask webhook using ngrok to connect to the backend. Voice recognition was also implemented so users can use a more hands-free approach to interact with the chatbot. Overall, the foodbot works well and gives the users ample options for choosing a good place to dine in. It is also easy to use, and asks questions to get user opinions in each step to choose a place better suited to their needs.

IV. PROPOSED MODULES

User Input Processing: This module will be responsible for understanding user input, including natural language processing, sentiment analysis, and entity recognition. Input can be text query or a voice query.

Recommendation Generation: Based on user input and preferences, this module will generate personalized recommendations for restaurants and dishes using backend recommendation systems.

Database Integration: The chatbot will need to integrate with external databases like yelp and dialog flow APIs to provide up-to-date information on restaurants, dishes, and nutritional information. This gives real time suggestions.

Location-Based Services: This module will use location data to provide recommendations for nearby restaurants and dishes. This will also help to suggest local dishes.

Dietary Restriction Management: This module will handle user dietary restrictions, such as allergies or vegetarian/vegan preferences, and generate recommendations based on these restrictions.

User Feedback and Ratings: The chatbot will allow users to provide feedback and ratings on recommendations, which will be used to improve future recommendations.

Multilingual Support: If the chatbot is intended for use in multilingual environments, this module will enable the chatbot to understand and respond to queries in multiple languages.

User Authentication: If the chatbot is intended for use by registered users, this module will manage user authentication and data privacy.

V. PROPOSED ARCHITECTURE

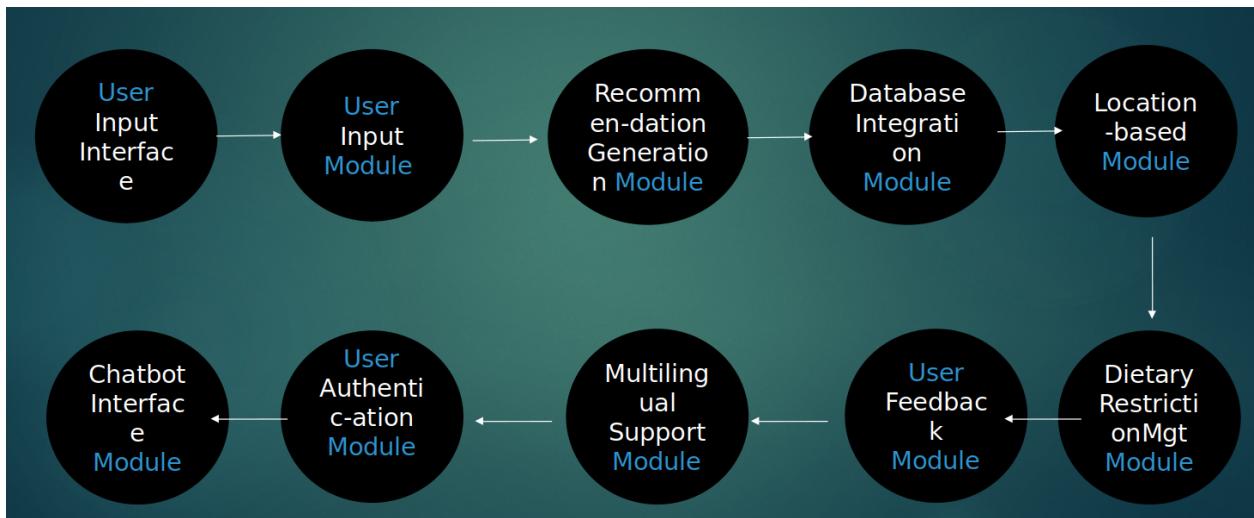


fig.1

The architecture of this model is as shown in the fig. 1. One the first we have a user interactive UI which has a text box and mic to provide queries for the chatbot. As the input query is given through the input module, that query is processed by Recommendation Generation Module which is integrated with the backend Database which has data like different food items in singapore, ratings of the food items of users, location based dishes and Dietary Restriction of user. Based on this data and user query the best solution and recommendation is provided. We also have Multilingual Support which can help user in different languages. We also have a User Authentication Module which Authenticates the user details with the backend database. Then this output will be displayed on the UI of the user.

VI. CODING

IntentGetDiningInfo.py

```
IntentGetDiningInfo.py > restaurantIntentHandler
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Created on Wed Aug 21 14:03:15 2019
5
6  @author: meiying
7
8  """
9
10 from flask import make_response, jsonify
11 from yelpAPIHelper import yelp_request
12 from richMessageHelper import displayResults_slack
13 import random
14
15 YELP_API_KEY = "rG5T0rDyCq0G-lIelg9XzKfBcSNrc2F7za3C99Nray3q_-Wz8YU1Jdi1rAu7-gSQwdKCuZA0b9GXcp5xMImW9_dxQo_9
16 API_HOST = 'https://api.yelp.com'
17 SEARCH_PATH = '/v3/businesses/search'
18 BUSINESS_PATH = '/v3/businesses/'
19
20 DEBUG_MODE = False
21 public_url = ""
22
23 def getPopularDiningInfoIntentHandler(PARAMETERS):
24     business_data = yelp_request(API_HOST, SEARCH_PATH, YELP_API_KEY, PARAMETERS, debug=DEBUG_MODE)
25     biz_array = []
26     if business_data["total"] > 0:
27         for biz in business_data["businesses"]:
28             biz_array.append({
29                 "name": biz.get("name", ""),
30                 "phone": biz.get("phone", "N/A"),
31                 "address": ", ".join(biz["location"]["display_address"]),
32                 "price_range": biz.get("price", "N/A"),
33                 "category": ", ".join(map(lambda x: x["alias"], biz["categories"])),
34                 "website": biz.get("url", "N/A"),
35                 "image": biz.get("image_url", "N/A"),
36                 "review_count": biz.get("review_count", "N/A"),
37                 "rating": biz.get("rating", "N/A"),
38             })
39     return biz_array
40
41 def processPopularDiningIntent(results, header, pre_header_msg = None):
42
43     if len(results) > 0:
44         rich_messages = displayResults_slack(results, public_url, header, default_header_msg = None, use_is_p
45     else:
46         rich_messages = {"fulfillmentText": "There are no results. Please try another search."}
47     return make_response(jsonify(rich_messages))
48
49 def hawkerCentreIntentHandler(req, url):
50     global public_url
51     public_url = url
52
53     for context in req["queryResult"]["outputContexts"]:
54         if "session var" in context["name"]:
55             limit = int(context["parameters"].get("number", None)) # must be integer
56             break
57     if limit == None:
58         limit = int(req["queryResult"]["parameters"].get("number", None))
59     PARAMETERS = {"term": "hawker centre", "limit": limit, "sort_by": "rating", "price":1, "location": "Singap
60     results = getPopularDiningInfoIntentHandler(PARAMETERS)
61     emojiArr = ["🍽️", "🍎", "🌍", "👉", "☀️", "😊", "☀️"]
62     emoji = random.choice(emojiArr)
63     resultText = "Here are some of the most popular hawker centres in Singapore!" + emoji
64     return processPopularDiningIntent(results, resultText)
```

```

66 def restaurantIntentHandler(req, url):
67     global public_url
68     public_url = url
69
70     budgetDict = {"moderatelypriced": 2, "somewhatexpensive": 3, "expensive": 4}
71     # dining = None
72     cuisine = None
73     budget = None
74     limit = None
75
76     for context in req["queryResult"]["outputContexts"]:
77         if "session_var" in context["name"]:
78             cuisine = context["parameters"].get("cuisine", None)
79             budget = context["parameters"].get("budget", None)
80             limit = int(context["parameters"].get("number", None))
81             break
82     if cuisine == None:
83         cuisine = req["queryResult"]["parameters"].get("cuisine", None)
84     if budget == None:
85         budget = req["queryResult"]["parameters"].get("budget", None)
86     if limit == None:
87         limit = int(req["queryResult"]["parameters"].get("number", None))
88
89     PARAMETERS = {"term": "restaurant", "categories": cuisine, "limit": limit, "sort_by": "rating", "price": budgetDict[budget.replace(" ", "")]
90     results = getPopularDiningInfoIntentHandler(PARAMETERS)
91     #print(results)
92     emojiArr = ["🍽️", "❓", "❓", "❓", "❓", "❓", "❓"]
93     emoji = random.choice(emojiArr)
94     resultText = "Here's a list of popular restaurants in Singapore!" + emoji
95     return processPopularDiningIntent(results, resultText)
96
97 def foodItemIntentHandler(req, foodItem, url, limit = 5, pre_header_msg = None):
98     global public_url
99     public_url = url
100
101     PARAMETERS = {"term": foodItem, "limit": limit, "sort_by": "best_match", "location": "Singapore", "is_closed": False}
102     results = getPopularDiningInfoIntentHandler(PARAMETERS)
103     return processPopularDiningIntent(results, "*Popular eateries selling " + foodItem + "*", pre_header_msg= pre_header_msg)
104

```

IntentHawkerInfo.py

```

1  from flask import Flask, request, make_response, jsonify
2  import requests
3  import json
4
5  from lxml import html
6  import urllib.parse as urllib
7  from urllib.error import HTTPError, URLError
8  from urllib.parse import quote
9  from urllib.parse import urlencode
10
11 DEBUG_MODE = False
12
13
14 # *****
15 # Hawker Intent Handlers funcs :
16 # *****
17
18 def getHawkerInfoIntentHandler(url):
19     try:
20         page = requests.get(url)
21     except HTTPError as e:
22         print(e)
23         return None
24     except URLError as e:
25         print('The server could not be found!')
26
27     try:
28         parser = html.fromstring(page.content)
29     except AttributeError as e:
30         return None
31
32     theXPath1 = '//h2//text()' # get text for the THE 5 BEST HAWKER CENTRES IN SINGAPORE
33     resultList1 = parser.xpath(theXPath1)
34     while ' ' in resultList1:
35         resultList1.remove(' ')
36
37     theXPath2 = '//h2/a/@href' # get the link
38     resultList2 = parser.xpath(theXPath2)
39

```

```

41     theXPath3 = '//*[@id="content"]//ul[2]/li[1]/p/text()' # get the highlights
42     resultList3 = parser.xpath(theXPath3)
43
44     return resultList1, resultList2, resultList3
45
46 def processHawkerInfoIntent(req):
47     intent_name = req["queryResult"]["intent"]["displayName"]
48     # Answer = req["queryResult"]["parameters"]["Answer"]
49     if (intent_name == "Hawker Info-yes"):
50         url = 'https://www.thebestsingapore.com/eat-and-drink/the-best-5-hawker-centres-in-singapore/'
51         nameList, urlList, highlights = getHawkerInfoIntentHandler(url)
52         if nameList == None:
53             fulfillmentMessage = [
54                 { "text": {
55                     "text": ["Error processing information ... Please try again"]}
56                 }
57             ]
58         else:
59             fulfillmentMessage = []
60             num = 1
61             for i in range(0, len(nameList)):
62                 # defaultPayload = "DEFAULT PAYLOAD"
63                 fulfillmentMessage.append({
64                     "card": {
65                         "title": nameList[i],
66                         "subtitle": highlights[i],
67                         "imageUri": "",
68                         "buttons": [{"text": "Go to site","postback": urlList[i]}]
69                     })
70             num = num + 1
71
72             fulfillmentMessage.append({
73                 "quickReplies": {
74                     "title": "Choose one:",
75                     "quickReplies": [
76                         "Get Restaurant Info",
77                         "Hawker Info",
78                         "Top 50 Restaurants"
79                     ]
80                 },
81                 "platform": "slack",
82                 "platform": "facebook",
83                 "platform": "ACTIONS_ON_GOOGLE"
84             })
85     elif (intent_name == "Hawker Info-no"):
86         if DEBUG_MODE: print("User has cancelled... Redirect to welcome")
87         followupEvent = {"name": "WELCOME"}
88

```

```
89     return make_response(jsonify({
90         "followupEventInput": followupEvent
91     }))
92
93     # fulfillmentMessage = [
94     #     { "quickReplies": {
95     #         "title": "Choose one:",
96     #         "quickReplies": [
97     #             "Get Restaurant Info",
98     #             "Hawker Info",
99     #             "Top 50 Restaurants"
100    #         ]
101   #     },
102   #     "platform": "facebook",
103   #     "platform": "slack",
104   #     "platform": "ACTIONS_ON_GOOGLE"
105  # }
106 # ]
107 else:
108     return make_response(jsonify({
109         "fulfillmentMessages": [
110             {
111                 "quickReplies": {
112                     "title": ["Would you like to know which are the best Hawkers in Singapore?"],
113                     "quickReplies": [
114                         "Yes", "No"
115                     ]
116                 }
117             }
118         ]
119     }))
120
121 RichMessages = {
122     # "fulfillmentText": defaultPayload,
123     "fulfillmentMessages": fulfillmentMessage
124
125 }
126
127 return make_response(jsonify(RichMessages))
128
```

IntentGetRestaurentInfo.py

```
1  from flask import Flask, request, make_response, jsonify
2  import requests
3  import json
4
5  from lxml import html
6  import re
7  import urllib.parse as urllib
8  from urllib.error import HTTPError
9  from urllib.parse import quote
10 from urllib.parse import urlencode
11 import random
12 # import pprint
13
14 from yelpAPIHelper import yelp_query_api
15 from richMessageHelper import displayResults_slack
16 DEFAULT_LOCATION = "Singapore"
17 DEBUG_MODE = False
18 public_url = ""
19
20
21 # *****
22 # UTIL FUNCTIONS : END
23 # *****
24 def validateParameters(params):
25     returnText = []
26     followupEvent = {}
27     passRestaurantName = True
28
29     RestaurantName = params["restaurantName"]
30     if (RestaurantName == "" or RestaurantName.lower() in ["hi", "hello", "how are you"]):
31         restNameArray = ["Please enter a restaurant", "Which restaurant would that be?", "Which restaurant are you interested in?", "Could you tell me the name of the restaur"]
32         returnText.append(random.choice(restNameArray))
33         passRestaurantName = False
34
35     # Invoke this event if it fails restaurant-name check
36     # TODO
37     followupEvent = {
38
39     }
40
41     if (passRestaurantName):
42         Location = params.get("location", DEFAULT_LOCATION)
43         Country = Location.get("country", None) if isinstance(Location, dict) else None
44         if (Location == "" or (Country != None and Country != "" and Country != "Singapore")):
45             # Invoke this event if it fails location check
46             returnText.append("Redirecting to GetRestInfo-Location")
47             if DEBUG_MODE: print("... Redirecting to GetRestInfo-Location")
```

```
47     if DEBUG_MODE: print("... Redirecting to GetRestInfo-Location")
48     followupEvent = {
49         "name": "actions_intent_restaurantInfo_getLocation",
50         "parameters": {
51             "restaurantName": RestaurantName
52         }
53     }
54
55     if (len(returnText) == 0): return None
56     else:
57         return {
58             "fulfillmentMessages": [
59                 {
60                     "text" : {
61                         "text": returnText # Note that this will not be used if there is followupEventInput
62                     }
63                 ],
64             "followupEventInput": followupEvent
65         }
66
67 def processRestaurantInfoConfirmIntent(req):
68     p = req["queryResult"]["parameters"]
69     RestaurantName = p.get("restaurantName")
70     Location = p.get("location", DEFAULT_LOCATION)
71     address = getLocationAsString(Location)
72
73     followupEvent = {}
74     returnText = []
75
76     intent_name = req["queryResult"]["intent"]["displayName"]
77
78     if (intent_name == "GetRestInfo-Confirm"):
79         if DEBUG_MODE: print("All parameters filled... We will seek user's confirmation")
80         confirmArr = [
81             "Okay great. I will get the top results for " + RestaurantName + " in " + address + "?",
82             "Okay. I will get you details on " + RestaurantName + " in " + address + ". Is that right?",
83             "Got it. " + RestaurantName + " in " + address + "?"
84         ]
85         returnText.append(random.choice(confirmArr))
86
87     return make_response(jsonify({
88         "fulfillmentMessages": [
89             {
90                 "quickReplies": {
91                     "title": returnText,
92                     "quickReplies": [
93                         "Yes", "No"
94                     ]
95                 }
96             }
97         ]
98     })
```

```

98     # If it's yes, we forward back to restaurntInfo with all entities filled
99     elif (intent_name == "GetRestInfo-Confirm-yes"):
100         if DEBUG_MODE: print("User has confirmed... Get results for user now")
101         # By this stage, everything should be filled up. Just forward back to restaurantInfo intent
102         followupEvent = {
103             "name": "actions_intent_restaurantInfo_execute",
104             "parameters": {
105                 "restaurantName": RestaurantName,
106                 "location": Location
107             }
108         }
109
110         return make_response(jsonify({
111             "followupEventInput": followupEvent
112         }))
113
114     # If it's no, we forward back to our default welcome
115     else:
116         if DEBUG_MODE: print("User has cancelled... Redirect to welcome")
117         followupEvent = {"name": "WELCOME"}
118         return make_response(jsonify({
119             "followupEventInput": followupEvent
120         }))
121
122 def processRestaurantInfoLocationIntent(req):
123     followupEvent = {}
124     returnText = []
125
126
127     # Slot filling check for required parameters
128     p = req["queryResult"]["parameters"]
129     RestaurantName = p.get("restaurantName")
130     bNoPreferredLocation = p.get("RestaurantNoPreferredLocation", "false").lower()
131
132     if (bNoPreferredLocation in ['true', 'yes', '1']):
133         # Passed location check, redirect to next intent
134         if (DEBUG_MODE): print("Passed location check (No preferred Location). Redirect to GetRestInfo-Confirm")
135         followupEvent = {
136             "name": "actions_intent_restaurantInfo_confirm",
137             "parameters": {
138                 "restaurantName": RestaurantName,
139                 "location": DEFAULT_LOCATION
140             }
141         }

```

```

142     else:
143         # Slot validation check for Location
144         Location = p.get("location", DEFAULT_LOCATION)
145         Country = Location.get("country", None) if isinstance(Location, dict) else None
146         if (Location == ""):
147             locationArray = [
148                 "Okay, " + RestaurantName + " at which location in Singapore?",
149                 "Sure. I will get you more details on " + RestaurantName + ". Any preferred location?",
150                 "Alright. Let's get details on " + RestaurantName + ". Any preferred location?"
151             ]
152             returnText.append(random.choice(locationArray)) # Failed validation - Ask user to try again
153
154     elif (Country != None and Country != "" and Country != "Singapore"):
155         countryArray = [
156             "I'm sorry, we cater only to food found in Singapore and not " + Country,
157             "Oops! " + Country + " is not supported!"
158         ]
159         returnText.append(random.choice(countryArray) + ". Please enter a valid location in Singapore") # Failed validation - Ask user to try again
160     else:
161         # Passed location check, redirect to next intent
162         if (DEBUG_MODE): print("Passed location check. Redirect to GetRestInfo-Confirm")
163         followupEvent = {
164             "name": "actions_intent_restaurantInfo_confirm",
165             "parameters": {
166                 "restaurantName": RestaurantName,
167                 "location": Location
168             }
169         }
170
171     return make_response(jsonify({
172         "fulfillmentMessages": [
173             {
174                 "text": {
175                     "text": returnText # Note that this will not be used if there is followupEventInput
176                 }
177             }
178         ],
179         "followupEventInput": followupEvent
180     }))
181
182 def processRestaurantInfoEntryIntent(req):
183     followupEvent = {}
184
185     # Slot filling check for required parameters
186     invalid = validateParameters(req["queryResult"]["parameters"])
187     if (invalid != None):
188         return make_response(jsonify(invalid))
189
190     # Slot filling passed
191     RestaurantNameRaw = req["queryResult"]["parameters"]["restaurantName"]
192     Location = req["queryResult"]["parameters"].get("location")
193     if (Location == None or Location == ""): Location = DEFAULT_LOCATION
194
195     # address = getLocationAsString(Location)
196
197     # Passed slot check, redirect to next intent
198     if (DEBUG_MODE): print("Passed location check. Redirect to GetRestInfo-Confirm")
199     followupEvent = {
200         "name": "actions_intent_restaurantInfo_confirm",
201         "parameters": {
202             "restaurantName": RestaurantNameRaw,
203             "location": Location
204         }
205     }
206
207     RichMessages = {
208         "followupEventInput": followupEvent
209     }
210     return make_response(jsonify(RichMessages))
211
212 def processRestaurantInfoExeIntent(req):
213
214     # Slot filling passed
215     RestaurantNameRaw = req["queryResult"]["parameters"]["restaurantName"]
216     Location = req["queryResult"]["parameters"].get("location")
217     if (Location == None or Location == ""): Location = DEFAULT_LOCATION
218
219     address = getLocationAsString(Location)
220     RestaurantName = RestaurantNameRaw.replace(" ", "-")
221     # RestaurantName = ''.join(e for e in RestaurantNameRaw if e.isalnum()) + "-Singapore" # remove non-alnum
222     results = getRestaurantInfoIntentHandler(RestaurantName, address, 1 if address != DEFAULT_LOCATION else 5)
223
224     if (results == None):
225         fulfillmentMessage = [
226             {
227                 "text": {
228                     "text": ["No businesses for " + RestaurantNameRaw + " in " + address + " found."]
229                 }
230             }
231         ]
232         RichMessages = {
233             "fulfillmentMessages": fulfillmentMessage
234         }
235     elif (isinstance(results, int)):
236         # Error code. 1 - Exceeded daily-limits calls
237         if (results == 1): fulText = "It appears that our magic server is overloaded. Please try again tomorrow"
238         else: fulText = "We have ran into an unknown problem. Please try again later."

```

```

237     else: fullText = "We have ran into an unknown problem. Please try again later."
238     RichMessages = {
239         "fulfillmentMessages": [
240             { "text": {
241                 "text": [fullText]
242             }
243         }
244     ]
245 }
246
247 else:
248     default_msg = "Here are your results for " + RestaurantNameRaw + " located in " + address
249
250     magArr = ["⚡Presto!", "⚡Abracadabra!", "⚡Swish!", "⚡Accio!"]
251     mag = random.choice(magArr)
252     emojiArr = ["🍔", "🍟", "🍕", "🌭", "🌖", "🌗", "🌖"]
253     emoji = random.choice(emojiArr)
254     resultText = mag + " Here are your results for *" + RestaurantNameRaw + "* located in *" + address + "* " + emoji
255
256     # check if results contain businesses in similar categories
257     hasOthers = not all(list(map(lambda x: RestaurantNameRaw.lower() in x["name"].lower(), results))[1:])
258     if (hasOthers):
259         resultText = resultText + "\nI have also included restaurants in similar categories."
260
261     RichMessages = displayResults_slack(results, public_url, resultText, default_msg)
262
263 # Return the Restaurant Information
264 return make_response(jsonify(RichMessages))
265
266
267 def processRestaurantInfoIntents(req, url):
268     global public_url
269     public_url = url
270
271     intent_name = req["queryResult"]["intent"]["displayName"]
272     if DEBUG_MODE: print("!!!! " + intent_name)
273
274     if intent_name == "GetRestInfo-Entry":
275         return processRestaurantInfoEntryIntent(req)
276     elif intent_name == "GetRestInfo-Exe":
277         return processRestaurantInfoExeIntent(req)
278     elif intent_name == "GetRestInfo-Location":
279         return processRestaurantInfoLocationIntent(req)
280     elif "GetRestInfo-Confirm" in intent_name:
281         return processRestaurantInfoConfirmIntent(req)
282

```

```

285     def getRestaurantInfoIntentHandler(restaurantName, location=DEFAULT_LOCATION, num_results=5):
286         """
287             Use Yelp APIs to get restaurant details
288         """
289
290         try:
291             templ = yelp_query_api(restaurantName, location, num_results=num_results, debug=DEBUG_MODE)
292         except HTTPError as error:
293             if (DEBUG_MODE): print("HTTP ERROR! " + error.code)
294             return None
295
296         if (templ == None): return None
297         elif (isinstance(templ, int)):
298             return templ
299         elif (isinstance(templ, list)):
300             query_result_array = []
301             for t in templ:
302                 query_result_array.append({
303                     "hours": "Open Now" if t.get("hours", None) != None and t["hours"][0]["is_open_now"] else "Closed",
304                     "name": t.get("name", ""),
305                     "phone": t.get("phone", "N/A"),
306                     "address": ", ".join(t["location"]["display_address"]),
307                     "price_range": t.get("price", "N/A"),
308                     "category": ", ".join(map(lambda x: x["alias"], t["categories"])),
309                     "website": t.get("url", "N/A"),
310                     "image": t.get("image_url", ""),
311                     "review_count": t.get("review_count", "N/A"),
312                     "rating": t.get("rating", "N/A"),
313                     "photo": t["photos"][0] if (t.get("photos", None) != None and len(t["photos"])>=1) else None
314                 })
315
316         # return infoCount, website, restName, address, phone, category, price_range, officeHours, take_out, delivery, reservation
317         return query_result_array
318
319     def getLocationAsString(Location):
320         address = Location if isinstance(Location, str) else Location.get("street-address")
321         if (address == "" or address == None): address = Location.get("city")
322         if (address == "" or address == None): address = Location.get("country")
323         if (address == "" or address == None): address = Location.get("business-name")
324         if (address == "" or address == None): address = DEFAULT_LOCATION
325         if (address.lower() in ["anywhere", "anything"]): address = DEFAULT_LOCATION      # quick-fix for bug
326         return address
327
328

```

IntentWhatis.py

```
1  from flask import make_response, jsonify
2  import csv
3  import random
4  from os import getcwd, chdir
5  from os.path import abspath, dirname, join, realpath, isfile
6  from IntentGetDiningInfo import foodItemIntentHandler
7
8  # *****
9  # UTIL FUNCTIONS : END
10 # *****
11 global foodDic
12 foodDic = {}
13
14 def initiateLookUpTable():
15     global foodDic
16
17     dir_path = dirname(realpath(__file__))
18     chdir(dir_path)
19
20     with open('_FoodCat.csv', mode='r', encoding="utf-8-sig") as infile:
21         reader = csv.reader(infile)
22         for row in reader:
23             foodDic[row[0].strip().lower()] = row[1]
24
25
26 def whatisIntentHandler(req, public_url):
27     returnText = []
28     foodItem = req["queryResult"]["parameters"].get("foodItem", None)
29
30     # lazy initialisation of foodDic
31     if (not bool(foodDic)): initiateLookUpTable()
32
33     # get food description from foodDic
34     foodDescription = foodDic.get(foodItem.strip().lower(), None)
35     if (foodDescription == None):
36         dunnoArray = [
37             "Hmm.. I am not sure what " + foodItem + " is too!",
38             "Oops! I don't know what " + foodItem + " is too!"
39         ]
40         returnText = [random.choice(dunnoArray)]
41     return make_response(jsonify({
42         "fulfillmentMessages": [
43             {
44                 "text" : {
45                     "text": returnText
46                 }
47             }
48         ]
49     }))
50
51
52     else:
53         returnText = "*" + foodItem.strip().capitalize() + "*"
54         returnText = returnText + "\n" + foodDescription
55         return foodItemIntentHandler(req, foodItem, public_url, pre_header_msg = returnText)
56
```

restaurentsIntent.py

```
0  from flask import make_response, jsonify
1  from yelpAPIHelper import yelp_request
2  from richMessageHelper import displayResults_slack
3
4  YELP_API_KEY = "rG5TOrDyCq0G-lIelg9XzKfBcSNrc2F7zsa3C99Nray3q_-Wz8YU1JdilrAu7-g5QwdKCuZA0b9GXCP5xMImW9_dxQo_9ib40AJ-PXRyqPGakfQD8WHL8BX7uDNJXXYx"
5  API_HOST = 'https://api.yelp.com'
6  SEARCH_PATH = '/v3/businesses/search'
7  BUSINESS_PATH = '/v3/businesses/'
8
9  DEBUG_MODE = False
10 public_url = ""
11
12 def getPopularDiningIntentHandler(PARAMETERS):
13     business_data = yelp_request(API_HOST, SEARCH_PATH, YELP_API_KEY, PARAMETERS, debug=DEBUG_MODE)
14     biz_array = []
15     if business_data["total"] > 0:
16         for biz in business_data["businesses"]:
17             #bizDict["hours"].append("Open Now" if biz.get("hours", None) != None and biz["hours"][0]["is_open_now"] else "Closed")
18             biz_array.append({
19                 "name": biz.get("name", ""),
20                 "phone": biz.get("phone", "N/A"),
21                 "address": ", ".join(biz["location"]["display_address"]),
22                 "price_range": biz.get("price", "N/A"),
23                 "category": ", ".join(map(lambda x: x["alias"], biz["categories"])),
24                 "website": biz.get("url", "N/A"),
25                 "image": biz.get("image_url", "N/A"),
26                 "review_count": biz.get("review count", "N/A"),
27                 "rating": biz.get("rating", "N/A"),
28                 "is_closed": biz.get("is_closed", "")
29             })
30             #bizDict["photo"].append(biz["photos"][0] if (biz.get("photos", None) != None and len(biz["photos"])>=1) else None)
31     print(biz_array)
32     return biz_array
33
34 def processPopularDiningIntent(dining, PARAMETERS, url):
35     # def response(dining, PARAMETERS):_
36     global public_url
37     public_url = url
38
39     results = getPopularDiningIntentHandler(PARAMETERS)
40     #print(results)
41     if len(results) > 0:
42         rich_messages = displayResults_slack(results, public_url, "Popular " + dining + "s", default_header_msg = None, use_is_closed=True)
43     else:
44         rich_messages = {"fulfillmentText": "no results were found."}
45     return make_response(jsonify(rich_messages))
```

RichMessageHelper.py

```
1 import urllib.parse as urllib
2 import random
3
4 def displayWelcome_slack(public_url, default_header_msg = None, additional_header = None):
5     fulfillmentMessage = []
6     slackBlocks = []
7
8     # Components of introduction message
9     introArr = [
10         "Hello I am Olive, your personal foodBot for Singapore food! 😊",
11         "Hello there! I am Olive, the foodBot for Singapore food 😊",
12         "👋 I am Olive. How can I help you today? 😊",
13     ]
14     intro = random.choice(introArr) if not additional_header else additional_header
15
16     eg1 = "Find popular eateries"
17     eg2 = random.choice(["Din Tai Fung in Orchard", "Swensens in Clementi"])
18     eg3 = random.choice(["What is Laksar", "What is Claypot Rice?"])
19
20     intro2 = "* Recommend a dining place. (eg _ss_)\n* Get a specific restaurant's details such as its location and rating. (eg _%s_)\n* Tell you about specific local food in Singapore. (eg _%s_)\n* (%s1, %s2, %s3)" % (intro, eg1, eg2, eg3)
21
22     body2 = "Or just click one of the buttons below:"
23
24     header_msg1 = intro + "\n\nHere's a couple of the things I can do:\n" + intro2
25     header_msg = header_msg1 + "\n\n" + body2
26
27     # Show header message with a varying image url from 1-9
28     img_num = random.randint(1, 9)
29     image_url = "https://www.secretfoodtours.com/images/singapore/singapore-tours-" + str(img_num) + ".jpg"
30     slackBlocks.append({
31         "type": "section",
32         "text": {
33             "type": "mrkdwn",
34             "text": header_msg
35         },
36         "accessory": {
37             "type": "image",
38             "image_url": image_url,
39             "alt_text": "Thumbnail"
40         }
41     })
42     slackBlocks.append({"type": "divider"})
43
44     # Show buttons
45     slackBlocks.append({
46         "type": "actions",
47         "elements": [
48             {
49                 "type": "button",
50                 "text": {
51                     "type": "plain_text",
52                     "text": "Get eatery information"
53                 },
54                 "value": "Get restaurant info"
55             },
56             {
57                 "type": "button",
58                 "text": {
59                     "type": "plain_text",
60                     "text": "Find popular hawker centres"
61                 }
62             }
63         ]
64     })
65
66     return slackBlocks
```

```

60     },
61     "value": "Find popular hawker centres"
62   },
63   {
64     "type": "button",
65     "text": {
66       "type": "plain_text",
67       "text": "Find popular restaurants"
68     },
69     "value": "Find popular restaurants"
70   }
71 }
72 }
73
74
75 if (default_header_msg == None): default_header_msg = ''.join([c for c in header_msg if c not in ['*', '_', '**']]) # remove markdown characters
76 fulfillmentMessage.append({
77   "text": {
78     "text": [default_header_msg]
79   }
80 })
81
82 RichMessages = {
83   "#fulfillmentText": defaultPayload,
84   "fulfillmentMessages": fulfillmentMessage,
85   "payload": {
86     "slack": {
87       "attachments": [
88         {
89           "blocks": slackBlocks
90         }
91       ]
92     }
93   }
94 }
95 return RichMessages
96
97 def displayResults_slack(results, public_url, header_msg, default_header_msg = None, use_is_popular = False, pre_header_msg = None):
98   fulfillmentMessage = []
99   slackBlocks = []
100  dict_stars = {
101    "0.0": "stars_0",
102    "1.0": "stars_1",
103    "1.5": "stars_1_half",
104    "2.0": "stars_2",
105    "2.5": "stars_2_half",
106    "3.0": "stars_3",
107    "3.5": "stars_3_half",
108    "4.0": "stars_4",
109    "4.5": "stars_4_half",
110    "5.0": "stars_5"
111  }
112
113  if (default_header_msg == None):
114    default_header_msg = ''.join([c for c in header_msg if c not in ['*', '_', '**']]) # remove markdown characters
115
116  if (pre_header_msg != None):
117    slackBlocks.append({

```



```
175     if (r["image"] != ""):
176         slackBlocks.append({
177             "type": "section",
178             "text": {
179                 "type": "mrkdwn",
180                 "text": slackText1
181             },
182             "accessory": {
183                 "type": "image",
184                 "image_url": r["image"],
185                 "alt_text": "Thumbnail"
186             }
187         })
188     else:
189         slackBlocks.append({
190             "type": "section",
191             "text": {
192                 "type": "mrkdwn",
193                 "text": slackText1
194             }
195         })
196
197     star_url = dict_stars.get(str(r["rating"]), None)
198     if (star_url != None):
199         slackBlocks.append({
200             "type": "image",
201             "image_url": public_url + "/static/yelp_stars/" + star_url + ".png",
202             "alt_text": "stars"
203         })
204     slackBlocks.append({"type": "divider"})
205     num = num + 1
206
207     RichMessages = {
208         # "fulfillmentText": defaultPayload,
209         "fulfillmentMessages": fulfillmentMessage,
210         "payload": {
211             "slack": {
212                 "attachments": [
213                     {
214                         "blocks": slackBlocks
215                     }
216                 ],
217             }
218         }
219     }
220
221     return RichMessages
```

TouristFood-main.py

```
1  from flask import Flask, request, make_response, jsonify
2  import requests
3  import json
4
5  from lxml import html
6
7  import builtins as exceptions
8  from time import sleep
9  # import re,urllib.parse as urllib
10 import argparse
11
12 from IntentGetRestaurantInfo import processRestaurantInfoIntents
13 from IntentGetHawkerInfo import processHawkerInfoIntent
14 from IntentGetDiningInfo import hawkerCentreIntentHandler, restaurantIntentHandler, foodItemIntentHandler
15 from IntentWhatIs import whatisIntentHandler
16 from richMessageHelper import displayWelcome_slack
17
18
19 API_KEY = 'rG5T0rDyCq0G-lIelg9XzKfBcSNrc2F7zsa3C99Nray3q_-Wz8YU1JdilrAu7-g5QwdKCuZA0b9GXcp5xMImW9_dxQo_91b40AJ-PXRyqPGakfQD8WHL8BX7uDNJXXYx'
20 ENDPOINT = 'https://api.yelp.com/v3/businesses/search'
21 HEADERS = {'Authorization': 'bearer %s' %API_KEY}
22
23 global RUN_NGROK
24 RUN_NGROK = True
25 import sys
26 if len(sys.argv) == 2:
27     RUN_NGROK = False if sys.argv[1] != '1' else True
28
29
30
31 # import colorama
32 # colorama.init()
33
34 # For running ngok directly from python
35 public_url = ""
36 if (RUN_NGROK):
37     from pyngrok import ngrok
38     from os.path import abspath, dirname, join, realpath, isfile
39
40     dir_path = dirname(realpath(__file__))
41     ngrok.DEFAULT_CONFIG_PATH = join(dir_path, "ngrok.yml")
42     # ngrok.DEFAULT_NGROK_PATH = join(dir_path, "ngrok.exe")
43     # tunnels = ngrok.get_tunnels(ngrok_path=join(dir_path, "ngrok.exe"))
44
45     ngrok.connect(port=5000, proto="http")
46     tunnels = ngrok.get_tunnels()
47     public_url = tunnels[0].public_url
48     print("-----")
49     print(" Flask will be run from ngrok")
50     print("-----")
51 else:
52     #public_url = "https://olivefoodbot.herokuapp.com" # Use heroku url
53     print("-----")
54     print(" Flask will be run from Heroku")
55     print("-----")
56
57 print(" * PUBLIC URL: " + public_url)
58 app = Flask(__name__)
```

```

61 # ****
62 # WEBHOOK MAIN ENDPOINT : START
63 # ****
64 from flask import render_template
65 @app.route('/', methods=['POST', 'GET'])
66 def webhook():
67     if (request.method == 'GET'):
68         message = "Flask Webhook is running @ " + public_url
69         return render_template('index.html', message=message, img="/static/olive.png")
70
71     elif (request.method == 'POST'):
72         req = request.get_json(silent=True, force=True)
73         intent_name = req["queryResult"]["intent"]["displayName"]
74         action = req["queryResult"].get("action", None)
75         foodItem = req["queryResult"]["parameters"].get("foodItem", None)
76         hawkerCentreIntents = ["GetHawkerCentre", "GetHawkerCentreComplete"]
77         restaurantIntents = ["GetRestaurant", "GetRestaurantComplete", "GetRestaurantCuisineFirst",
78                               "GetRestaurantBudgetFirst", "GetRestaurantNumberFirst",
79                               "GetRestaurantCuisineBudgetFirst", "GetRestaurantCuisineNumberFirst",
80                               "GetRestaurantBudgetNumberFirst"]
81         # restaurantIntentsConfirmation = ["UserRepliesNumberRestaurant"]
82
83         if ("GetRestInfo" in intent_name):
84             return processRestaurantInfoIntents(req, public_url)
85
86         elif ("Hawker Info" in intent_name):
87             return processHawkerInfoIntent(req)
88
89         elif action in ["WELCOME"] or "Default Welcome Intent" in intent_name:
90             wasRedirected = (req["queryResult"].get("outputContexts") != None and any("Welcome" in d["name"] for d in req["queryResult"].get("outputContexts")))
91             additional_header = None if not wasRedirected else "I am sorry, but I could not understand. Try rephrasing your query."
92             return make_response(jsonify(displayWelcome_slack(public_url, additional_header = additional_header)))
93
94         elif (action in hawkerCentreIntents):
95             return hawkerCentreIntentHandler(req, public_url)
96
97         elif (action in restaurantIntents):
98             return restaurantIntentHandler(req, public_url)
99
100        # replaced knowledge-base intent with explicit intent for more control
101        elif (intent_name == "WhatIsDish"):
102            return whatisIntentHandler(req, public_url)
103
104        elif foodItem != None:
105            print(foodItem)
106            return foodItemIntentHandler(req, foodItem, public_url)
107
108        else:
109            # Cannot understand. Just redirect to welcome screen
110            followupEvent = {"name": "WELCOME",
111                            "parameters": {
112                                "unknown": True
113                            }
114                        }
115            return make_response(jsonify({"followupEventInput": followupEvent}))
116
117 # ****
118 # WEBHOOK MAIN ENDPOINT : END
119 # ****
120
121
122
123 import os
124 #from pml import app
125
126 if __name__ == '__main__':
127     port = int(os.environ.get('PORT', 5000))
128     print("Starting app on port %d" % port)
129     app.run(debug=True, host='0.0.0.0', port=port)
130

```

yelpAPIHelper.py

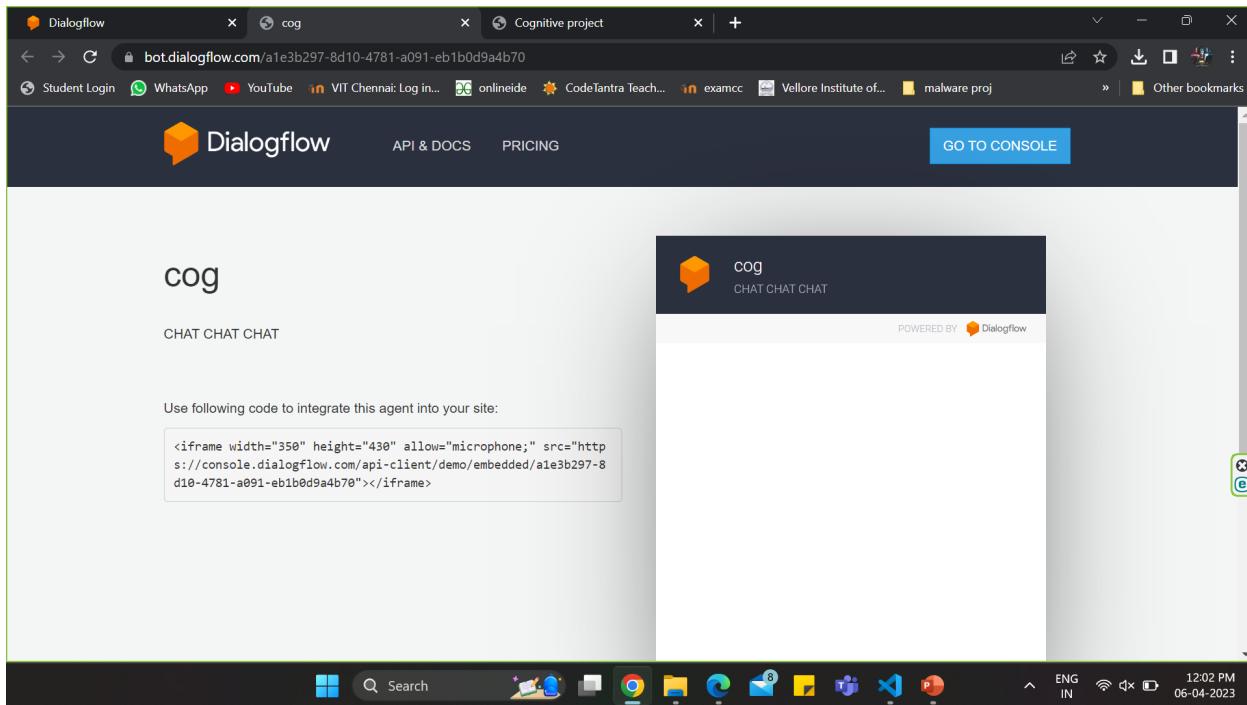
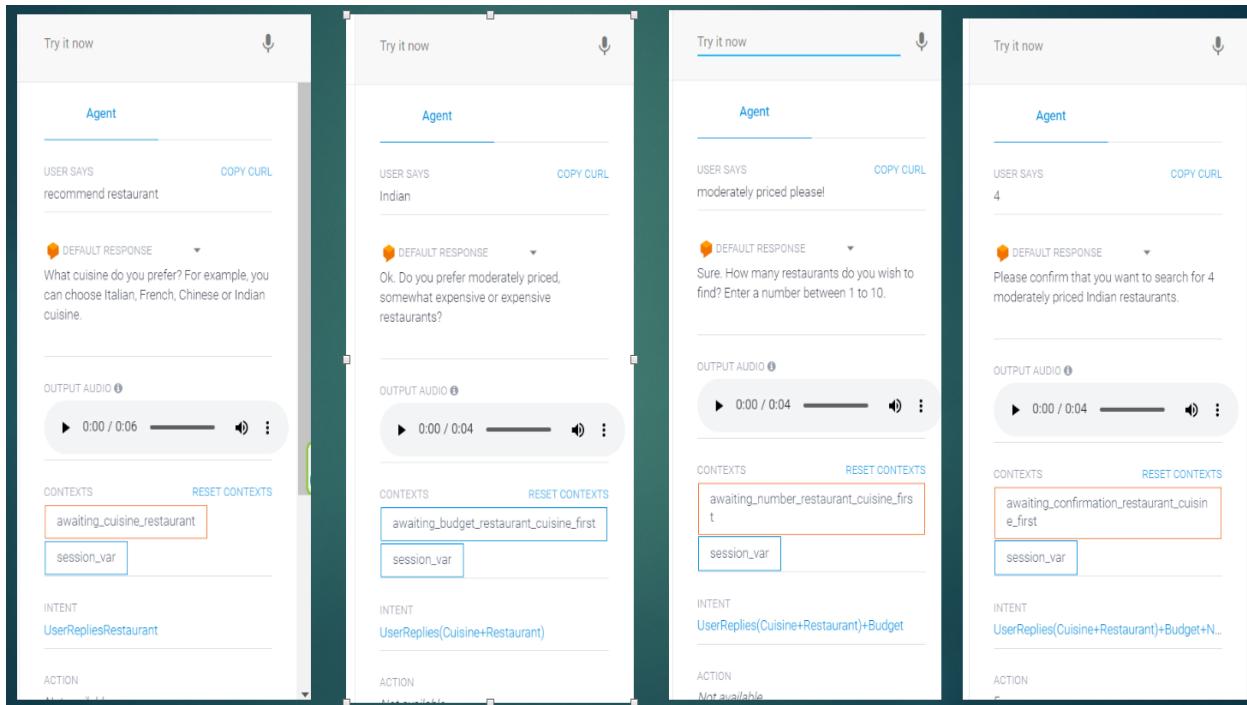
```
1  from urllib.parse import quote
2  import requests
3  import pprint
4  import time
5
6  # Using Yelp API v3
7  YELP_API_KEY = "rG5TOrDyCq0G-lIelg9XzKfBcSNrc2F7za3C99Nray3q_-Wz8YU1JdilrAu7-gSQwdKCuZA0b9GXCP5xMImW9_dxQo_9ib40AJ-PXRyqPGakfQD8WHL8BX7uDNJXXYx"
8  API_HOST = 'https://api.yelp.com'
9  SEARCH_PATH = '/v3/businesses/search'
10 BUSINESS_PATH = '/v3/businesses/'
11
12 # Defaults
13 SEARCH_LIMIT = 5
14 DEBUG_MODE = False
15
16 # Cache
17 Cache_Businesses = {}
18
19 def yelp_request(host, path, api_key, url_params={}, debug=DEBUG_MODE):
20     """Given API_KEY, send a GET request to the Yelp REST API.
21     Args:
22         host (str): The domain host of the API.
23         path (str): The path of the API after the domain.
24         API_KEY (str): Your API Key.
25         url_params (dict): An optional set of query parameters in the request.
26     Returns:
27         dict: The JSON response from the request.
28     Raises:
29         HTTPError: An error occurs from the HTTP request.
30     """
31     url = '{0}{1}'.format(host, quote(path.encode('utf8')))
32     headers = {'Authorization': 'Bearer %s' % api_key}
33     if (debug): print(u'Querying {0} ...'.format(url))
34
35     start_time = time.time()
36     response = requests.request('GET', url, headers=headers, params=url_params)
37     elapsed_time = time.time() - start_time
38     print("... API call: " + url + " took " + str(elapsed_time) + ". RateLimit-Remaining: " + response.headers.get("RateLimit-Remaining"))
39     if (response.headers.get("RateLimit-Remaining") == '0'):
40         print("Oops.. We have exceeded Yelp Fusion API daily access limit... Please try again tomorrow!")
41     return response.json()
42
43
44 def yelp_search(api_key, term, location, sortBy='best_match', debug=DEBUG_MODE):
45     """Query Yelp Search API by a search term and location.
46     Args:
47         term (str): The search term passed to the API.
48         location (str): The search location passed to the API.
49     Returns:
50         dict: The JSON response from the request.
51     """
52     url_params = {'term': term.replace(
53         ' ', '+'), 'location': location.replace(' ', '+'), 'limit': SEARCH_LIMIT, 'sort by': sortBy}
54     return yelp_request(API_HOST, SEARCH_PATH, api_key, url_params=url_params, debug=debug)
55
56
```

```

57
58     def yelp_business(api_key, business_id, debug=DEBUG_MODE):
59         """Query Yelp Business API by a business ID.
60         Args:
61             business_id (str): The ID of the business to query.
62         Returns:
63             dict: The JSON response from the request.
64         """
65         business_path = BUSINESS_PATH + business_id
66         return yelp_request(API_HOST, business_path, api_key, debug)
67
68
69     def yelp_query_api(term, location, debug=DEBUG_MODE, num_results=1):
70         """Queries the API by the input values from the user.
71         Args:
72             term (str): The search term to query.
73             location (str): The location of the business to query.
74             Note: Businesses returned in the response may not be strictly within the specified location.
75         Returns:
76             array of responses that match num_results
77         """
78         sortBy = 'best_match'
79         if (location.lower() != 'singapore'): sortBy = 'distance'    # don't just use 'best-match', we want 'distance'
80
81         response = yelp_search(YELP_API_KEY, term, location, sortBy= sortBy, debug=debug)
82
83         # Check if daily access limit has been reached
84         # https://www.yelp.com/developers/documentation/v3/rate_limiting
85         if ("error" in response):
86             if (response["error"].get("code") == "ACCESS_LIMIT_REACHED"):
87                 return 1
88             else:
89                 return 2
90
91         # Check if this business exists. If exists, we get its ID
92         businesses = response.get('businesses')
93
94         # if (location.lower() != 'singapore'):
95         #     businesses = sorted(businesses, key= lambda b: b['distance'])    # don't just use
96
97
98         if not businesses:
99             print(u'No businesses for {0} in {1} found.'.format(term, location))
100            return None
101
102         # If exists, then we get its details
103         num_results = min(len(businesses), num_results)
104         print(u'{0} businesses found, querying business info ' \
105             'for the top {1} results ...'.format(
106                 len(businesses), num_results))
107
108         responseArr = []
109         for b in businesses[:num_results]:
110             business_id = b['id']
111             responseArr.append(yelp_business(YELP_API_KEY, business_id))
112
113             if (debug):
114                 print(u'Result for business "{0}" found:'.format(business_id))
115                 pprint.pprint(response, indent=2)
116
117         return responseArr
118

```

VII. OUTPUT



Try it now



Agent

USER SAYS

COPY CURL

what is carrot cake



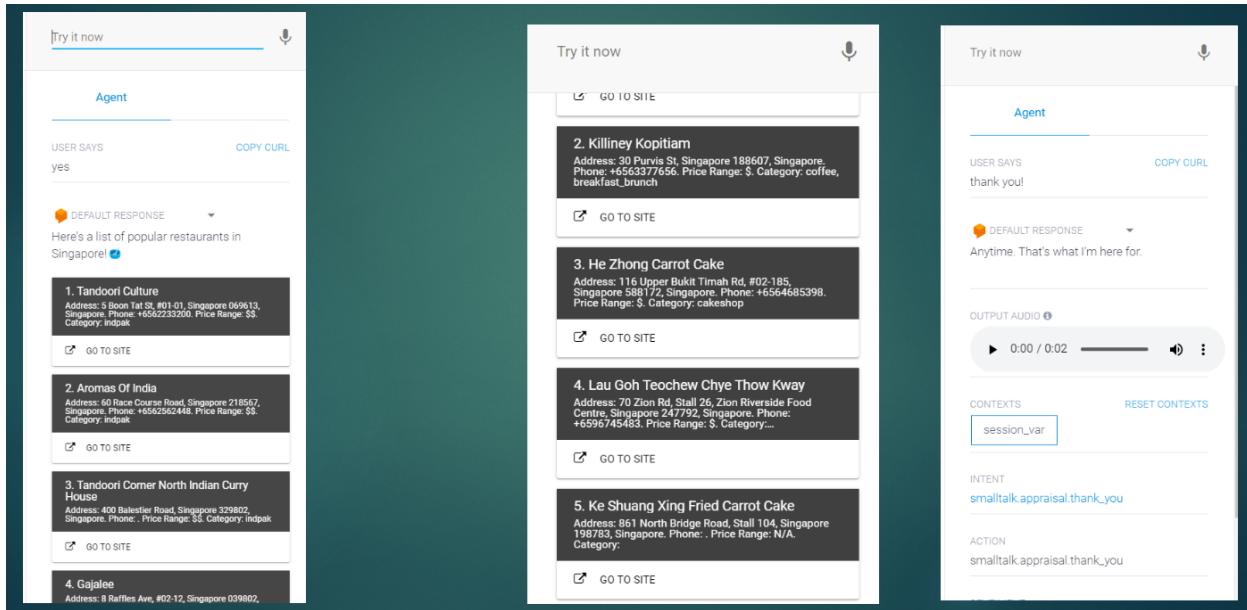
DEFAULT RESPONSE



Carrot cake: Dialect name "chye tow kway" or in Chinese, 菜头粿. Contrary to its name, "carrot cake" contains no carrots and is not a cake in the usual sense of the word. "Carrot cake" in Singapore is usually steamed minced radish mixed with flour. It is normally fried with eggs with served with some light garnishing of spring onions or chye-poh (preserved radish). It is also served piping hot in its steamed form at tim-sum shops. Popular eateries selling carrot cake

1. Guan Kee Fried Carrot Cake

Address: 270 Queen Street, #01-59 Albert Centre
Market & Food Centre, Singapore 180270, Singapore.
Phone: +6562453989. Price Range: \$. Category:....



Viewing inside UI after embedding the iframe into the frontend html:

The screenshot shows a web browser window with multiple tabs open. The active tab is titled "Cognitive Project Food Chatbot". The page content includes a list of "Members" (Sankalp Hegde, Navya Verma, Emmay Koushal) and a message from the bot: "Hey yo! I am Olive, the foodBot for Singapore food 😊 Here's a couple of the things I can do: Get a specific restaurant's details such as its location and rating (eg Swensens in Jurong). Recommend a dining place. (eg Find popular eateries) Tell you about specific local food in..."

RESULT

VIII. PERFORMANCE AND COMPARISON

Here are several existing methods for food chatbots in Singapore, such as Burpple, HungryGoWhere, and Chope. Here's a comparison of a food chatbot for Singapore with these existing methods:

Burpple: Burpple is a popular food recommendation app in Singapore that provides personalized restaurant recommendations based on user preferences. The app uses a combination of user reviews, expert reviews, and AI-driven algorithms to generate recommendations. However, Burpple is a standalone app and does not offer a chatbot interface for users to interact with.

HungryGoWhere: HungryGoWhere is another popular food recommendation app in Singapore that provides personalized recommendations based on user preferences. The app allows users to search for restaurants by cuisine, location, and price range, and provides reviews and ratings for each restaurant. Like Burpple, HungryGoWhere is a standalone app and does not offer a chatbot interface.

Chope: Chope is a restaurant reservation app in Singapore that allows users to make reservations at over 4,000 restaurants. The app also provides reviews and ratings for each restaurant, but does not offer personalized recommendations or a chatbot interface.

Mainly our project is inspired from the chatbot of eleow. There were many differences made to this chatbot. The existing chatbot asks one or two questions and based on the response it suggests restaurants and cuisine. But our chatbot asks many questions based on the user's queries and then analyzes all replies for the user and gives appropriate results. Our results are better than the existing chatbots since it considers all constraints of the user. This is represented in Table 1. The other difference in our chatbot is that we have voice input as well. This helps a lot for the tourists and saves a lot of time.

These are the main changes and differences in our project compared to existing projects.

The screenshot shows a conversational AI interface with two columns of messages. The left column represents the user's input, and the right column represents the chatbot's response. Each message includes a timestamp, audio playback controls, and context reset buttons.

User Input	Chatbot Response
USER SAYS recommend restaurant	DEFAULT RESPONSE Which cuisine do you like? For instance, do you want Mexican, French or Japanese cuisine?
USER SAYS French	DEFAULT RESPONSE Ok. Do you prefer moderately priced, somewhat expensive or expensive restaurants?
USER SAYS expensive	DEFAULT RESPONSE Sure. How many restaurants do you wish to find? Enter a number between 1 to 10.

USER SAYS COPY CURL

 DEFAULT RESPONSE ▾
Here's a list of popular restaurants in Singapore! 🇸🇬

1. Rhubarb Le Restaurant
Address: 3 Duxton Hill, Singapore 089589, Singapore.
Phone: +6581275001. Price Range: \$\$\$\$. Category: french
[GO TO SITE](#)

2. Odette
Address: 1 Saint Andrew's Rd, #01-04, Singapore 178957, Singapore. Phone: +6563850498. Price Range: \$\$\$\$. Category: french, wine_bars
[GO TO SITE](#)

3. Gunther's
Address: 36 Purvis St, Singapore 188613, Singapore.
Phone: +6563388955. Price Range: \$\$\$\$. Category: french
[GO TO SITE](#)

4. Les Amis
Address: 1 Scotts Rd, #02-16, Singapore 228208, Singapore. Phone: +6567332225. Price Range: \$\$\$\$. Category: french

4

 DEFAULT RESPONSE ▾
Please confirm that you want to search for 4 expensive French restaurants.

OUTPUT AUDIO 

▶ 0:00 / 0:04  ⏴ ⏵ ⏷

Question1	Question2	Question3	Question4	Question5
Which cuisine do you prefer? french , mexican	Do you like moderately priced or less expensive	How many restaurants you wish to find 1-10	Please confirm to search 4 expensive restaurants	Gives the list of recommended restaurants

Table 1

IX. CONCLUSION

In conclusion, food chatbots are becoming an increasingly popular tool for restaurants, food delivery services, and food bloggers. They provide a quick and efficient way for users to search for recipes, place orders, and get recommendations. With the advancements in AI and natural language processing, food chatbots are becoming more conversational and personalized, making them an essential part of the food industry. Additionally, they help businesses to reduce costs and improve customer satisfaction. Overall, food chatbots are an innovative solution that is changing the way we interact with food and technology.

X. FUTURE WORK

The chatbot can be linked to past medical records of the user so that health concerns such as cholesterol, allergens, GERD triggers, etc can directly be mapped by the AI to remove certain search results.

The chatbot can be integrated with social media platforms such as Instagram and facebook messenger to reach a wider user base, as well as to better recommend places based on popularity as well.

For a more personalized experience, the chatbot can query for certain user details, for example name, which can then be used during the session to reserve a table or pre-order a meal via a single click alone.

the chatbot can further be enhanced to support other languages, as well as be integrated with google maps and voice assistants like alexa for a more hands-off experience.

REFERENCES

- [1] Chen, Y., Cao, X., & Cao, Y. (2021). Factors influencing customers' acceptance of food chatbots. *Journal of Hospitality and Tourism Technology*.
- [2] Grewal, J., & Sharma, S. (2021). Chatbots in the food industry's digital transformation. *International Journal of Information Management*.
- [3] Hu, J., Li, J., & Li, X. (2021). Personalizing customer experience with chatbots in the restaurant industry. *International Journal of Hospitality Management*.
- [4] Liu, X., Chen, Y., & Ye, Q. (2019). How can chatbots improve customer engagement and satisfaction? A survey of restaurant customers. *Journal of Foodservice Business Research*.
- [5] Gupta, A., & Yadav, V. (2020). A literature review on chatbots and its algorithms. *International Journal of Advanced Computer Science and Applications*.