Hungry Hippos: Restaurant Bot Documentation

Index

1. Introduction

Conversational bots have many applications throughout the industry today as businesses are looking to take advantage of deploying such services on messaging platforms, voice assistant devices like Google Home, Alexa or customer service bots with the goal of improving user interaction, reducing time or freeing up time taken to service simple tasks.

Our goal was to explore building a task oriented chatbot into an initial minimum viable product that can aid in providing information about business (restaurants, grocery stores, retail stores, pharmacies, etc) from the Yelp directory, such as finding business location, operating hours or notifying the user on nearby events. Additionally it would be helpful to automate some of the tasks that go into a reservation system or placing an order via a chatbot.

We believe collecting feedback on an initial version of such a service and future development cycles would lead to a fully fleshed out agent that could be used by Yelp or Opentable and provide alternate ways to engage with the customer in lieu of the traditional website search.

Some key aspect that we were interested in focusing and learning about were:

1. Understand how all the Google Cloud Platform and Dialogflow's components work together during the building of such a service.
2. How to make the user interaction fast and easy to use?
3. Understand the data needs for building such a service.
4. Integrating with platforms and end devices and apps like Slack, Google Home and Google Assistant enabled devices.

2. Design and Architecture

2.1 Requirements

Certain requirements that we think should apply towards building such a service, which were considered while designing the agent architecture are as follows:

Functional:

The Dialogflow agent must respond when a conversation is triggered, regardless of the medium of the end client/device.

The agent must work with text based clients like Slack, i.e. it should be able to parse such queries

The agent must work with voice enabled devices like Google Home and other Google Assistant enabled devices.

Usability :

The chatbot is able to identify user requests to one of the intents defined within Dialogflow.

If a matching intent is not found the agent should fail gracefully by letting the user know that it could not process their request.

The architecture should be able to process multiple user requests being initiated from various end clients & devices processed simultaneously. This is possible due to the use of GCP Cloud Functions, where each user request can be processed in parallel and the service scales up to the number of requests being generated.

Non-functional:

Performance/ Response time requirement - The chatbot agent should reply to the user in a timely manner to keep the conversation flowing without interruption.

Reliability requirement- The chatbot agent must behave consistently, i.e. trigger the same intents and provide the same responses to user queries each time.

Availability:

The agent must be available to chat at all times.

This is achieved by deploying the code on Google Cloud Platform services. This has a couple of benefits, any updates to the code will initiate a new deployment on the GCP server using serverless functions. This does not impact the end clients using the agent.

2.2 Dialogflow - System Architecture

Google's Dialogflow is a natural language understanding platform that helps developers to design conversational interfaces for websites, mobile apps.

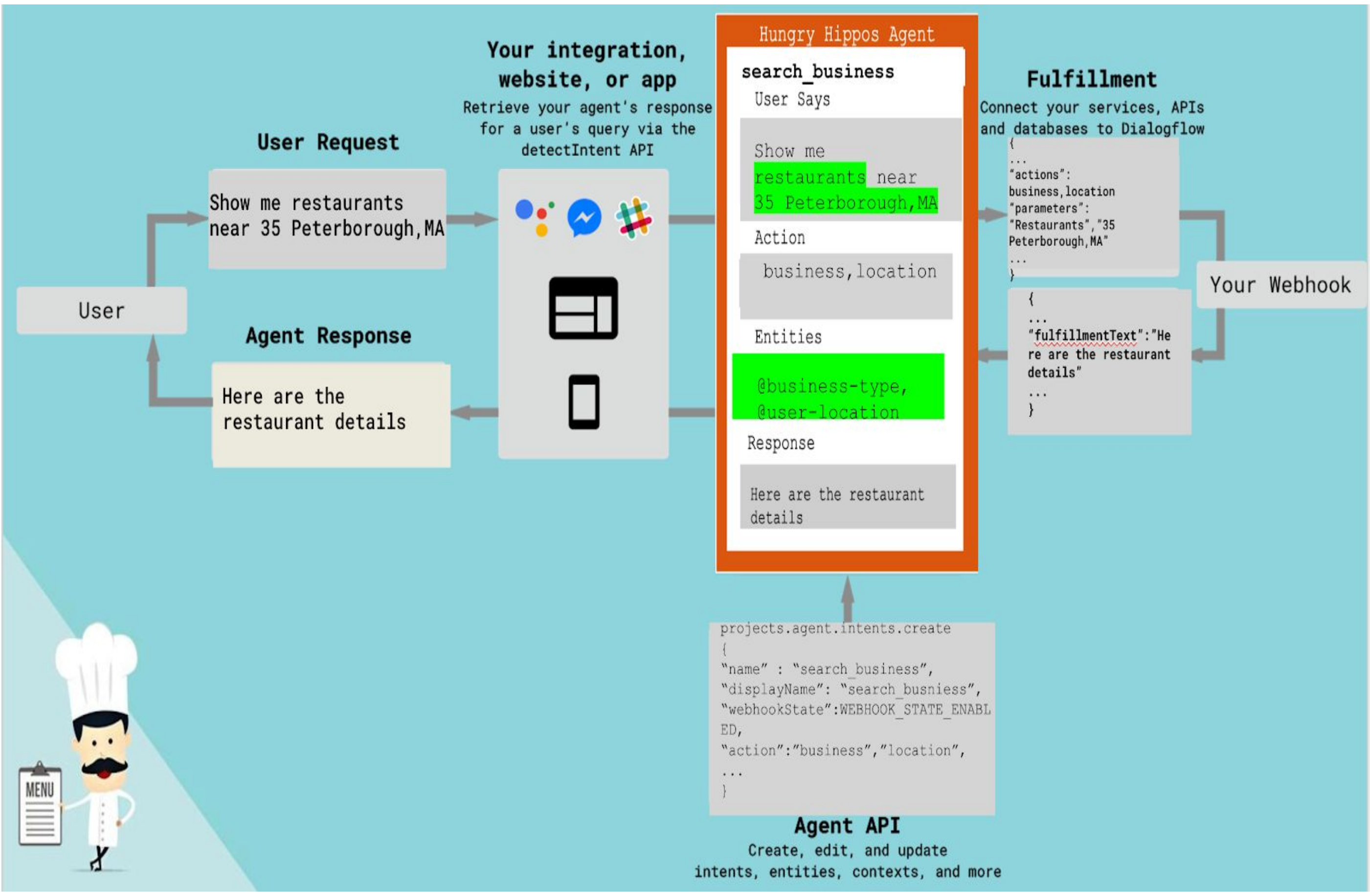


Fig A: Components and Architecture of Dialogflow

Agent

A Dialogflow agent is a virtual agent that handles conversations with our end-users. It is a natural language understanding module that helps in identifying user's queries. The training does not need to be overly explicit, as long as we train it to handle expected conversation scenarios.

Dialogflow also helps in translating end-user text or audio during a conversation to structured data that our apps and services can understand.

Agents also serve as a top-level container for settings and data:

- Agent settings for language options, machine learning settings, and other settings that control the behavior of your agent.
- Intents to categorize end-user intentions for each conversation turn.
- Entities to identify and extract specific data from end-user expressions.
- Knowledge to parse documents (for example, FAQs) and find automated responses.
- Integrations for applications that run on devices or services that directly handle end-user interactions for us (for example, Google Assistant).
- Fulfillment to connect our services when using integrations.

NLU engines used by the agent consists of a supervised intent classifier that is trained on a variety of sentences as inputs and intents as target and a Entity Recognition model which can be a pretrained model like Spacy or Stanford's NLP Library to extract the entities present in the input.

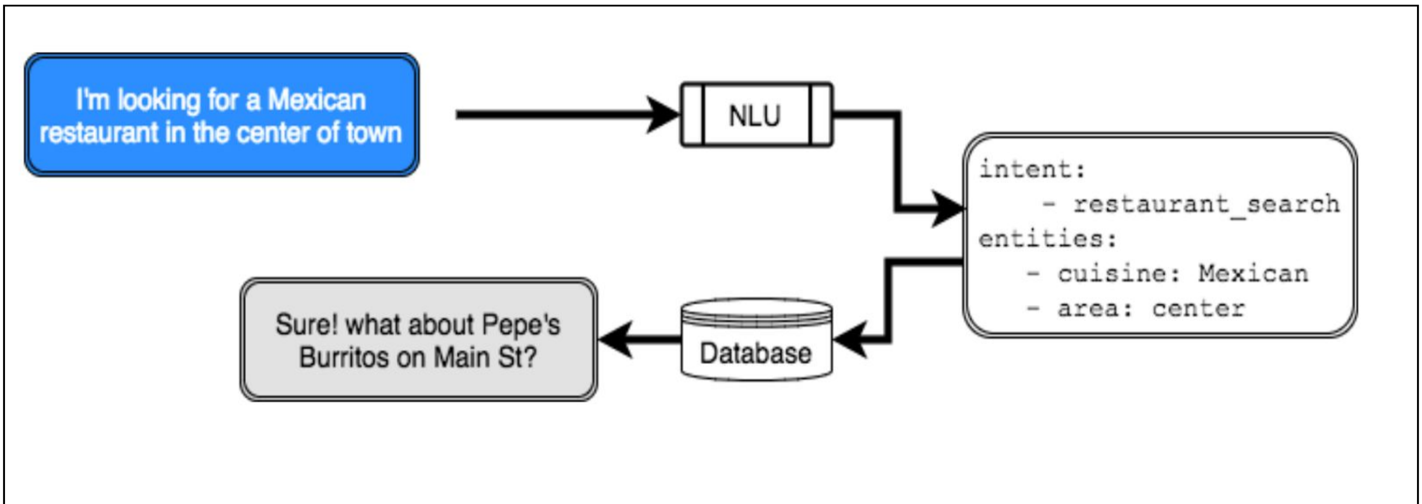


Fig B: NLU Architecture

In Dialogflow, we can create Intents that map the user's input to the responses. In each intent, you define examples of user statements that can trigger the intent, what to extract from the statement, and how to respond. Entities are extracted from user queries and used as parameters for processing that particular Intent. Entities are of two types namely- system entities and custom entities. System entities are predefined entities that can match the common types of data such as (Location, time etc.,). Custom entities are entities that can be created by us for matching custom data i.e, **`@pizza-type`** entity that can match the type of Pizza available for purchase in a restaurant.

In Fig A, we can see that the user wants or intends to search for restaurants near 35 Peterborough Street, MA. The intent 'search_business' is triggered by the user query since the query has matched the format of the training phrases that we have used to train the agent for that intent label. We enter multiple training phrases for each intent to capture all possible user queries. The entities extracted are the **`@business-type`** and @**`user-location.`** The entities can be made either optional or mandatory for the user. If the entities are mandatory, then the bot would perform an **Action** by asking the user to enter more details to get parameter values.These parameters are stored in the backend or Firestore database. **Context** represents the current state of the user's request and allows our agent to carry the information from one agent to another. It represents a necessary condition that has to exist before an intent can be triggered.

The Dialogflow can connect with external systems using the Fulfillment code,which is deployed as a webhook. During a conversation, the fulfillment lets you use the information extracted by Dialogflow's natural language processing to generate dynamic responses or trigger actions on your backend.

2.3 Model choices for Intent Detection in Dialogflow

Dialogflow provides an option to use Hybrid or ML only models for intent detection process based on the number of training examples. If the training examples are limited, Dialogflow uses the Hybrid model which comprises Rule based approach and a traditional classification model. The ML only model is adopted when the number of examples in intents is less. It may use a state-of-the-art Deep learning approach for intent classification.

We chose to use the Hybrid Model due to the limited size of our training phrases, as we manually created the training phrases for each of our intents.

The classification model's performance is evaluated by the confidence score. This confidence score indicates how certain the agent is that the intent has been assigned correctly to a given user query. The score can have a value between 0 and 1. A score is calculated for each intent for each user input and the one with the highest score is returned as the result. If the confidence score falls below the threshold, which is 0.3 by default but can be changed, the fallback intent is triggered. This indicates that our agent does not recognize the user input. The fallback intent re-prompts the user to provide necessary inputs for the action. If no fallback intent is defined, then we won't get any response.

3. Implementation Details:

We will provide a few high level details on the Dialogflow Console, GCP Console and some of its components in relation to our project and our use cases. (For more details please refer to the detailed documentation here)

3.1 Dialogflow Console - This is the entrypoint and the main web user interface to create, build and test our agents. This is where we start defining our Intents, Entities, add our training phrases, setup our integrations, and validate how our model is performing on new user queries sent from end clients apps like Slack, Google Home.

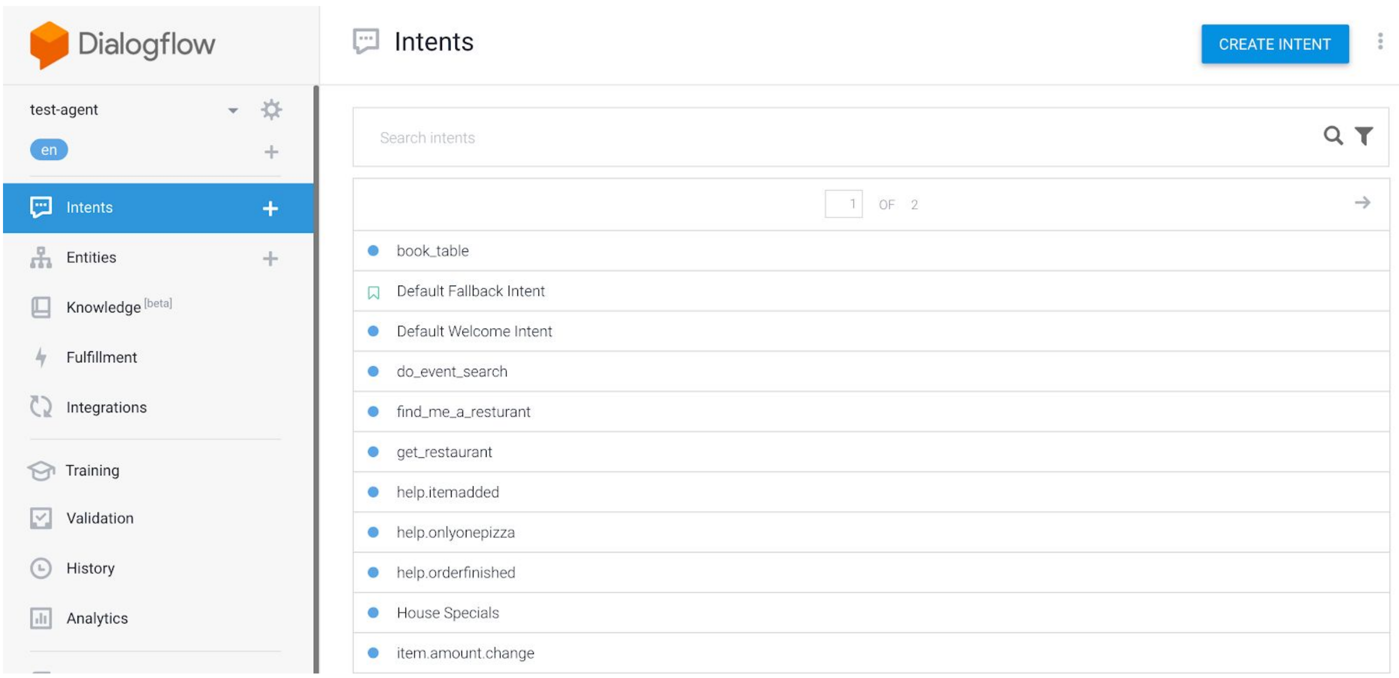


Fig C . Dialogflow Console - Main Page

(**Note**: Dialogflow Console is different from GCP Console, the later is used to manage GCP resources like Cloud functions, Data Store, Firebase, ML modules)

3.2 Create New Agent

First, we have to create a new Dialogflow agent and associate it with a new Google Cloud project which would allow us to enable Google Cloud Functions and Actions on Google SDK for this project, these would be required in order to service incoming requests from client applications.

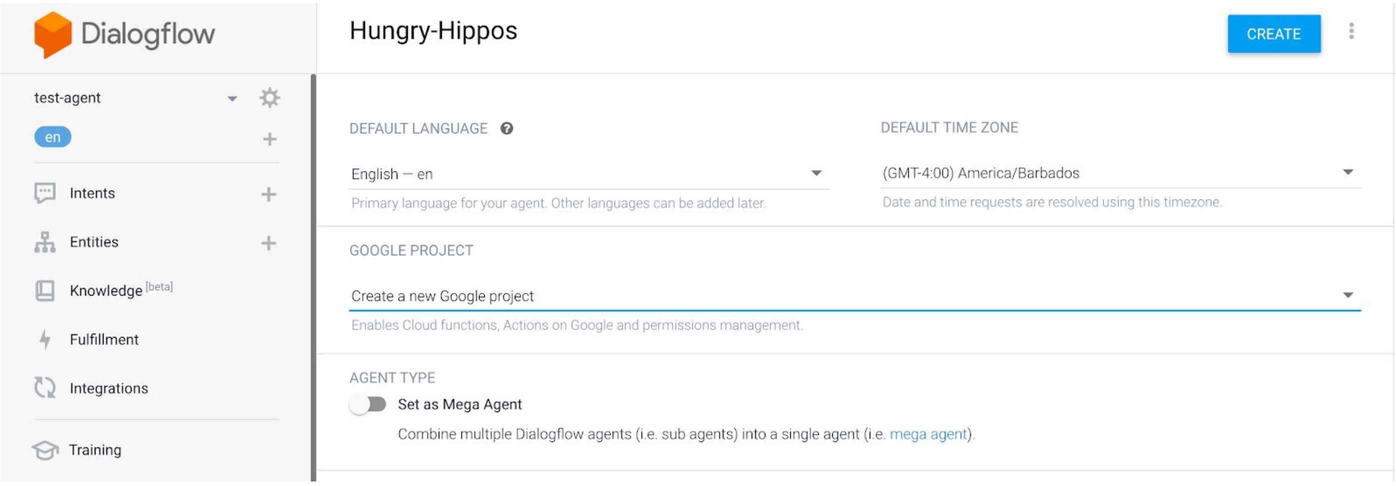


Fig D. Creating New Test Agent

(**Note**: **For information on importing our exported agent files and code into Dialogflow see Appendix 1)**

3.3 Creating Intents

Next we start adding the intents that we wanted our agents to recognize. An intent categorizes an end-user's intention for one conversation turn. For our agent, we define a total of **30** intents, these combined intents can handle a complete conversation or a subset can be triggered for executing any of the **7** Use Cases defined in section 3.8. When an end-user writes or says something, referred to as an end-user expression, Dialogflow matches the end-user expression to the best intent in our agent. Matching an intent is also known as intent classification.

A basic intent in our agent contains the following:

- **Training phrases**: These are example phrases for what end-users might say. When an end-user expression resembles one of these phrases, Dialogflow matches the intent. We don't have to define every possible example, because Dialogflow's built-in machine learning expands on our list with other, similar phrases.

- **Action**: We can define an action for each intent. When an intent is matched, Dialogflow provides the action to our system, and you can use the action to trigger certain actions defined in our Cloud Functions.
- **Parameters**: When an intent is matched at runtime, Dialogflow provides the extracted values from the end-user expression as *parameters*. Each parameter has a type, called the entity type, which dictates exactly how the data is extracted. Unlike raw end-user input, parameters are structured data that can easily be used to perform some logic or generate responses.
- **Responses**: We define text, speech, or visual responses to return to the end-user. These may provide the end-user with answers, ask the end-user for more information, or terminate the conversation.

For a complete list of intents for each Use Case refer to APPENDIX 2

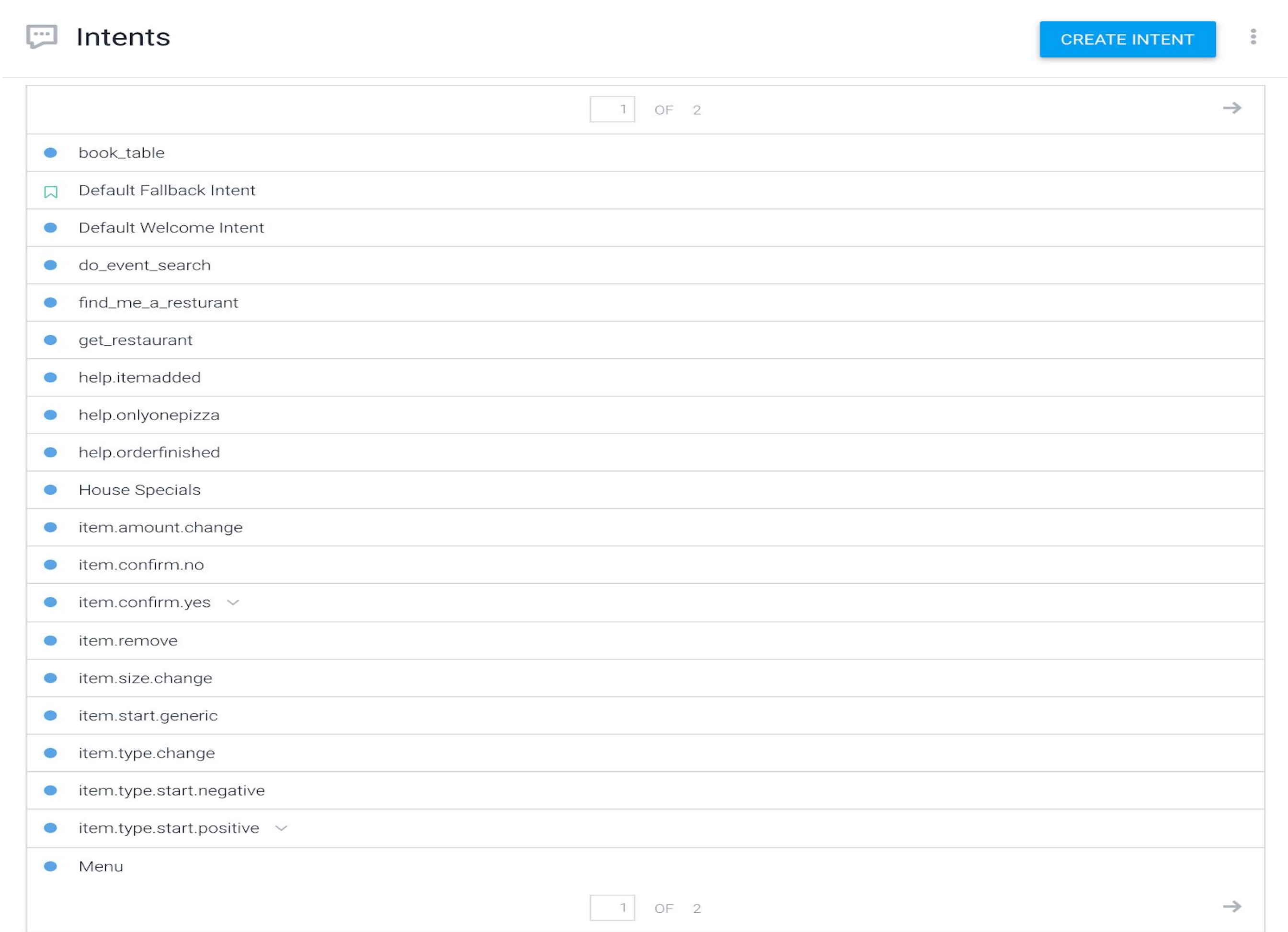


Fig E. Sample Intent List that we have defined for our agent.

3.4 Creating Entities

We created custom entities that could be used across all intents and would help in identifying business names, business types, cuisine types, business ratings, facility type, order type and thereby bootstrap the identification of such occurrences in conversational queries and somewhat circumvent the process of needing to tag too many entities in training phrases manually. (See Appendix 3. for additional info on these entity types)

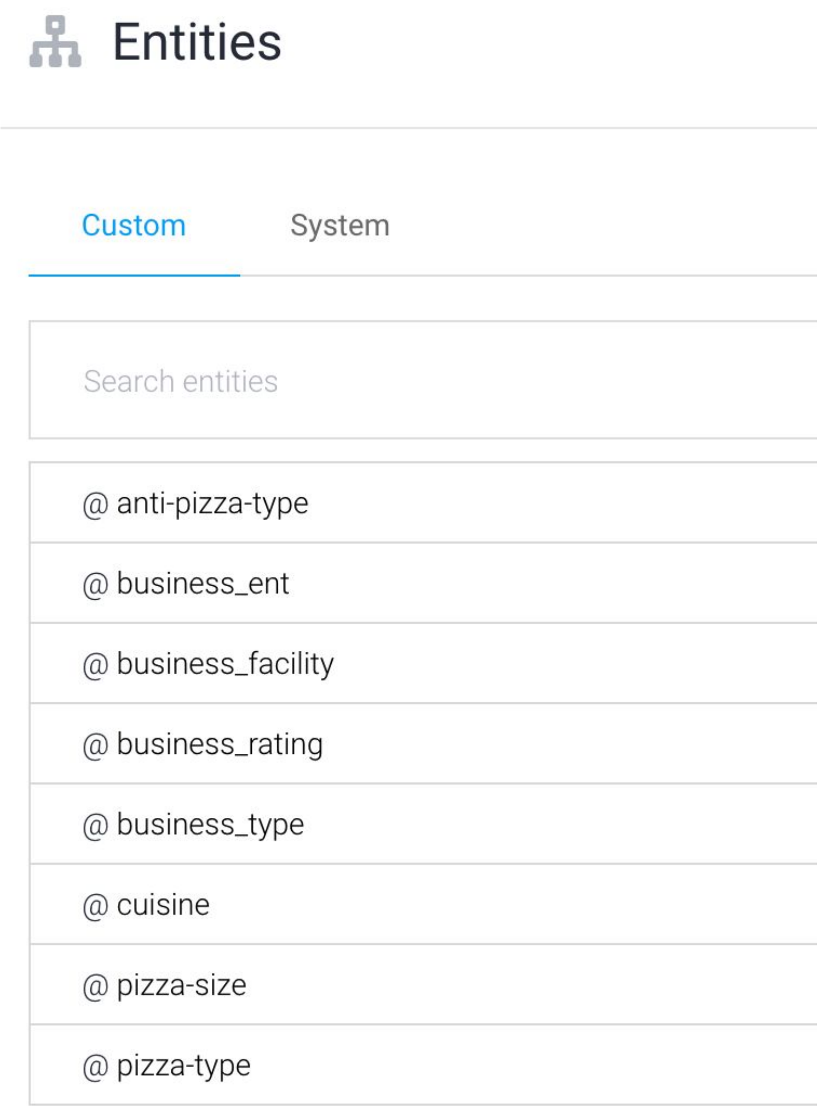


Fig F. Creating custom entity types

3.5 Adding Training Phrases

For each intent we add our training phrases, and tag the entities with their corresponding entity type that we have defined. In the figure below, we show a sample of the training phrases defined for the business search intent.

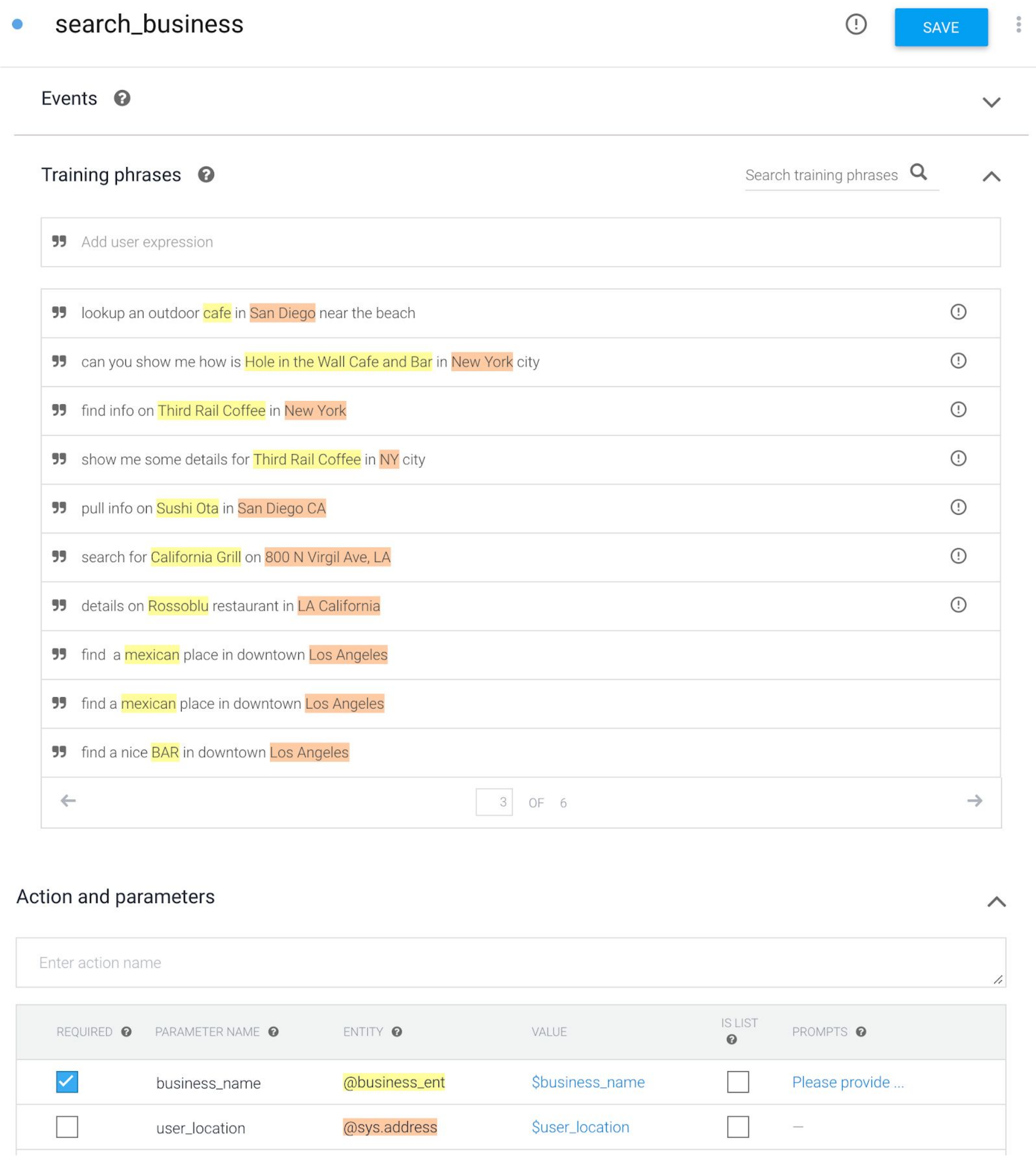


Fig G. Sample of training phrases and params defined for Yelp Business Search Intent in Dialogflow

The entities that are tagged in the training phrases include Business_name and location entities which when extracted from the user request would be passed to the cloud function for fulfillment and fetch information pertaining to that business from the Yelp API. On adding new training phrases in an intent we can simply trigger a retraining and deployment of the new model, this seamlessly is available to all the users using the agent without any interruption to the service.

3.6 Setting up the Google Cloud components: For webhook calls and fullfillment:

So far the steps we have listed above only train a model on the provided training phrases and detect/trigger an intent from the user query. This alone would not provide any dynamic response back to the user.

In order to do that we need to enable fulfillment for an intent. Each intent has a setting to enable fulfillment.

This allows us to respond to that intent by calling a service that we define. For example, if an end-user wants to reserve a table on Friday, your service can check your database and respond to the end-user with availability information for Friday.

When an intent with fulfillment enabled is matched, Dialogflow sends a request to our webhook service with information about the matched intent and the extracted entities and parameters.

The actions or responses that need to be executed for that intent are mapped with a corresponding cloud function which will then be executed. We have defined corresponding cloud functions for all of our intents, defining actions and responses appropriate for that use case. **(See Appendix 4 for a sample cloud function for the business_search intent)**

The following diagram shows the processing flow for fulfillment.

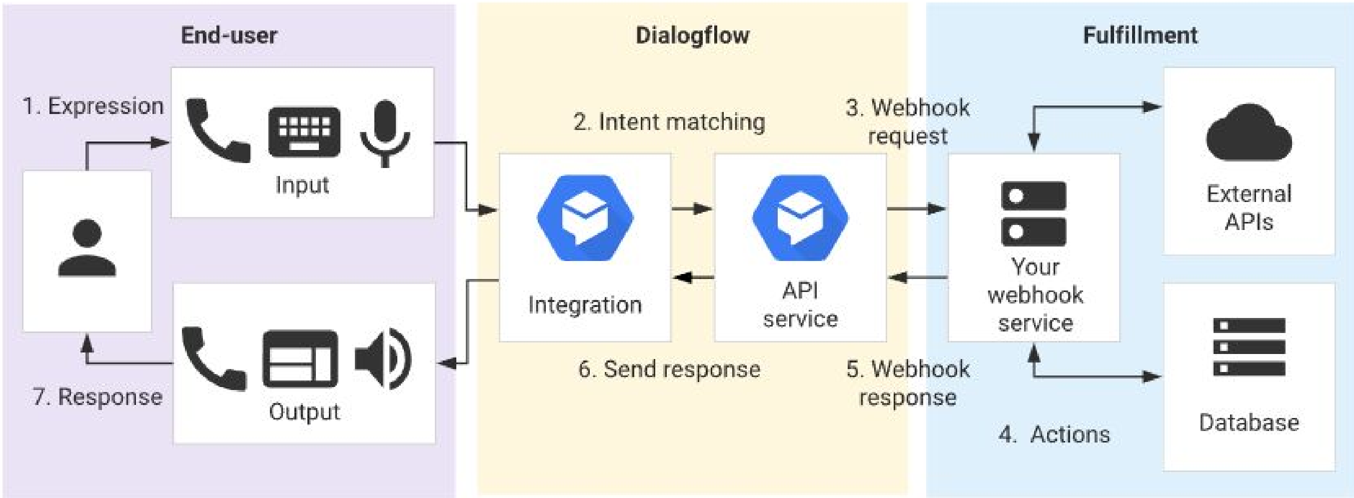


Fig H. Fulfillment Workflow

1. The end-user types or speaks an expression.
2. Dialogflow matches the end-user expression to an intent and extracts parameters.
3. Dialogflow sends a webhook request message to your webhook service. This message contains information about the matched intent, the action, the parameters, and the response defined for the intent.
4. Your service performs actions as needed, like database queries or external API calls.
5. Your service sends a webhook response message to Dialogflow. This message contains the response that should be sent to the end-user.
6. Dialogflow sends the response to the end-user.
7. The end-user sees or hears the response.

3.7 Train and Publish the Model

After adding the intents and entities, the model is trained and published. After the model is published, REST API can be accessed from the service to access the LUIS endpoint to get the intent and entity from the user's query.

3.8 Components of Hungry Hippos Agent

In this section, we will be discussing in depth about the different Use Cases and the intents defined within each use case and their respective workflows. **Note**: All the use cases and their respective intents have outputs defined for Slack and Google Home end clients. (**Note**: For Conversational Flow Diagrams for all our Use Cases please refer to **Appendix 8**)

Use Case 1: Search for Business from Yelp

This Use Case involves querying the Yelp database using the Fusion API endpoint to lookup the parameters extracted from the user query on matching one of the intents created for this use case. (see Reference 6. for details on the Yelp Fusion API endpoints or https://www.yelp.com/developers/documentation/v3)

Intent 1: Search for business info (search_business)

The end goal here is to bring back business information fields such as **address, no. of reviews, average rating, contact details, price, latest business image, business reviews.**

This intent will be triggered when the model is confident that the user query matches with the training phrases defined within this intent. The default confidence threshold for intent matching is 0.30.

Intent	**Sample Training Phrases**	**Functional name in Dialogflow/ Cloud Function**
Search Business Info	show me some details for Rock On Pizza in Escondido, California	search_business
	lookup an outdoor cafe in San Diego	
	find me a pharmacy near me	

For this intent we have set Business Name as a required param. If not provided in the user query, the agent prompts the user to give one before proceeding. **Location** is an optional param, if provided overwrites default user location. For the MVP we have set a default user location within the agent. Ideally we would want to pull this location metadata from the end device using the agent, but this required getting some additional permissions and steps that needed some more legwork, which we skipped for the purposes of this MVP, although the code is designed keeping that extension in mind. Once we have extracted the highlighted params, we next query the **Yelp Business Search endpoint** & **Yelp Review endpoint** with these parameters. This allows us to fetch information on our field of interest as mentioned earlier and build an appropriate response for the end client (Slack vs Google Home). All of these steps are coded and defined by us within the cloud function. (See Appendix for the Cloud function for this use case).

Sample Slack Output Response:

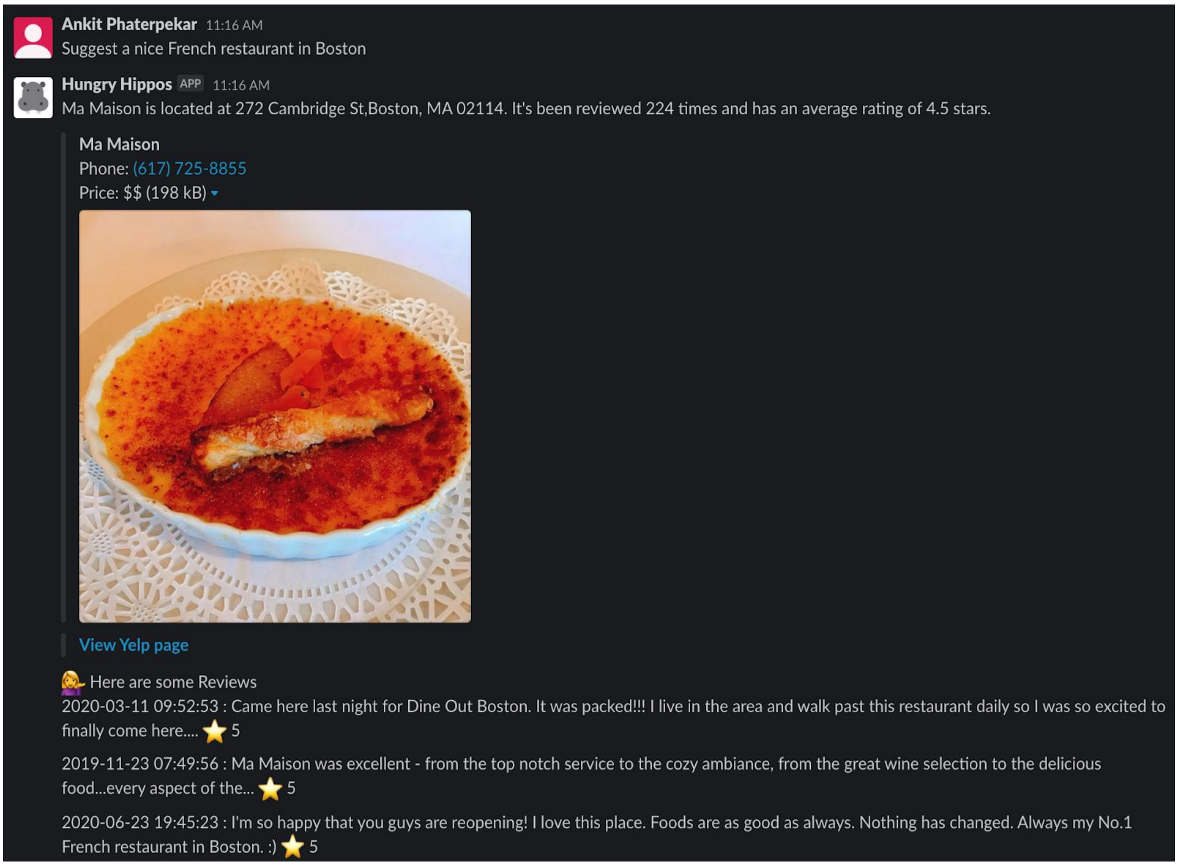


Fig I. Slack Output Response for Business Search

Intent 2: Search for business details

For this use case we wanted to answer user queries concerning the 'operating hours' for a store, or answer user queries regarding establishments open for food pickup or delivery.

Intent	Sample Training Phrases	Functional name in Dialogflow/ Cloud

		Function
Search Business Details	When does Giant close tonight ?	search_business.details
	show me hours of operation for Rossoblu restaurant in LA California	
	Till what time is Laliguras open today ?	

For this intent we have set Business Name as a required param. If not provided in the user query, the agent prompts the user to give one before proceeding.

Location and Time are optional params, if provided overwrites defaults. The default for time is today. The process is similar to the one mentioned for the prior intent (business info intent), except that we query the Yelp's **Business-Details endpoint** while execution. Yelp response provides as with a static list of weekly hours for the business, so we carry out some additional mappings for date-time objects to identify for which day of the week the user is requesting details regarding.

Sample Slack Output:

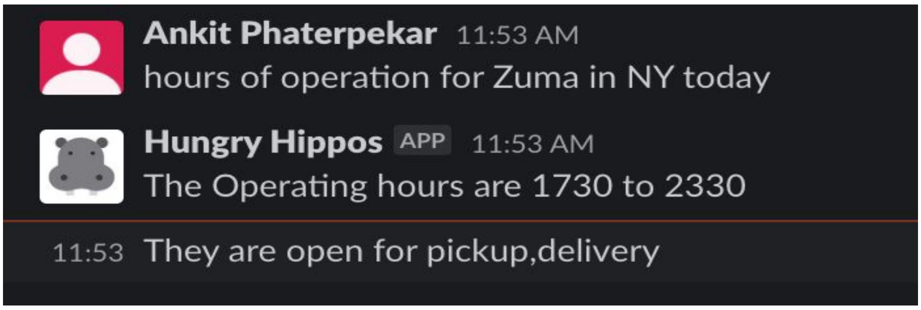


Fig J. Slack Output Response for Business Details query

Use Case 2: Search for Events

Here we wanted to design a functionality that would allow the user to search for nearby Events from the Yelp directory. For this feature we query the Yelp Events endpoint

Yelp has multiple categories to search from, the default category that we search for an event if none is provided by the user is 'food-and-drink' category.

The events are sorted by popularity, other options available are "time-sort". Defacto, we search for events occurring within the next 24 hour time range. This can be extended to do an unbounded search as well for any time range.

We grab the first event, from the Yelp response and retrieve Event Name, Event Description, Event Location and build the agent response.

Slack Sample Interaction:

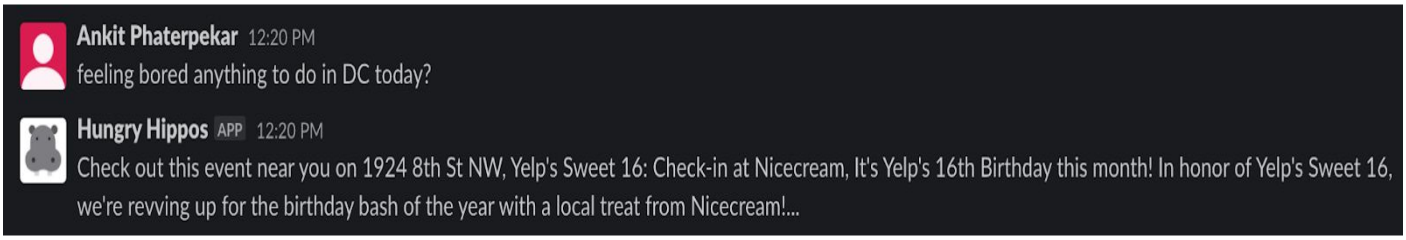


Fig K. Slack Output Response for Event search query

This pulled the correct information that we can validate from on the Events page.

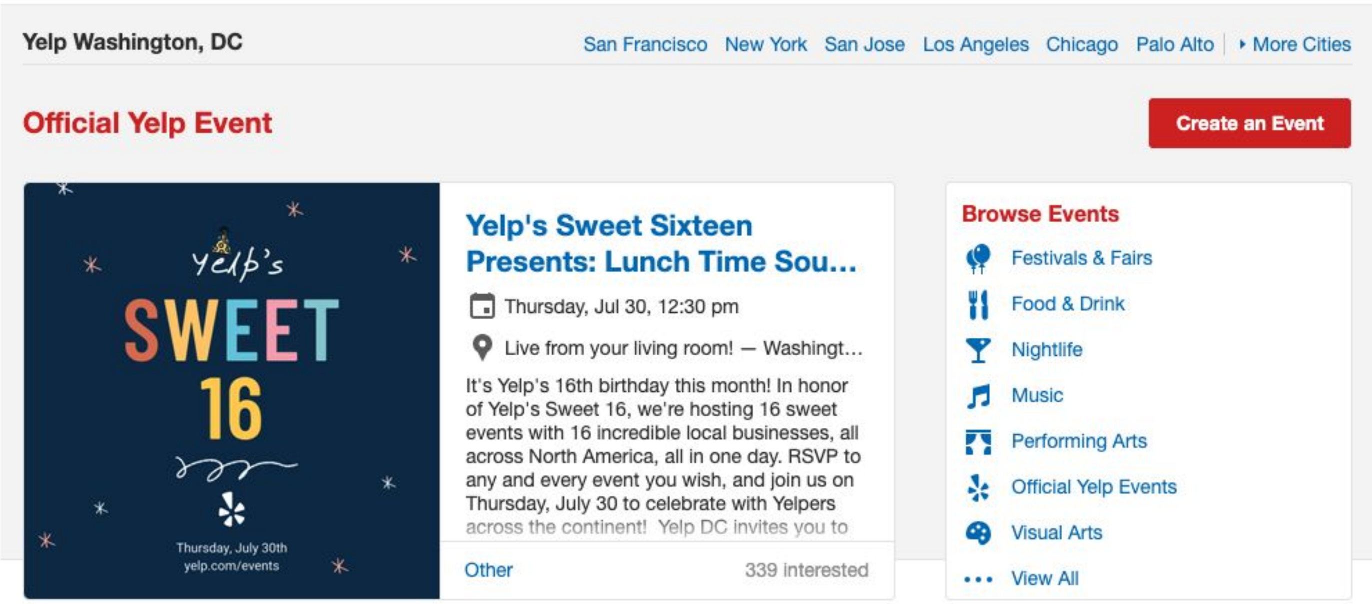


Fig L. Yelp Events page for Washington DC, sorted by popularity on July 30th, 2020

Use Case 3: Reserve a Table

Here we wanted to design a functionality that would allow the user to initiate a conversation for reserving a table. **For this intent we have set all the entities within the training phrases i.e. guests, date, time as required parameters.** This means that agents will not close this particular conversation until all the required params are fetched from the user responses which is also called as slot filling.

When Slot filling is triggered the agent detects which parameter is missing from the user response, and comes back with the followup question for the user to request the missing details, before initiating the webhook call to the cloud function for further processing.

Slot Filling is desirable when we know in advance that a certain action, api request or code logic as defined in the Cloud function would fail due to a missing parameter/entity, and we would want to avoid our chatbot agent from crashing or returning unintended results.

Intent	**Sample Training Phrases**	**Functional name in Dialogflow/ Cloud Function**
Reserve a table	book a table for part of 10 for August 7th at 6pm	book_table
	reserve a spot for 4 for Joes Kitchen for saturday morning at 7 pm	
	plz make reservations for 5 at 9pm tomorrow	
	book a table tomorrow at 9pm	
	I need to book a table	

Sample Slack Interaction

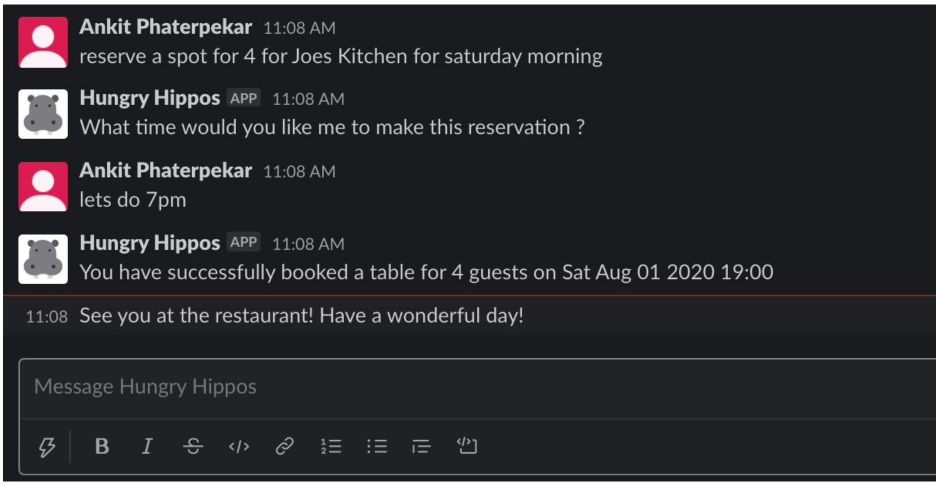
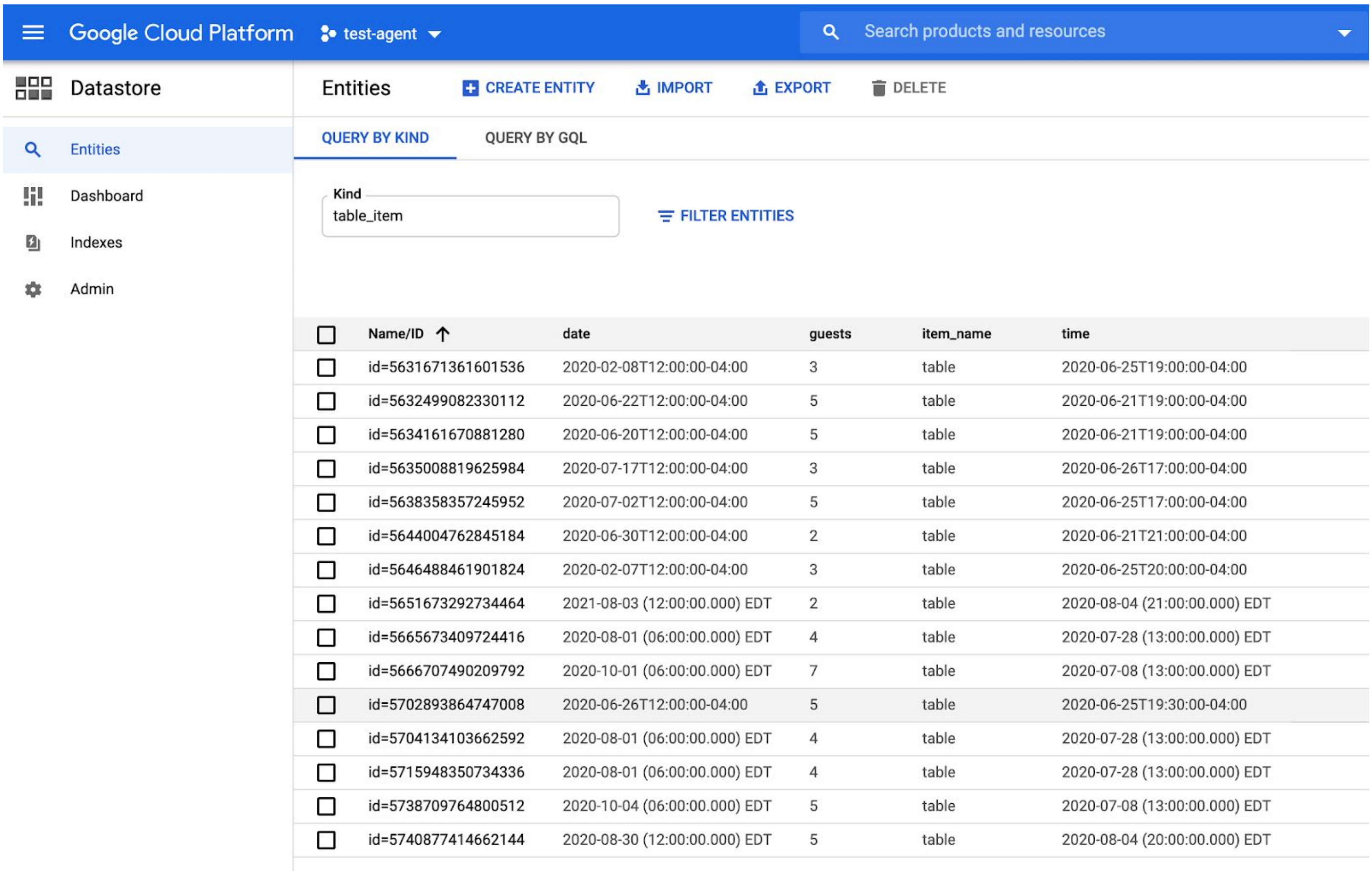


Fig M. Slack Output Response for Reserve Table query

Once the request is successfully completed we store the reservation date in the Google Cloud Datastore table that we created for storing the reservation dates. The cloud Datastore is invoked within the Cloud Function defined for processing this intent.



Name/ID	date	guests	item_name	time
id=5631671361601536	2020-02-08T12:00:00-04:00	3	table	2020-06-25T19:00:00-04:00
id=5632499082330112	2020-06-22T12:00:00-04:00	5	table	2020-06-21T19:00:00-04:00
id=5634161670881280	2020-06-20T12:00:00-04:00	5	table	2020-06-21T19:00:00-04:00
id=5635008819625984	2020-07-17T12:00:00-04:00	3	table	2020-06-26T17:00:00-04:00
id=5638358357245952	2020-07-02T12:00:00-04:00	5	table	2020-06-25T17:00:00-04:00
id=5644004762845184	2020-06-30T12:00:00-04:00	2	table	2020-06-21T21:00:00-04:00
id=5646488461901824	2020-02-07T12:00:00-04:00	3	table	2020-06-25T20:00:00-04:00
id=5651673292734464	2021-08-03 (12:00:00.000) EDT	2	table	2020-08-04 (21:00:00.000) EDT
id=5665673409724416	2020-08-01 (06:00:00.000) EDT	4	table	2020-07-28 (13:00:00.000) EDT
id=5666707490209792	2020-10-01 (06:00:00.000) EDT	7	table	2020-07-08 (13:00:00.000) EDT
id=5702893864747008	2020-06-26T12:00:00-04:00	5	table	2020-06-25T19:30:00-04:00
id=5704134103662592	2020-08-01 (06:00:00.000) EDT	4	table	2020-07-28 (13:00:00.000) EDT
id=5715948350734336	2020-08-01 (06:00:00.000) EDT	4	table	2020-07-28 (13:00:00.000) EDT
id=5738709764800512	2020-10-04 (06:00:00.000) EDT	5	table	2020-07-08 (13:00:00.000) EDT
id=5740877414662144	2020-08-30 (12:00:00.000) EDT	5	table	2020-08-04 (20:00:00.000) EDT

Fig N. GCP Cloud Datastore stores the reservation

Use case 4: Placing an Order

 # Reference : https://www.youtube.com/channel/UCmBGWgaTx1_KwbNm2PJF0_A

Ordering a Pizza

1. Asking details about configuration of Pizza

   Some of the training phrases that could trigger the intents to start the order are:

Operation	Training Phrases	Intent
Start Order	“Hi” or “Order”	item.start.generic
Configure Pizza	“Can I have 4 large pepperoni?”	item.start.generic
Enquiring about a particular type of pizza	“Do you have Americanas”	item.type.start.positive

Generally, while ordering a pizza in a restaurant, we generally enquire whether a particular type of Pizza is available and if yes, then we could start our order.

This action is mimicked by our bot via two custom entities namely `@pizza-type` and `@pizza-type-no`. `@pizza-type` consists of all the pizza types that can be offered whereas `@pizza-type-no` contains all the pizza types that are not offered by the restaurant.

When the customer asks the bot for a pizza type that is on the `@pizza-type` list, it will be matched to via *item.type.start.positiv*e,applying `@pizza-type` and the user is asked whether they want to order this type of pizza. The slot filling process is completed in the follow-up intent *item.type.start.positive* and the configuration of pizza is thus obtained.

In case, if a particular type of pizza is not available, this will be matched via *item.type.start.negative*, applying `@pizza-type-no`.

2. Confirmation of Item

When a pizza is configured, the bot will repeat the current configuration and confirms if the user is happy with his current order.

If the user confirms his order, then the order is added to the shopping cart/basket and the user is guided throughout the rest of the conversation by asking whether the user wants to place an additional order. If yes, then the bot can start configuring another pizza. If the user doesn’t want to add anything to the basket, the bot asks if they want to finish the order. If the user says yes, that is the end of the conversation and bot will reply "Hurray! you have completed your order".

If the user is unhappy with the current order, before confirming it, he can change the configurations of the pizza.

Modifying the Pizza Configuration

The configurations of a pizza includes the type of pizza, amount of pizza/number of pizza to be ordered and the size of the pizzas. These configurations can be changed in a number of ways.

For example, if the user query to the bot is " 4 small Pepperoni please". Post confirming this configuration, the *item* context is set. While the *item* context is still intact, the configuration of the most recent order can be changed by the below mentioned intents.

Operation	**Training Phrases**	**Intent**
Change type of Pizza	"Can I have a Margherita instead?"	item.type.change
Change amount of Pizza	"I want 3"	item.amount.change
Change size of Pizza	"Can I have large instead"	item.size.change

Contents of the Basket

In order to mimic a shopping cart, we have added a shopping basket functionality to our bot. Every time a user confirms the pizza configuration, the order is added to the basket.

An overview of the intents that help in modification of the basket is as shown below

Operation	**Training Phrases**	**Intent**
Finish the order	“I am done with my order”	order.finish
Remove an item from the basket	“Please remove pepperoni”	item.remove
Modify an item configuration	“Can I get a Mushroom instead”	item.type.change , item.size.change, item.amount.change
Delete the basket	“Delete basket”	order.cancel
Start the order again post deletion	“Start order”	order.start
View items in basket	“Show basket”	order.showbasket

1. Structure of Basket

The basket is a context that is programmatically set in the fulfillment. The items themselves are then passed along through the parameters of the context.

```
basketContext = {'name': 'basket', 'lifespan': 50, 'parameters': {}}
```

The items in the basket could just be an array of objects. However, the bot uses an object as a data structure, due to the fact that the Google Cloud products were designed for concurrency and removing or adding items from the array could result in inconsistent states.

```
"items": {
```

"900f2fb2-9ef7-47ea-bb96-3c2b6eb64131-0820055c": {
    "amount": 2,
    "type": "Margherita",
    "size": "14''"
},
"e6b0da6a-1186-452d-b263-31ea401d080d-0820055c": {
    "size": "10''",
    "type": "Pepperoni",
    "amount": 1
}

Since the basket is a list of items, we are simulating an array in that object. For the keys, ResponseID of Dialogflow conversation is used which serves as an unique ID.

2. Modifying the Last Item in Basket

   The bot provides flexibility of changing the last item that was added . This is taken care of by the *confirmItem()* method in the fulfillment. If an item exists in the basket, then the record in basket is updated and if it doesn't exist, a new record of item is created in the basket. The confirmItem method adds the response ID to the item context, so that it can be used to get access to that item in the basket when the user wants to change its configuration later.

3. Removing a specific item

   In case the user wants to remove a specific item from the basket, he can mention the type of Pizza that he wants to remove. For example, “Remove Pepperoni”. The particular item is deleted from the basket and the remaining items are retained. This is taken care of by the *removeItem()* method in Fulfillment. The method iterates through the items present in the basket and removes those items based on the type of pizza provided by the user.

4. Deleting the basket

   The basket can be emptied and reset if the user wants to start over the order again thus triggering the intent *order.start*. The basketContext can be deleted by setting the lifespan to 0.

```
 let basketContext = {'name': 'basket', 'lifespan': 0};
agent.context.set(basketContext)
```

Use case 6: Helping the User

There are no user interface elements in the bot that can guide the user on how it is intended to be used. So, at certain points in the conversation, the user should be able to ask for help and get information on the conversation flow.The bot fullfils this requirement of the user by matching the “help” query to different intents by using contexts.

Point in conversation	**Context**	**Intent**
After adding the item	*item*	help.itemadded
Post placing an order	*order-done*	help.orderfinished

1. Fallback in Help

   There is a special help intent: help.onlyonepizza

   The users might try to order two pizzas in the same query.Currently, the bot has no way of dealing such a case.Thus, it outputs a simple message letting the users know that they can place only one order.

Use case 7: Small Talk

Smalltalk intents were added to make the bot more conversational. These smalltalk intents can be extended,depending on what conversational style we want our bot to achieve.

1. Context-Dependent Smalltalk

   These intents are context dependent. The user gets different responses depending on where he is in the conversational flow. For example, after adding an item, the user says “Great!”, the *smalltalk.user.positive.itemdone* intent is matched through the *item* context.This is interpreted as a sign that the user is happy about having a pizza added to their basket.

2. Muti-Step Smalltalk

   After placing the order,if the user is unhappy with it and types “no!”,the *smalltalk.user.negative.inorder* intent is triggered.The bot tries to pacify the user by telling him that he can always modify his order. Depending on what the user says next, the bot goes deep into the branch of the conversation, by going into different followup intents.

4. Integrations

Once the Dialogflow agent is ready, we can use the one-click integrations to connect to various platforms. The different types of integrations offered are:

<u>Text based Integrations:</u>

1. Slack:

   The user can create a new slack bot and enable integration with Dialogflow via the Client ID, Client Secret and Verification Token that could be found in the Slack developer console.

   Additionally ,OAuth URL from Dialogflow’s Slack setting page, Event Requests URL and Event subscriptions need to be enabled to complete the setup.

   Finally, the slack bot can be added to a team and can be tested.

   Even Though Slack can support a variety of rich responses it is impossible to display multiple card responses unlike Google Assistant.

<u>Telephony based Integrations:</u>

Google’s Dialogflow provides telephone interfaces to the agent via Telephony gateway ,Voximplant, Signal Wire etc. This could facilitate in building conversational IVR which could be integrated with the rest of the call center network.

1. Dialogflow’s Phone Gateway:

   The phone gateway could be configured to get a unique number provided by Dialogflow. A phone call to this number will be answered by the Dialogflow agent that is integrated with the phone gateway thus becoming a conversational IVR. Rich response messages like audio files, call transfer can be used by the intents for phone gateway responses. However, external noise and interferences might affect the performance of the agent.

2. Voximplant:

   Voximplant is a third party telecommunications provider which offers greater flexibility than Google’s telephone gateway.It provides outbound and inbound calling functionality, advanced interaction such as recording, audio playbacks and SMS. However, it is not possible to integrate the agent with both Voximplant and Google Assistant as such flexibility is not offered by Google’s Dialogflow.

<u>Google Assistant/Google Home:</u>

Google Assistant is an artificial intelligence powered virtual assistant developed by Google that is primarily available on mobile and smartphone devices.It offers a wide range of voice controls.

Dialogflow agent can be integrated with Google Assistant by one-click integration which will redirect the users to Actions console page where they can publish the agent for wider distribution and availability. The agent can be tested using the simulator provided by Google. The agent can be deployed to an Alpha/Beta release channel and tested via any device that supports Google Assistant i.e Google Home, Android phone, etc.

Google Assistant can support both text as well as voice based commands unlike Slack. Through the assistant, the agent is exposed to a wider range of services like Google Maps, Phone, Mail, Health tracking apps in Android, thus leveraging its application.

The detailed steps on integration of Dialogflow agent with the above mentioned platforms can be found in the **Appendix 5**.

5. Testing/Evaluation

We tested the efficiency of the chatbot for both voice and text interactions. In particular we looked at the following:

1. Simulated Conversation Testing:

For text, we checked if the intents were recognized, which dives were better at recognizing the intents, even if the certain key phrases were missing or were misspelt.

Text Based Interactions: With Slack, we found that Dialogflow was able to overcome grammatical errors at least 50% of the time and still successfully detect the intent. We found that text based systems were a bit more error prone when it came to the intents not getting mapped when provided with malformed texts. In particular when it failed we found that the queries would not get mapped to an intent due to a typo or if the text query had some grammatical errors. (**Note**: Dialogflow has a feature for auto-grammar correction, but that seemed to be purely experimental and we did not find that to be very helpful here)

Voice Enabled Interactions: For Voice enabled devices we tested with devices like Google Assistant on a Pixel 2 phone as well as on Google Home. During the initial iterations we tested with the Google Assistant Simulator on Dialogflow console. The simulator returns a JSON object holding information regarding the chatbots understanding level identifying whether the utterance was matched to an intent.

We found voice interaction to be more robust to malformed queries or missing key terms. We believe this is due to the fact that additional ML steps are executed while converting speech-to-text which corrects some of those inconsistencies and detect incomplete words.

In Summary we preferred Google Home as our prime mode for interaction

2. Response Times:

Average response times between text and voice interactions were also recorded.

We found the Slack Interactions were slightly faster as compared to those on Google Home. With Slack our average response times were < 1 sec, while with Google Home it was between 1-2 secs and some intents taking around 3-4 seconds to deliver the response. This makes sense as Slack interactions only use text-to-text detection while interactions with a Google Home device needs an additional step of converting Speech-to-text.

This was probably also due to the fact that some of our intents queried the Yelp API endpoint, which has its own response lag associated with it when fetching that data and formatting it for our use. Building a database schema for fetching structured data, to specifically service the Dialogflow Use Cases would certainly help reduce the burden and computation time spent on formatting the raw data and reduce the response times. Nevertheless, we found the response times with Google Home to be reasonably satisfactory.

Validation Console, Dialogflow CLI

Due to the limitation on the amount of training data to be had manually entered and annotator bias, we believe the best way to accumulate testing metrics and improve on this initial framework for an MVP would be to let a few users interact with the agent via an alpha or beta release.

We would expect the initial testing phases with these alpha users to uncover most of the failures and allow us to add more heterogeneity to the vocabulary and training phrases. Using that and additional data augmentation approaches would allow us to build a more robust system.

One neat feature in Dialogflow is that it has a Training/Validation page which allows us to view all the incoming requests to our agent from every device and see which requests correctly mapped and which failed. We can also simply approve and add the user queries that failed into our our desired intents as training phrases. (See Appendix 7)

6. Project planning and Development Cycle Breakdown

Week	Milestone Description	Tasks Completed

Week 1	Analysis & requirement specifications	1. Researching existing NLU systems: Dialogflow, Amazon Lex, Rasa 2. Looking for available training data sets for NLU (SNIPS dataset) **Outcomes**: 1.We decided to go with Dialogflow for building our agent.
Week 2	Design presentation	1.Gathered available resources for Dialogflow 2. Explored use cases for building our agent **Outcomes**: 1.Initial Dialogflow Setup. 2.Integration of Dialogflow with primary GCP billing account to link up GCP components like Cloud Function, Firebase, etc.
Week 3	Iteration 1	1.Explored open APIs like Places API, OpenTable, Yelp Fusion, Resy, OpenMenu for accessing Menu, reservation systems (emailed Resy API for developer access, no response) 2. Built a Slack Developer account and Integrated the agent to Slack channels and as a personal bot. 3. Tested a base use case with Google Assistant **Outcomes**:

		1.Realized access to Menus and Reservations systems were only available to partners and affiliates. **(roadblocks)** 2.Finalised Yelp Fusion API and Places API as they were open source and free of cost 3. Successfully tested integrating a single test intent with Yelp Fusion API to send a request and fetch the desired response.
Week 4	Iteration 2	1. Defined our Use Cases and their corresponding intents 2. Started Designing Flow Diagrams for each of our Use Cases. (See **Appendix 7**)
Week 5	Iteration 3 & Status update 1	1. Integrated our agent with Phone Gateway 2. Explored other use cases that could be implemented 3**.** Built a list of Custom Entities to be used by all intents, helped the model for detecting these occurrences and reduced the burden to manually add training phrases that contain such entity occurrences in each intent. **Outcomes**: 1.Faced performance issues with Phone Gateway integration

		2. Further expanded on integration with Google Assistant 3.Build intents,training phrases for fetching details about events happening near a particular location 4. Build richer workflows, added more training phrases to each of our intents to mitigate conflicts while detecting overlapping intents
Week 6	Iteration 4-Final build & Status update 2	1. Deployed the agent on Google Home by initiating an Alpha release. 2. Explored Voximplant to check the feasibility of the agent placing an outbound call **Outcomes:** 1. Voximplant couldn't be integrated with Google Assistant and the use case was dropped 2. Final agent was successfully deployed on Google home and
Week 7	Testing & Evaluation	Refer **Testing & Evaluation** section for more details
Week 8	Packaging & Documentation	Refer **Packaging & Documentation** section for more details

7. Limitations & Future Advancements

The goal of our project was to build a restaurant bot that could automate different restaurant services and could be integrated with different platforms. However, there are some roadblocks that we faced while building the agent..

1. Lack of flexibility to integrate Voximplant with Google Assistant: To initiate an outbound call, Voximplant or Dialogflow's telephony could be used. However, these apps cannot be integrated with Google Assistant as such features are not yet available.
2. Manual work involved in Training : In many instances, Dialogflow makes it harder than it should be to automate processes and expand your conversational agent's learning. This can get annoying because you have to input many things manually, especially when you consider the need to train your bot over time. Therefore, the ability to even make your chatbot better is hindered, which unfortunately defeats one of the platform's main purposes
3. API server return error: We use the Yelp fusion API to fetch details about different businesses. Sometimes, the server might throw a '500 status return error' when it encounters an unexpected condition which prevents it from fulfilling the request from a client. This could be triggered by different issues like error in scripting, failed disk or non-functional software module.
4. Bias in Training Phrases - Since the developer is defining the training phrases for each intents, there is a creator bias. Due to this, the training phrases are more tuned to the developer's vocabulary and might fail to capture the heterogeneity in different users' queries. This could be overcome by data augmentation where we can explore simulating more training phrases using synonyms and verb replacement strategies. Large selection of custom entities can be created to mitigate the limitations of less training phrases.

   We expect that Alpha/Beta release will help us in capturing the failure due to lack of diversity in training phrases and make our agent more robust.

8. Exporting and Packaging Agent

1. The agent can be backed up or transferred to a different account via the Import and Export settings tab, found in Agent settings. This feature allows us to export the agent and its intents,entities into a zipped folder that can be uploaded on Github.
2. The cloud functions can be exported separately and reimported within the end user's Dialogflow console.
3. A documentation regarding the workflow for each intent and the required parameters can be maintained along with the steps for integration for future reference.

9. Pricing of Dialogflow Services

1. The Dialogflow is available in two pricing plans: Essentials and Plus. Dialogflow is priced monthly based on the Edition and Pricing plan in addition to the number of requests made. The main difference between Essentials and Plus plans is that the Plus plan offers enterprise-ready quotas for knowledge connectors.
2. It is to be noted that a request is defined as any call to the Dialogflow service,whether direct via API usage or indirect with integration or console usage.Unless a feature is indicated as included, prices are cumulative for all features used by a request.

   Refer **Appendix 6** to get more information on price comparison for the two editions and pricing plans.

10. Conclusion

Thus,an end to end conversational agent was built and deployed using Google's Dialogflow platform that could give real time information on businesses and events happening near a specific location. The vast amount of research that has been carried out along with the evolution of intelligent and sophisticated conversational agents like Dialogflow , Amazon's Lex , would prove to be immensely beneficial in providing convenient and accessible customer service at a rapid scale.

11. Future Advancements

Dialogflow has a growing developer base and is striving hard to add value and remove mundane steps in communication and collaboration in the future. Combining Dialogflow with Amazon's Lex or other NLU might offer an edge over other chatbots in terms of better intent detection and collaboration with a wide range of devices.

Knowledge base features have also shown promising results in detecting intents of simple requests like FAQs thereby eliminating the need to create Intents and training phrases for such trivial cases in the future.

12. References

1. https://cloud.google.com/dialogflow/docs
2. https://www.coursera.org/learn/conversational-experiences-dialogflow
3. https://firebase.google.com/docs/database
4. https://www.youtube.com/watch?v=Z3SR3gS8rF4&list=PLJLSPq0cTRmYmsO3kXuN-a2sg3ruoZV6x
5. https://javascript.info/
6. https://www.yelp.com/developers/documentation/v3
7. https://chatbotslife.com/building-chatbots-using-dialogflow-on-google-assistant-for-beginners-1e91cf355abb?gi=7445dfa68548
8. https://cogint.ai/3-methods-for-connecting-a-phone-call-to-dialogflow/
9. https://comparecamp.com/dialogflow-review-pricing-pros-cons-features/
10. https://link.springer.com/chapter/10.1007/978-1-4842-5741-8_2

APPENDIX 1

Importing an agent from exported files

Install Dialogflow Agent

1. Go to the Google's Dialogflow Console using your Google credentials : https://dialogflow.cloud.google.com/.
2. Click on 'Create New Agent" from the agent selector dropdown.
3. Give it a name "Hungry Hippos" or "test-agent" and click "Create".
4. Now go to Agent settings which is a small icon near the agent selector dropdown.
5. Select "Export and Import" tab.
6. Click on "RESTORE FROM ZIP" and upload the test-agent.zip file provided.

Install Fulfilments Functions

1. Click on the ***Enabled*** switch on the Inline Editor in the **Fulfillment** Tab of the Dialogflow Console.
2. Copy the contents of index.js and package.json into the Inline Editor window

Connect to Firebase Database/GCP Console

Enable Billing Account to link the Dialogflow Agent to access the services provided by Google Cloud.

APPENDIX 2.

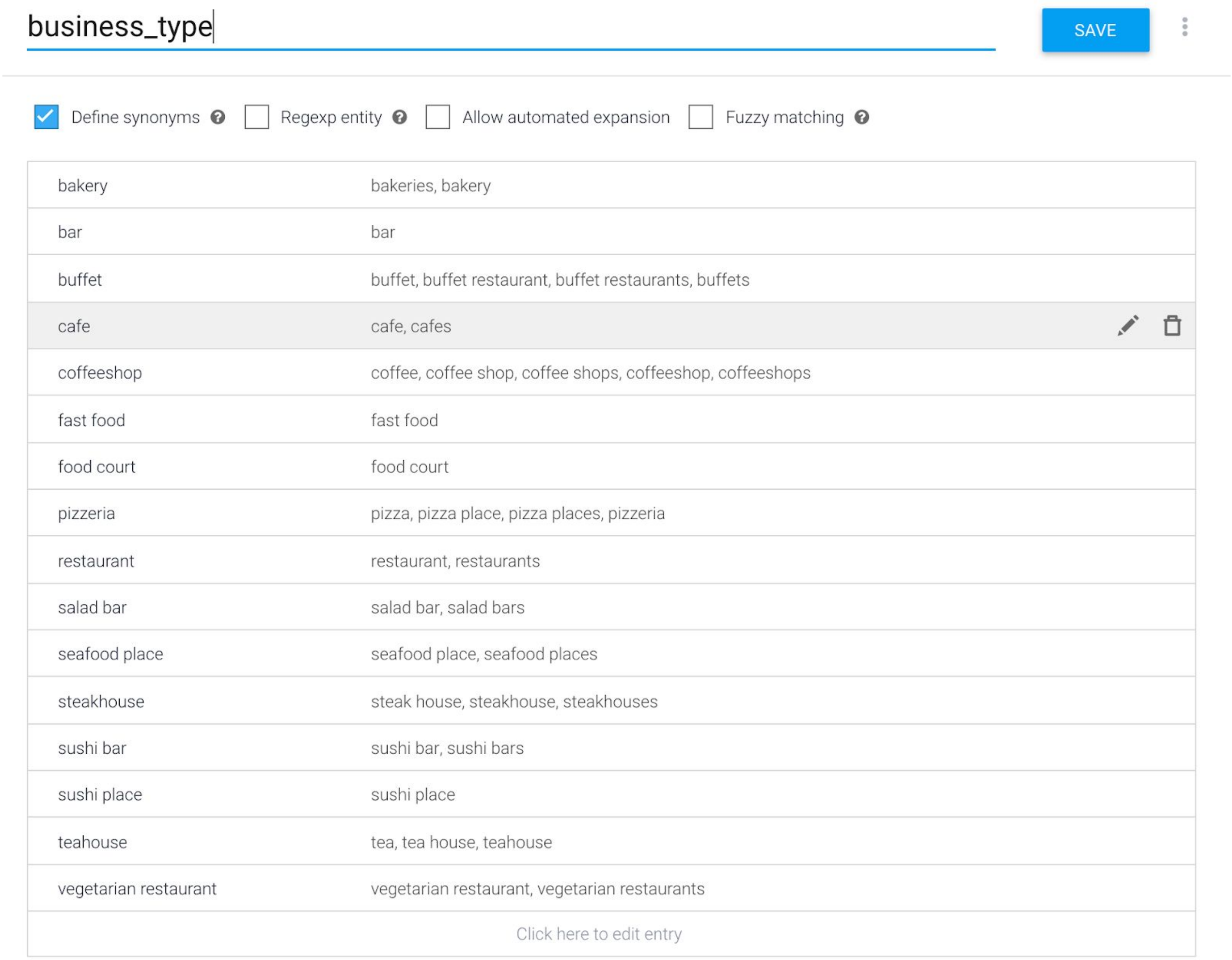
Use Cases and their corresponding intents.

- Welcome : Default welcome intent for the user
- Fallback : Default intent when any intent is not matched
- Reserve a table : book_tables
- Yelp business information: search_business, search_business.details, restaurant Info, get_restuarant
- View events/specials : do_event_search
- Place an order : item.start.generic, item.type.change , item.size.change, item.amount.change, item.remove, item.confirm.yes, item.confirm.no, item.type.start.negative, item.type.start.positive, order.start, order.finish, order.cancel, order.showbasket
- Small Talk: smalltalk.user.negative.inorder, smalltalk.user.positive.itemdone

APPENDIX 3.

We created the following entity lists at a global level, which are available to all of our intents for detection in the training phrases and new user queries. These reduce the burden of manually training too many training phrases with different entity names to detect their occurrences.

Business Type predefined entity list



business_type

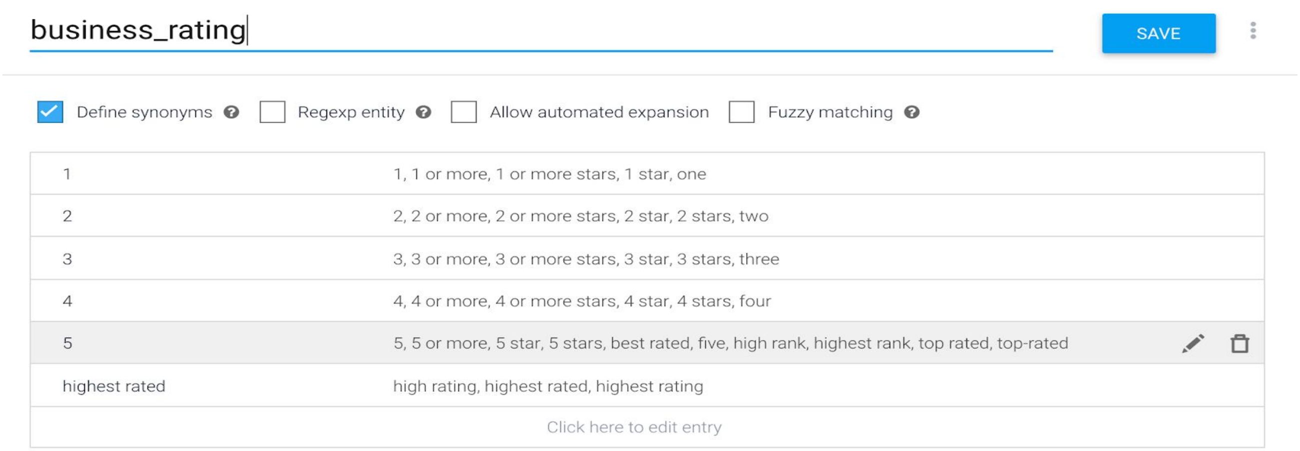
SAVE

Define synonyms | Regexp entity | Allow automated expansion | Fuzzy matching

bakery	bakeries, bakery
bar	bar
buffet	buffet, buffet restaurant, buffet restaurants, buffets
cafe	cafe, cafes
coffeeshop	coffee, coffee shop, coffee shops, coffeeshop, coffeeshops
fast food	fast food
food court	food court
pizzeria	pizza, pizza place, pizza places, pizzeria
restaurant	restaurant, restaurants
salad bar	salad bar, salad bars
seafood place	seafood place, seafood places
steakhouse	steak house, steakhouse, steakhouses
sushi bar	sushi bar, sushi bars
sushi place	sushi place
teahouse	tea, tea house, teahouse
vegetarian restaurant	vegetarian restaurant, vegetarian restaurants

Click here to edit entry

Business Rating predefined entity list



business_rating

SAVE

Define synonyms | Regexp entity | Allow automated expansion | Fuzzy matching

1	1, 1 or more, 1 or more stars, 1 star, one
2	2, 2 or more, 2 or more stars, 2 star, 2 stars, two
3	3, 3 or more, 3 or more stars, 3 star, 3 stars, three
4	4, 4 or more, 4 or more stars, 4 star, 4 stars, four
5	5, 5 or more, 5 star, 5 stars, best rated, five, high rank, highest rank, top rated, top-rated
highest rated	high rating, highest rated, highest rating

Click here to edit entry

Cuisine Types - Predefined entity list

cuisine

SAVE

☑ Define synonyms ☐ Regexp entity ☐ Allow automated expansion ☐ Fuzzy matching

Search entries

1 OF 3

Cajun Creole	Cajun Creole
Cethelsea	Cethelsea
Czech Slovakian	Czech Slovakian
Dairy	Dairy
Diners	Diners
Himalayan Nepalese	Himalayan Nepalese
Latin American	Latin American
Persian Iranian	Persian Iranian
Rooftop	Rooftop
Soul food	Soul food
Tibetan	Tibetan
afghan	afghan
african	african
american	american
american new	american new
american traditional	american traditional

Business Facility - custom defined entity list

business_facility

SAVE

☑ Define synonyms ☐ Regexp entity ☐ Allow automated expansion ☐ Fuzzy matching

beauty salon	beauty salon, hair and nail salon
delivery	deliver, delivery
indoor	indoor
internet	acceess to internet, free internet, internet
non-smoking room	non smoking room, non-smoking room, nonsmoking room
outdoor seating	outdoor seating, outdoor seats
parking	available parking, free parking, parking, parking lot
pet friendly	cat friendly, dog friendly, pet friendly
pool	pool, swimming pool
quick serve	quick serve, quick service, quick serving
smoking room	smoking room
spa	spa
transfer	transfer, transfer service, transportation
wifi	access to wi-fi, access to wifi, free wi-fi, free wifi, wi-fi, wi-fi connection, wifi, wifi connection

Click here to edit entry

Business/Establishment Names (We added a total of 10000 names)

business_ent

SAVE

☑ Define synonyms ☐ Regexp entity ☐ Allow automated expansion ☐ Fuzzy matching

Search entries

1 OF 196

Italian Pizza Kitchen	Italian Pizza Kitchen
bar	bar
grocery	grocery
Olive Bistro	Olive Bistro
Guapo's	Guapo's
Donuts	Donuts
1833	1833
241 Pizza	241 Pizza

APPENDIX 4.

Cloud function that will execute in GCP when Business_Search Intent is triggered

```
 // Business Search

 function business_search(params){

   return client.search(params)

 };

 // Reviews search

 function reviews(params){

   return client.reviews(params)

 };


async function search_business(agent){

   // agent parameters

   console.log(agent.parameters.user_location);

   console.log(typeof agent.parameters.user_location);

   console.log(agent.parameters.business_name);

   console.log(typeof agent.parameters.business_name);

   var business_name = JSON.stringify(agent.parameters.business_name);

   var business_location = agent.parameters.user_location;

   console.log(typeof business_name);

   console.log(typeof business_location);

   // Get Business info from Business Endpoint

   try {

   const bus_resp = await business_search({term: business_name , location:business_location});

   console.log("Got the business info")

   // Get the First Business from the response

   const firstResult = bus_resp.jsonBody.businesses[0];

   const id = firstResult.id;

   const resolved_name = firstResult.name;
```

```
// Get the Review for the first Business
const reviews_body = await reviews(id);
const reviews_list = reviews_body.jsonBody.reviews;
// Process the review List
const text_0 = reviews_list[0].text;
const rating_0 = reviews_list[0].rating;
const time_created_0 = reviews_list[0].time_created;
const text_1 = reviews_list[1].text;
const rating_1 = reviews_list[1].rating;
const time_created_1 = reviews_list[1].time_created;

const text_2 = reviews_list[2].text;
const rating_2 = reviews_list[2].rating;
const time_created_2 = reviews_list[2].time_created;

//  Extract desired params
const rating = firstResult.rating;
const address = firstResult.location.display_address.toString();
const review_count = firstResult.review_count;
const phone = firstResult.display_phone;
const image_url = firstResult.image_url;
const yelp_url = firstResult.url;
const price = firstResult.price;
// Logging
console.log(firstResult)
console.log(rating)
console.log(address)
console.log(reviews_body)
console.log(rating_0, text_0)
```

```
// Prepare Agent Card
   agent.add(`${resolved_name} is located at ${address}. It's been reviewed ${review_count} times and has an average rating of ${rating} stars.`);
   agent.add(new Card({
         title: resolved_name,
         imageUrl: image_url,
         text: `Phone: ${phone} \n Price: ${price}`,
         buttonText: 'View Yelp page',
         buttonUrl: yelp_url
       })
     );
      agent.add(`💁 Here are some Reviews  \n ${time_created_0} : ${text_0} ★ ${rating_0} \n\n ${time_created_1} : ${text_1} ★ ${rating_1} \n\n ${time_created_2} : ${text_2} ★ ${rating_2}`)
   } catch(err){
     console.log(err)
     agent.add("Oops ! Something went wrong. Please try again.")
   };
 };
```

APPENDIX 5.

Steps for integrating Dialogflow with various platforms.

1. Slack - https://cloud.google.com/dialogflow/docs/integrations/slack
2. Phone Gateway - https://cloud.google.com/dialogflow/docs/integrations/phone-gateway
3. Voximplant - https://cloud.google.com/dialogflow/docs/integrations/voximplant
4. Google Assistant- https://developers.google.com/assistant/conversational/df-asdk/overview

APPENDIX 6.

Pricing details.

1. **Dialogflow services:**

Features	Standard Edition	Enterprise Edition
Text	Free *(180 requests per minute)*	$0.002 per request *(600 requests per minute)*
Speech recognition Speech-to-text	Free • *100 requests per minute* • *1,000 requests per day* • *15,000 requests per month*	$0.0065 per 15 seconds of audio • *300 requests per minute*
Speech synthesis Text-to-speech	Free	$0.0065 per 15 seconds of audio • *Standard voices: $4 per 1 million characters* • *WaveNet voices: $16 per 1 million characters*

2. **GCP Services:**

	Free limit per day	Price above free limit (per unit)	Price Unit
Stored data	1 GB storage	$0.099	GB/Month
Entity Reads	50,000	$0.033	per 100,000 entities
Entity Writes	20,000	$0.099	per 100,000 entities
Entity Deletes	20,000	$0.011	per 100,000 entities
Small Operations	50,000	Free	-

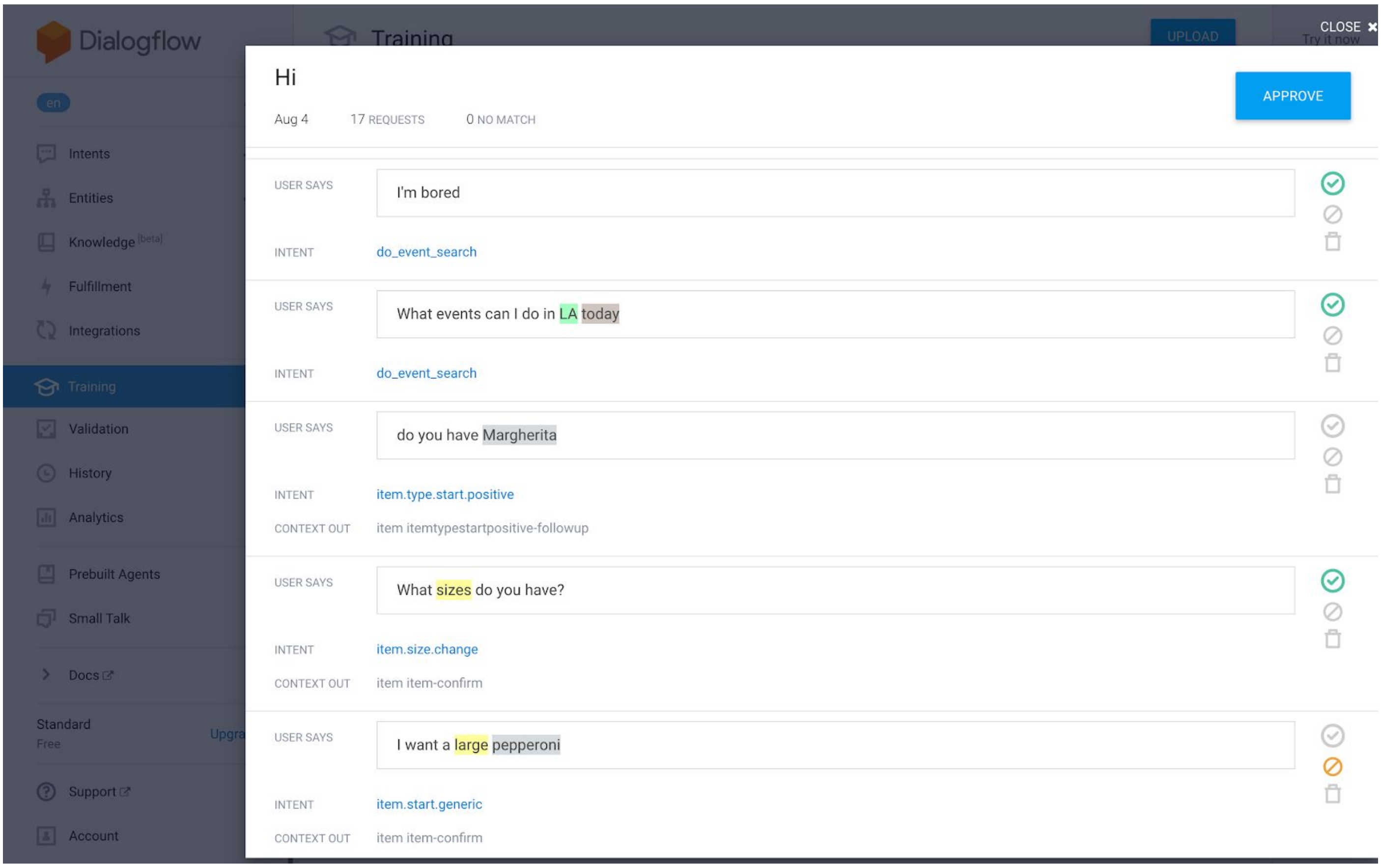
For more details, please refer the below links:

GCP Firebase packages

Agent GCP Billing Console

APPENDIX 7.

View Incoming user requests from all devices interacting with the agent.

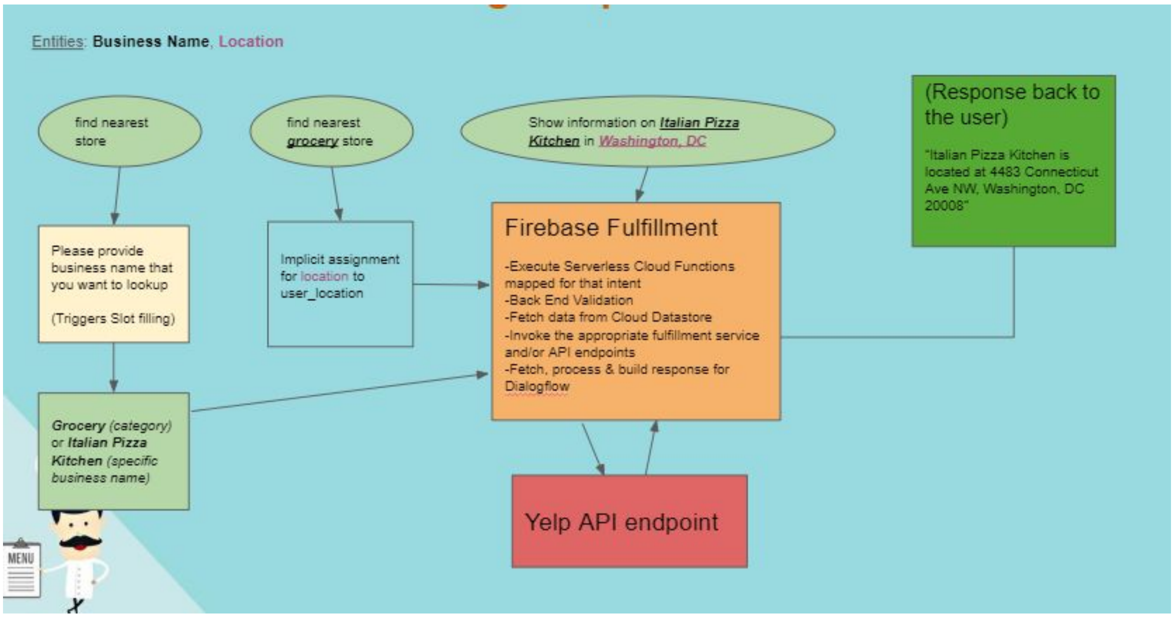


APPENDIX 8.

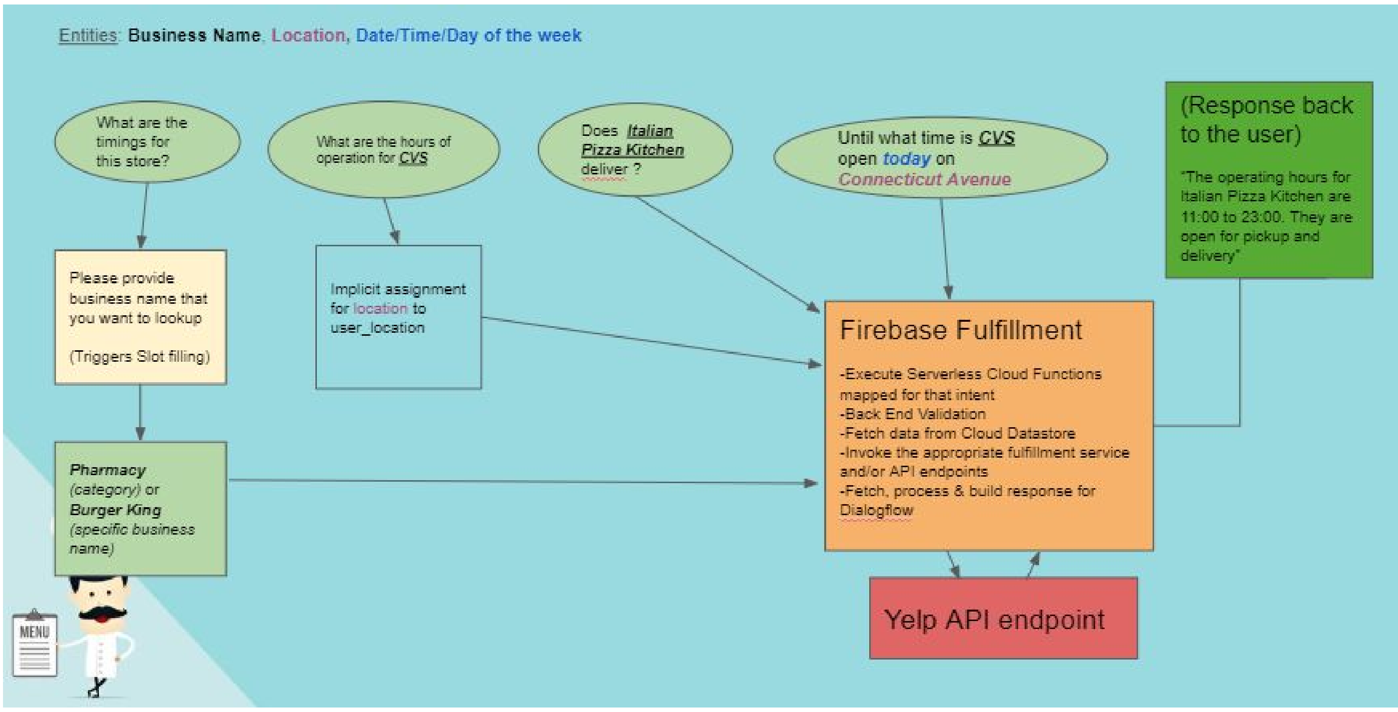
Conversational Flow Diagrams for our Use Cases

1. Get business details via Yelp Fusion API

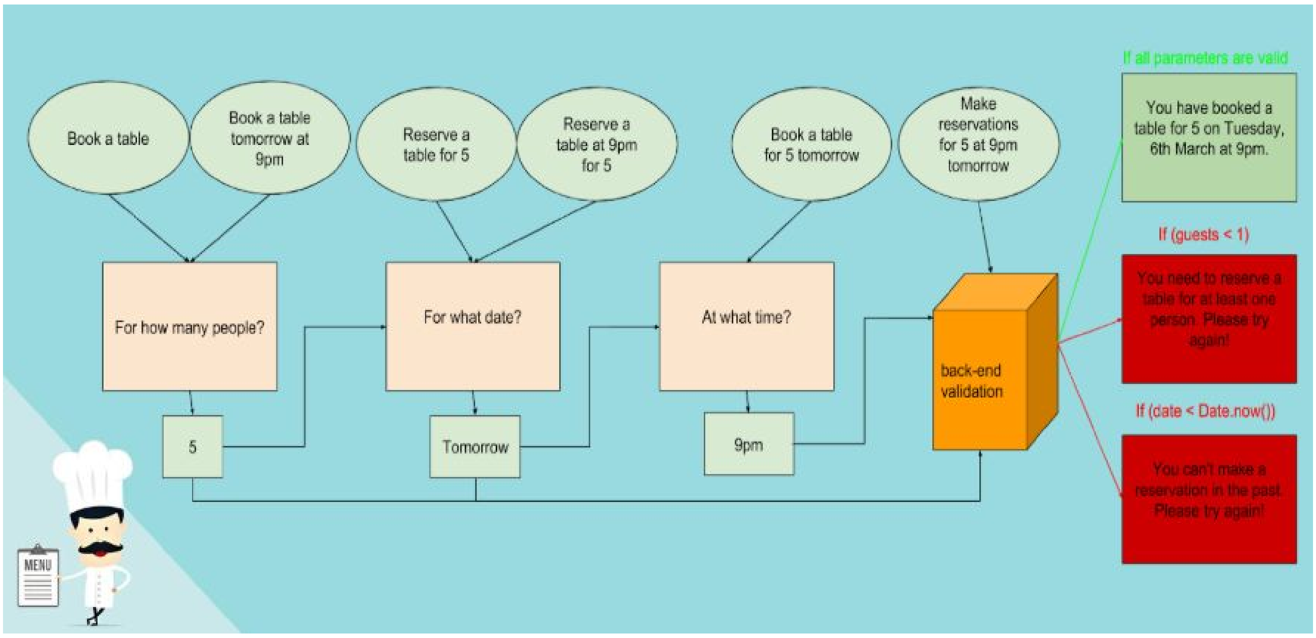
1a. Searching through yelp for business information



1b. Searching for hours of operation, delivery information

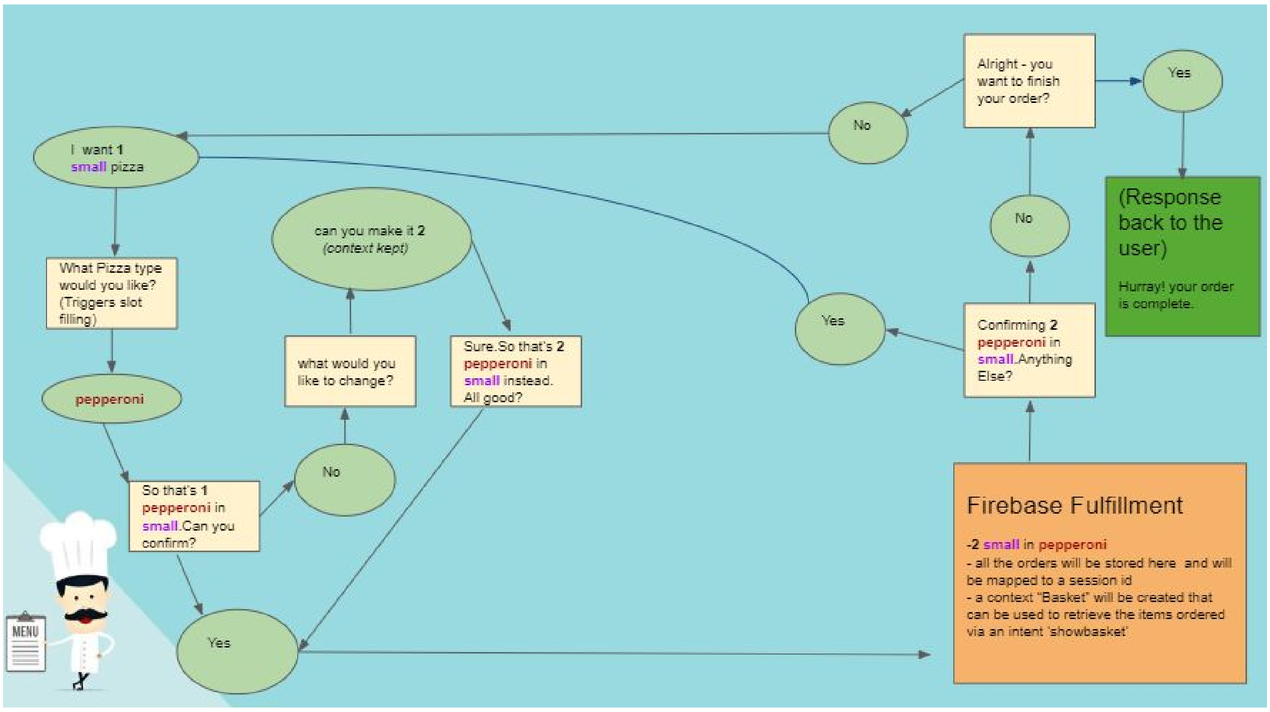


2. Reserve a table

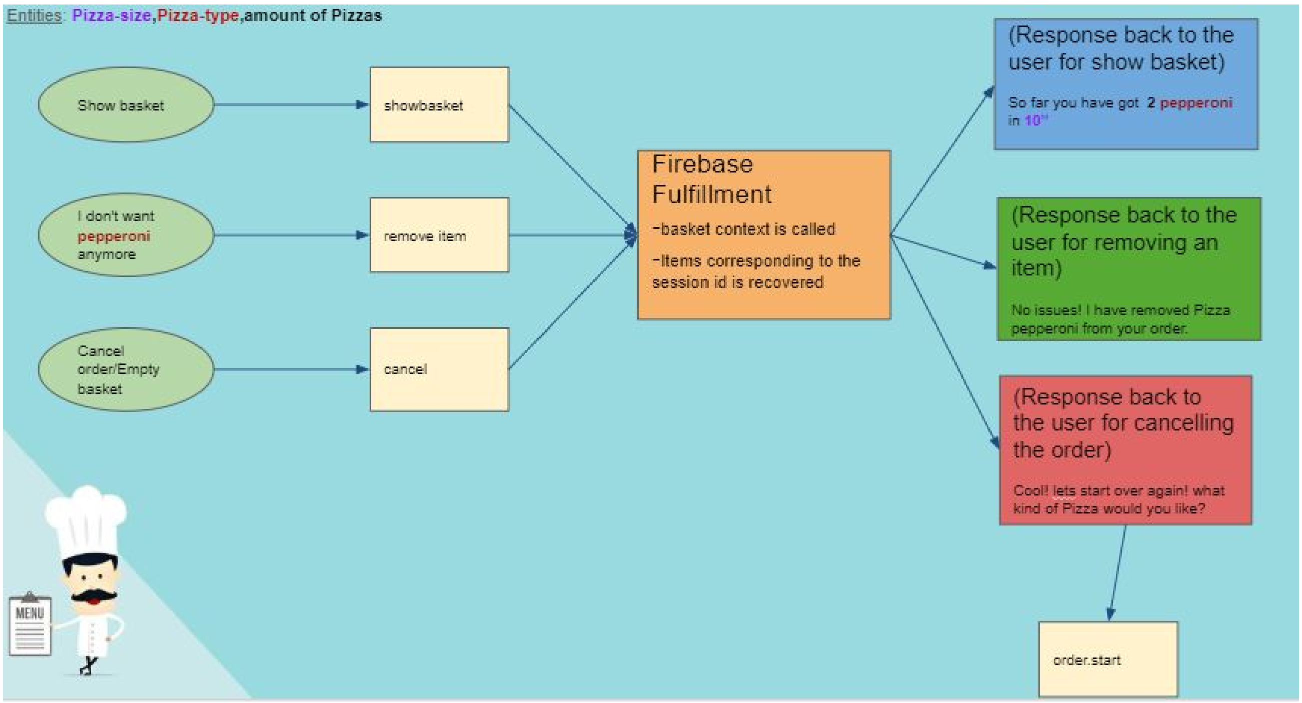


3. Placing an order

3.a)Configuring the Pizza



3b) Modify shopping basket



Appendix 9

Sample Conversations with Agent

Use Case: Reservation of Table

User: Hi

Agent: *Hi there! Welcome to Hungry Hippos, you can say things like "Where is the nearest Starbucks? ", "What are the operating hours for Giant?", "Reserve a table" or "Place an order". What would you like help with?*

User: Reserve a Table for 2

Agent: *Which date would you like me to reserve?*

User: 31st July 2020

Agent: *What time would you like me to make this reservation?*

User: 7pm

Agent: *You can't make a reservation in the past. Please try again!*

User: Reserve a table for 5 on 20th August 2020 at 8pm

Agent: *You have successfully booked a table for 5 guests on Sat Aug 20 2020 20:00*

See you at the restaurant! Have a wonderful day!

Use Case: Order a Pizza

User: Hi

Agent: *Hi there! Welcome to Hungry Hippos, you can say things like "Where is the nearest Starbucks? ", "What are the operating hours for Giant?", "Reserve a table" or "Place an order". What would you like help with?*

User: Do you have Pepperoni?

Agent: *Yes, we do offer Pepperoni. Would you like to order one?*

User: I want a large Pepperoni!

Agent: *How many would you like*

User: 7

Agent: *Fantastic! so that's 7 Pepperoni in 14". Can you confirm?*

User: No! Make it 5

Agent: *Sure. So that's 5 pizza Pepperoni in 14" instead, all good?*

User: Yes

Agent: *Confirming 5 pizza Pepperoni in 14". Anything else?*

User: Nah

Agent: *Alright, do you want to finish your order?*

User: Yeah

Agent: *Hurray! you have completed your order!*