Building Chatbots Using Google DialogFlow

This **project** doesn't require you to do lots of coding but still gives you a fair idea of how to build chatbots in an easy way and make it public.

Do scan the home page of DialogFlow on : https://dialogflow.com/

Note : Dialogflow was formerly known as Api.ai.

Introduction to Dialogflow

Dialogflow gives users new methods to interact with their product by building engaging voice- and text-based conversational interfaces, such as voice apps and chatbots. Dialogflow is powered by AI. It helps you connect with users on your website, mobile app, the Google Assistant, Amazon Alexa, Facebook Messenger, and other popular platforms and devices.

The following diagram from Dialogflow shows how they handle a user request.

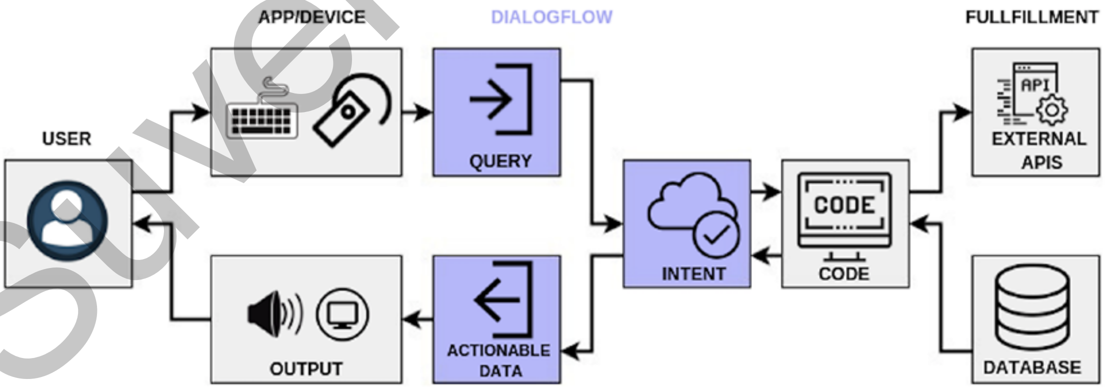


Figure 1. *Working diagram of Dialogflow architecture*

Here is what happens:

1. User talks to the input device.
2. User query goes into the Dialogflow engine.
3. Dialogflow tries to recognize the intent.
4. Based on the intent, a fulfillment is done and data is returned from database.
5. Response is returned to the intent.
6. Response is converted into actionable data.
7. User's request for information is given back to the output device.

There is a concept of agents in Dialogflow that are best described as Natural Language Understanding (NLU) modules. These can be included in your app, product, or service and transform natural user requests into actionable data. This transformation occurs when a user input matches one of the intents inside your agent.

Agents can also be designed to manage a conversation flow in a specific way. This can be done with the help of contexts, intent priorities, slot filling, responsibilities, and fulfillment via webhook.

Project1: Building a Food-Ordering Chatbot

We are going to create a chatbot with help of Dialogflow for a specific restaurant. Let's name it **OnlineEatsBot**. In short we can call it OnlineEats product. You can choose any other use-case for which you want to build the chatbot. For this project we are going to build a food-ordering chatbot.

Deciding the Scope

Let's decide the scope of this chatbot—that is, what it can do and to what extent.

- It should be able to greet the user dynamically.
- It should be able to understand the menu items and their quantity requested.
- Chatbot should be able to place an order on the user's behalf.
- Give the user the status of the order when asked.

Listing Intents

Here we list the intents we want our chatbot to train with so that it can understand those intents when asked by the user.

Intents

- Default welcome intent: when the user messages the chatbot
- Place order intent: when the user asks the bot to order food
- Item description intent: when the user tells what item and quantity they want
- Order status: when the user wants to know his order status
- Order_ID: the bot needs to understand the user's order ID for tracking.
- User thanks: when the user thanks the bot

Listing Entities

We'll try to list all possible entities for all intents here. Later in the chapter we'll see them being defined for whichever intent they are applicable.

Entities

- food_items: what food does the user want to order?
- quantities: what is the quantity of the food item the user is willing to order?
- order_id: order_id of the place order for user

Building a Food Ordering Chatbot

Let's also level-set the expectation of the chatbot in terms of the minimum things we should be able to see it doing. For that, let's create a script of a conversation between our chatbot and the user. This helps us to stick to the plan to getting a basic working chatbot ready, assuming the user is doing the conversation in a nice and correct manner.

Chatbot and User Conversation Script:

User: Hello

OnlineEatsBot: Hello and Welcome! How may I help you?

User: I am looking to order Chinese food.

OnlineEatsBot: Sure, what would you like to order today?

User: One chicken dumpling and two spring rolls.

OnlineEatsBot: Done. Your final amount is XYZ, and your order is placed for 1 Chicken dumpling and 2 spring rolls.

User: I haven't received my order yet. Where is my food?

OnlineEatsBot: Can you please help me with your order ID?

User: 123456

OnlineEatsBot: Order Status for Order ID: 123456. The delivery guy is in your locality, your food will arrive in about 5 minutes.

User: Thanks.

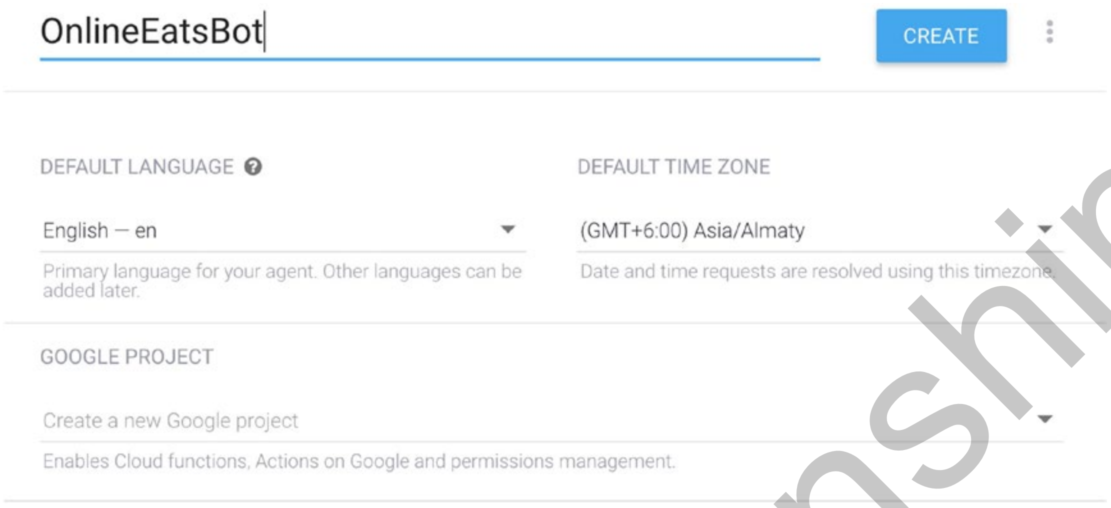
OnlineEatsBot: Thanks a lot for your cooperation.

Now that we have written a basic script to follow to build our chatbot, we'll now dive into Dialogflow.

Getting Started With Dialogflow

Let's follow these steps to create an account in Dialogflow and then create an agent. (An agent is nothing but another name for a chatbot.)

1. Create an account at `https://dialogflow.com`, and log in to the account.
2. Create an agent.


OnlineEatsBot

DEFAULT LANGUAGE

English — en

Primary language for your agent. Other languages can be added later.

DEFAULT TIME ZONE

(GMT+6:00) Asia/Almaty

Date and time requests are resolved using this timezone.

GOOGLE PROJECT

Create a new Google project

Enables Cloud functions, Actions on Google and permissions management.

Figure 2. *Creating a new agent in Dialogflow*

Enter the details, like name of the agent, time zone, default language, and Google Project that you want to choose or create a new Google project.

3. Create intents.

   If you see Figure 3, you will see that we are given two intents already.

- **Default fallback intent:** Fallback intents are triggered if a user's input is not matched by any of the regular intents or enabled built-in small talk. When you create a new agent, a default fallback intent is created automatically. You can modify or delete it if you wish.
- **Default welcome intent:** We can extend this welcome intent for our own chatbots. You should add some of your own user expressions and default responses.

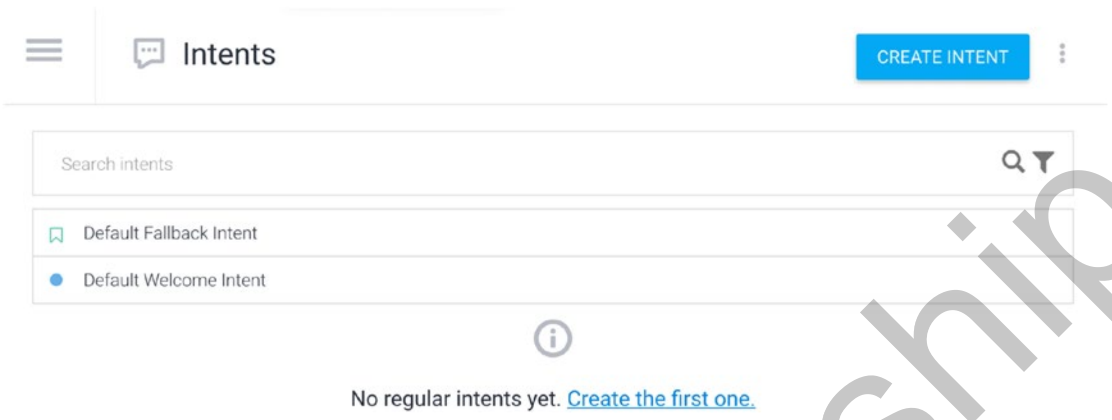
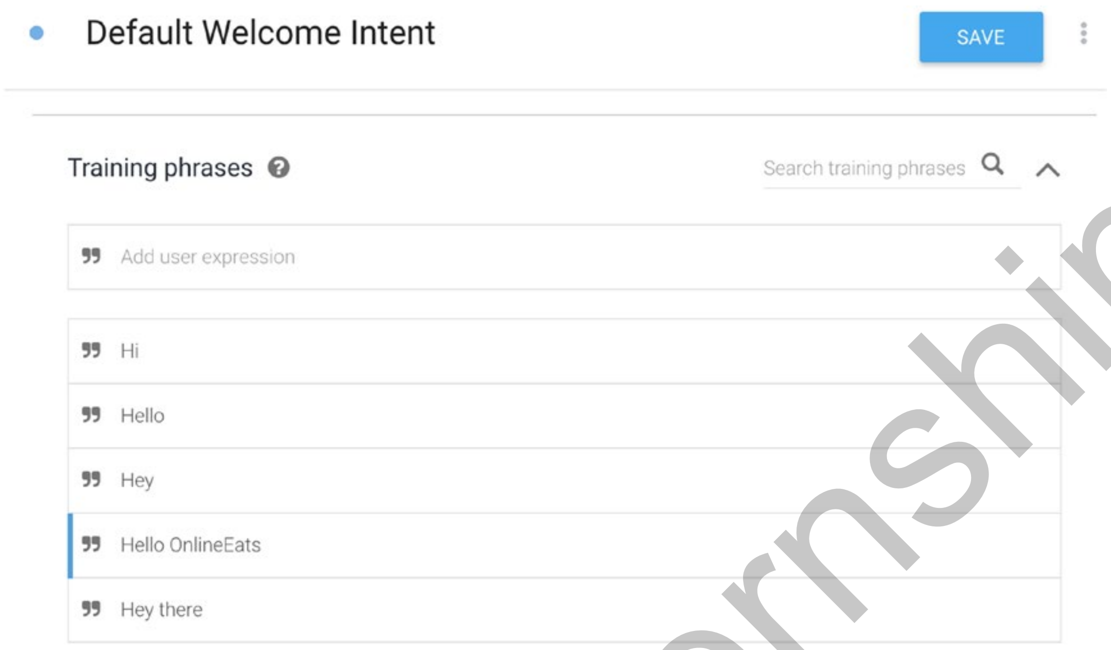


Figure 3. *Creating intents in Dialogflow*

Before we create our own intents, let's first add some utterances in default welcome intent and make it ready using the following steps:

1. Click on the default welcome intent.
2. Add your own user expressions in Training Phrases.
3. Click on **SAVE.**

When we click on save, the machine learning models behind the scenes run and train the data that we gave (i.e., the user expressions). Training the data means letting the machine understand what kind of intent it is based on the data that we provide and being able to predict when we give any new data to it. For example, if we look at Figure 4, where we have defined five user expressions that the machine already knows belong to "welcome intent," what if the user says "Hello there," which is not defined anywhere? The machine will still categorize "Hello there" as **default welcome intent**, as the features used in training and the machine for welcome intent are similar in the new user expression.



Hi

Hello

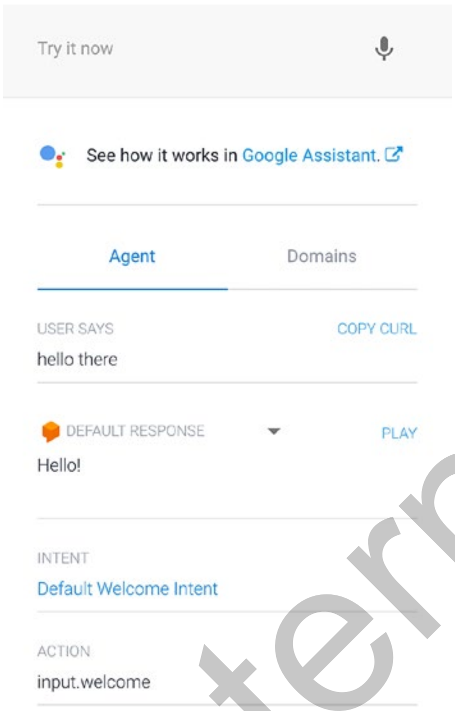
Hey

Hello OnlineEats

Hey there

Figure 4. *Defining a default welcome or greeting intent in Dialogflow*

Let's try to see if the welcome intent works for us. With Dialogflow we can do that in the dashboard itself. See Figure 5.


Try it now

See how it works in Google Assistant.

Agent | Domains

USER SAYS | COPY CURL
hello there

DEFAULT RESPONSE | PLAY
Hello!

INTENT
Default Welcome Intent

ACTION
input.welcome

Figure 5. *Testing the welcome intent in Dialogflow*

Points to Remember When Creating Intents

Let's look at some of the points important to know while creating intents on Dialogflow.

- Dialogflow's intent also has the facility to have a default response of every intent. A default response is a response given back to the user every time that intent is recognized. In our example, when a user says "Hello there," we get "Hello!" as the response from the bot.
- If you want to you can add more responses or delete existing ones, having more than one response makes the bot look realistic so that it doesn't reply with the same response every time and feels human to the user talking to the bot.
- Intents in Dialogflow also have the capability of being marked as the end of the conversation. In other words, you can let the bot assume that the user will not be participating in the conversation anymore, and the bot can do the necessary action, based on this information, to end the conversation.

Creating Intents and Adding Utterances

Now that we have created the welcome intent, let's create the **order intent**. I named it **place_order_intent**. The following are my user expressions that I entered:

I want food

I want to order food asap

Can you please take my order for food?

Take my order please

I want to place an order for Chinese food

I want to place an order

Would you please help me to order food?

Can you please order food for me?

I want to order food

I am looking to order Thai food

I am looking to order Chinese food

Now, we have built the intent to identify the aforementioned user expressions or related user expressions. Now, it's time to add a response back to the user using *default response* to the intent.

Adding Default Response to the Intent

We'll be adding three possible responses that will be given back to the user once **place_order_intent** is encountered.

Sure, What would you like to order today?

Definitely, What would you like to have today?

Certainly, I'll try to help you with that. What are you feeling like eating today?

Now, the next step is to wait for the user to input the items he wants and parse the items.

Now we'll create a new intent that tells us what the user actually intends to order (i.e., the food items).

We create a new intent named **items_description**
First, we add our standard user expression.
One chicken dumpling and two spring rolls.
When we add the user expression then we can select specific words that we want to specify as entities of the intent. This could be quantity, date or time, location, etc., which are predefined, but we can create our own entities by clicking on the Create New button on the bottom right after we get the pop-up box.

Highlight the word in the utterance for which you want to make that selected word an entity. After that, it opens the pop-up box to create our own entity.

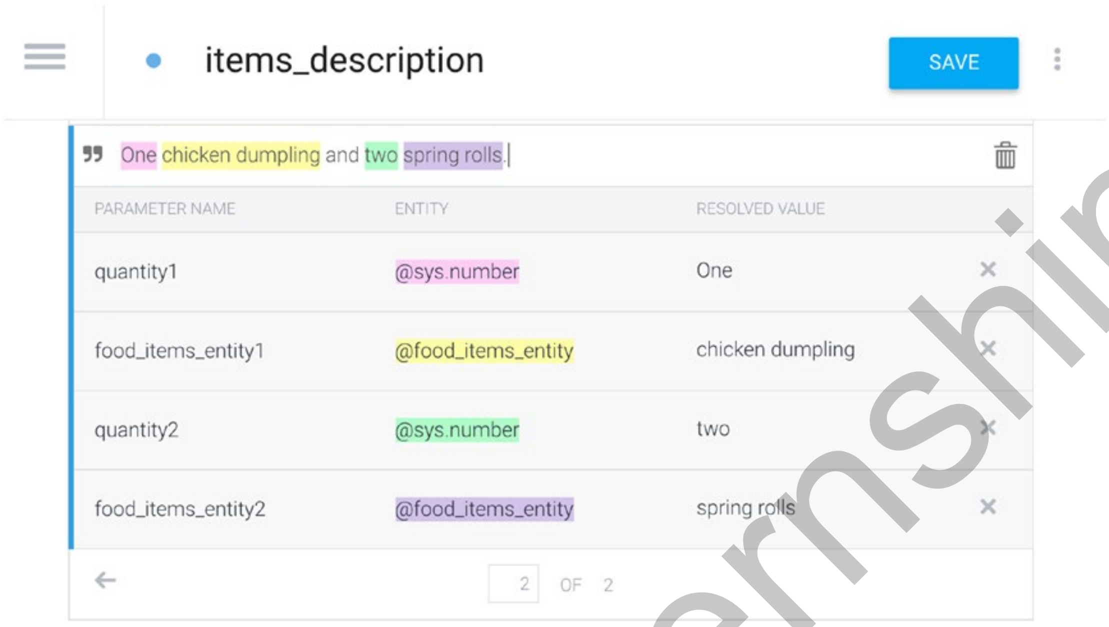
In this example, we should be able to parse the data in a nice readable format so that we can use that using any programming language. JSON format is the best format we can have in today's cross-platform applications. Dialogflow returns the data in JSON format by default, which can be parsed to look something like the following code. It's always suggested to keep your data as minimal as possible; don't overwhelm the API response by giving too much data. Remember all of these at scale cost money.

```
{
  "food_items": {
    "chicken dumpling": 1,
    "Spring rolls": 2
  }
}
```

Item Description Intent and Belonging Entities

We can select One and Two and define them as @sys.number, which is nothing but the data type. We'll create a new entity called **food_items_entity** to identify food items.

If you look at Figure 6, you'll find that we have ENTITY named as **food_items_entity**, but when we select the words, then we name the parameters as **food_items_entity1** and **food_items_entity2**; this is similar for the food quantity, which is a number where we name the first and second parameters as **quantity1** and **quantity2,** respectively.



One chicken dumpling and two spring rolls.

PARAMETER NAME	ENTITY	RESOLVED VALUE
quantity1	@sys.number	One
food_items_entity1	@food_items_entity	chicken dumpling
quantity2	@sys.number	two
food_items_entity2	@food_items_entity	spring rolls

Figure 6. *Item description intent*

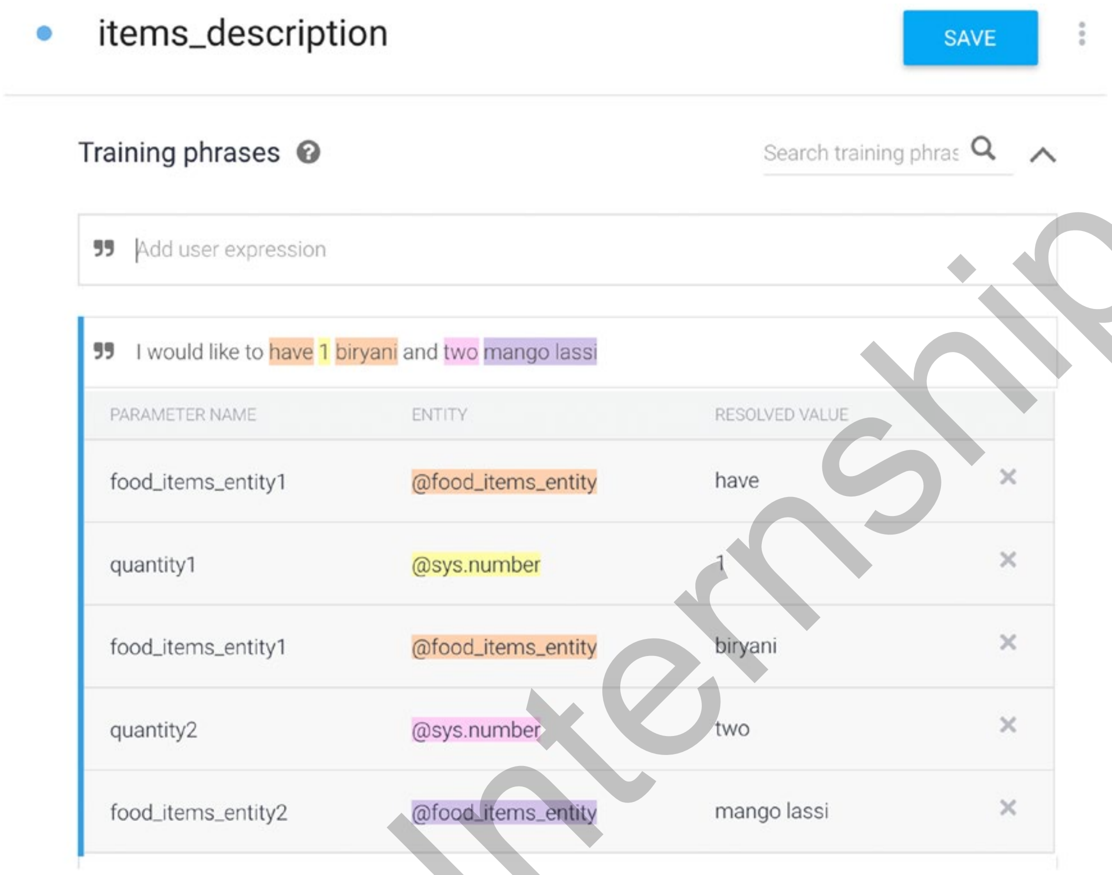
What we define here will help us understand the JSON response, which we'll be getting after intent is triggered. We should have all these values there to move forward with the chatbot flow. So, select the entire word or combination of words and click on Create New. A new screen will come to create entities; just enter the name for this new entity and save.

Now, come back to our intent for **items_description** and you should see something like Figure 6.

Keep adding more user expression in the training phrases, and keep defining the entities within it.

We have added four utterances so far, and this is how they look. **You will add as many as possible so that our agent's accuracy for intent classification is better.**

As you can see in the Figure 7 where we try to add some more examples in the item description intent in our dialogflow agent.



I would like to have 1 biryani and two mango lassi

PARAMETER NAME	ENTITY	RESOLVED VALUE
food_items_entity1	@food_items_entity	have
quantity1	@sys.number	1
food_items_entity1	@food_items_entity	biryani
quantity2	@sys.number	two
food_items_entity2	@food_items_entity	mango lassi

Figure 7. *Adding more utterances in item description intent*

Now, at this point once we have saved our intent, and our agent has finished training the models. If we enter the following sentence on the right side, we should be able to see the following JSON response:

One chicken dumpling and two spring rolls

Response from the intent:

```
{
  "id": "e8cf4a44-6ec9-49ae-9da8-a5542a80d742",
  "timestamp": "2018-04-01T21:22:42.846Z",
  "lang": "en",
  "result": {
    "source": "agent",
    "resolvedQuery": "One chicken dumpling and two spring rolls",
    "action": "",
    "actionIncomplete": false,
    "parameters": {
      "quantity1": 1,
      "food_items_entity1": "chicken dumpling",
      "quantity2": 2,
      "food_items_entity2": "spring rolls"
    },
    "contexts": [],
    "metadata": {
      "intentId": "0b478407-1b37-4f9a-8779-1866714dd44f",
      "webhookUsed": "false",
      "webhookForSlotFillingUsed": "false",
      "intentName": "items_description"
    },
    "fulfillment": {
      "speech": "",
      "messages": [
        {
          "type": 0,
          "speech": ""
        }
      ]
    },
    "score": 1
  },
  "status": {
    "code": 200,
    "errorType": "success",
    "webhookTimedOut": false
  },
  "sessionId": "e1ee1860-06a7-4ca1-acae-f92c6e4a023e"
}
```

If you look at the *parameters* section of the JSON response we see

```
{
"quantity1": 1,
"food_items_entity1": "chicken dumpling",
"quantity2": 2,
"food_items_entity2": "spring rolls"
}
```

Understanding and Replying Back to the User

Now, next in the conversation is to make the bot reply back to the user that the order is understood along with any new information. New information could be the generated order_id, the order amount, or the expected delivery time. These things will be populated at your server end, and you can formulate it with the bot's response to give it back to the user.

Now, let's try to add the order amount in our case; to do that, we can use Dialogflow's **Default Response** feature and add this inside the intent. Let's hardcode the amount
for now, as this amount will vary depending on the food items, their quantity, or the restaurant. Later in the **project**, we'll discuss how to make it dynamic by invoking an API. The interesting thing here is that we can access the params we got from the intent (i.e. , the food items and their quantities). Responses can contain **references to parameter values**. We'll understand this in a moment.

If a parameter is present in the parameter table, we can use the following format to refer to its value in the 'Text response'field: $parameter_name.

We can use this params in the default response so that the bot confirms the order back to the user.

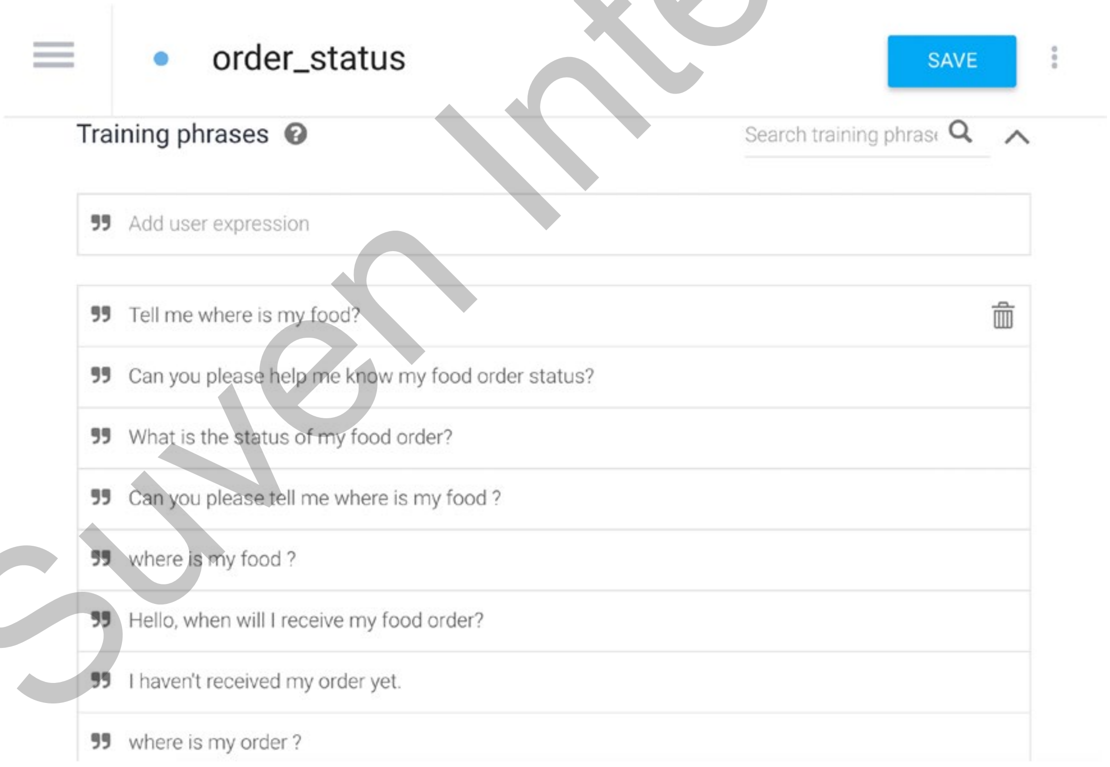
Add *"Done. Your final amount is XYZ and your order is placed for $quantity1 $food_items_entity1 and $quantity2 $food_items_entity2"* as the response.

Note In case our intent is not able to parse the food items or their quantity, we need to give a different default response asking to explain what our bot couldn't understand or to at least confirm. We already learned how to add default response to an intent in the section "**Adding Default Response to the Intent**."

Order Status Intent

Now, let's create the order_status intent, where the user may be trying to ask for the status of the order after the order is placed.

Figure 8 provides some training phrases we added for order status intent, and we name the intent **order_status**.



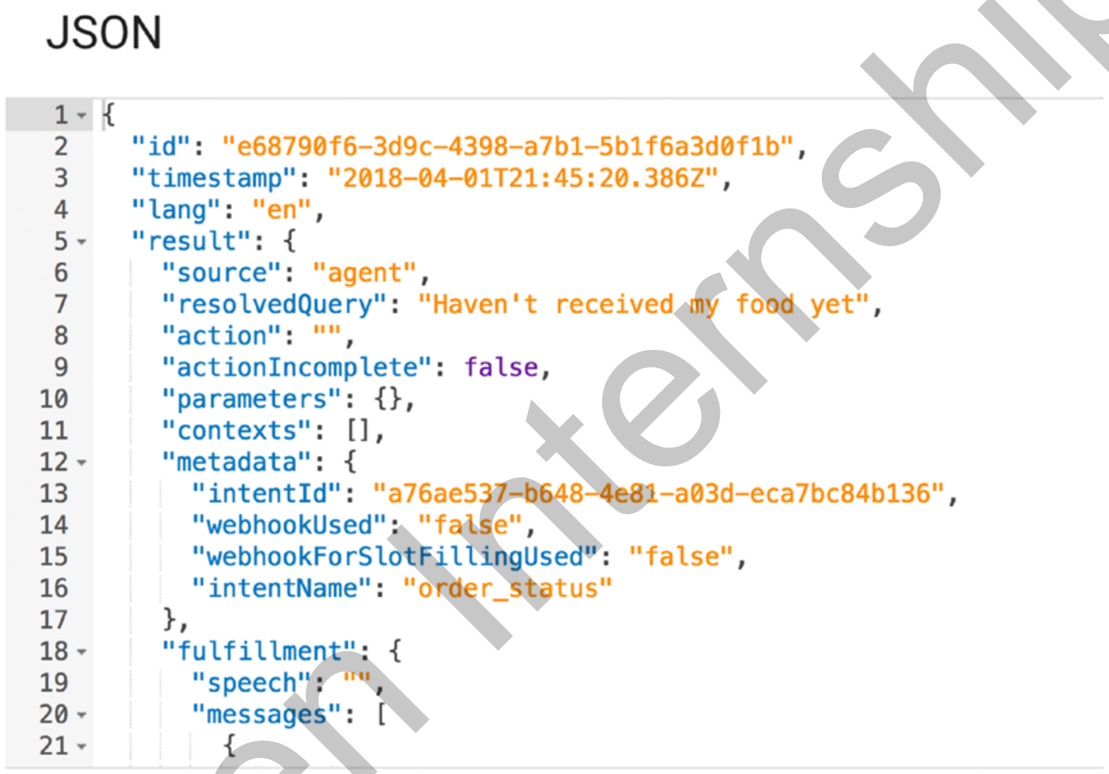
Tell me where is my food?
Can you please help me know my food order status?
What is the status of my food order?
Can you please tell me where is my food ?
where is my food ?
Hello, when will I receive my food order?
I haven't received my order yet.
where is my order ?

Figure 8. *Creating order status intent*

Now, let's try some random order status asking utterances and see if our agent is intelligent enough to identify the intent.

I tried, *"Haven't received my food yet,"* and voila—my agent got it perfectly right that it's an **order_status** intent.

See the *resolvedQuery* and its intentName in the JSON in Figure 9.



```
{
  "id": "e68790f6-3d9c-4398-a7b1-5b1f6a3d0f1b",
  "timestamp": "2018-04-01T21:45:20.386Z",
  "lang": "en",
  "result": {
    "source": "agent",
    "resolvedQuery": "Haven't received my food yet",
    "action": "",
    "actionIncomplete": false,
    "parameters": {},
    "contexts": [],
    "metadata": {
      "intentId": "a76ae537-b648-4e81-a03d-eca7bc84b136",
      "webhookUsed": "false",
      "webhookForSlotFillingUsed": "false",
      "intentName": "order_status"
    },
    "fulfillment": {
      "speech": "",
      "messages": [
        {
```

Figure 9. *JSON response from Dialogflow after query is resolved*

User_Order_ID Intent

Now, next is to ask the user for Order ID, so let's set the default response of this intent to ask the following question.

Can you please help me with your order ID?

Now, the user will be giving their order ID, and our task is to identify that and give a response again.

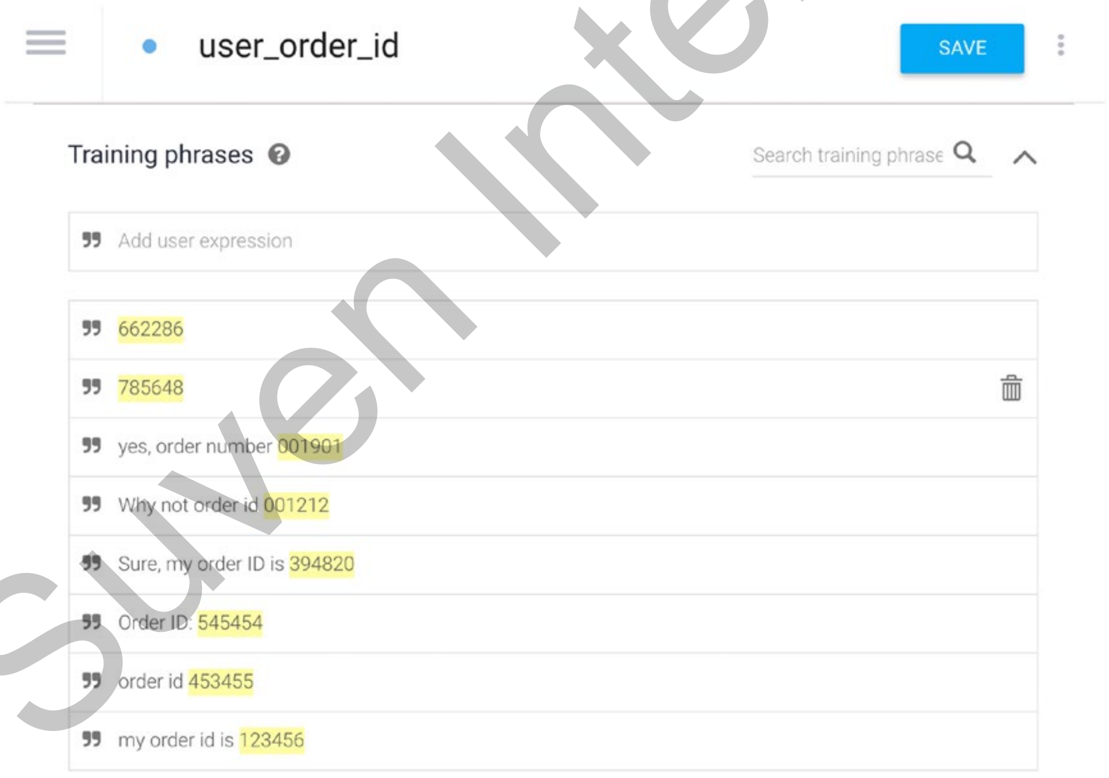
So, for that we need to create another intent to identify when the user is talking about the order ID.

Note that the intents we are creating are pretty much independent of each other. In this case we know that the user is going to give the order ID, and it will mostly be right. If it's wrong you can always go back to the user and ask again.

We should also note that in some cases, order_id and phone number both may be integers. In such cases, we need to do some validation, like number of digits in order_id or phone number. Also, based on the context of the previous question, you can figure out if the user is giving an order_id or a phone number. As discussed in ***Pre-read doc : Terminology Related to Chatbots***, we can always use decision trees for better flow management of the chatbot. Also we can programatically keep track that after **order_status** intent we ask for the order ID, and the user will be sending some order ID (some number), which is easier to parse in code rather than creating a new intent altogether.

In this project, we'll create user_order_id intent, as there is no conflict as such.

Now, we create a new intent called **user_order_id**
Figure 10 shows how our **user_order_id** intent looks.



662286
785648
yes, order number 001901
Why not order id 001212
Sure, my order ID is 394820
Order ID: 545454
order id 453455
my order id is 123456

***Figure 10.** Defining the user order ID intent in our agent*

I tested a couple of expressions, and it works well to classify them correctly as **user_ order_id** intent. Always test using the Dialogflow console to see if your intent is behaving as expected.

Now, let's set the default response of **user_order_id** intent to the following response from the bot:

Order Status for Order ID: $order_id .The delivery guy is in your locality, your food will arrive in about 5 minutes.

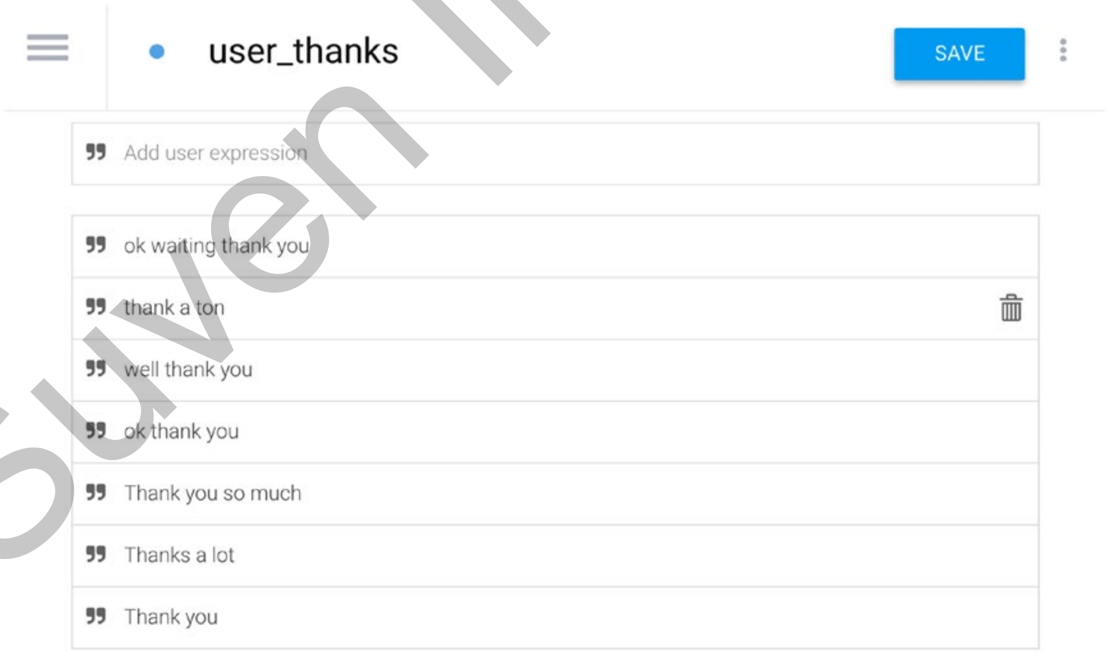
We are again using the parameter parsed from the **user_order_id** intent to prepare a reply to the user.

User_Thanks Intent

Next, the user will possibly be thanking, if not something else, so we create a new intent called **user_thanks** to identify different ways the user is saying thank you. This is important because once the user says *thank you* in some way or another, our bot should reply the same.

We shouldn't just expect the user to say thanks after the delivery status default response and reply blindly but try to identify it using custom intents.

Figure 11 shows how our **user_thanks** intent looks.

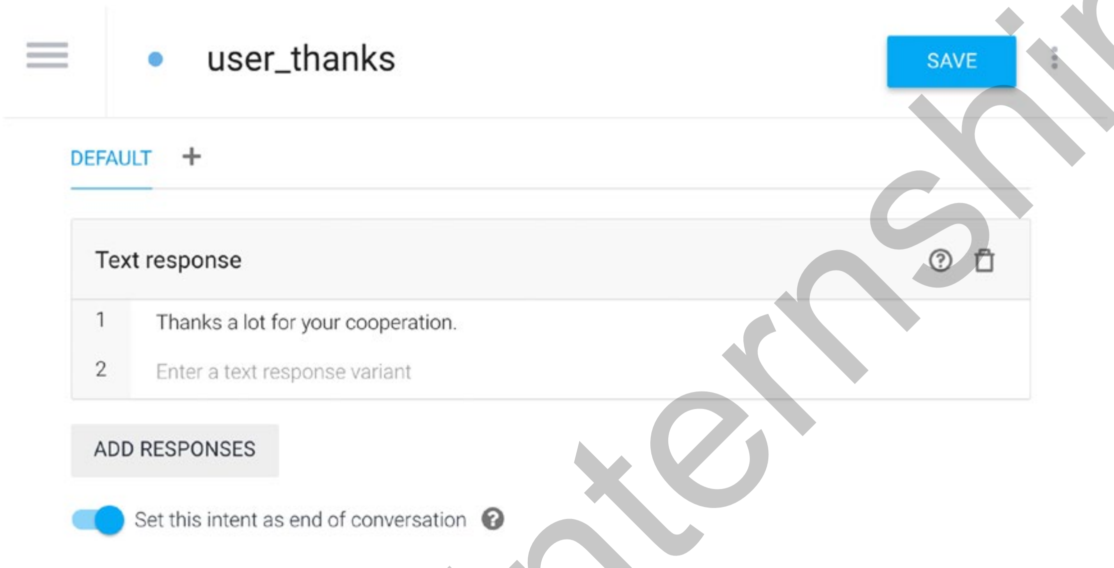


ok waiting thank you
thank a ton
well thank you
ok thank you
Thank you so much
Thanks a lot
Thank you

***Figure 11.** Defining an intent when the user says thank you*

Now, it's time to say thank you to the user using the default response feature and mark this intent as the end of the conversation.

Let's add some text like, "*Thanks a lot for your cooperation,*" as our default response. We can add more such responses so that the bot should look more realistic (see Figure 12).



1 Thanks a lot for your cooperation.
2 Enter a text response variant

Set this intent as end of conversation

Figure 12. *Adding a response against a user's intent in the agent*

Look at Figure 12 and see that we have enabled this intent as the end of conversation.

If we try to integrate our bot with Google Assistant, then enabling this means to close the microphone in Google Assistant when intent is finished.

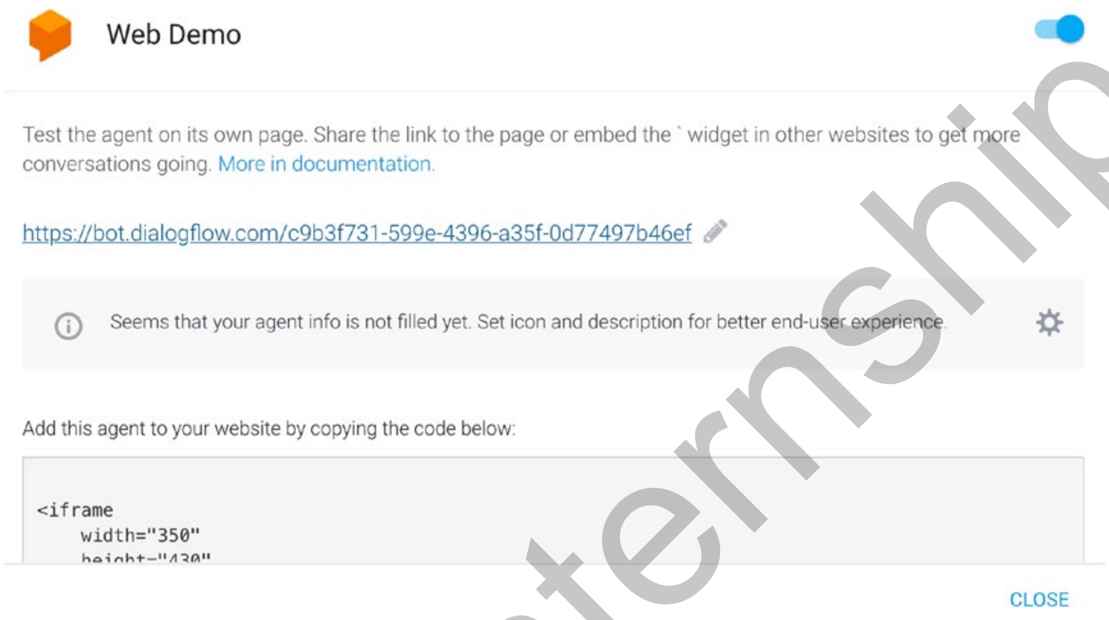
Now at this point we have created our bot, built it as per our initial design and script, and trained it. Now, it's time to deploy it on web and see how it looks.

Deploying Dialogflow Chatbot on the Web

In this part, we are going to integrate our bot with various platforms like Facebook Messenger, Twitter, Slack, etc., and see if they work. There are many more platforms where you can integrate this bot easily.

We will test our bot with Web Demo and Facebook Messenger for now.

Let's go to the Integrations page in our Dialogflow account and enable the **Web Demo.** You will be getting a pop-up like Figure 13. Click on the link in the pop-up.


Web Demo

Test the agent on its own page. Share the link to the page or embed the ` widget in other websites to get more conversations going. More in documentation.

https://bot.dialogflow.com/c9b3f731-599e-4396-a35f-0d77497b46ef

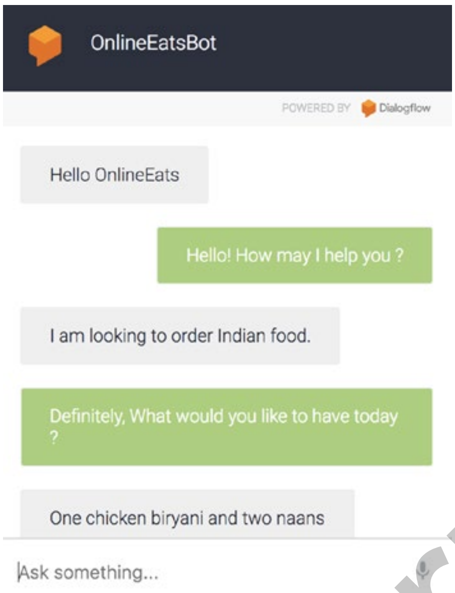
Seems that your agent info is not filled yet. Set icon and description for better end-user experience.

Add this agent to your website by copying the code below:

```
<iframe
    width="350"
```

Figure 13. *Dialogflow's web demo link*

You will be seeing something similar to Figures 14.1 through 14.4. I tested my bot with the conversation we wrote, and my bot works like a charm.


OnlineEatsBot

POWERED BY Dialogflow

Hello OnlineEats

Hello! How may I help you ?

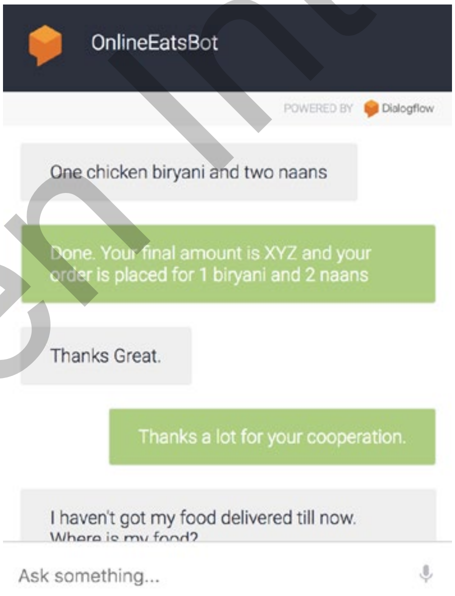
I am looking to order Indian food.

Definitely, What would you like to have today ?

One chicken biryani and two naans

Ask something...

Figure 14.1. OnlineEatsBot Demo Conversation Screen I


OnlineEatsBot

POWERED BY Dialogflow

One chicken biryani and two naans

Done. Your final amount is XYZ and your order is placed for 1 biryani and 2 naans

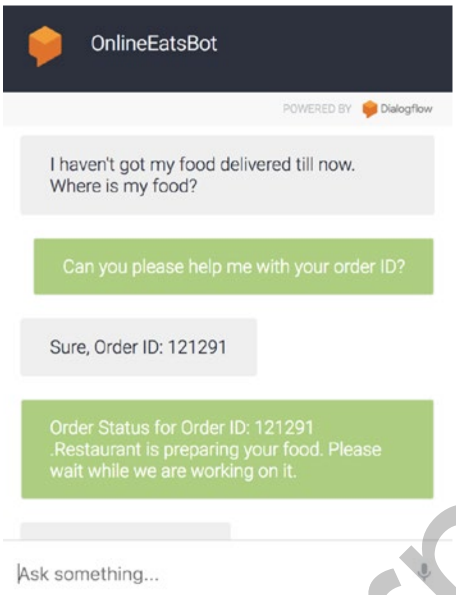
Thanks Great.

Thanks a lot for your cooperation.

I haven't got my food delivered till now. Where is my food?

Ask something...

Figure 14.2. OnlineEatsBot Demo Conversation Screen II


OnlineEatsBot

POWERED BY Dialogflow

I haven't got my food delivered till now.
Where is my food?

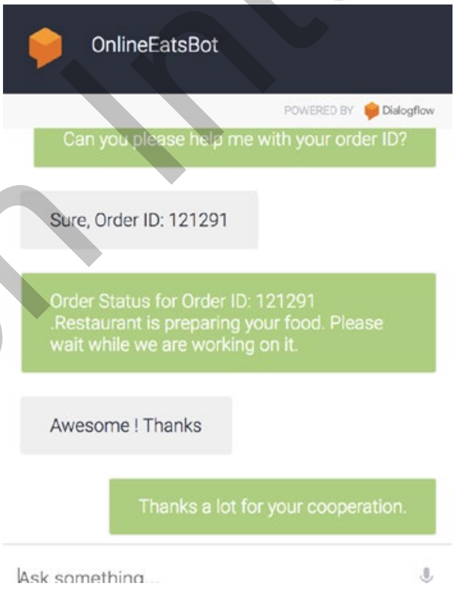
Can you please help me with your order ID?

Sure, Order ID: 121291

Order Status for Order ID: 121291
.Restaurant is preparing your food. Please
wait while we are working on it.

Ask something...

Figure 14.3. OnlineEatsBot Demo Conversation Screen III


OnlineEatsBot

POWERED BY Dialogflow

Can you please help me with your order ID?

Sure, Order ID: 121291

Order Status for Order ID: 121291
.Restaurant is preparing your food. Please
wait while we are working on it.

Awesome ! Thanks

Thanks a lot for your cooperation.

Ask something...

Figure 14.4. OnlineEatsBot Demo Conversation Screen IV

Apart from this we can also embed this bot on our own website using the iframe code found in the pop-up window.

Talk to my **OnlineEatsBot** here:
`https://bot.dialogflow.com/c9b3f731-599e-4396-a35f-0d77497b46ef`
Share your own bot with your friends and family as well, and see the way they interact with the bot in a legitimate manner. If your chatbot is not doing something expected, then try to fix that.

Integrate Dialogflow Chatbot on Facebook Messenger

Now-a-days Facebook is taking too long for validating Hence you may keep this step optional

In this section we'll try to integrate our same chatbot to Facebook Messenger so that our users on the Facebook platform can also use it without having to come to our website.

Let's go back to the integrations page in the Dialogflow dashboard and enable the Facebook Messenger icon, and then click on that, which should bring up a similar pop-up as before.

Here we need to go to Facebook, register an app, and get the required token.

- Verify Token (any string and is solely for your purposes)
- Page Access Token (Enter the token generated in the Facebook Developer Console)

The Dialogflow Facebook integration is very helpful to easily create a Facebook Messenger bot with NLU, based on the Dialogflow technology.

Setting Up Facebook

To make our bot work the same way it worked in Facebook, we would need to do the following:

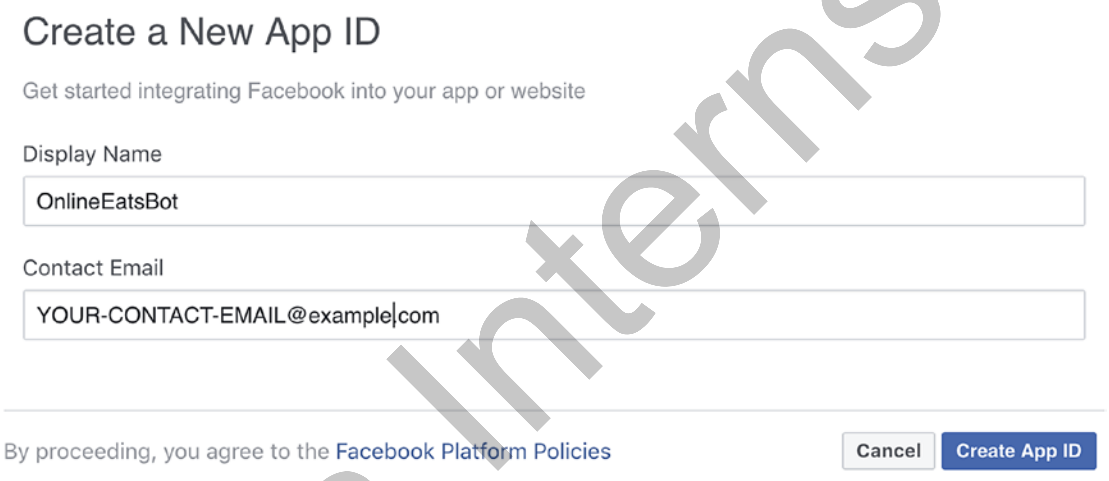
1. Create a Facebook account, if you haven't already.
2. Create Facebook page to which you can add your bot.

When a user visits your Facebook page and sends you a message, they'll be talking to your bot directly.

Creating a Facebook App

The following are the steps to create an app:

1. Log into the Facebook Developer Console.
2. Click on **My Apps** in the upper right-hand corner.
3. Click on **Add a New App** and enter a name and contact e-mail address.
4. Click **Create App ID** as shown in the Figure 15 below.



Create a New App ID

Get started integrating Facebook into your app or website

Display Name

OnlineEatsBot

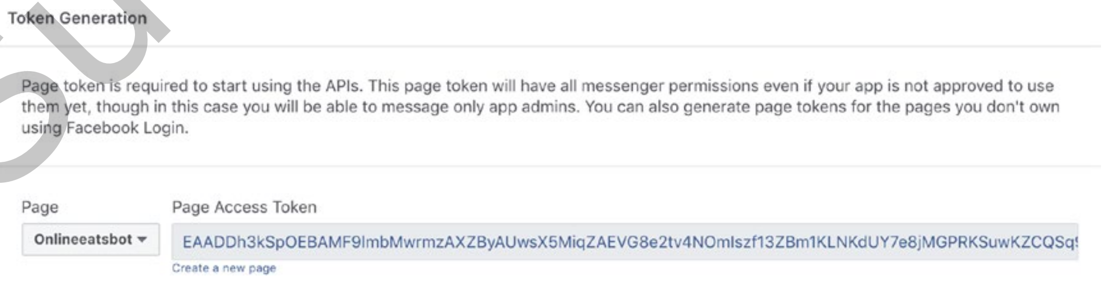
Contact Email

YOUR-CONTACT-EMAIL@example|com

By proceeding, you agree to the Facebook Platform Policies

Figure 15. *Creating a new app at Facebook Developer Platform*

5. On the next page, click the **Set Up** button for the **Messenger** option.
6. Under the **Token Generation** section, let's choose the Facebook page to which we want our bot to connect (see Figure 16).



Token Generation

Page token is required to start using the APIs. This page token will have all messenger permissions even if your app is not approved to use them yet, though in this case you will be able to message only app admins. You can also generate page tokens for the pages you don't own using Facebook Login.

Page: Onlineeatsbot

Page Access Token: EAADDh3kSpOEBAMF9lmbMwrmzAXZByAUwsX5MiqZAEVG8e2tv4NOmlszf13ZBm1KLNKdUY7e8jMGPRKSuwKZCQSq

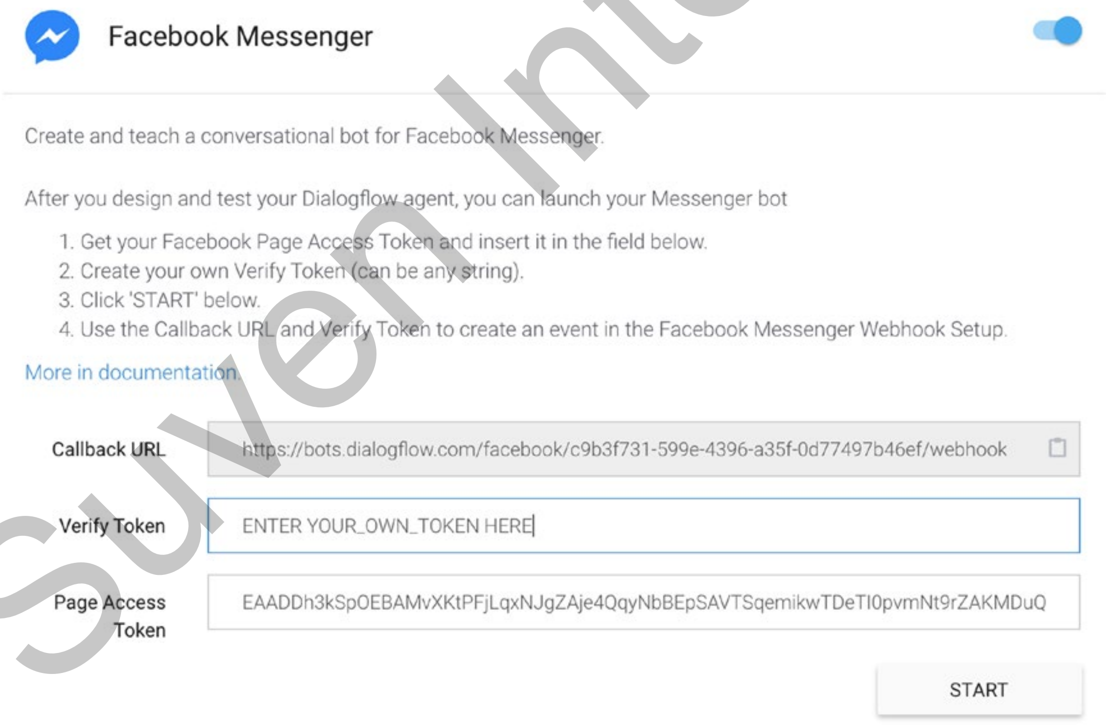
Figure 16. *Generating token by selecting your Facebook page for your bot*

This will generate a **Page Access Token**. Keep this token handy, as we'll need to enter it in Dialogflow.

Setting Up the Dialogflow Console

Here are the steps:

1. Click on the **Integrations** option in your Dialogflow console in the left menu and switch on **Facebook Messenger** if you haven't already. In the pop-up that opens, enter the following information as shown in the Figure 17 Setting up and integrating Dialogflow with Facebook Messenger:
   - **Verify Token:** this can be any string that you want and for your own purposes
   - **Page Access Token:** enter the token generated in the Facebook Developer Console
2. Click the **Start** button.


Facebook Messenger

Create and teach a conversational bot for Facebook Messenger.

After you design and test your Dialogflow agent, you can launch your Messenger bot

1. Get your Facebook Page Access Token and insert it in the field below.
2. Create your own Verify Token (can be any string).
3. Click 'START' below.
4. Use the Callback URL and Verify Token to create an event in the Facebook Messenger Webhook Setup.

More in documentation

Callback URL	https://bots.dialogflow.com/facebook/c9b3f731-599e-4396-a35f-0d77497b46ef/webhook
Verify Token	ENTER YOUR_OWN_TOKEN HERE
Page Access Token	EAADDh3kSpOEBAMvXKtPFjLqxNJgZAje4QqyNbBEpSAVTSqemikwTDeTI0pvmNt9rZAKMDuQ

Figure 17. *Setting up and integrating Dialogflow with Facebook Messenger*

You should get a message saying, “Bot was started.” That means we are good to go. You must be wondering what a callback URL, Verify Token, and Page Access Token are. **Let’s try to understand and implement those.**

Callback URLs

A callback URL is nothing but a publicly accessible URL where Facebook will POST any real-time requests coming from your Facebook page.

Suppose you are trying to make payment for your food on **OnlineEats** and then you are redirected to a bank’s payment page. Now, **OnlineEats** must be giving a callback URL to banks to which they can redirect the user after payment is done.

Here Facebook will not do any redirection but will take everything that our user messages in the page’s chatbox and POST that to the webhook or callback URL.

Now, once we get the message on our server we do our intent classification and entities parsing and then formulate what you want to reply back to the user.

VerifyT oken

A verify token is an arbitrary string sent to your endpoint when the subscription is verified. The reason why it’s needed is to ensure that our server knows that the request is being made by Facebook and relates to the subscription we just configured.

Suppose somebody else gets to know your webhook and posts messages posing as Facebook, then *verify_token* will come into the picture, and you will verify if the source is correct or not. Based on this token you can handle POST requests from multiple sources as well because there will be different tokens defined for different sources but the same callback url.

Access Tokens

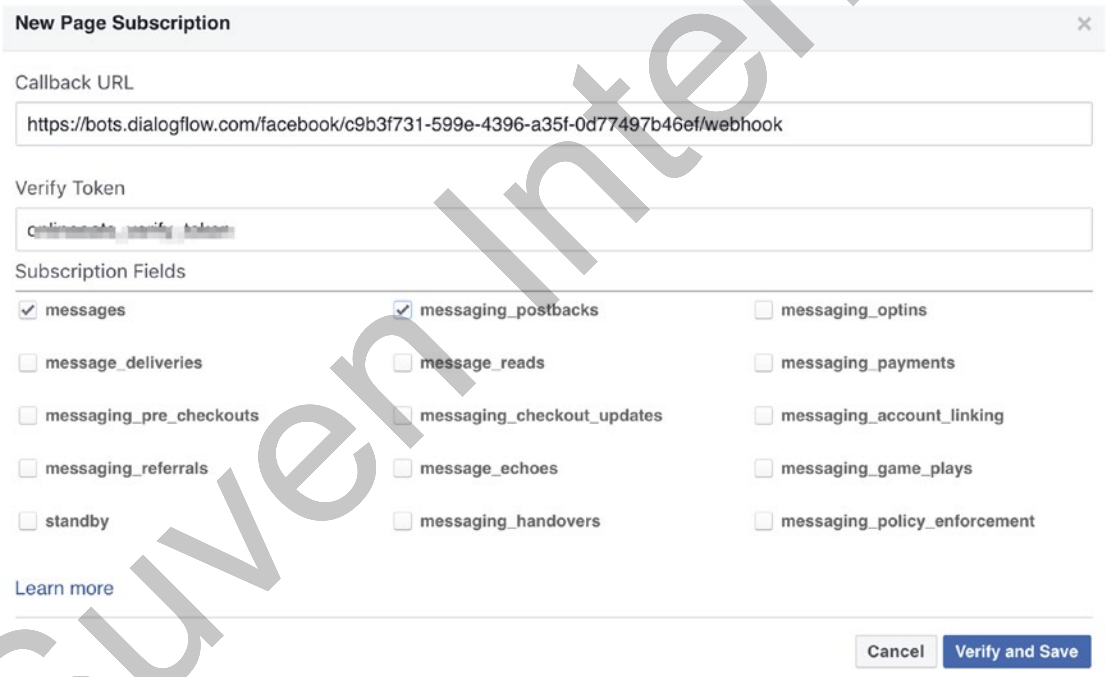
Facebook APIs require Page Access Tokens to manage Facebook pages. They are unique to each page, admin, and app and have an expiration time.

Note : Keep the callback URL and Verify Token handy for configuring the webhook now.

Configuring Webhooks

To configure our bot's webhook, let's go back to the Facebook Developer Console:

1. Click the **Setup** button under the **Add a product** section for webhooks when you click on the dashboard. If you have not already subscribed to webhooks, then you will get an option saying "subscribe to this object." Click on this to get a new pop-up and enter the following information:
   - **Callback URL**: this is the URL provided on the Facebook Messenger integration page.
   - **Verify Token**: this is the Token you created.
2. Go to Messenger ➤ Settings ➤ Setup Webhooks. You will get a pop-up like Figure 18. Add your callback url and verify the token.



New Page Subscription

Callback URL: https://bots.dialogflow.com/facebook/c9b3f731-599e-4396-a35f-0d77497b46ef/webhook

Verify Token: [redacted]

Subscription Fields

✓ messages	✓ messaging_postbacks	messaging_optins
message_deliveries	message_reads	messaging_payments
messaging_pre_checkouts	messaging_checkout_updates	messaging_account_linking
messaging_referrals	message_echoes	messaging_game_plays
standby	messaging_handovers	messaging_policy_enforcement

Figure 18. *Setting up webhooks in Facebook for Dialogflow bot*

3. Check the **messages** and **messaging_postbacks** options under **Subscription Fields.** You can definitely choose whichever is needed for your use-case.

4. Click the **Verify and Save** button. Check Figure 18 for reference.

You'll be taken back to the settings page and **Webhooks** should have a "Complete" status. Make sure to select a page to which you can subscribe your webhook for page events.

Testing the Messenger Bot

To be able to make our bot available for testing, we'll need to make our app public:

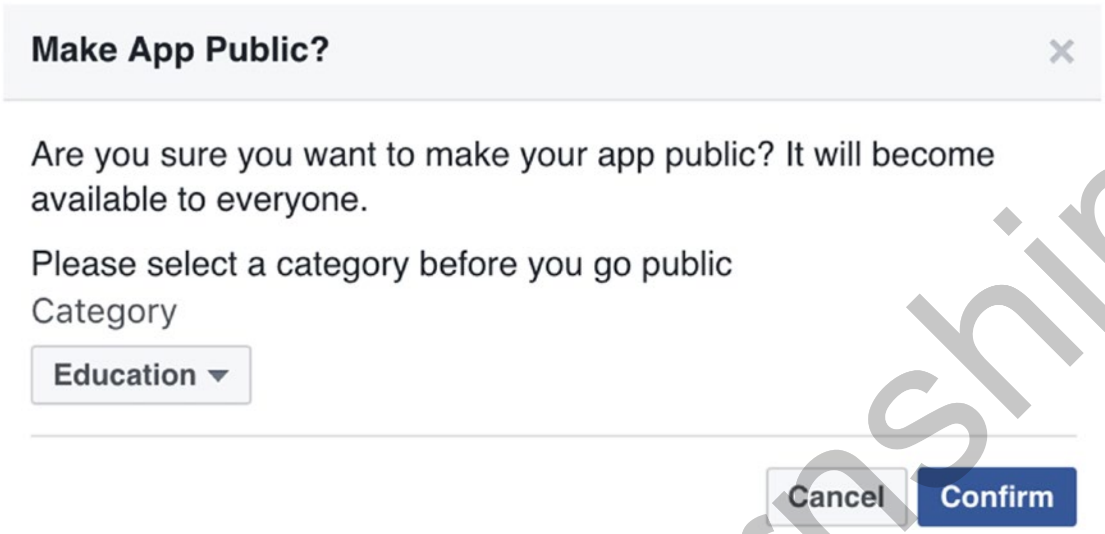
1. Click on **App Review** in the left menu of the Facebook Developer Console.

2. Click on the switch under **Make <Your APP Name> public?** If you get an **Invalid Privacy Policy URL** prompt, then go to the Basic Settings link in the dialog box and, if you haven't already, put any URL for the Privacy Policy URL, for the time being and then click on Save Changes. Now, go back to the **App Review** page and try to switch the app to public again.

3. You'll be prompted to choose a category for your app.

4. Choose **Education** from the list. Feel free to choose whichever suits your bot best.

5. Click the **Confirm** button as shown in the Figure 19, Making your facebook app public.


Make App Public?

Are you sure you want to make your app public? It will become available to everyone.

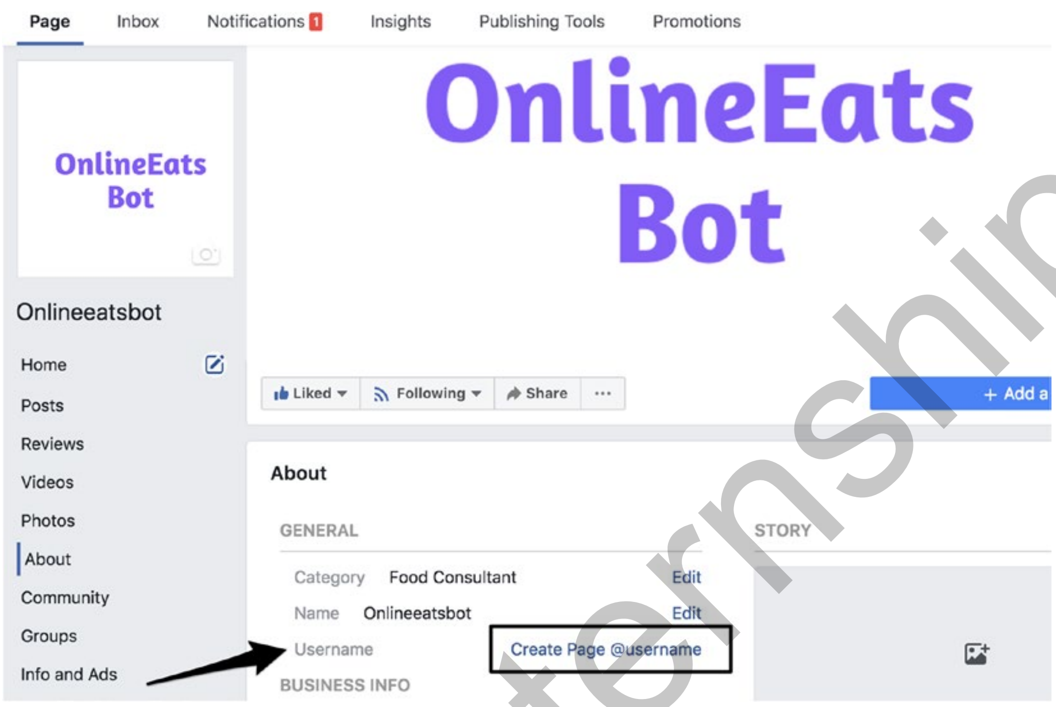
Please select a category before you go public

Category

Education

Figure 19. *Making your facebook app public*

We will also need to create a username for our page. This is the username users will chat with when using our bot. To set the username, click the **Create Page @Username** link under the About section of your page, as shown in Figure 20. This is helpful to share your page or bot with people using just a short name.

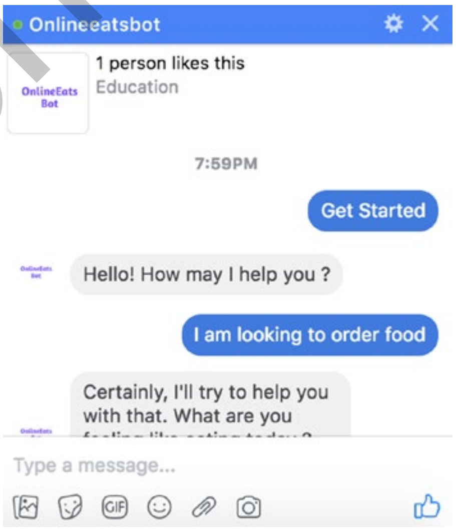


Category Food Consultant Edit
Name Onlineeatsbot Edit
Username Create Page @username

Figure 20. *Creating your Facebook bot's page username*

Let's test the same flow of our bot on Facebook Messenger that we tested on Dialogflow's website. You should be able to see how my Facebook Messenger bot responded by referring from Figure 21.1 up-to Figure 21.4.

Now-a-days Facebook is too long for validating Hence you may keep this step optional



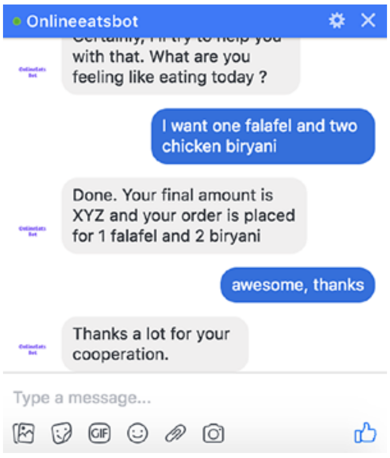
Get Started

Hello! How may I help you ?

I am looking to order food

Certainly, I'll try to help you with that. What are you feeling like eating today ?

Figure 21.1. *Facebook Messenger OnlineEatsBot Demo Screen I*


Onlineeatsbot

with that. What are you feeling like eating today ?

I want one falafel and two chicken biryani

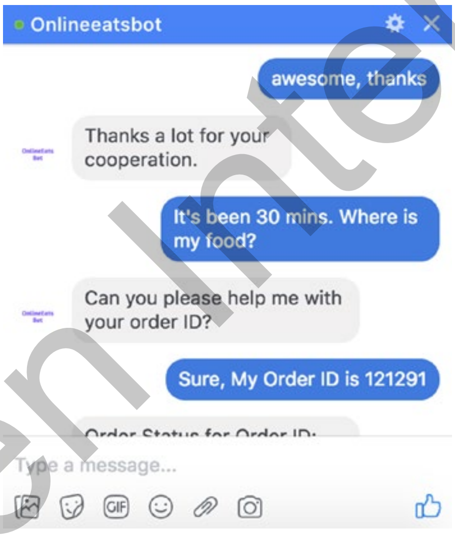
Done. Your final amount is XYZ and your order is placed for 1 falafel and 2 biryani

awesome, thanks

Thanks a lot for your cooperation.

Type a message...

Figure 21.2. *Facebook Messenger OnlineEatsBot Demo Screen II*


Onlineeatsbot

awesome, thanks

Thanks a lot for your cooperation.

It's been 30 mins. Where is my food?

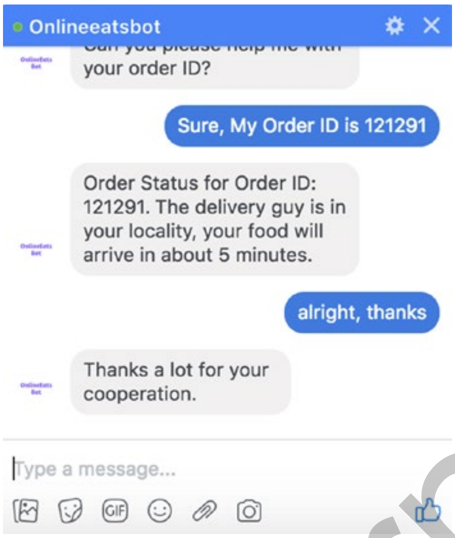
Can you please help me with your order ID?

Sure, My Order ID is 121291

Order Status for Order ID:

Type a message...

Figure 21.3. *Facebook Messenger OnlineEatsBot Demo Screen III*


Onlineeatsbot

Can you please help me with your order ID?

Sure, My Order ID is 121291

Order Status for Order ID: 121291. The delivery guy is in your locality, your food will arrive in about 5 minutes.

alright, thanks

Thanks a lot for your cooperation.

Figure 21.4. *Facebook Messenger OnlineEatsBot Demo Screen IV*

And that, folks, is how we build our bot.

Chapter 4 is going be even more interesting. In Chapter 4, we will try to achieve the same without having to depend on Dialogflow's API or dashboard.

It's always good when you have full control over everything you have, isn't it?

Note: You can go to your account settings and export an agent or import other agents directly. You can download the zip file (**OnlineEatsBot.zip**). You can use this zip to import this into Dialogflow and play with the chatbot we built in this chapter.

You must be wondering, what if I wanted to make the order placement in real-time and find the order status using APIs of the vendor/restaurant and reply to the user accordingly? It could be any API call you want to make—retrieve the data in real-time and formulate the bot's response. It's time to know how to do that before we wrap up this chapter and get ready for the next one.

Let's learn about something called "**Fulfillment**" in Dialogflow.

Fulfillment

In order to get some real-time information requested by a user, we need to develop some API or use existing ones. To achieve that using Dialogflow, we would need to set up fulfillment, which requires deploying a service and calling an API.

We will not go into the nitty-gritty of building APIs and how to deploy it but if you have ever tried using any Google or Facebook APIs, you should be familiar with at least how to call them.

We have built a small Flask-based API and deployed it to Heroku. We will be using it for fulfillment, which just takes any order_id in the url and returns a random order_status. If you are not familiar with Heroku, then please do a little **Google**.

In the code you can read how order_identity, intentName, etc., are parsed.

Find the code here: **flask_onlineeats_demo.zip**

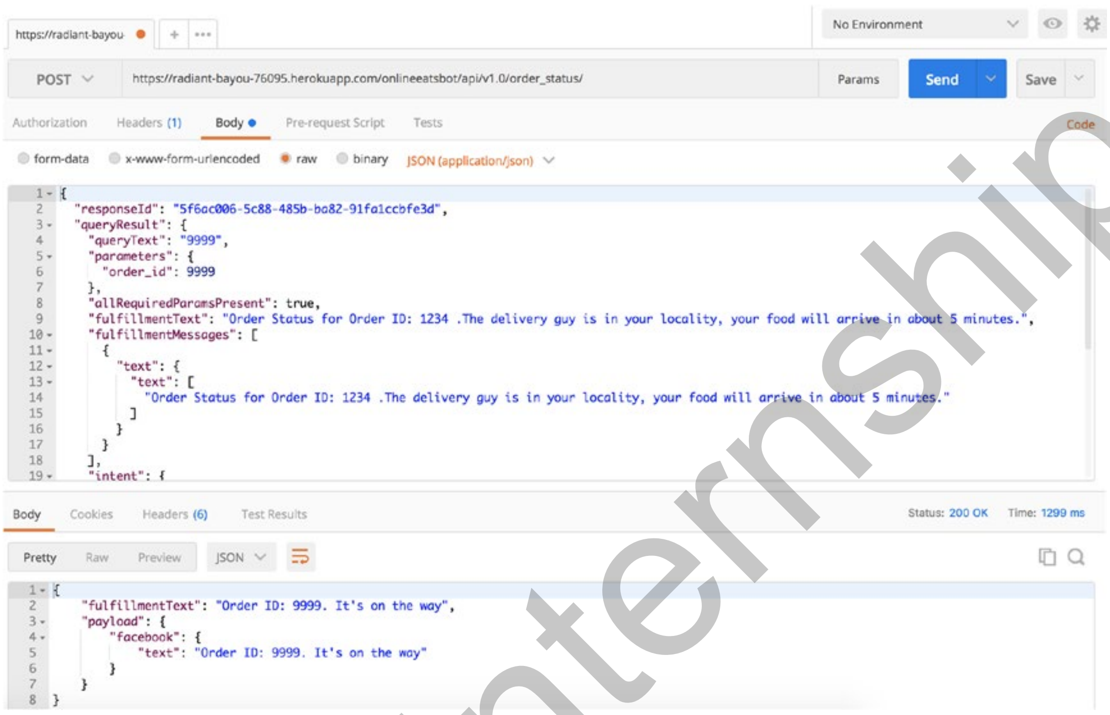
Request URL: `https://radiant-bayou-76095.herokuapp.com/onlineeatsbot/api/v1.0/order_status/`

So, in Dialogflow Fulfillment will POST the JSON response from the intent to this URL, which you would need to parse to get the relevant entities and their values and do specific actions.

You can also try to deploy the sample Flask App code on Heroku and have your own endpoint working in your bot for fulfillment.

Now, Dialogflow will POST the JSON response of the intent for which the webhook call is enabled on our endpoint. It has the code to parse the **order_id** entity and take actions based on that. Currently, the code only returns a randomly chosen status from a list.

To test if the API is working, go to POSTMAN and test it using the sample data in Figure 22. If you are running the Flask app on local, then use the local url.



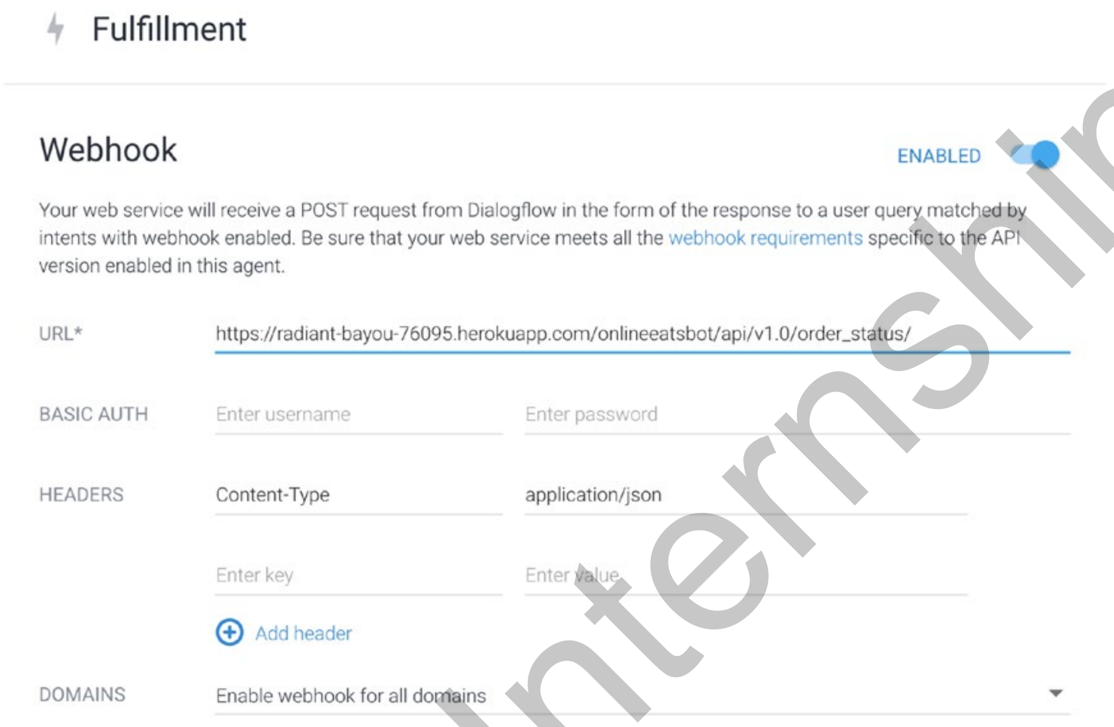
```
{
  "responseId": "5f6ac006-5c88-485b-ba82-91fa1ccbfe3d",
  "queryResult": {
    "queryText": "9999",
    "parameters": {
      "order_id": 9999
    },
    "allRequiredParamsPresent": true,
    "fulfillmentText": "Order Status for Order ID: 1234 .The delivery guy is in your locality, your food will arrive in about 5 minutes.",
    "fulfillmentMessages": [
      {
        "text": {
          "text": [
            "Order Status for Order ID: 1234 .The delivery guy is in your locality, your food will arrive in about 5 minutes."
          ]
        }
      }
    ],
    "intent": {
```

```
{
    "fulfillmentText": "Order ID: 9999. It's on the way",
    "payload": {
        "facebook": {
            "text": "Order ID: 9999. It's on the way"
        }
    }
}
```

Figure 22. *Testing the fulfillment API deployed on heroku in POSTMAN*

Enabling Webhook

So, let's go to the fulfillment page in Dialogflow and try to enable webhook (see Figure 23).


Fulfillment

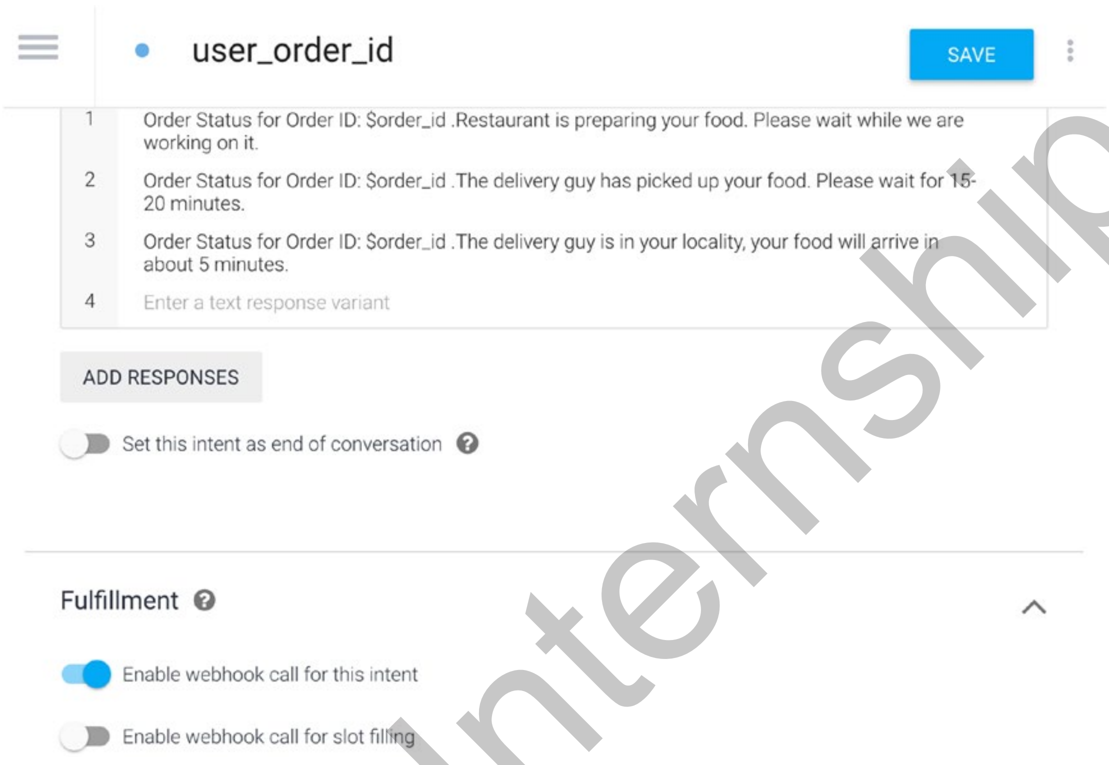
Webhook ENABLED

Your web service will receive a POST request from Dialogflow in the form of the response to a user query matched by intents with webhook enabled. Be sure that your web service meets all the webhook requirements specific to the API version enabled in this agent.

URL*	https://radiant-bayou-76095.herokuapp.com/onlineeatsbot/api/v1.0/order_status/	
BASIC AUTH	Enter username	Enter password
HEADERS	Content-Type	application/json
	Enter key	Enter value
	Add header	
DOMAINS	Enable webhook for all domains	

Figure 23. *Setting up webhook for fulfillment in Dialogflow*

Make sure you have enabled webhook call for **user_order_id** intent (see Figure 24).


user_order_id

SAVE

1 Order Status for Order ID: $order_id .Restaurant is preparing your food. Please wait while we are working on it.

2 Order Status for Order ID: $order_id .The delivery guy has picked up your food. Please wait for 15-20 minutes.

3 Order Status for Order ID: $order_id .The delivery guy is in your locality, your food will arrive in about 5 minutes.

4 Enter a text response variant

ADD RESPONSES

Set this intent as end of conversation: off

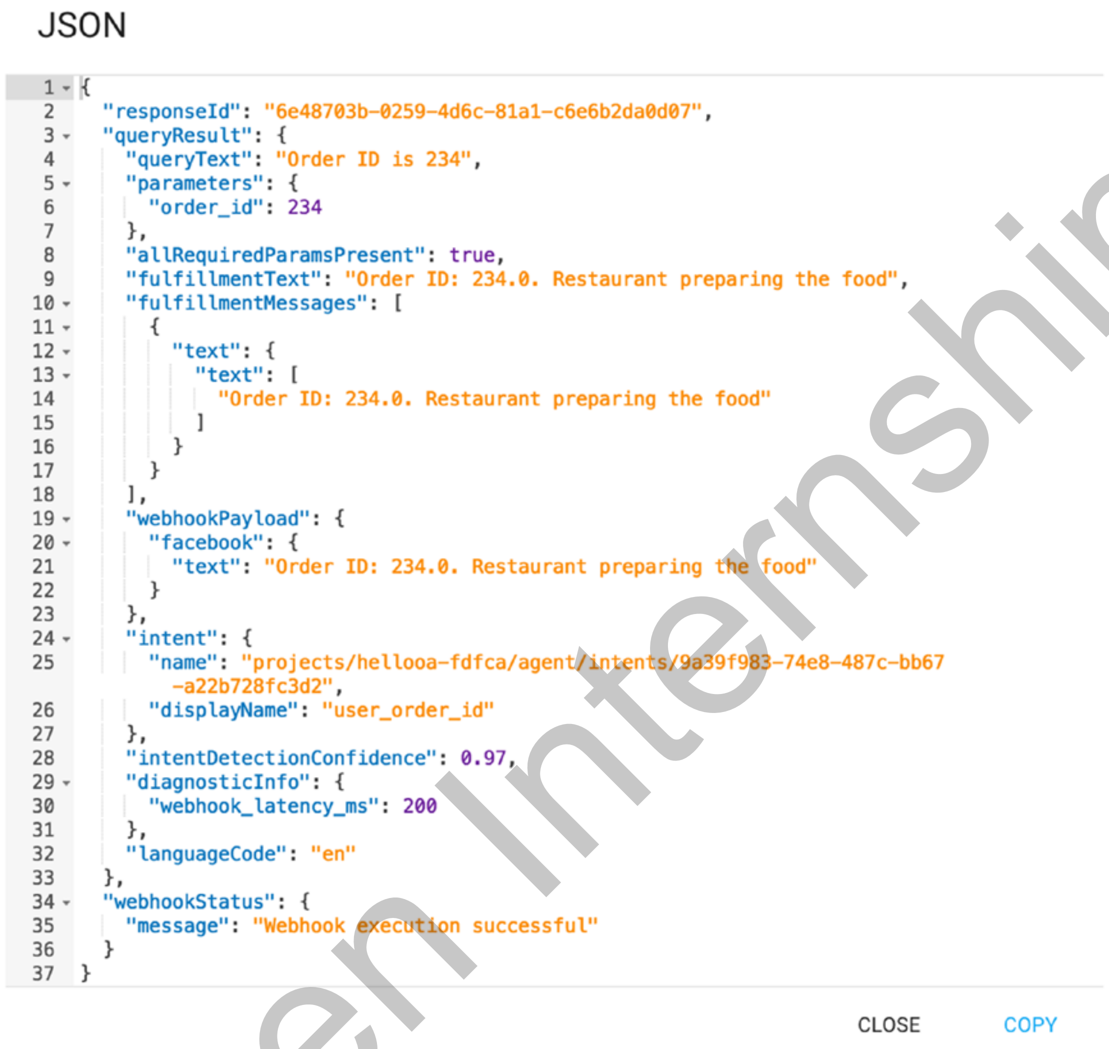
Fulfillment

Enable webhook call for this intent: on

Enable webhook call for slot filling: off

Figure 24. *Enabling webhook call for specific intent*

Dialogflow will POST a JSON body to your webhook URL, which will look like Figure 25.

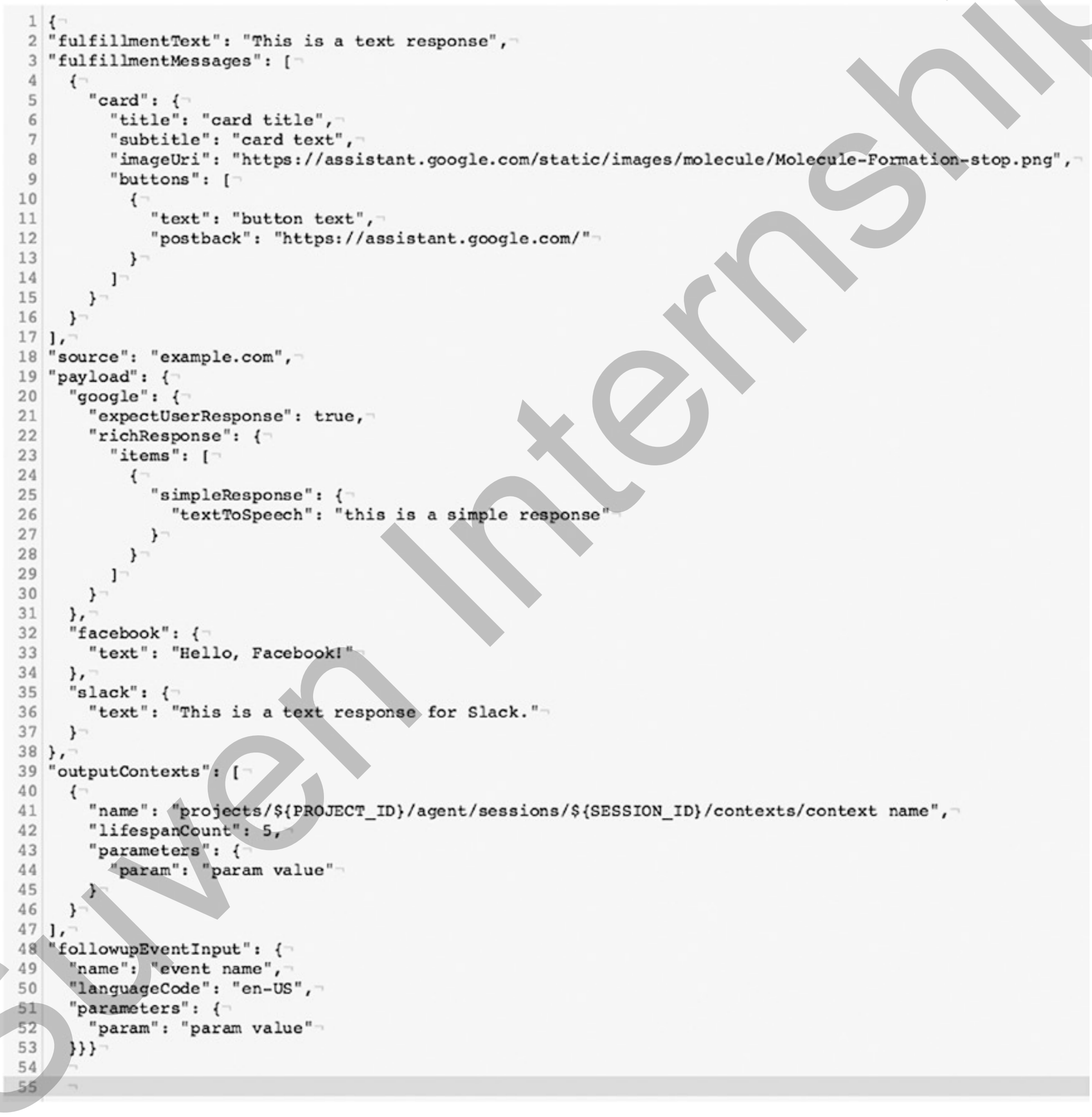


```
{
  "responseId": "6e48703b-0259-4d6c-81a1-c6e6b2da0d07",
  "queryResult": {
    "queryText": "Order ID is 234",
    "parameters": {
      "order_id": 234
    },
    "allRequiredParamsPresent": true,
    "fulfillmentText": "Order ID: 234.0. Restaurant preparing the food",
    "fulfillmentMessages": [
      {
        "text": {
          "text": [
            "Order ID: 234.0. Restaurant preparing the food"
          ]
        }
      }
    ],
    "webhookPayload": {
      "facebook": {
        "text": "Order ID: 234.0. Restaurant preparing the food"
      }
    },
    "intent": {
      "name": "projects/hellooa-fdfca/agent/intents/9a39f983-74e8-487c-bb67
        -a22b728fc3d2",
      "displayName": "user_order_id"
    },
    "intentDetectionConfidence": 0.97,
    "diagnosticInfo": {
      "webhook_latency_ms": 200
    },
    "languageCode": "en"
  },
  "webhookStatus": {
    "message": "Webhook execution successful"
  }
}
```

Figure 25. *Incoming JSON data from Dialogflow to our webhook endpoint*

Checking the Response

Dialogflow expects a response from your web service in the format shown in Figure 26 whenever they POST an intent's JSON response (shown in Figure 25) to your web service.



```
{
"fulfillmentText": "This is a text response",
"fulfillmentMessages": [
  {
    "card": {
      "title": "card title",
      "subtitle": "card text",
      "imageUri": "https://assistant.google.com/static/images/molecule/Molecule-Formation-stop.png",
      "buttons": [
        {
          "text": "button text",
          "postback": "https://assistant.google.com/"
        }
      ]
    }
  }
],
"source": "example.com",
"payload": {
  "google": {
    "expectUserResponse": true,
    "richResponse": {
      "items": [
        {
          "simpleResponse": {
            "textToSpeech": "this is a simple response"
          }
        }
      ]
    }
  },
  "facebook": {
    "text": "Hello, Facebook!"
  },
  "slack": {
    "text": "This is a text response for Slack."
  }
},
"outputContexts": [
  {
    "name": "projects/${PROJECT_ID}/agent/sessions/${SESSION_ID}/contexts/context name",
    "lifespanCount": 5,
    "parameters": {
      "param": "param value"
    }
  }
],
"followupEventInput": {
  "name": "event name",
  "languageCode": "en-US",
  "parameters": {
    "param": "param value"
  }}}
```

***Figure 26.** Response from the webhook URL expected by Dialogflow*

If you are thinking that your API's response should be exactly in the same format as Figure 26, then relax—that's not the case. Your intent won't throw errors because all of the keys in the JSON body are optional.

Here is how my API response looks and works perfectly:

```
{
    "fulfillmentText": "Order ID: 9999. It's on the way",
    "payload": {
        "facebook": {
            "text": "Order ID: 9999. It's on the way"
        }
    }
}
```

When I try to hit the same API again I get a different order status text but with the same format expected by Dialogflow engine.

```
{
    "fulfillmentText": "Order ID: 9999. Rider has picked up your food,
    please wait for another 10-15 minutes",
    "payload": {
        "facebook": {
            "text": "Order ID: 9999. Rider has picked up your food, please
            wait for another 10-15 minutes"
        }
    }
}
```

fulfillmentText is the key that matters for the agent to reply something back to the user.

Now, try the bot with the public URL or in the Dialogflow agent itself to see the responses coming from the API instead of the default static responses we added earlier.

This is how we can integrate an external API or our own API using Dialogflow's fulfillment feature into our chatbot to make things dynamic and real-time.

Outcome of this Project

In this project we learned about Dialogflow and how to use Dialogflow to build a chatbot. We learned to define intents and their respective entities.

We built a simple food ordering chatbot that understands the order food intent from the user and also understands what food items the user has asked for as well as the quantity. We also enhanced our chatbot to let users ask about the status of their order and take their order ID from them and formulate a response with different order statuses.

We also learned about fulfillment in Dialogflow, where we pulled the status of the order from our own API and gave the response to the user based on that. We learned to create a web demo of our chatbot, and we also integrated our bot with Messenger.

It is expected that the Chatbot coded by the Intern should ensure as many possible cases to provide a smooth experience to the end-user. For smoothness, one needs to work hard on the training phrases.

After completing the project , one needs provide the bot link on submission. On submitting , Our AI Engine would assess your work through a 5 mins MCQ test.