A STEP BY STEP INTRODUCTION

BEGINNER TO MASTER

Dialogflow tutorial

NLP MADE EASY

JANA BERGANT

DialogFlow introduction

DialogFlow can help us and our apps to **understand and parse natural language**. I's build on **Google's infrastructure** and uses **machine learning** to process natural language.

In simple words, it helps us understand what our users want.

It is Google's service so naturally, it is **optimized for Google Assistant**, but it can be integrated into other chat platforms too. Platforms like these:





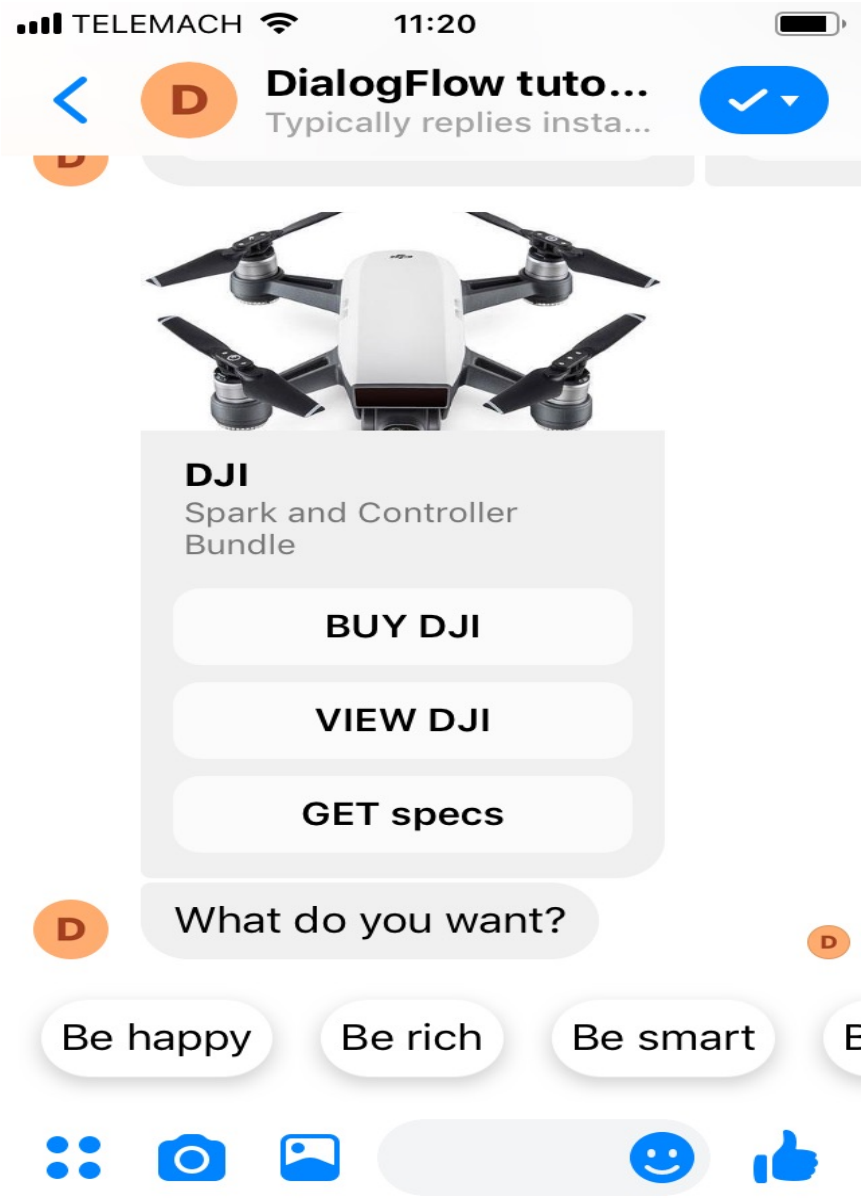




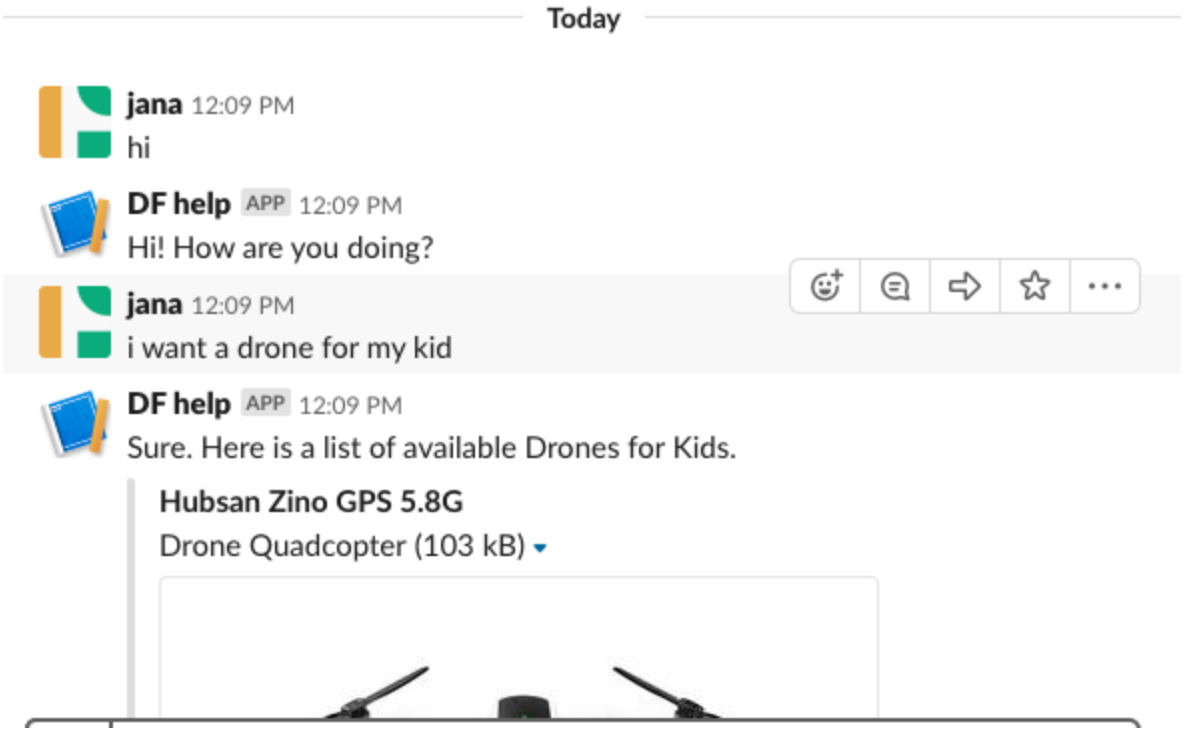
I find it super intuitive and easy to use.

What I will show you in this tutorial is how to create chatbots that can be hosted on different platforms.

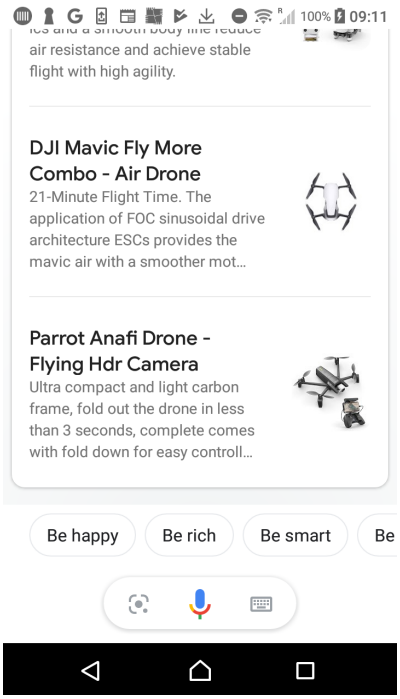
I'll show you how to integrate with Facebook Messenger. Like this:



Also, we will integrate with Slack. Like this:



And of course Google Assistant:



In the process, you will learn all the building blocks of DialogFlow and how to use them efficiently to build dialogues and get information from the conversation.

We'll:

- Create an agent and go through its settings
- I'll show you how to create bots that can speak in different languages
- And we'll use the building blocks of DialogFlow; that is intents, entities, contexts, and fulfillment
- We'll use entities to get parameters from the conversation
- And we'll use fulfillment to store that information into a database
- I'll also show you different ways of implementing code
- We'll simplify things with follow up intents
- And create powerful dialogues with contexts
- In the end, I show you how to integrate with one-click integration
- And how to integrate with a backend app as a middle man to DialogFlow

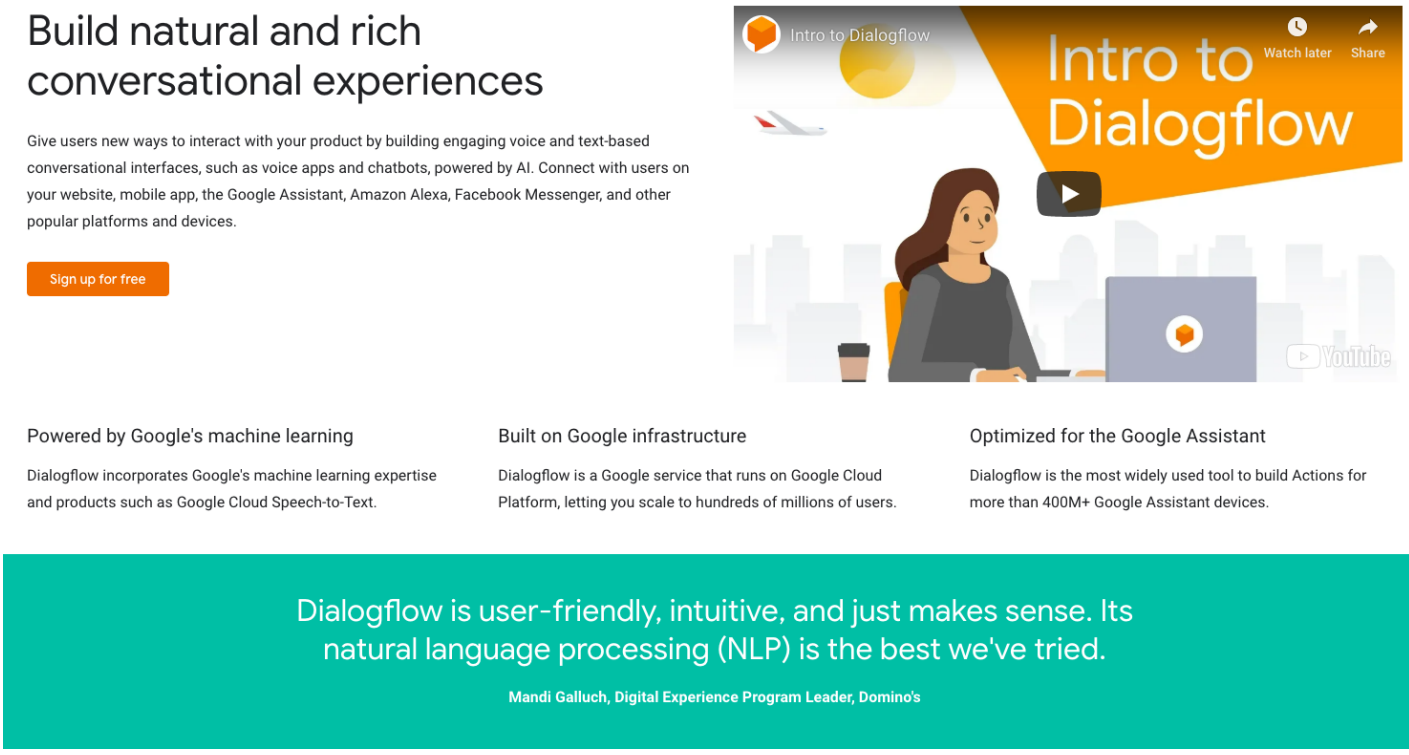
Besides this free tutorial, you will get free updates if you get in touch with my chatbot: m.me/365868714267545.

Chatbot will also send you links to my articles so you can stay up to date. Once in a while while I have time between my project I also create a new course. I update courses regularly so they stay up to date. Therefore talk to my bot: m.me/365868714267545 and she'll get you all the information you need including some course coupons special for you.

(Actually, that is me behind the chatbot, she just helps me to broadcast news and give you quick help, but you already know that ;))

DialogFlow agent

DialogFlow is a **Google service for natural language processing**. It's powered by machine learning, built on Google's infrastructure and it's optimized for Google Assistant, but it can be integrated to other chat platforms.



It supports more than 20 languages including Spanish, French, and Japanese. Here is a list of supported languages: https://cloud.google.com/dialogflow-enterprise/docs/reference/language

Name	Tag	STT	TTS	Telephony	Knowledge	Sentiment
Chinese - Cantonese	zh-HK	✔				
Chinese - Simplified	zh-CN	✔				✔
Chinese - Traditional	zh-TW	✔				✔
Danish	da	✔	✔			
Dutch	nl	✔	✔			
English	en	✔	✔	✔	✔	✔
English - Australia	en-AU	✔	✔		✔	
English - Canada	en-CA	✔	✔		✔	
English - Great Britain	en-GB	✔	✔		✔	
English - India	en-IN	✔	✔		✔	
English - US	en-US	✔	✔	✔	✔	✔
French	fr	✔	✔			✔
French - Canada	fr-CA	✔	✔			
French - France	fr-FR	✔	✔			✔
German	de	✔	✔			✔
Hindi	hi	✔				
Indonesian	id	✔				
Italian	it	✔	✔			✔
Japanese	ja	✔	✔			✔
Korean	ko	✔	✔			✔
Norwegian	no	✔				
Polish	pl	✔	✔			
Portuguese	pt	✔	✔			✔
Portuguese - Brazil	pt-BR	✔	✔			
Russian	ru	✔	✔			
Spanish	es	✔	✔			✔
Spanish - Latin America	es-419	✔				
Spanish - Spain	es-ES	✔	✔			✔
Swedish	sv	✔	✔			
Thai	th	✔				
Turkish	tr	✔				
Ukranian	uk	✔	✔			

All of them are supported in a speech to text, and some of them support text to speech.

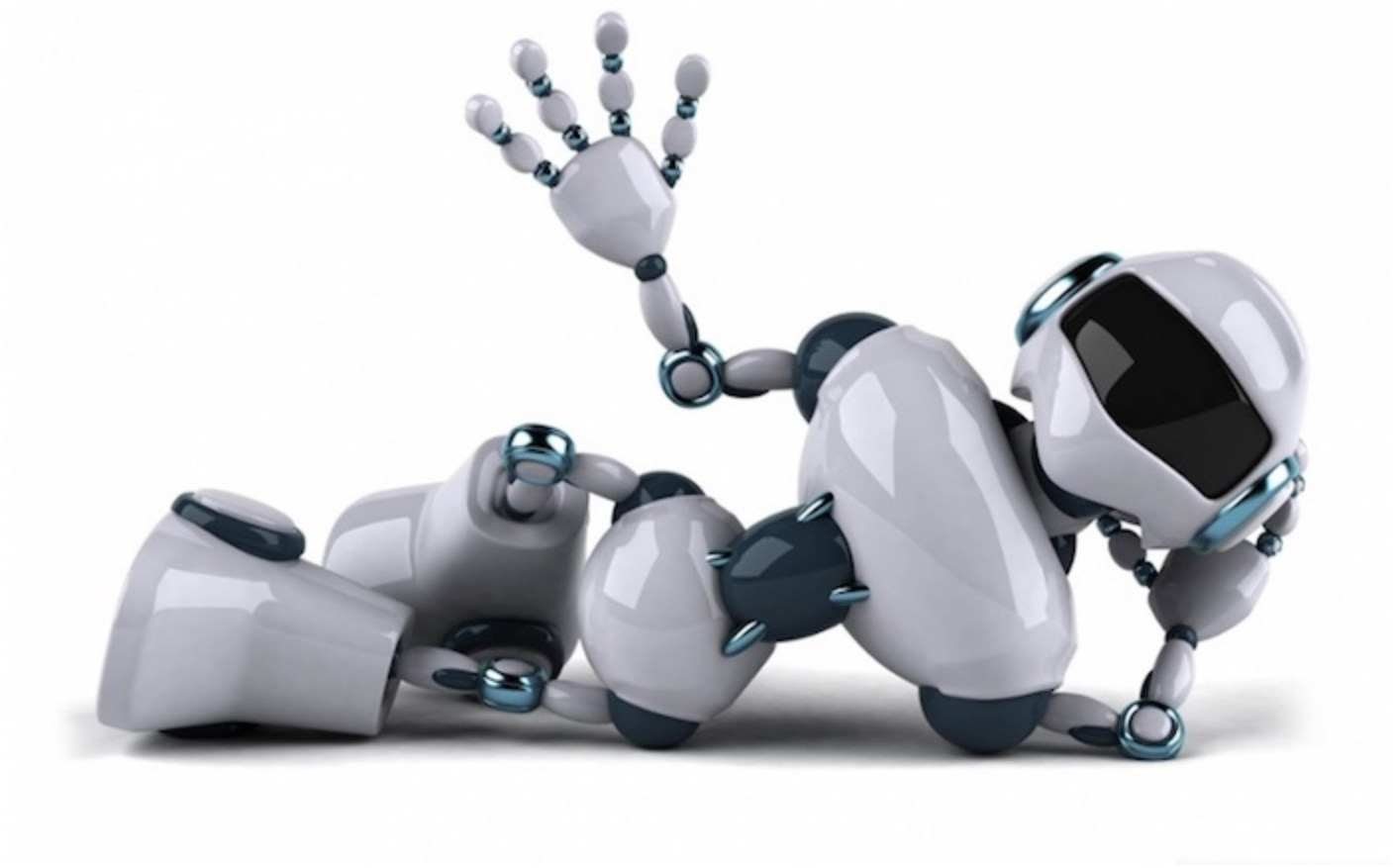
We'll use DialogFlow for parsing natural language. In a simple form:

Natural language parsing means that we can interpret what the people want and what information they have provided and transform all that into something a program can understand and act upon.

Like a black box. In the DialogFlow we send text and voice and outcome intents, actions and parameters that we can then use in the program. Don't worry! I'll soon tell you everything you need to know about each of them.

What you need to do first, is to create an **agent**. Let me tell you what an agent is.

As we mentioned previously, an agent helps you process user's input into structured data, that you can use to return an appropriate response. You can define all of these things inside one or many intents, which define how to map user's input into a corresponding response. An agent is like a module with all the intents we need to recognize.



First, you need to sign up. It's free. It's free for the standard edition. And standard addition has unlimited text queries. Here you can compare packages and view pricing: https://dialogflow.com/pricing, https://cloud.google.com/dialogflow-enterprise/pricing

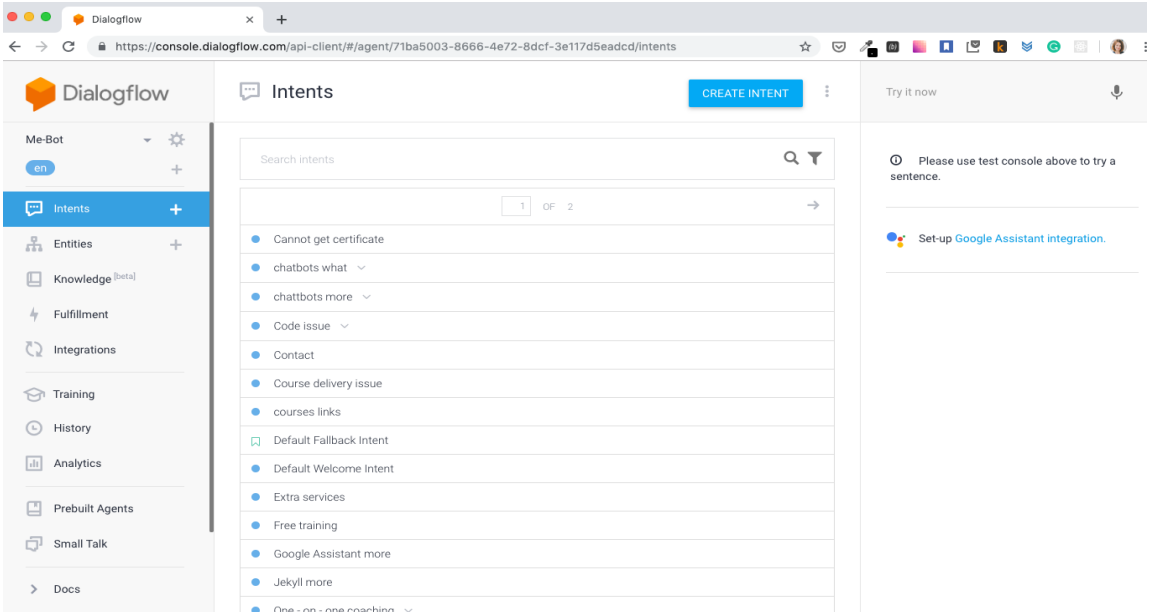
	Free Standard Edition	Pay as you go Enterprise Edition	
		Essentials	Plus
Knowledge Connectors (Beta)	Limited	Limited	Unlimited*
Text or Google Assistant	Unlimited*	Unlimited* $0.002 per request	Unlimited* $0.004 per request
Audio Includes speech recognition and synthesis (Beta)	Limited	Unlimited* $0.0065 per 15 sec of audio	Unlimited* $0.0085 per 15 sec of audio
Phone Call (Beta) Includes phone connectivity, speech recognition, natural language understanding, speech synthesis	Limited	Unlimited* $0.05 per min of phone call processed	Unlimited* $0.065 per min of phone call processed
Toll-free phone call (Beta)	None	$0.06 per min of phone call processed	$0.075 per min of phone call processed
Service Level Agreement	None	SLA	
Support	Community support and email	Eligible for Cloud Support package with commited response times for supporting production applications	
Terms of Service	Dialogflow ToS	Google Cloud Platform ToS	
Choose a plan →	Ideal for small to medium businesses or those who want to experiment with Dialogflow Start now	Ideal for businesses that need to easily scale to support changes in demand from their users Learn more	

If you will want to add voice interaction in the future, then you would have a thousand requests per day and 15 thousand per month.

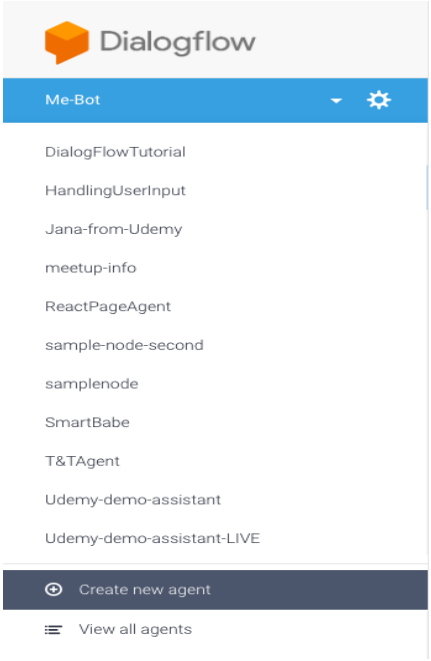
While the use of the DialogFlow Standard Edition is free, there are limits on the number of requests that you can make.

That is what we'll use, we don't need more. So, you can sign up.

Now I'm in the console. When you log in, you might have a different screen. I already have some agents created.



Click the Create agent button. I have it here under the arrow of the current agent and a list of existing ones.

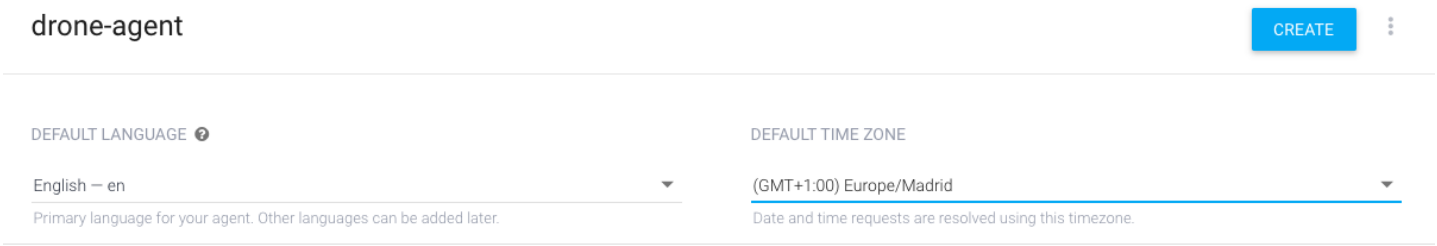


We can give our agent a name. I will call this agent drone-agent since we'll be doing a bot for drone selling website. Like this.

Then we can choose a language. As said, DialogFlow supports more than 20 languages. The popular ones. Mine, that is Slovenian is not among them. But I want to create an English-speaking bot anyway.

Are you asking me if an agent can speak and understand more languages? Yes, of course. I can show you later how to add them.

Choose your timezone. Date and time will be resolved by this time zone.



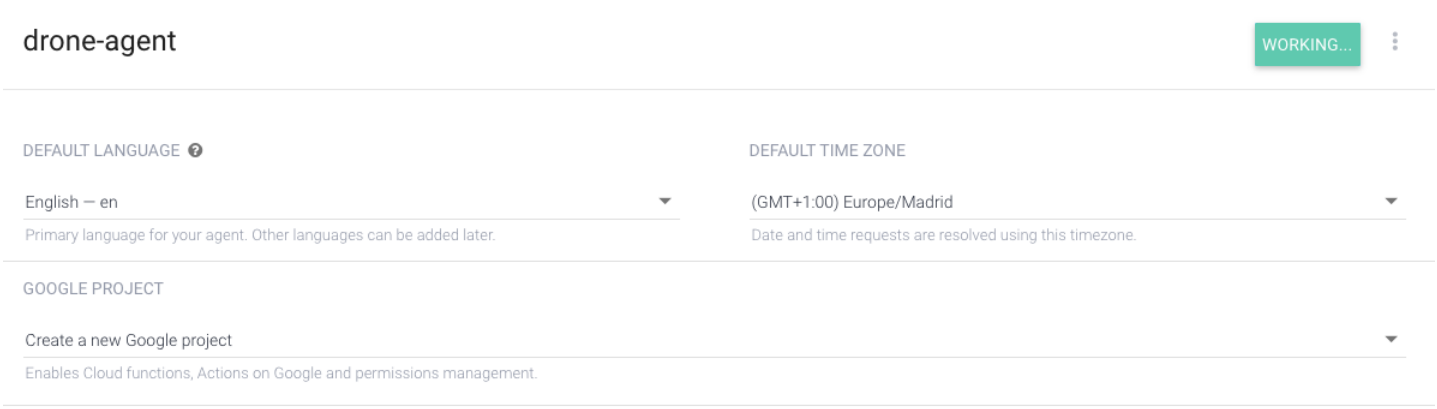
And here is the Google project.

DialogFlow is linked to the Google project. I will explain how things are linked together in the background soon. Let's first finish this.

Your interface might look a bit different if you don't have any Google projects yet. I have some, so I have to choose to Create a new Google project. If you don't have a project yet, then you might have here a sign saying: New project will be automatically linked to the agent after saving. Ok?

Click the Create button up here, so that your agent will be generated. It will take a while. Google must create a project and an agent.

Ok. You might be wondering what does a Google project have to do with my DialogFlow agent. Well, DialogFlow is Google's service. But let me explain more about what is going on in the background.



I'll do that in the next article. Bye.

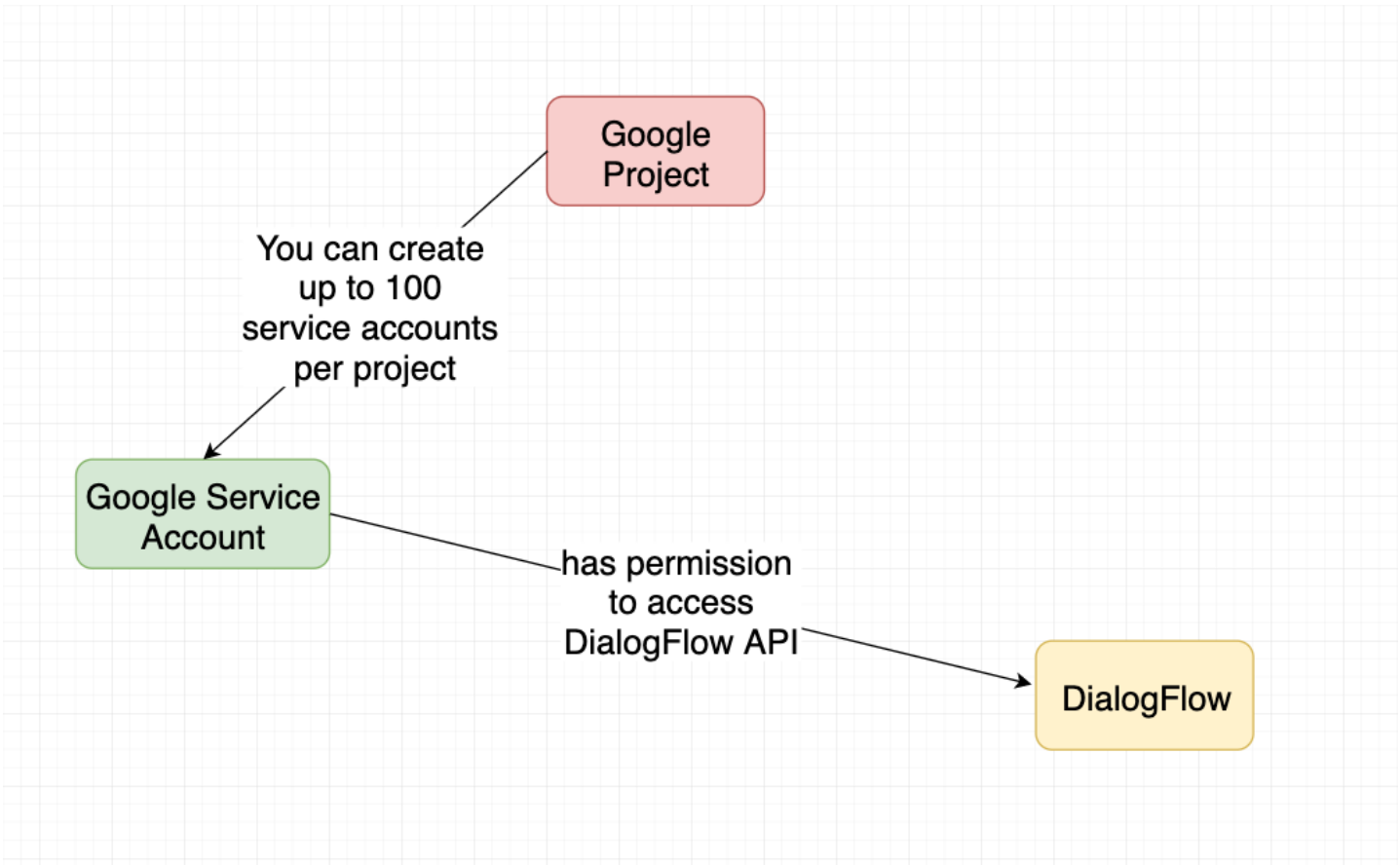
Google project and service accounts

Hi and welcome back. In the previous article, we have created DialogFlow agent. We have said that:

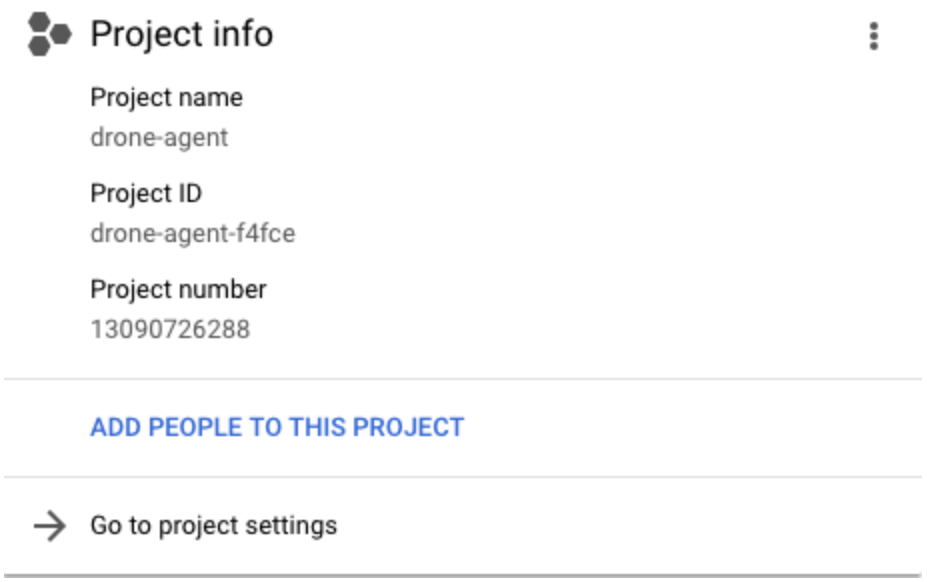
DialogFlow agent is a module that contains all the intents we need to recognize in our bot.

But while creating agent, the Google project was created. In this article, I'll show you what is going on behind the scenes when we create a new agent.

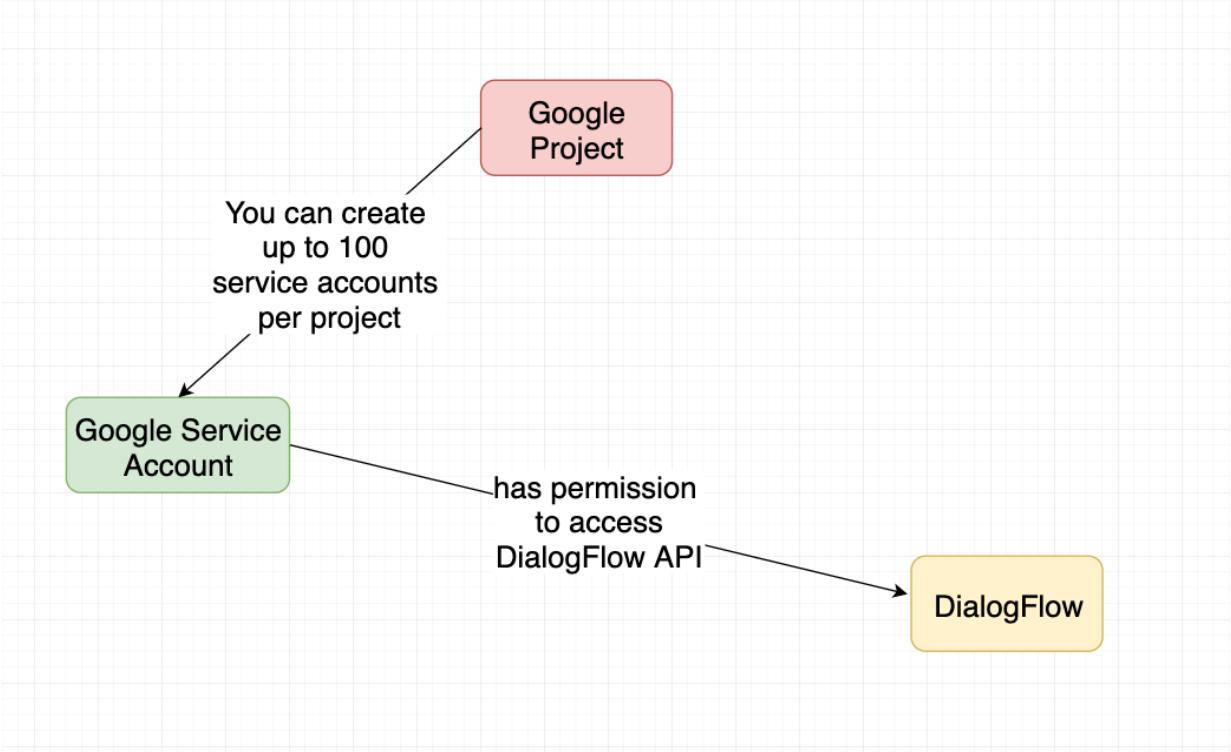
Google services share the same authentication flow. Here I have a diagram that shows you what is created behind the scenes when we create an agent.



First, we see that a google project is created. Go to Google console: https://console.cloud.google.com. You might have to login first. Here you'll see all your Google projects. A new one will appear here. The one for the DialogFlow agent. But it is not just a Google project that is created.



Along with the project a service account is created. Google service account is like a user, a virtual user account to which permissions are attached. Like permission to DialogFlow service. And each Google project can have up to 100 service accounts.



When we create an agent a project is created and a service account that has permission to DialogFlow. This is how permissions are handled in Google. And a Google service account would get a pair of keys in order to authenticate. But this is done behind the scenes when we create a new agent. For more information about service accounts check put these links:

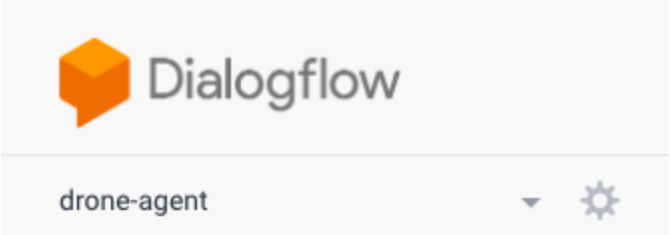
https://cloud.google.com/iam/docs/understanding-service-accounts

https://cloud.google.com/iam/docs/service-accounts

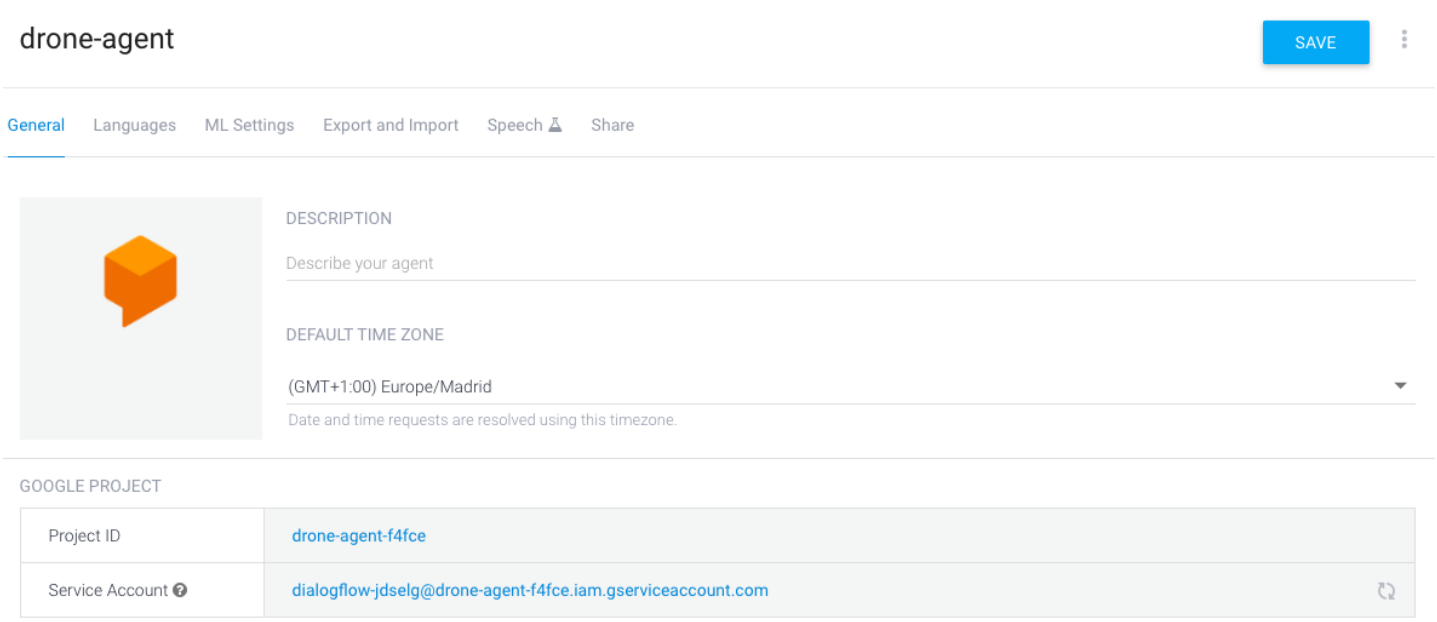
https://dialogflow.com/docs/concepts/google-projects-faq

DialogFlow agent settings

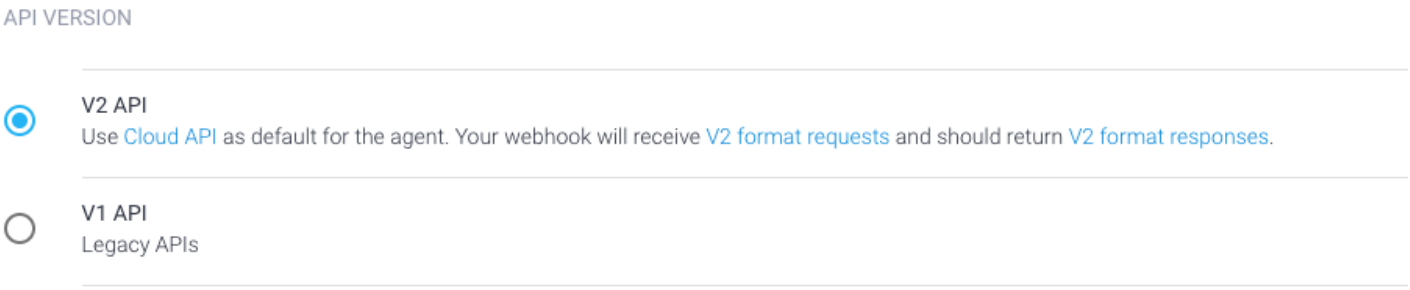
Go to DialogFlow to your agent. Here, right beside your agent's name, you see a gear icon. Click on it.



This is the settings of your agent. And in the General tab, you can see Your Google project Id And the service account, that was created for you.



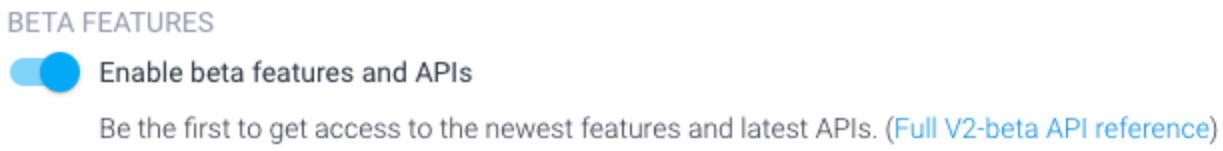
DialogFlow also has two versions, one and two. The first version will soon be terminated.



Beta features

Beta features enable you to have three powerful features of DialogFlow:

- Knowledge connectors,
- Multiple versions of your agent for different environments
- And Text to speech feature.



Let me give you a little introduction to each topic.

Knowledge connectors

If you have a web page or a text or PDF document with questions and answers that are frequently used that you can connect DialogFlow to them as a source for your FAQ bot.

Simply put, Dialoglow will create intents from your FAQ sources and Knowledge Base Articles.

You do this with Knowledge connectors. With knowledge connectors, you can parse documents and web sites to find questions and answers for bot users. Questions become training phrases as DialogFlow calls the input text in the intents and answers to become responses. Knowledge connector helps you create FAQ intents with training phrases and responses, so super simple intents.

You can add one or more knowledge bases.

But I will warn you.

Knowledge connectors offer less response precision and control than intents.

Also, knowledge connectors are currently only available for English and related locale languages.

When using both intents and knowledge connectors, you should define your intents to handle complex user requests that require special handling and precision. You can let knowledge connectors handle simple requests with responses automatically extracted from your documents.

When you identify content in FAQs that users may want to know more about, you can convert questions into explicit intents, which gives you full control.

Ok, what about Versions and Environments?

Versions and Environments

You can have different versions of your agent. You can use different versions of agent for different environments.

DialogFlow environments allow users to publish different versions of their agent. When calling the API endpoints you then specify which version of an agent you wan to use. This is how you can easily separate development and production and also different environments.

The biggest benefit is that a version of your agent can be tested in one environment without affecting the production version of your agent.

Text to speech

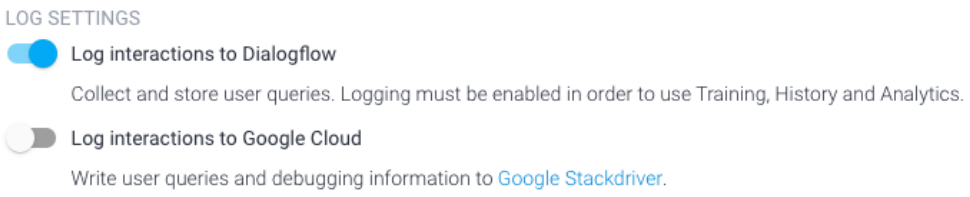
Applications often need a bot to talk back to the user. When you enable beta features, you can also use text to speech.

If you enable it DialogFlow will automatically convert default text responses to speech in all conversations. The output audio will be included in the response.

DialogFlow will use Cloud Text-to-Speech powered by DeepMind WaveNet to generate speech responses from your agent.

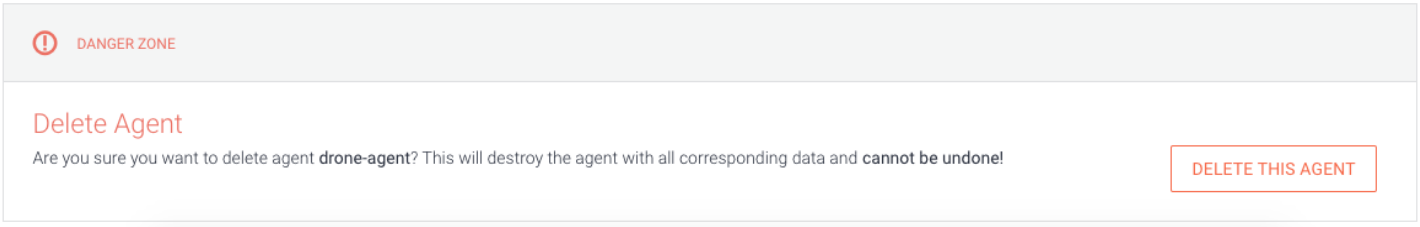
Other Setting

What is also important here in the settings is that you need to have logging enabled if you want Training, History, and Analytics to work. I'll show all these features to you in the next few articles.



Pic: s204_dialogflow-agent-settings/dialogflow-logging.png

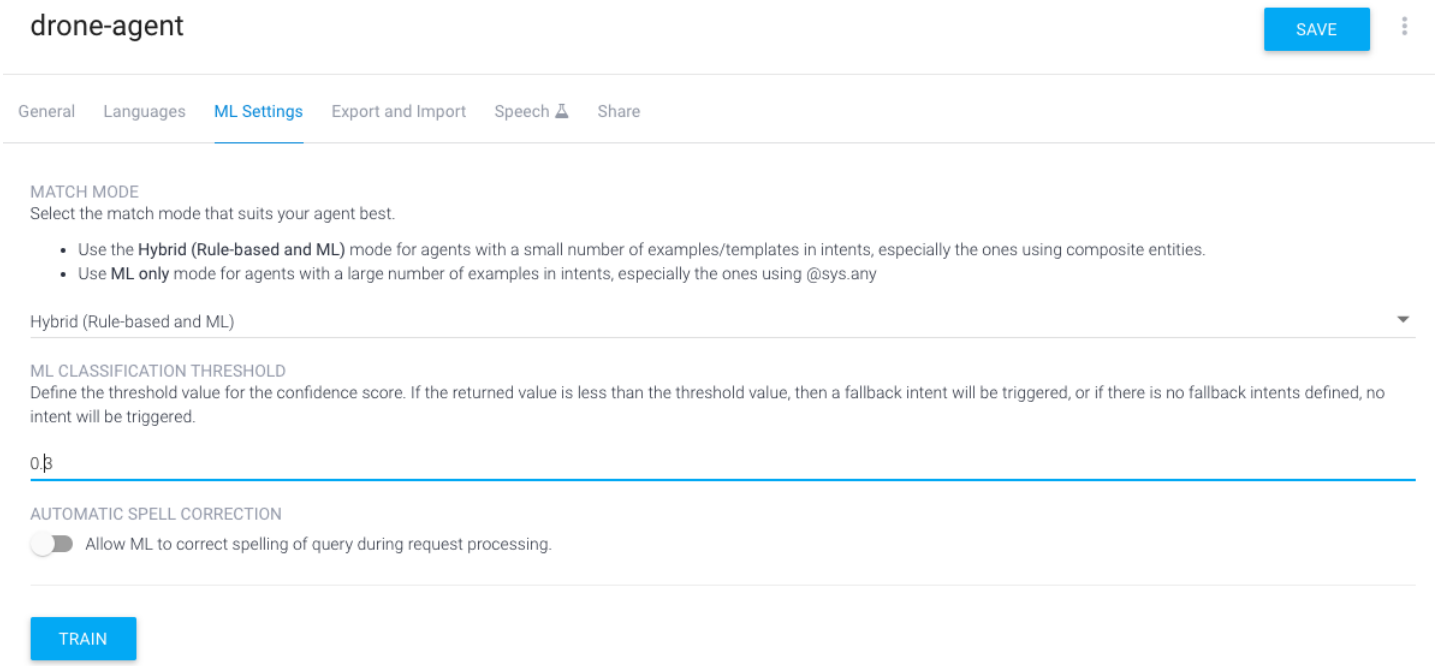
And down here you can delete an agent. This is obvious.



So, ok. While we are here, we can take a look at what else we have here in the settings.

Machine Learning settings

Ok, the next thing is Machine Learning settings. This down here is the settings you'll be playing around with.



This is a threshold for the confidence score. That means, what is the lowest percentage of certainty DialogFlow has to have, to confirm one string belongs to the intent. In simple terms How sure DialogFlow has to be that some string means something.

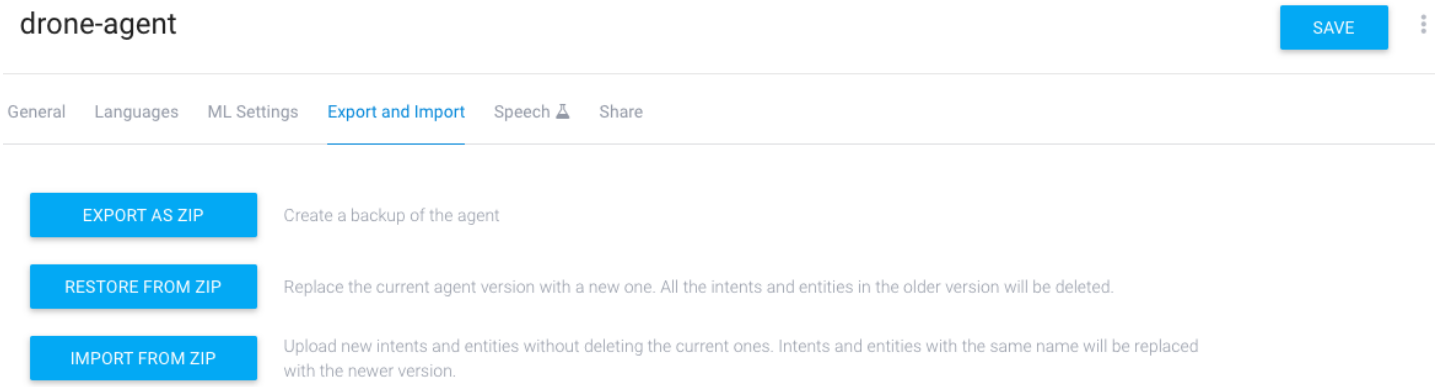
If the returned value is less than the threshold value, then a fallback intent will be triggered.

So, as we said DialogFlow uses machine learning to recognize the intents from a string. DialogFlow agent helps you process user input into structured data that you can use to return an appropriate response. By default, DialogFlow uses rule-based and machine learning match mode.

Rule-based mode is best for agents with a smaller number of intents and examples. Use machine learning only when you have a large sum of entities and examples. Ok?

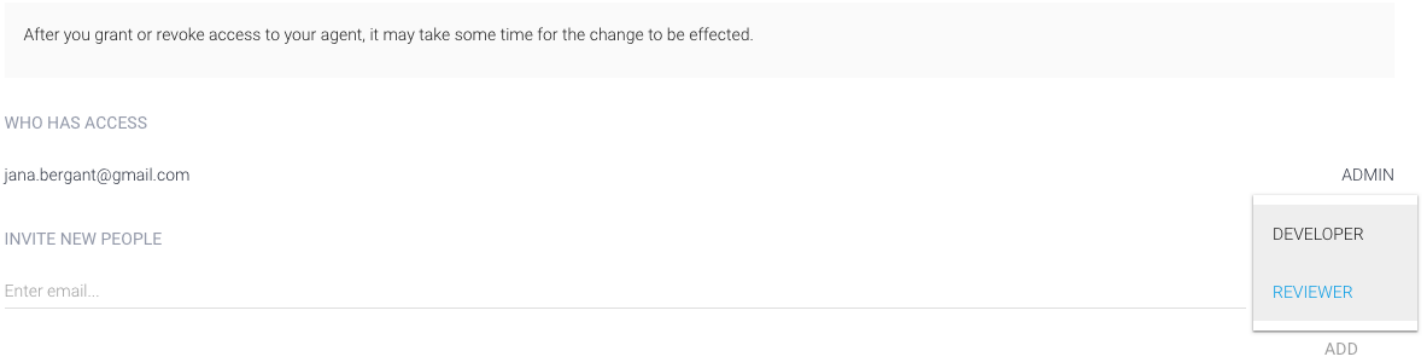
Import and export data

You can simply import and export all the information about the agent. If you don't want the existing intents to be deleted, then be sure to use the import option.



User roles

And the last tab. You can invite other developers or collaborators to work with you on setting up the intents. It is common for multiple team members to collaborate on building an agent. Using roles, you can control the level of access granted to team members.



If you will use DialogFlow's API you will need to set up authentication. By setting set up authentication you will control access with roles and service accounts. We talked about roles and service accounts in the previous article.

You will need to create a new service account and select the right role. We do that in my course Chatbot for a web page. Where we call DialogFlow API from a web interface we build ourselves with React and node. You may also have one or more applications that send requests to an agent.

If you want to share access to an agent with your team you have three roles available. Admin has full access to DialogFlow Console to create and edit agents. And also to Google project and it's resources. The editor has edit rights for an agent, but cannot create or delete an agent. The same with google project. And the reviewer can only read. Simple.

We looked at almost all the settings. The last one is languages.

Since I have more to say about this topic I will put it in a separate article.

So, now you know where to adjust the confidence score, switch between rule-based mode and machine learning.

DialogFlow agent and multiple languages

Hi and welcome back. What happens when your target audience speaks French and English? What can you do? Well, there is a way to talk in both languages with them. To see how to look at this article.

Here I have a list of all the languages that are currently supported by DialogFlow. English, French, German, Chinese, Russian, Italian ... the most used languages are here.

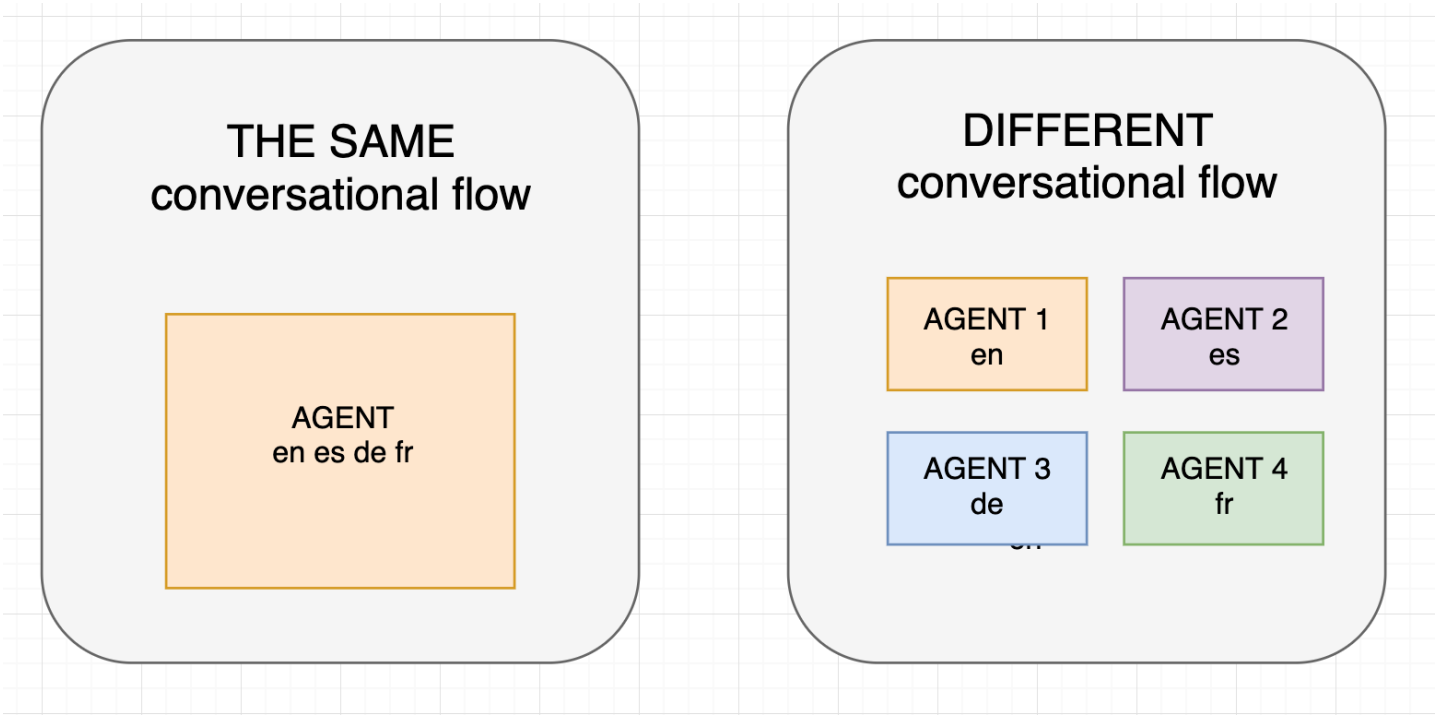
Name	Tag	STT	TTS	Telephony	Knowledge	Sentiment
Chinese - Cantonese	zh-HK	✔				
Chinese - Simplified	zh-CN	✔				✔
Chinese - Traditional	zh-TW	✔				✔
Danish	da	✔	✔			
Dutch	nl	✔	✔			
English	en	✔	✔	✔	✔	✔
English - Australia	en-AU	✔	✔		✔	
English - Canada	en-CA	✔	✔		✔	
English - Great Britain	en-GB	✔	✔		✔	
English - India	en-IN	✔	✔		✔	
English - US	en-US	✔	✔	✔	✔	✔
French	fr	✔	✔			✔
French - Canada	fr-CA	✔	✔			
French - France	fr-FR	✔	✔			✔
German	de	✔	✔			✔
Hindi	hi	✔				
Indonesian	id	✔				
Italian	it	✔	✔			✔
Japanese	ja	✔	✔			✔
Korean	ko	✔	✔			✔
Norwegian	no	✔				
Polish	pl	✔	✔			
Portuguese	pt	✔	✔			✔
Portuguese - Brazil	pt-BR	✔	✔			
Russian	ru	✔	✔			
Spanish	es	✔	✔			✔
Spanish - Latin America	es-419	✔				
Spanish - Spain	es-ES	✔	✔			✔
Swedish	sv	✔	✔			
Thai	th	✔				
Turkish	tr	✔				
Ukranian	uk	✔	✔			

And you can add up to 15 languages to one agent. That is a lot. When you do that, DialogFlow agent can understand and parse the languages you add to an agent. When you add a language to an agent, you get a **multilingual agent**. With this your bot can support multiple languages and locales within the same agent, allowing you to specify **different Training Phrases and Responses for each language your agent supports**.

We'll take a look at Training Phrases and responses soon. But basically, **Training Phrases is everything a user might say to the bot** when having one kind of intent. And Responses is what the bot says back.

If you add several languages to the same agent, all languages will have the same conversational flow. All intents will be duplicated across languages.

It depends on the use case but sometimes conversation flow may not be the same in every language. If that is the case you would need to create different agents for different languages. But if the structure is the same for every language, then you just add languages to the same agent. It depends on what you want. Like everything in development. It depends on the use case. If you have the same conversational structure, then you add languages to one agent. But if you have a different structure, then you create several agents.



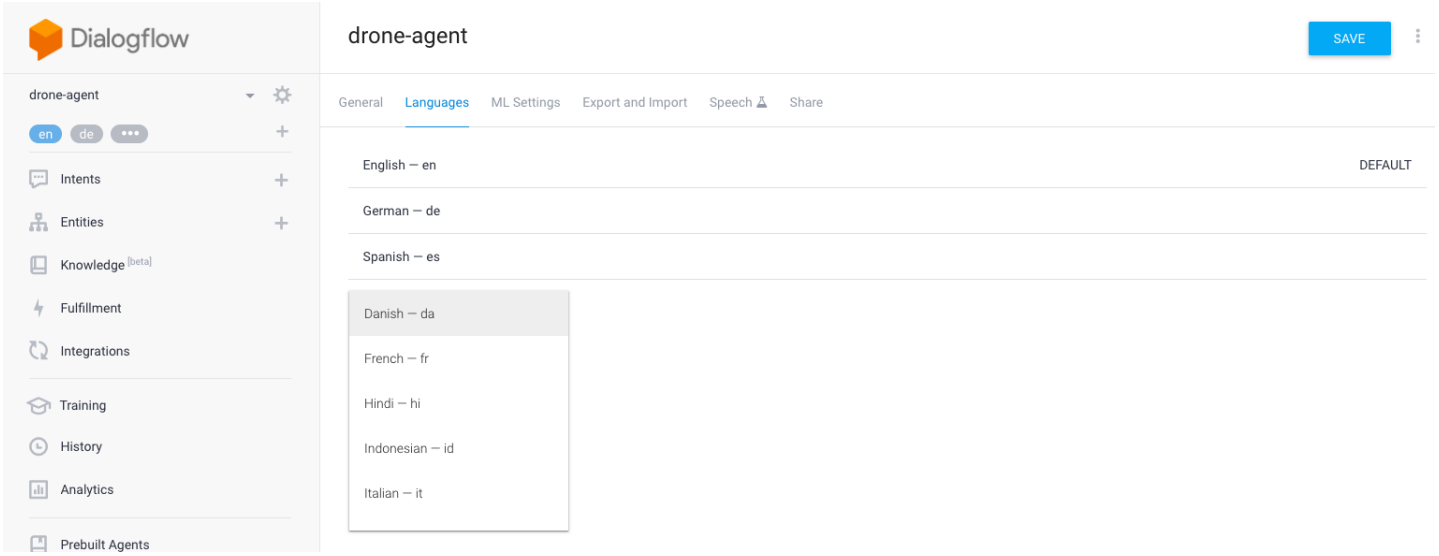
Well, in most cases, your conversation structure and parameters won't change from language to language, so you'll just add additional languages to the same agent and customize the Training Phrases and Responses as needed. All the newly-created intents and entities will be copied to all languages.

If you do need to change the conversation structure or parameters to support another language, you should set up a new agent. With the new agent, it will be easier to manage the differences.

And how do we add a language to an agent?

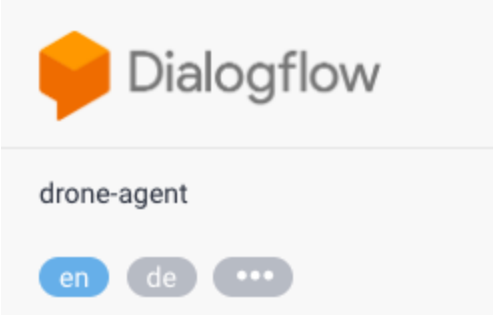
I am in the DialogFlow agent. And I clicked on settings and Languages.

To add a language, just select it here.

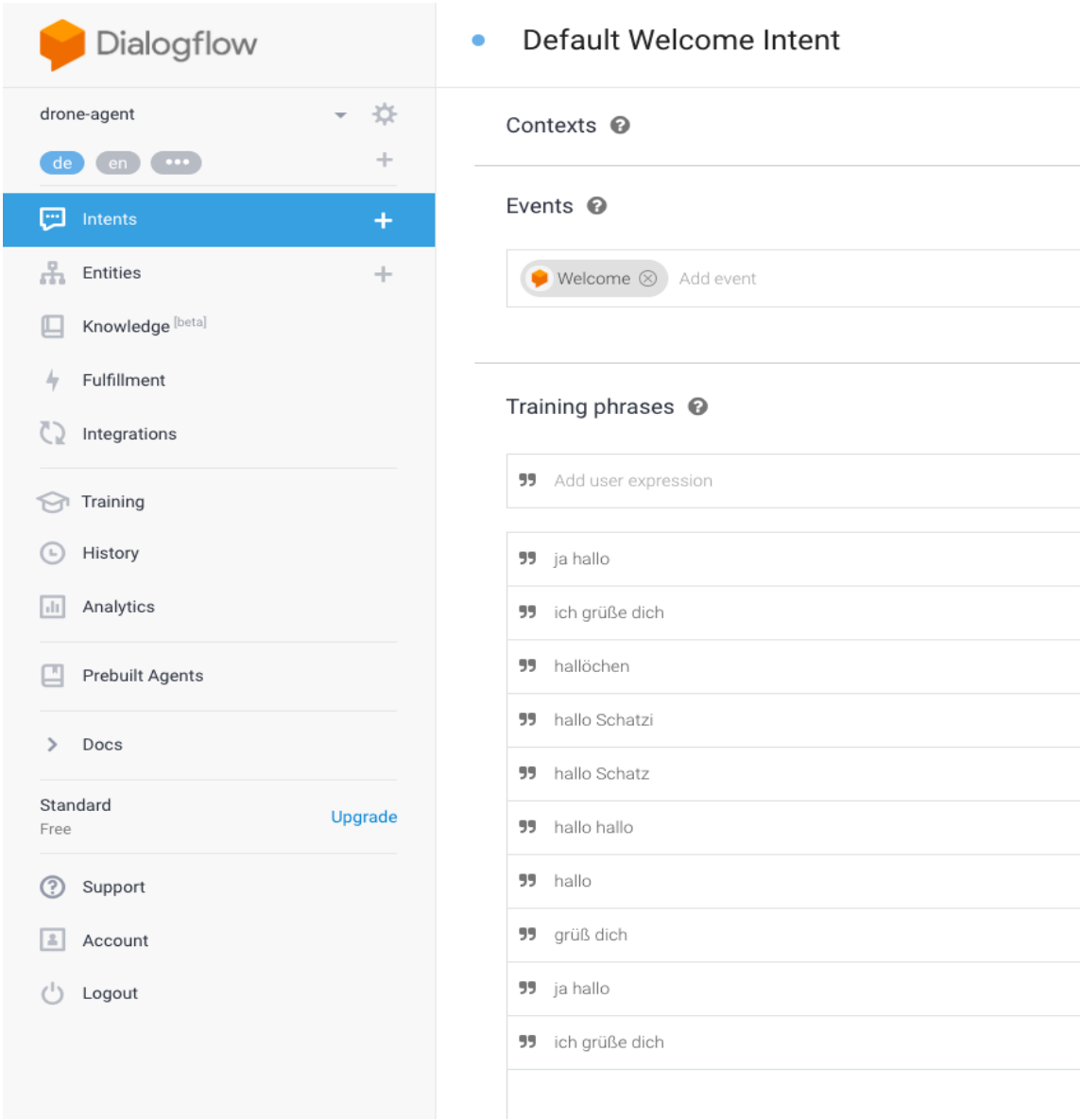


I don’t need more languages, so I’ll just stay with English.

After you add more languages to the agent you will see buttons in the navigation. You will **switch between languages** here.



Each intent will be duplicated for each agent. For example, if I click the german language, the default Welcome intent will now have german Training Phrases.



Let's start learning the basics of DialogFlow. I will introduce the basic building blocks. That is intents, parameters, entities, contexts, fulfillment and so on.

I'll show them in the next article.
And here are a few resources about multiple languages in the agent:

https://cloud.google.com/dialogflow-enterprise/docs/reference/language

https://dialogflow.com/docs/multi-language

DialogFlow basics

Hi and welcome back. So far you saw how DialogFlow is connected to Google project. And what settings are available for DialogFlow agent. And you're about to learn about intents.

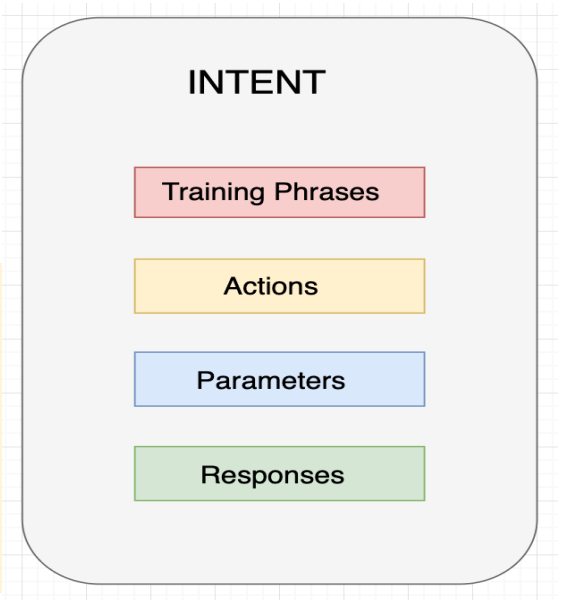
Before we take a look at anything, let me tell you what intent is all about and what it is made of.

We've said that **agent helps you process user input into structured data that you can use to return an appropriate response**. You define all of these inside intents.

DialogFlow will take users input, text or speech and it will try to map that to the right response. Intents are what it will look at. It will try to find the right intent.

In the intent we define training phrases, actions and parameters and responses.

If I had to explain in a simple way, I would say that **Training phrases** is all that a user can say, when it has a certain intent. If the user wants to buy. Then she could say. I want a dress. Or I need to buy a dress. Or Can you help me find a nice summer dress. So, training phrases is all the user say. So, all of these are similar sentences.



> DialogFlow takes training phrases and naturally expands them to many more similar phrases. It uses them to build a language model that accurately matches user input.

Based on training and **machine learning** the **model becomes robust** and varied and it can match user input better.

When we want to extract information from a user (like for example what kind of a summer dress our customer is looking for) we use **parameters** . Parameters is all the extra information we would need in our app to satisfy the user need. We would want to get the color of the dress, the size, the shape.

To get the parameters from the conversation you can use **entities** to define them. Entity is like a category of a parameter. It defines what kind of parameter you want from a user. Is it a date, a range, a name, a certain array of words.

By defining entities you tell DialogFlow that you want a particular type of a parameter, like for example when the user orders pizza, you want to get a set of toppings the user can order extra. Therefore toppings would be an entity with all available topping for pizza. And a topping parameter would be recognised from users input based on toppings entity. I will show you soon.

DialogFlow extracts matched entities as parameters from the training phrases.

You can then process these parameters in your app or fulfillment. We'll talk a lot more about this soon.

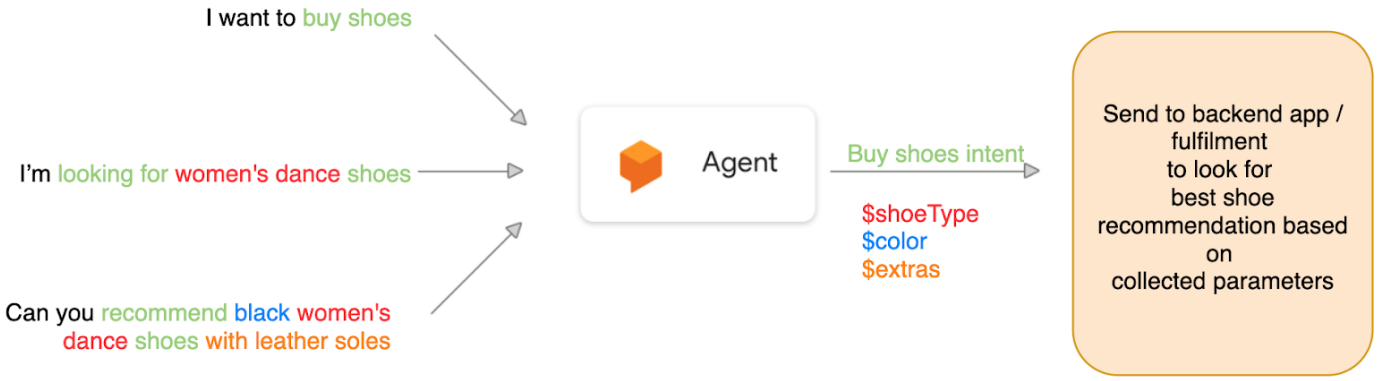
And finally in the intent we define **responses**. We define text, speech or visual responses like cards, quick replies, chips, depending on what is the media we will integrate DialogFlow with. We will use different kinds of visuals for Facebook Messenger than on Google Assistant.

Responses should be designed in a way that tells the user what to say next. And it should be clear when the conversation ended.

To send responses, you can use DialogFlow's built-in response handler or call fulfillment to process the extracted data and return a response back to DialogFlow.

Now let me show you how all this works together.

What we will do is sell shoes. Bot will help user to find the right shoes by collecting the parameters and looking for the best offer.



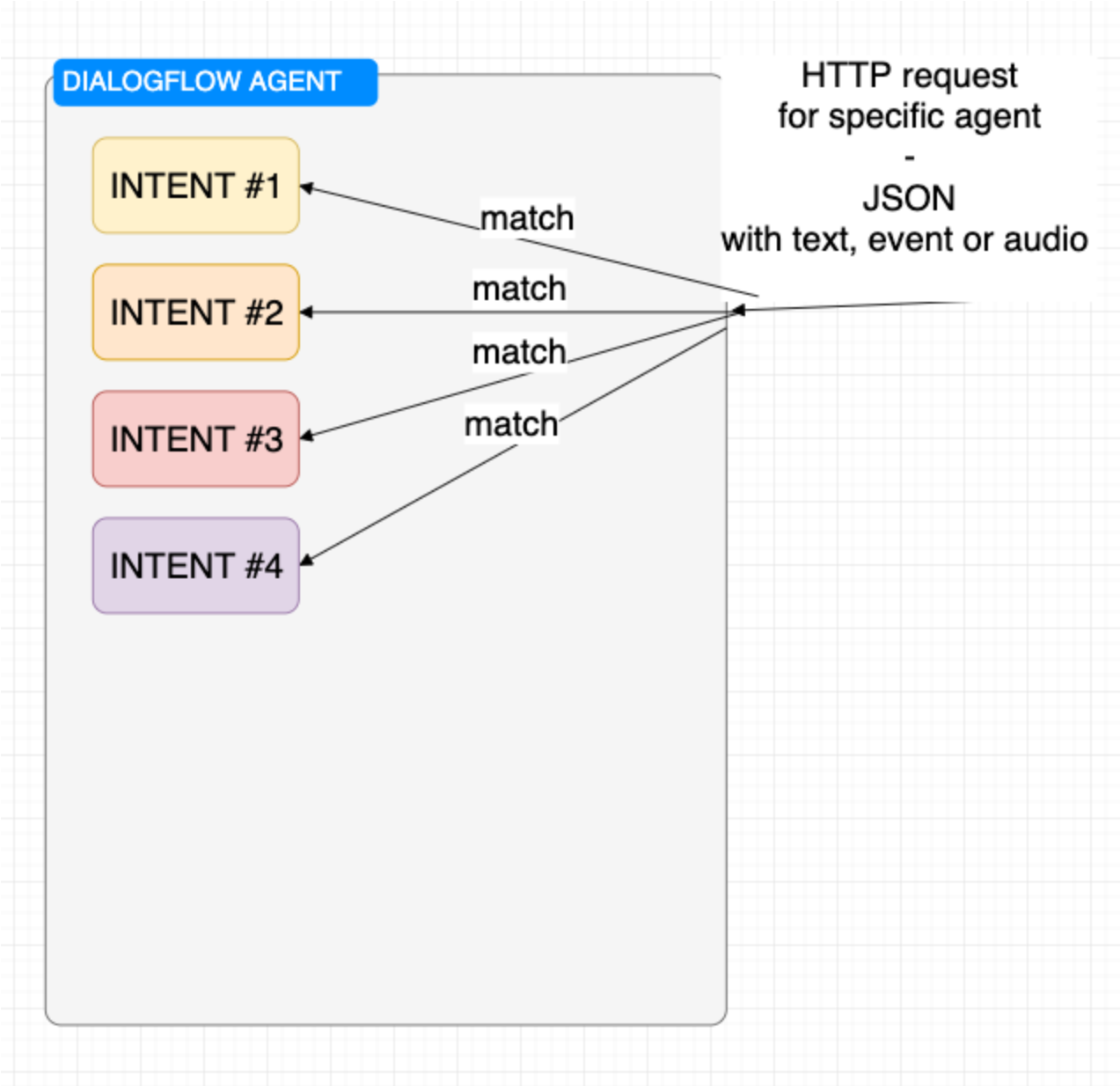
1. When users say something, your **agent matches that text/utterance to an appropriate intent**.
2. To intent to match the text should closely or exactly match the Training Phrases specified in the intent. Training phrases are examples of things users might say. DialogFlow expands training phrases to create the intent's language model.
3. DialogFlow also matches **extracts *parameters*** that you need from the sentence. This parameter can be a city, product, id, name, date, or anything else you would need to use in the backend app to give better responses to the user.
4. You define the **type of parameter with *entities*** . DialogFlow offers a lot of predefined entities, but you can also create your own custom entities.
5. You would tell DialogFlow what to extract in the training phrases. You do this by **linking specific parts of the training phrases to parameters** (described with entities). For example in a string "Can you recommend black women's dance shoes with leather soles" you would link black to parameter color, woman's dance to shoeType, and leather soles to extras.
6. You can even make some of the parameters required. For example, if you need the user to specify shoeType, you would make this parameter required. If the user would not say shoe type you could define a prompt that would prompt users for more information to continue the conversation.
7. DialogFlow includes an easy-to-use response handler to **return simple static responses**.
8. If you want to have **dynamic responses**, for example containing information from a database or external API service, you can use logic called ***fulfillment*** or you can link a backend app. I talk about this later on.
9. What is always done is this: **DialogFlow agent matches text with an intent, extracts parameters, and returns a response**.
10. One more thing is important. We need to glue intents somehow together. In a conversation sentences usually rely on something said before. We don't just say unrelated strings. And how do we form dialogues? How do we know what was previously discussed? With contexts. **Contexts are what glues intents together. It is like a topic of the discussion**. With contexts, we can create dialogues, make branches, just by adding the right input and output contexts.

When you create your first agent, you already have two intents defined by default.

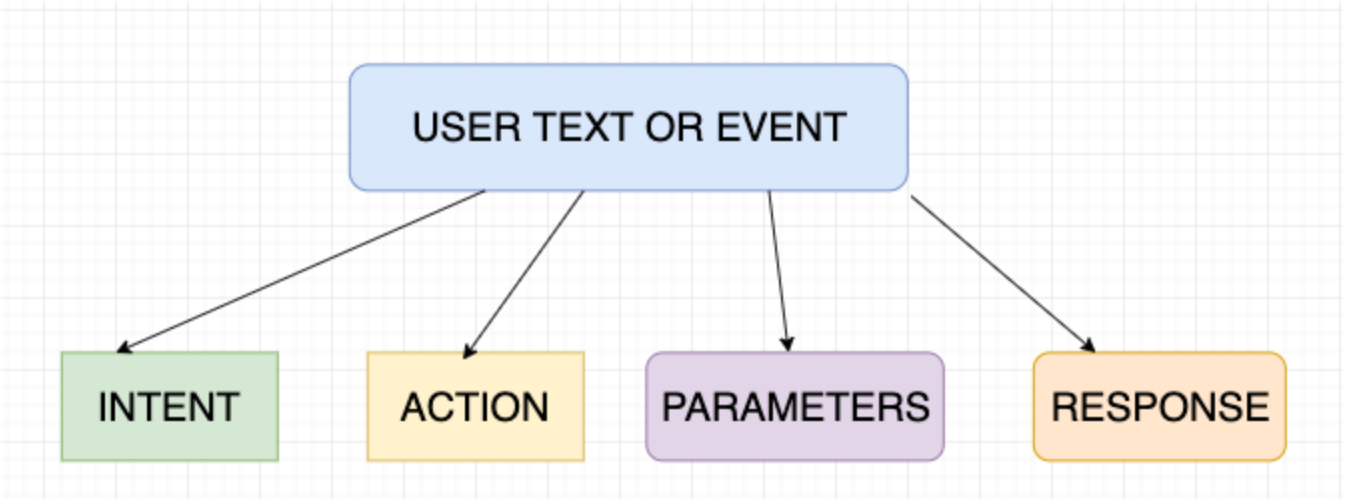
These are Welcome and Fallback intent.

Before I tell you more about them, let me show you a diagram. Here is DialogFlow agent. In the agent, we can define more intents and we have two already predefined for us.

DialogFlow first receives an HTTP request. In the request, it writes for what agent the request is for. This request is basically a JSON structure with text, event or audio.



The agent then takes this input and matches it with all intents in the agent using machine learning and rules to find the best fit. And when it finds the best fit it knows how to convert the input to intent, action, parameters, and response.



In the intent, we define intent's name, an action that needs to be taken, parameters to be read from the string and a response.

Intents Basics – let's define first intents

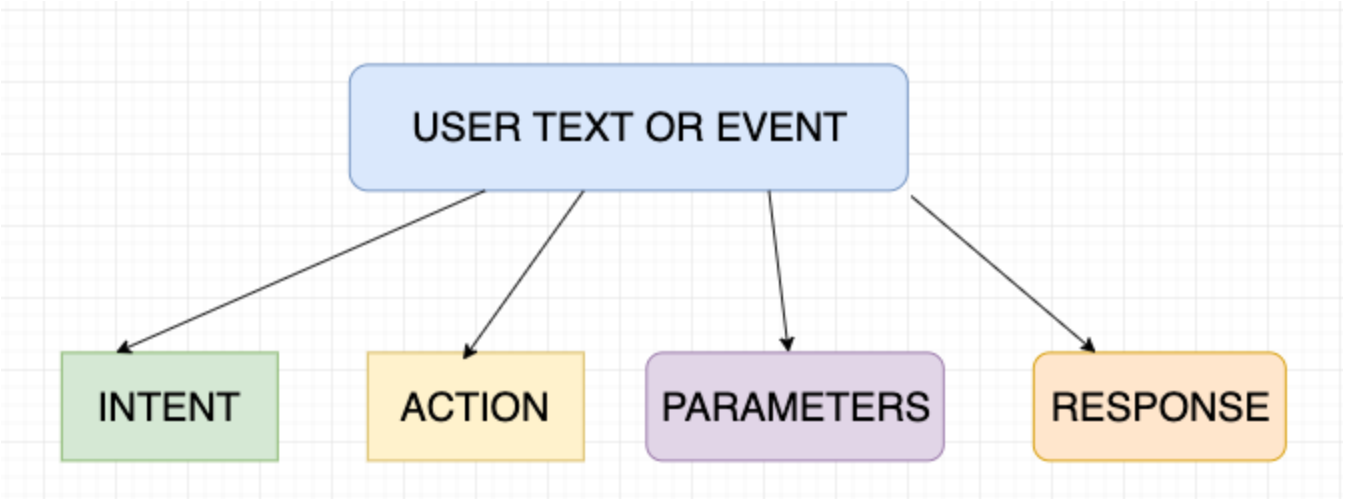
Now that you know how DialogFlow works theoretically, let's see all this in practice. In this article, you'll create the first intent.

Ok, let's create the first intent. You can click the Create Intent button

or the plus sign here, near Intents menu.

Here we go.

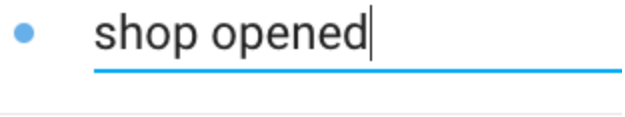
Before we proceed, let's remember an intent is a building block of DialogFlow. The most important building block that maps what the user says, that is the input string or event into intent, action, parameters, and response.



And all of this we need to define inside the intent.

First, we'll add a name. This intent will be just a demo. We'll be asking when is the shop opened. Let's pretend this is a bot for a shop. A shop for Drones.

And we'll be asking for open hours. Therefore let's name the intent shop opened. Like this.



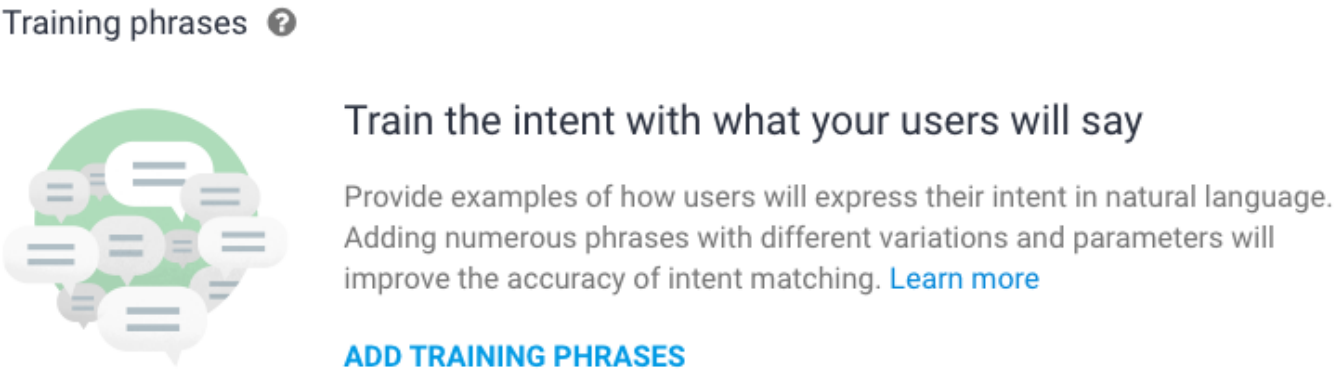
The first thing we see are contexts. Contexts are what glues intents together. It is like a topic of the discussion. **We need to know in what context the user has said something.** But I'll tell you all about this very soon. It does not apply to this intent.

The next thing is the **events**.

Intents can be triggered by text or audio. But not only that. Intent can also be triggered by an event, an event like **click on the button, click on the list**. We can add **custom events** that trigger the intent. And yes, sure, we'll do exactly that in this course. I'll show you everything. But in this intent, we don't need to add it.

Every intent has the **training phrases**. This is what the user says to the agent. Every possible sentence that could be said to trigger the specific intent. Well, we don't need one here.

And in the Training Phrase, we'll add what the user says to trigger the intent.

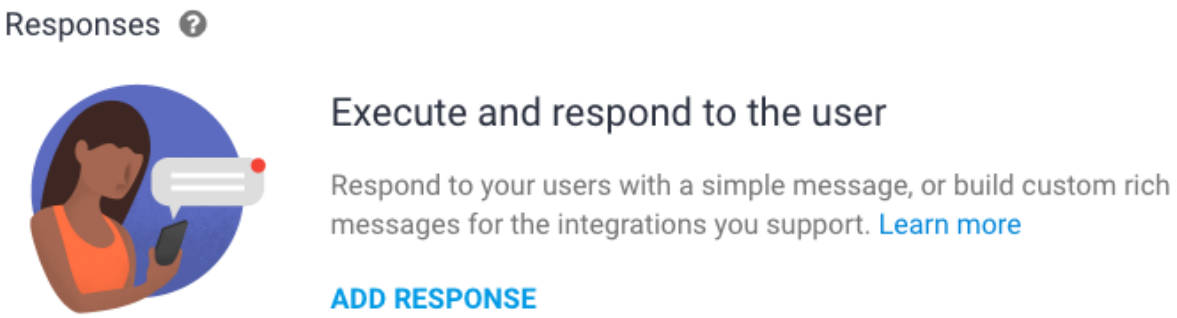


Click Add training phrases

Like "*When is the shop opened?*" Or the next one. "*When does the shop open?*" or "What are the working hours?". The more you add, the better DialogFlow can recognize the intent. Make sure the training phrases are related.

The more Training Phrases that say the same thing we add, the more DialogFlow will be able to recognize the intent. It's recommended to add **at least 10 variations**. Ok?

And now I'll add a response. That is the response that will be sent back from the bot if DialogFlow recognizes this intent.

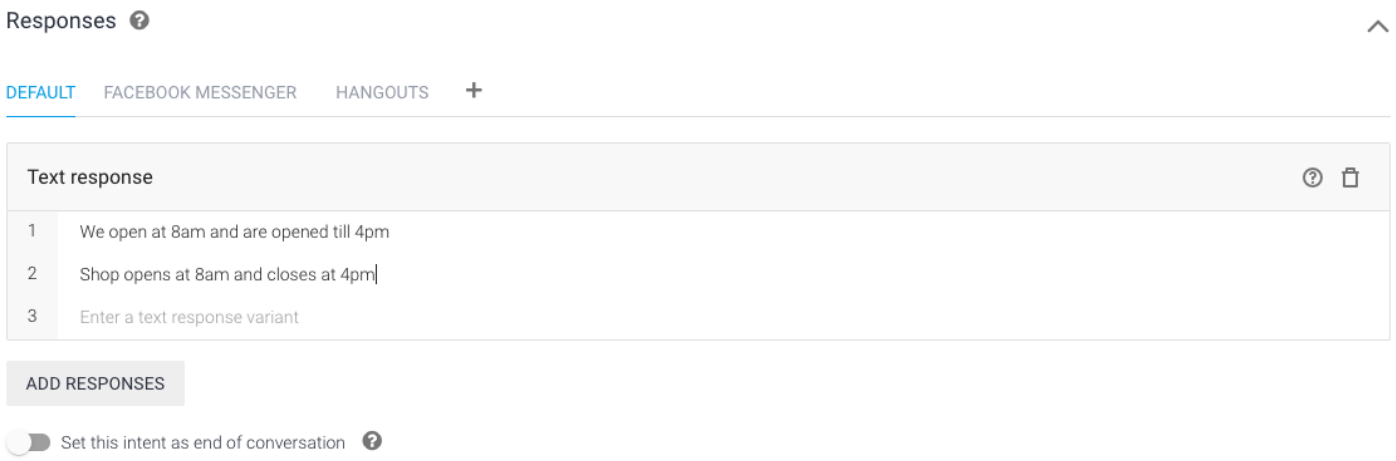


Click add response link.

I'll write "*We open at 8 am and are open till 4 pm*". And let's add another. If we add more than DialogFlow will choose one of these answers and return it. I'll add: "*Shop opens at 8 am and closes at 4 pm*". And I'll add it under the Default tab.

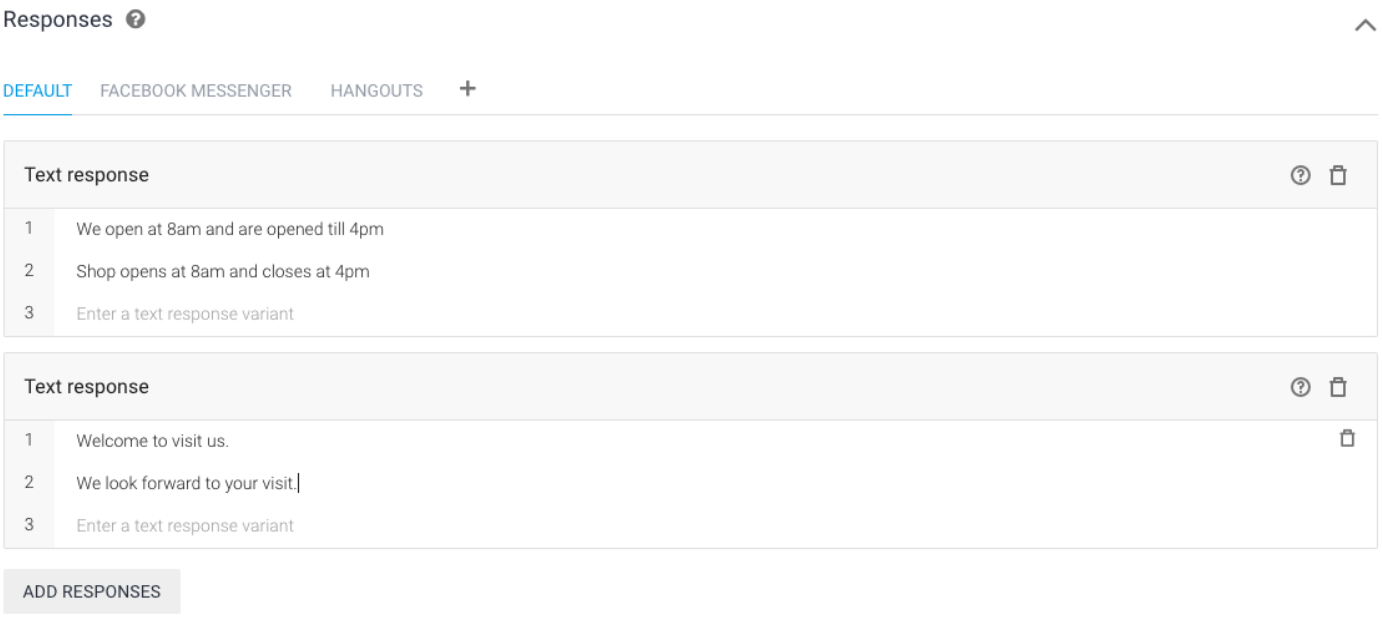
Responses are grouped together based on the platform the bot response is for. For example, all the answers for Facebook Messenger bot should be under the Facebook Messenger tab. And all the responses for Hangouts or Google Assistant should be under the appropriate tab.

Currently, DialogFlow supports Facebook Messenger, Hangouts, Google Assistant, Slack, Telegram, Kik, Viber, and Telephony. Each platform has different kind of responses. One has chips other quick replies, other cards, and lists. It depends on the platform. For now, we'll add simple text responses listed under Default tab.

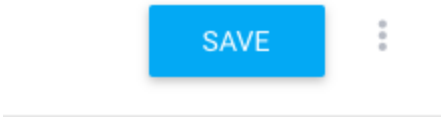


To remember: **If we add multiple answers to one response DialogFlow will choose randomly between them.**

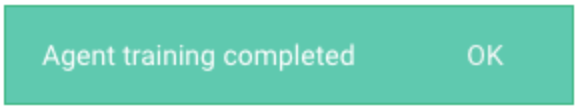
You can also **add more responses**. And **each response can have multiple variations to choose from**. This is how we make the conversation more natural. We don't always reply the same one to one question. And we many time write sending multiple messages.



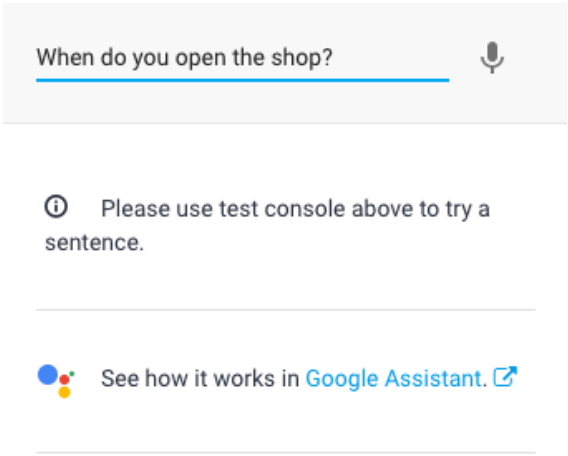
Ok. Before we proceed let's save the intent. The button here.



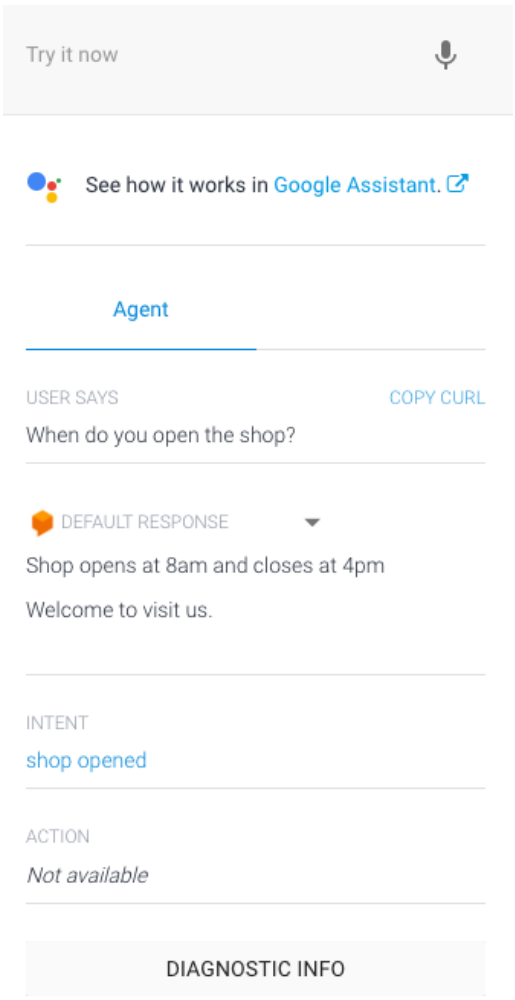
As you can see something is happening in the background. The agent is being trained. That is machine learning behind the scenes. Ok.



Here in the right. Here you can test an agent. Her in the try it now we can write "*When do you open the shop?*"



And now wait a minute. And woohooo. We have an answer. It opens at 8 am. Let's try another. "*When do you open the shop?*" Now, wait a while. And did you see that? We did not write "When is the shop opened?" to Training Phrases but DialogFlow was still able to recognize the intent.



So, great! And remember, the more you help DialogFlow, the more accurate responses will be. And you help by adding more Training Phrases. Ok?

Now, in this article, you found out how intent can transform an input string to the response. Let's look at the Welcome and Fallback intent in the next article.

Welcome and Fallback intent

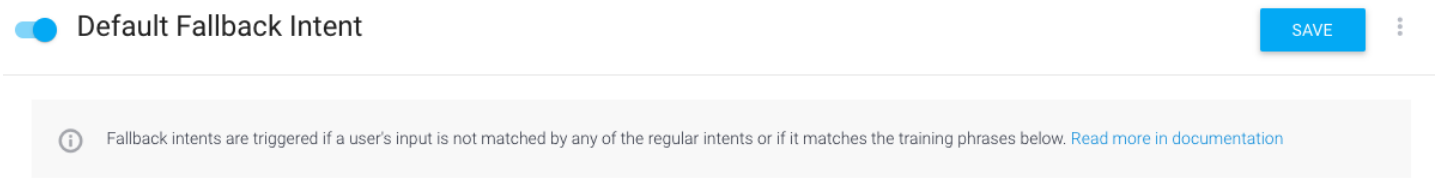
Ok! Let's go back to DialogFlow.

Let's take a look at the intent we have already created for us.

First one is the **Fallback Intent**.

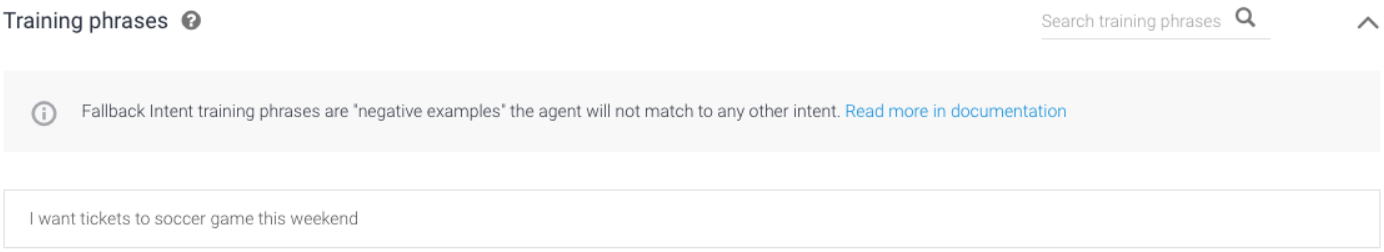
This is the intent that will be triggered if no other intent will be matched.

So, an agent will take the string and match it with all intents and if no one matches, then it will trigger this intent. Let's click on it.

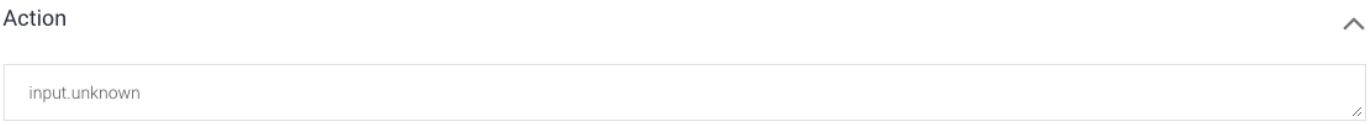


Fallback intent will be triggered anytime no other intent is matched.

And with fallback intent is a little bit different. Everything that we would write in this intent as a Training Phrase would not be then matched in any other intent. For example, if we were selling tickets for concerts. And the user wrote to the bot: I want tickets to a soccer game this weekend. We would write this sentence here. We would not want this phrase to trigger any other intent since we're not selling tickets to soccer games, but just music concerts. Ok? Does that make sense?



What we have here is the action. And it is already defined as input.unknown.



This action could be handled by a backend app. So, what do I mean by handling the action? By handling action I mean we run some extra code that does something that is required by that action. Let me give you a few examples.

Let's say we want to read some extra information from a database and generate dynamic responses. Or we want to trigger some API service call, maybe place an order user has given or set up an appointment.

Well, what we could do is maybe send an email to customer support, like here in the case of Fallback intent. We would send a notification, some user has problems interacting with the bot.

We could also implement the rules and winning conditions for a game.

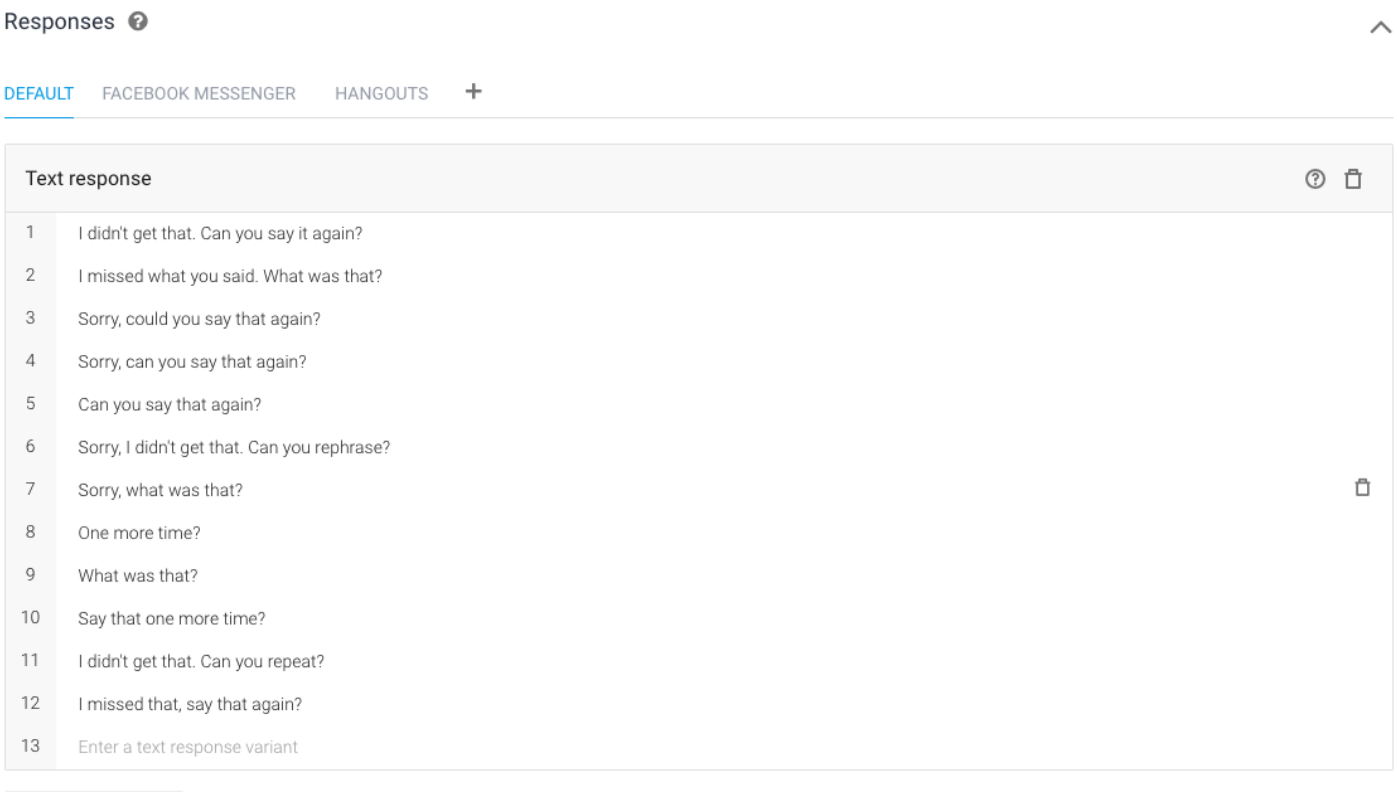
In the fallback intent, for example, we could implement switching to live agent after three attempts.

So, actions can be handled in the backend app, in the code.

I will show you in the next section how we can run this extra code. There are two ways of implementing it.

Ok, back to DialogFlow.

Way down here you have defined responses. These are all the possible answers bot can say when it does not understand the input string.



More than one response here is defined. Only one of these will be returned. This is because the conversation needs to sound more natural. If the bot would always reply to the user something like 'I didn't get that. Can you say it again?' that would start to be annoying.

Remember Star Track and Unable to Comply.

https://getyarn.io/yarn-clip/d71806e2-1c18-48cd-99ee-d773c63c3b5c

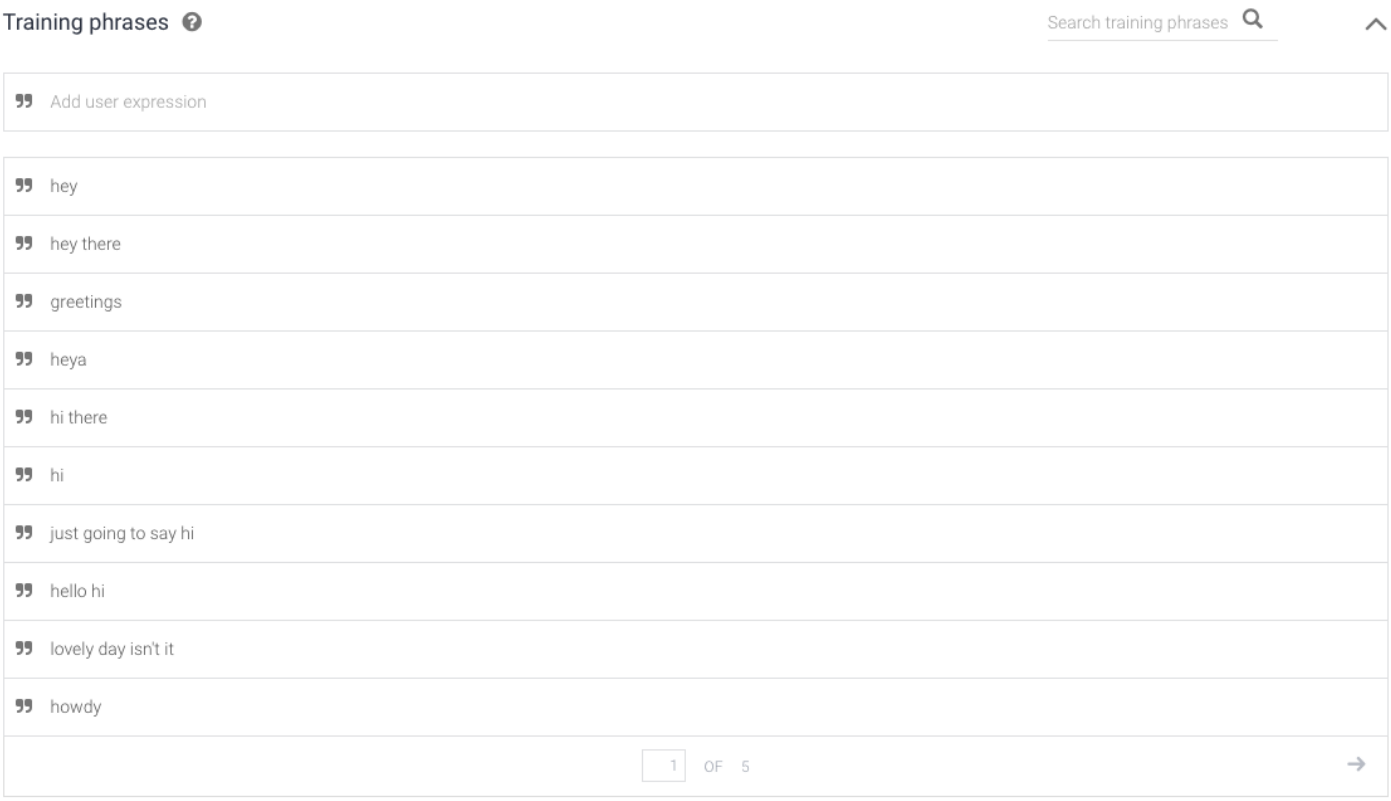
Well, they might start using DialogFlow now for better user experience.

What we could also do is we could add more responses. Like this. Like in real conversations. We send more than one message at once. The bot needs to emulate this natural behavior. And we can add more messages by adding them here.

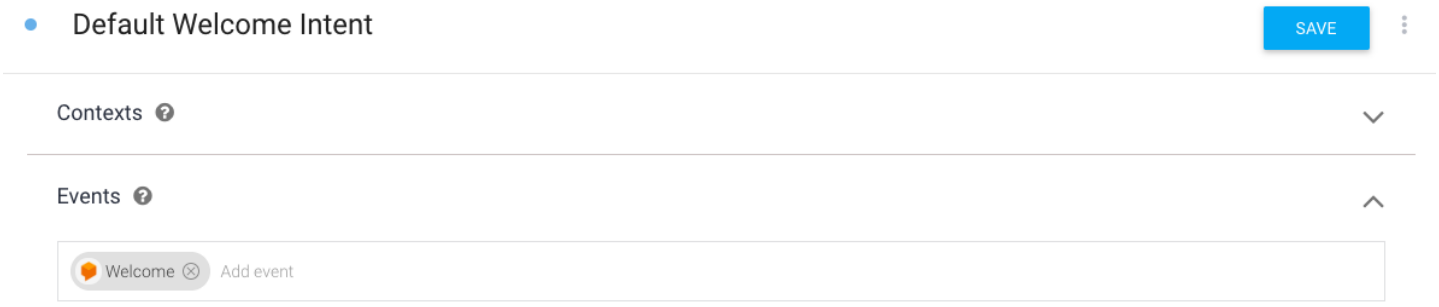
But still, **if we design an agent in a way the Fallback intent would be triggered to many time, that would still annoy bot users**. And that is the no. 1 reasons why users are not satisfied with bots. **Poor user experience design**. But that is another story we won't cover in here. Let's go back to all intents.

Let's take a look at the **Welcome intent.**

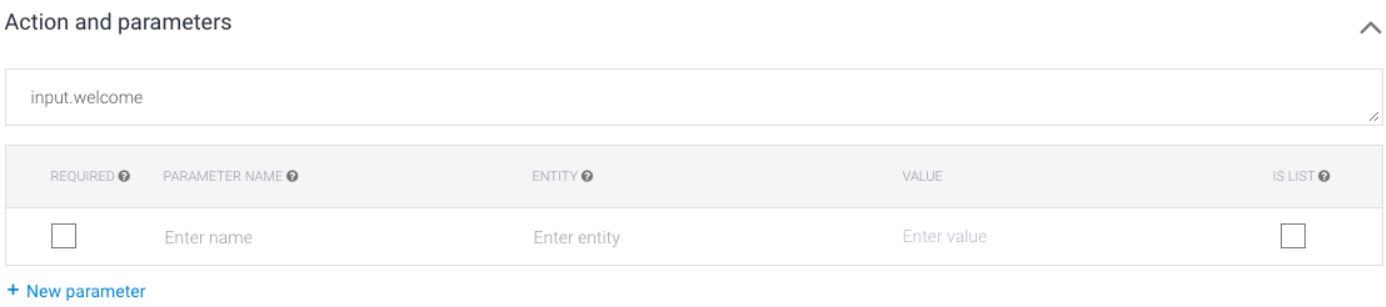
Welcome intent can be triggered with an event and with a text. It can be triggered an event called Welcome. I will show you how to trigger an intent with an event in the future. The difference between event and text is in the JSON you send to detectIntent DialogFlow API. Here are the training phrases that can trigger the welcome intent. These are defined by default and we can add our own.



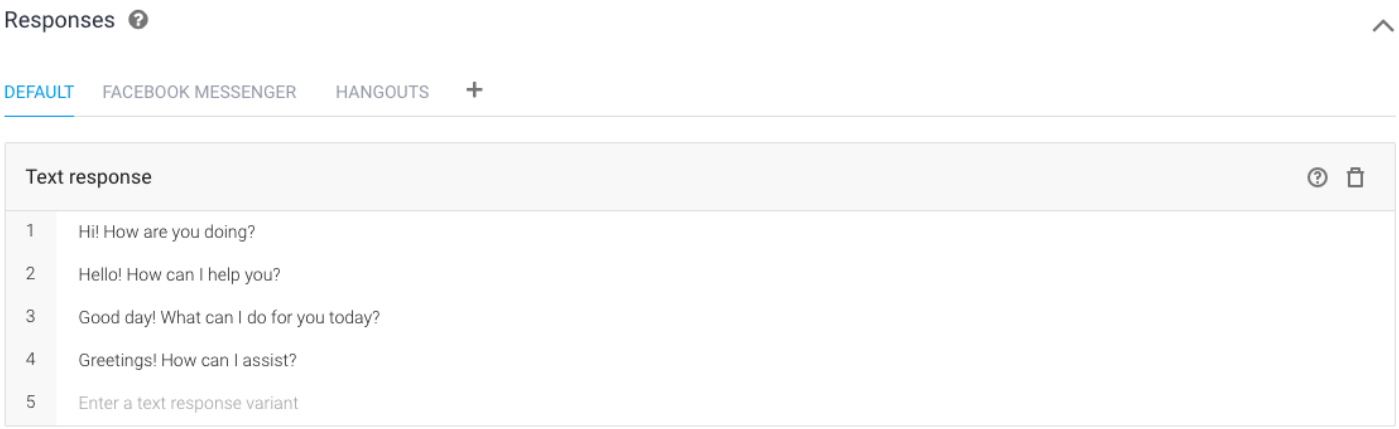
Welcome intent can be triggered with Welcome event.



The other difference is that this intent also has an option to define parameters to be collected from user string. Every intent you will define from now on will have this option. Well, all except fallback intents.



The same as the Fallback intent it has responses down here. What you can see is that we can define different responses for different platforms. For example, Google Assistant and Messenger support different kinds of responses. Messenger, for example, has quick replies and Google Assistant has chips. We can define different responses for each. And they would be grouped under the appropriate tag to make it more clear. So these are the responses that are returned to the bot that will show them to the user.



We've now looked at the Fallback and Welcome intent.

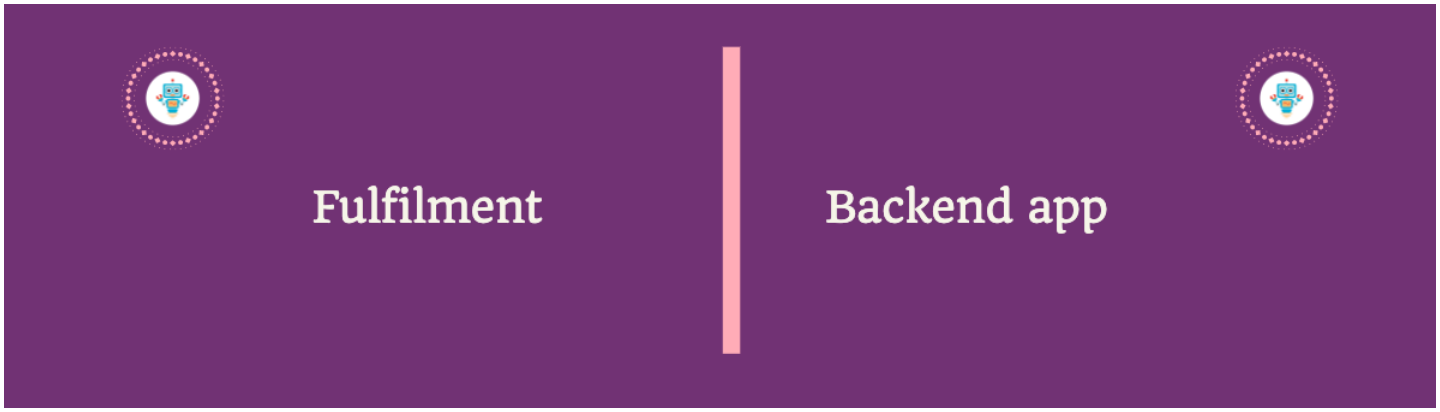
I have mentioned that for every intent we can run some extra code. With this, we can build more dynamic responses, for example with information from a database. Or we could run some extra calculation, call a service, build in some logic. Yes, way richer experience than just static responses to questions. Let's see how we can run this logic.

Backend app and fulfillment integration / two ways of integration

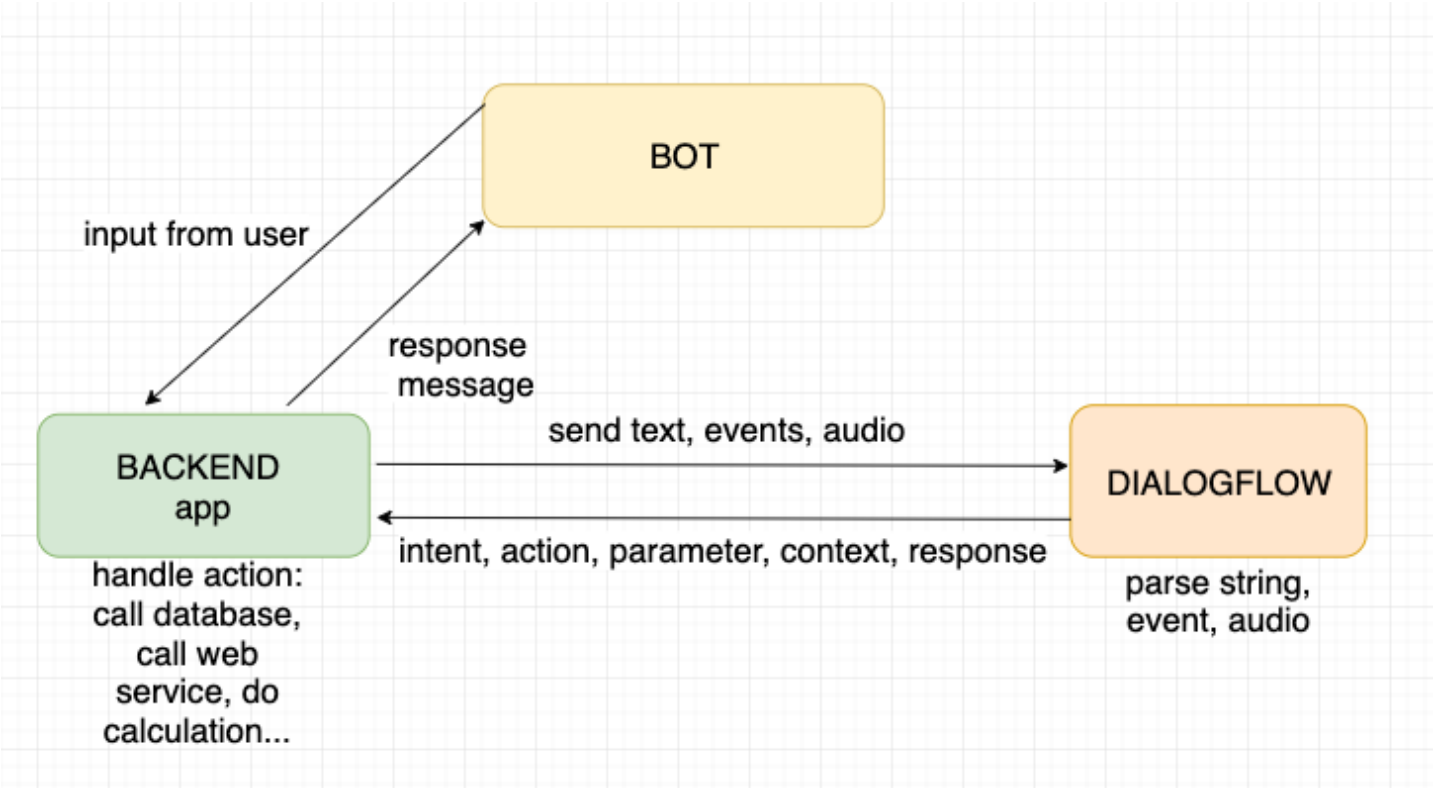
Hi and welcome back. In the previous article, we looked at Fallback intent and Welcome intent.

We have said that Fallback intent is triggered when no other intent is matched. So, if no intents match then the user gets a response from fallback intent. And also, we took a look at the Welcome intent, a predefined intent that is triggered by an event and greets the users. In this article, we'll create a custom developer-defined intent.

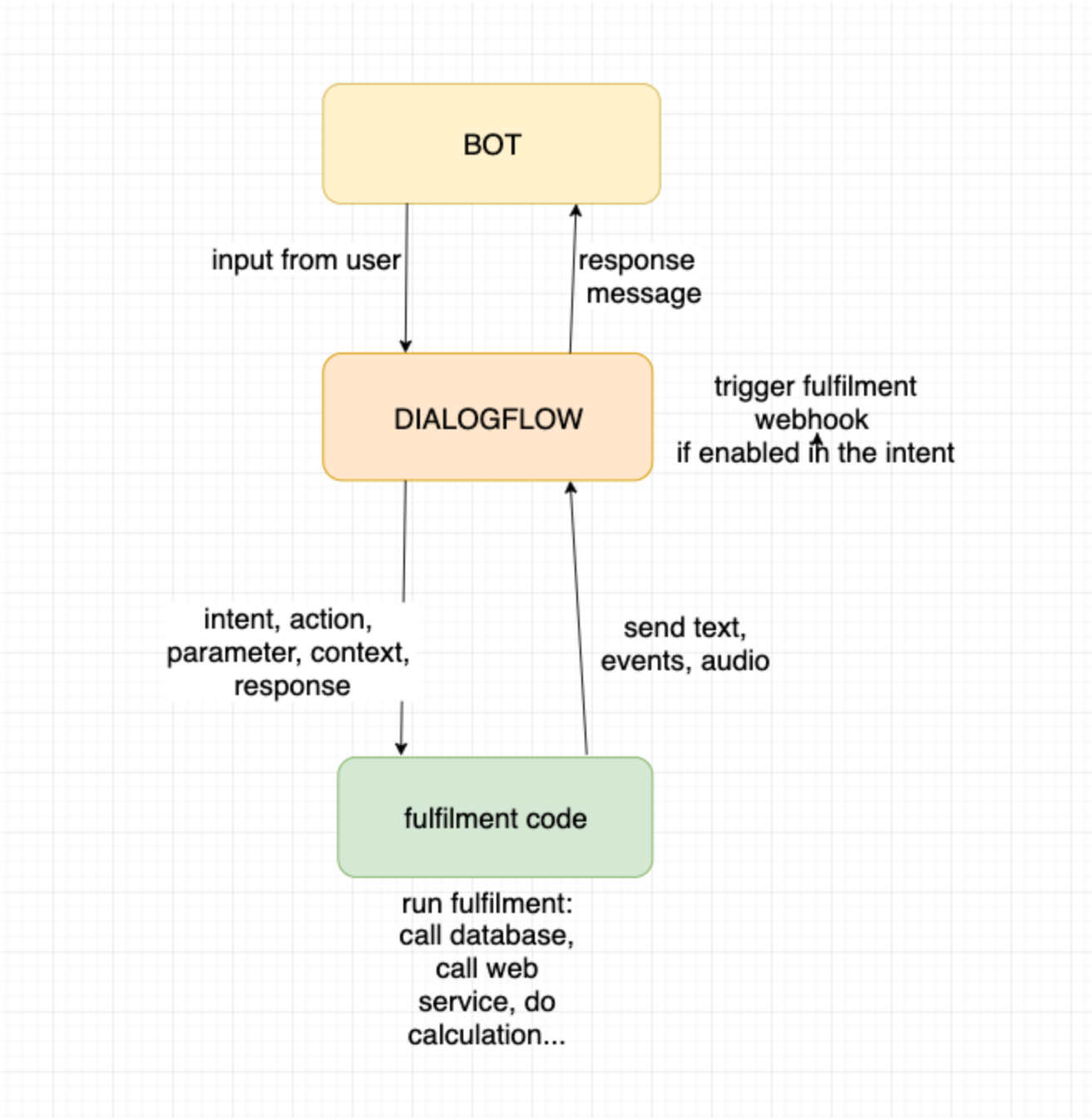
We have **two ways to implement backend code** when the intent is triggered. Let me tell you about them.



In the first case, we would connect our app, that can be a bot or an assistant with a backend app. And then this backend app would forward text requests to DialogFlow. With forwarding I mean that backend app would call DialogFlow's API service. The endpoint for text and event query is called detectIntent. Upon receiving the response from DialogFlow it would handle action if needed. It could perform extra queries, do a calculation, call another webservice. And after that return the response to the user. So, this is the first case. The backend app communicating with DialogFlow.



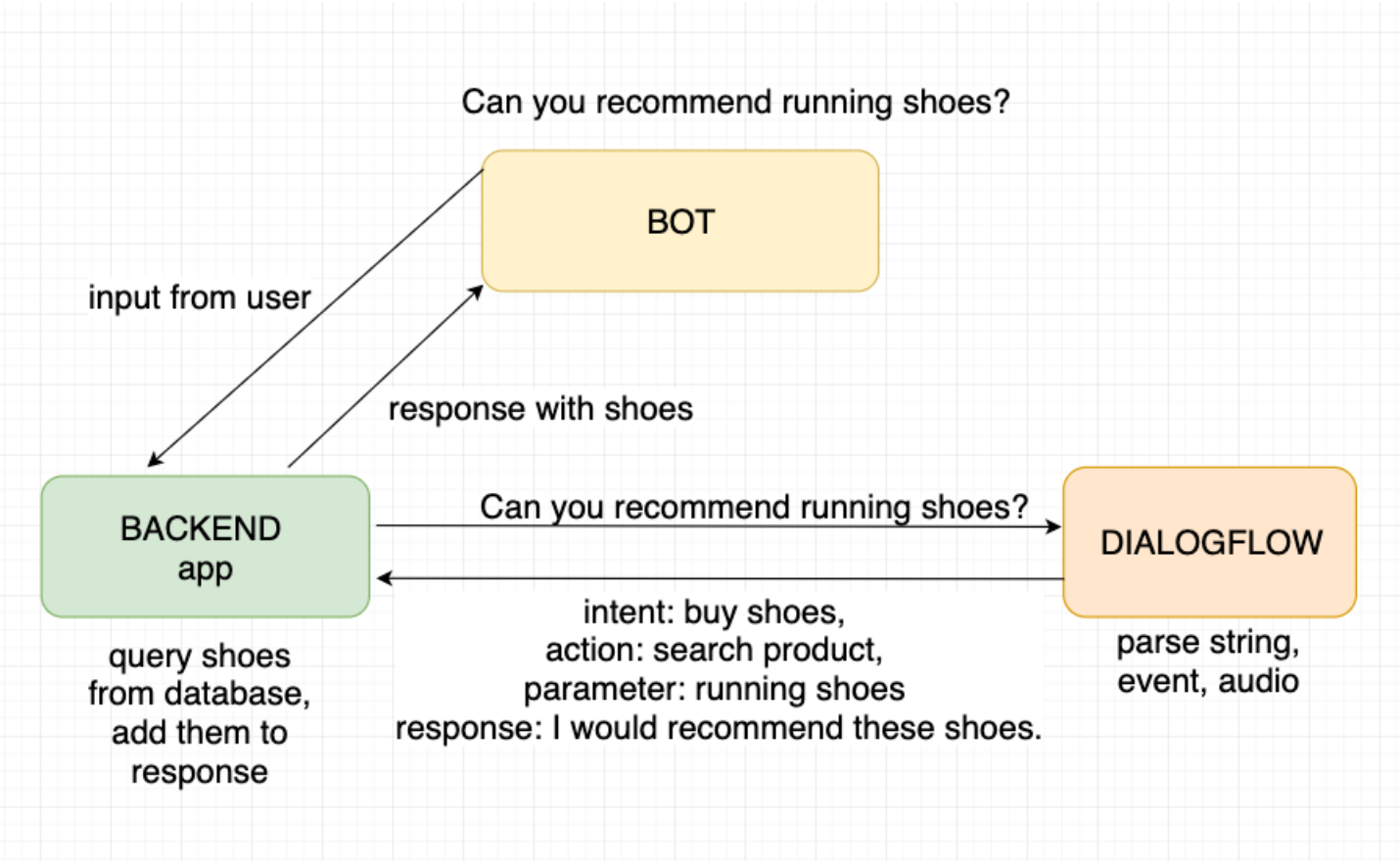
In the second case, we would trigger a fulfillment code if the fulfillment is enabled in the intent. Fulfillment is a webhook that runs some code, usually on a cloud function.



Let's talk about the first case. Where we connect our bot or an assistant with a backend app. And then backend app forwards text requests to DialogFlow.

Let me give you an example.

User asks a bot: Can you recommend running shoes. That input is sent via the HTTP to Backend app. Backend app forwards the input to DialogFlow to get it parsed. This is here, send queries, events or audio to DialogFlow. DialogFlow then does its thing and returns the intent, action, parameters and context and response. For example intent: buy shoes, action: search product, parameter: running shoes and response: I would recommend these shoes. As you can see it did not recommend any shoes. Well because DialogFlow is not meant to be a database with all your shop information. It just returns action search products and parameter running shoes. It just tells your app what the user wants. And then your backend app can query shoes from a database, add them to the response and send the response message to the user.



So, again. Flow goes like this. Input from the bot is sent to backend app, that sends it to DialogFlow for processing. After our backend receives response with the action and parameters and all the information needed, it runs extra code, extra queries, extra calculation if needed, transforms the response if needed and sends it back to the bot.

This is the first way, so the backend app in front of the DialogFlow. If we use the backend app then what we do is trigger DialogFlow's API to get the response from it. The API endpoint for sending texts and events to DialogFlow is called detectIntent. Here is a link to the documentation:

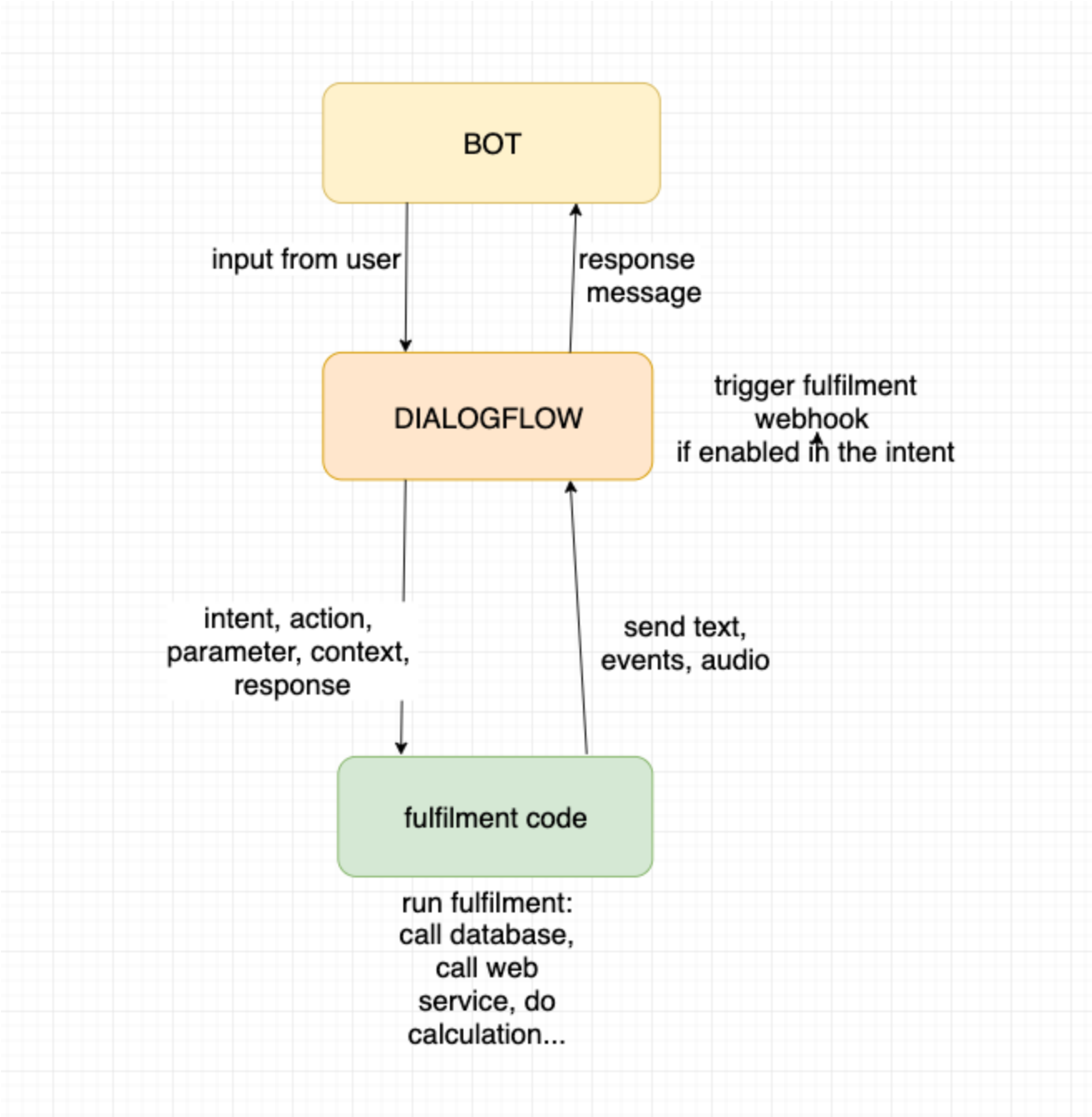
https://cloud.google.com/dialogflow-enterprise/docs/reference/rest/v2/projects.agent.sessions/detectIntent

The second one is called fulfillment. In this case, our bot or virtual assistant would be connected directly to DialogFlow. And DialogFlow would trigger an extra webhook if the matched intent would have fulfillment enabled.

Fulfillment is code that's deployed as a webhook that lets your Dialogflow agent call business logic on an intent-by-intent basis.

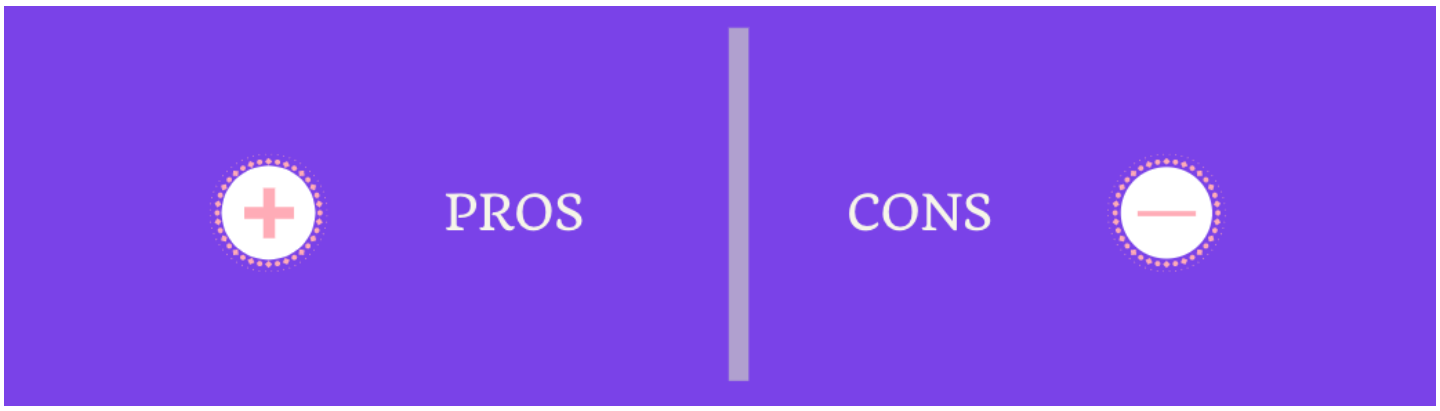
You can enable fulfillment for any of your agent's intents. To use fulfillment, you need to set up a webhook. A webhook is a web server endpoint that you create and host. The recommended way is to use Google functions, but you can use any other.

DialogFlow offers fulfillment webhooks for intents. What does that mean? Well, it means that for every intent that you add to DialogFlow you can enable a webhook that will send a request to this hosted app and get extra information if needed. So, in that app is where the queries and calculations would be done. DialogFlow would get the response back and not the Bot. Than DialogFlow would send it to the bot. So, in this case, a backend app is after the DialogFlow. This is the difference.



So, in this second case, DialogFlow would find the intent, then it would see fulfillment is enabled and it would send a fulfillment webhook request to your app. Fulfillment code would handle intent and sent back a response to DialogFlow. And DialogFlow would return then that response to the bot or the Assistant.

Let me tell you about the pros of each approach.



What are the pros of the first approach:

- we can choose if we want to send queries to DialogFlow or not. If the request is text, then we forward it to DialogFlow. But if the user clicked a button or sent an image, then we don't want to send this to DialogFlow
- we can send messages to different agents. For example, we have a different conversation path in two different languages, so we want to send a text to the right agent
- we don't want our request to timeout

I would use this approach when communicating with several agents. And where we use all sorts of rich messages, from images, buttons and so on. Not just text and voice. I would also use it when I rely on other API services where they might be a delay and the fulfillment could timeout.

And the pros of the second approach:

- the code will run on a serverless function and it will be triggered only when fulfillment is called
- we only call the code in the intents that have fulfillment enabled, not for all intents

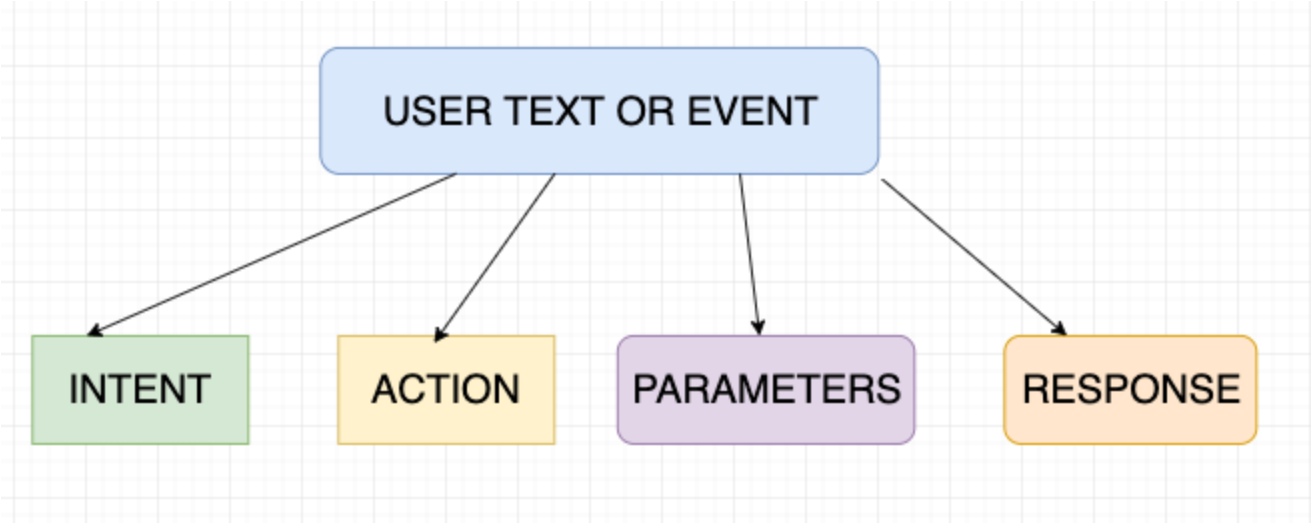
I would use this approach when I would be doing a simple application, like a game or a simple bot that need to take some information from the user and save it to a database or post it to some web service. Or read some information from a database or a web service. Basically for simple applications, where the timeout is unlikely to happen. And we only communicate with one agent. And use simple text or voice messages.

Ok. Let's see how to extract information from a conversation. How do we get parameters from a conversation?

Entities

Hi and welcome back. We have already created the first intent. I created Shop open and we saw how the input string is transformed to response or even two responses. In this article, you'll get to know entities. Entities help you get parameters out of the input string.

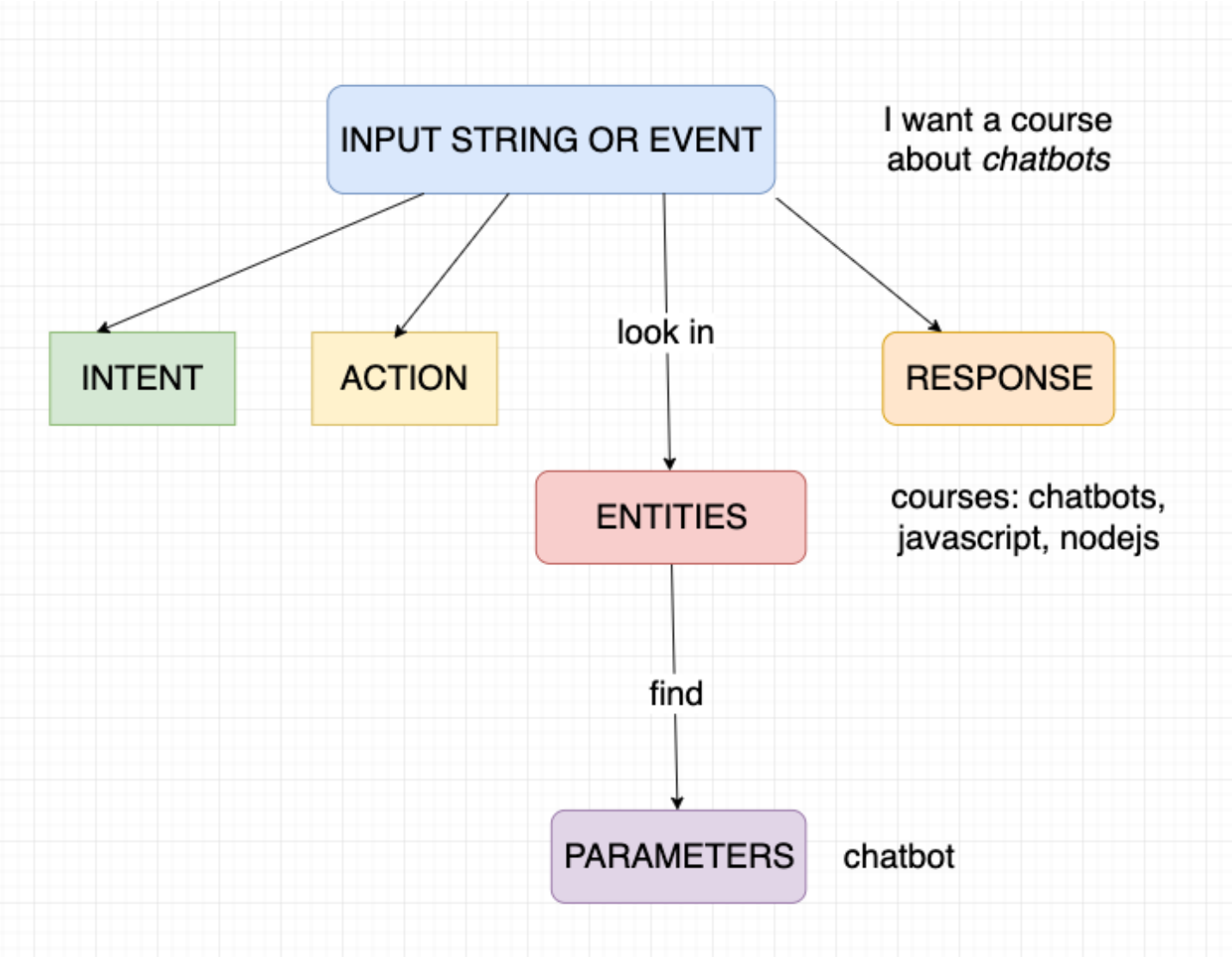
DialogFlow transforms input string, event or audio to intent and action, parameters and response.



But how is it able to read parameters from the string?

For extracting parameter values from natural language inputs DialogFlow uses Entities. Entities are a powerful tool.

> Any important data you want to get from a user's request will have a corresponding entity.



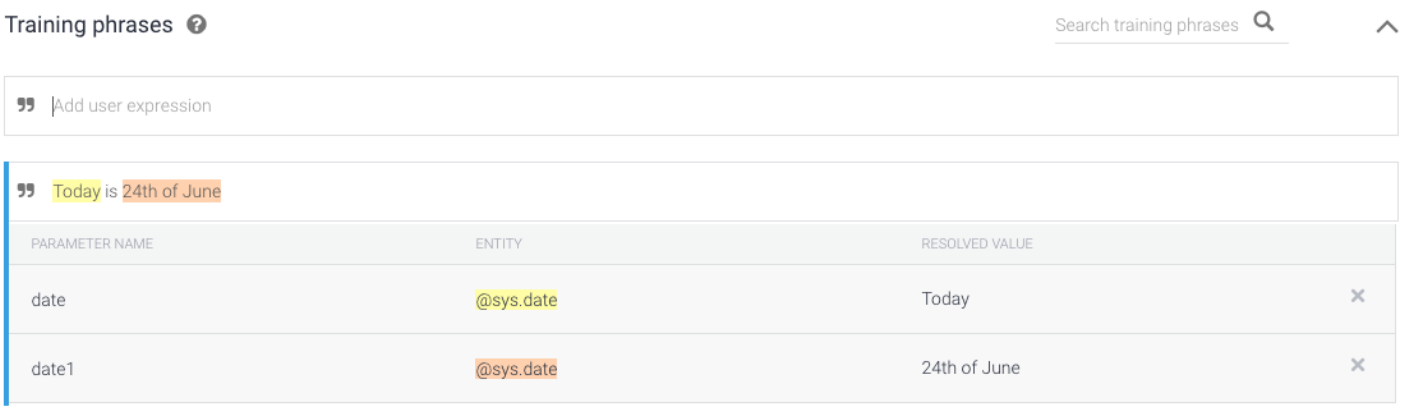
Like for example in this picture. We define all possible courses as an entity. Entity courses would have possible values: chatbots, javascript, and node.js. So these could be all of my courses. And my student would come to my chatbot and say to chatbot: I want a course about chatbots. And since I have added chatbots as one of the possible entity values, DialogFlow would recognize it as a parameter. Yes? So what are entities?

Entities are like predefined words, objects you'd like to get from the conversation.

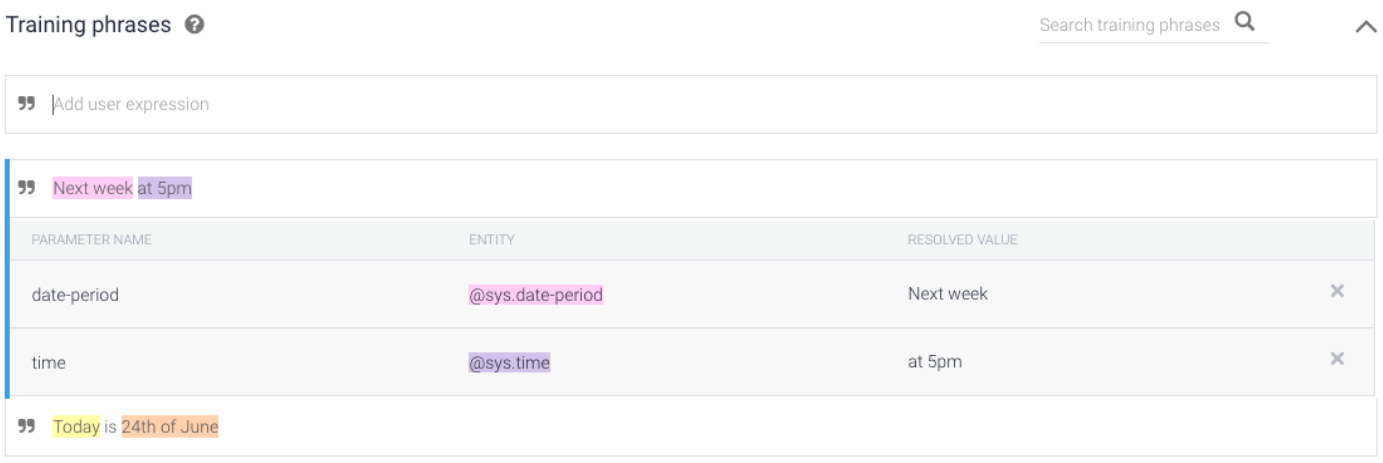
In other words, a developer does not need to create entities for every possible concept mentioned in the agent – only for those needed for actions, for the backend app.

Some more common entities, like for extracting dates, names, numbers, addresses are already predefined for us. These are called system entities: https://cloud.google.com/dialogflow-enterprise/docs/reference/system-entities

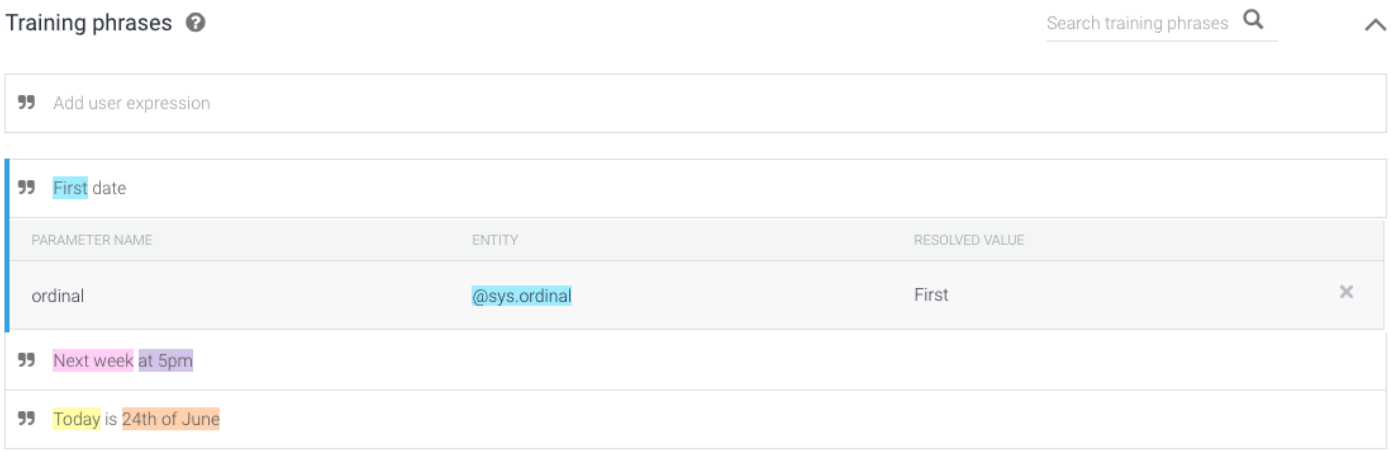
Let me show you something. I'll show you in an example. Just watch, you don't have to do what I do. And I'll write Today is the 24th of June. And woohooo. See. DialogFlow knows this are dates. Today and the 24th of June.



One more example. Next week at 5 pm. Again. Next week is the date period. And it is time entity. Cool ha!

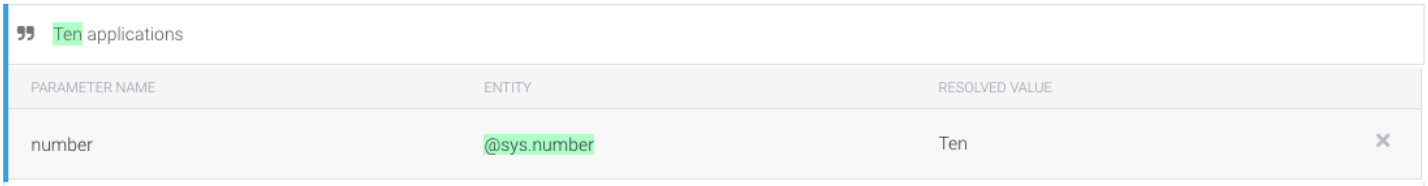


Ok. Now I'll write the First date. And first is ordinal. True.



10 attendees. And yes, it's a number.

Ten applications. Also, a number. Ok.



Not all languages have the same entities. English has date and time entities we looked at. And numbers, units, amounts, geography. That would be for example zip-code, geo-city. Another one would be contacts, like email and phone-number. Than names, music and more.

The table below provides input and output examples for all system entities. You can filter the table by one or more languages and/or system entity names. If an entity has no examples for a specific language, the entity does not support that language.

Filter by language

Chinese-Traditional (zh-TW)

Danish (da)

Dutch (nl)

English (en)

French (fr)

		Input Examples	Output Examples
	e	2:30 pm 13 July April morning tomorrow at 4:30 pm tomorrow afternoon	"2018-04-05T14:30:00-06:00" "2018-07-13T18:00:00-06:00" {"startDate":"2018-04-01T12:00:00-06:00","endDate":"2018-04-30T12:00:00-06:00"} {"endTime":"2018-04-06T12:00:00-06:00","startTime":"2018-04-06T08:00:00-06:00"} {"date_time":"2018-04-06T16:30:00-06:00"} {"startDateTime":"2018-04-06T12:00:00-06:00","endDateTime":"2018-04-06T16:00:00-06:00"}
English (en)	@sys.date	tomorrow	2018-04-06T12:00:00-06:00
English (en)	@sys.date-period	April	{"startDate":"2018-04-01T12:00:00-06:00","endDate":"2018-04-30T12:00:00-06:00"}
English (en)	@sys.time	4:30 PM	2018-04-05T16:30:00-06:00
English (en)	@sys.time-period	afternoon	{"startTime":"2018-04-05T12:00:00-06:00","endTime":"2018-04-05T16:00:00-06:00"}

Here is a list of all available system entities.

You don't need to know them by heart.

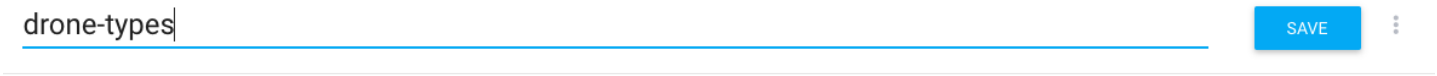
User Entities

The next type of entities are user entities. User entities can be redefined on a session ID level, allowing for specific concepts, like drone types.

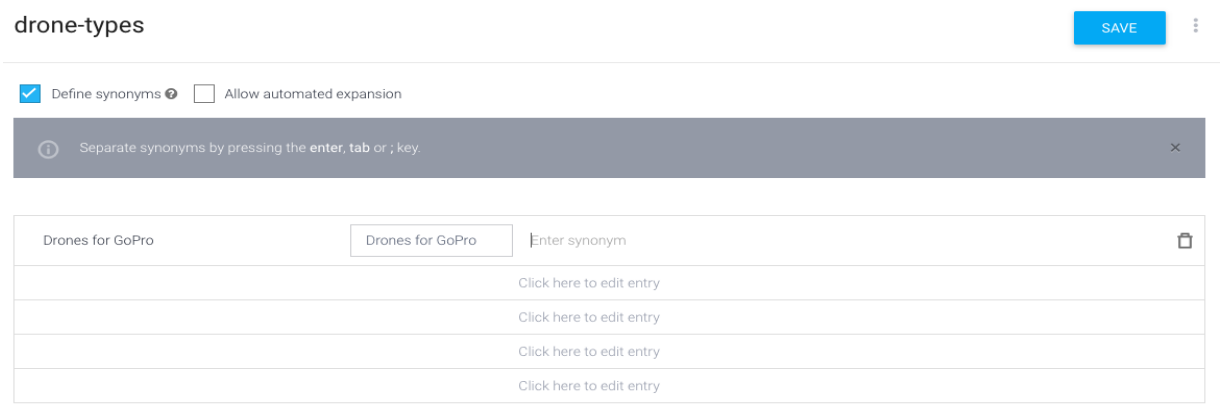
And what we'll use in this course is developer-defined entities. These are the entities will define ourselves.

What I want to do is have an intent where buyers will ask for a specific drone type. Like drones for beginners. Or drones with a camera. And I will want to extract what kind of a drone type they are interested in.

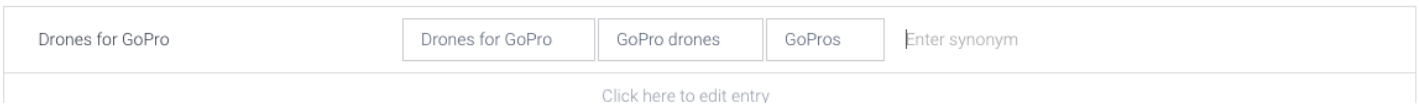
For my agent to be able to recognize drones as parameters I need to define all the type of drones I have available in the shop. Let's go and do this. Go to tab entities. And click Create entity or just the plus sign here near entities. I will call this entity drone-types.



And what I need to do is add each type of drone and its synonyms. I'll start with Drones for GoPro. Add it here.



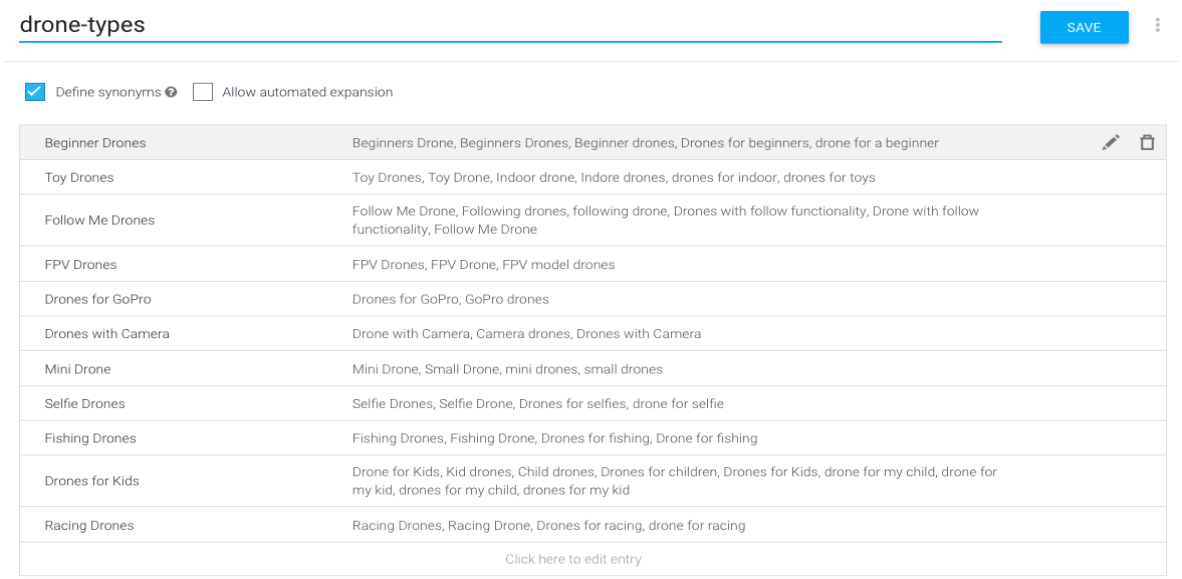
I'll add a GoPro drones as a synonym. Like this. Also, GoPros. Ok? So, all the possible parameters with their synonyms. And DialogFlow will recognize the word no matter if the user writes GoPro drones or Drones for GoPro.



Next one will be Drones with the camera. Drones with a camera. Ok, what the user could also write is Camera drones. Like this.

And then, another type I have is Drones for kids. A synonym can be just short version of kid drones. Kid drones. Drones for my kid. And maybe also Drones for my child.

And in the same manner I will add all the other type of drones. I will first add the name that I want to extract. And then all the possible synonyms for it. Try to figure out all the synonyms. Like this.



We have now created the first entity, we looked at system entities and we are now ready to create an intent that will be able to recognize parameters from user input. Like we have here in the diagram. With the help of entities.

Let's review again.

> Entities are nothing more than predefined words, that we want our bot to recognize and send to backend app. When we need a parameter, we add an entity. Then we add all possible values with synonyms.

And now it's time to use this entity to extract a parameter from a string.

Ok, let's go to the next article where we'll create an intent that will be able to recognize parameters from user input.

Here is the link to documentation:

https://dialogflow.com/docs/reference/system-entities

How to extract parameters from the conversation

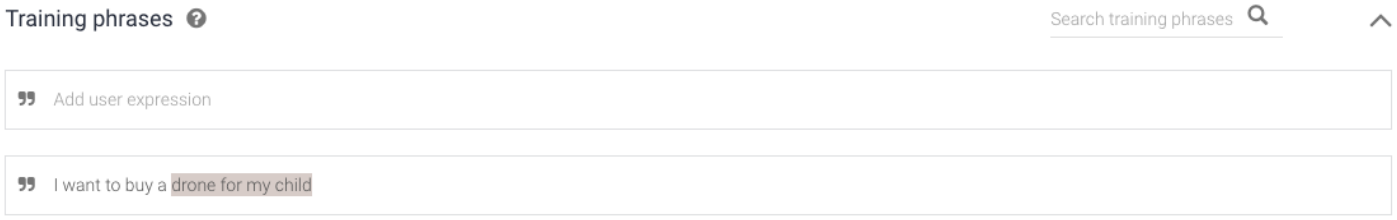
Hi and welcome back. In the previous article, we've created entities.

Entities are all the predefined words we want to read from input string as extra parameters.

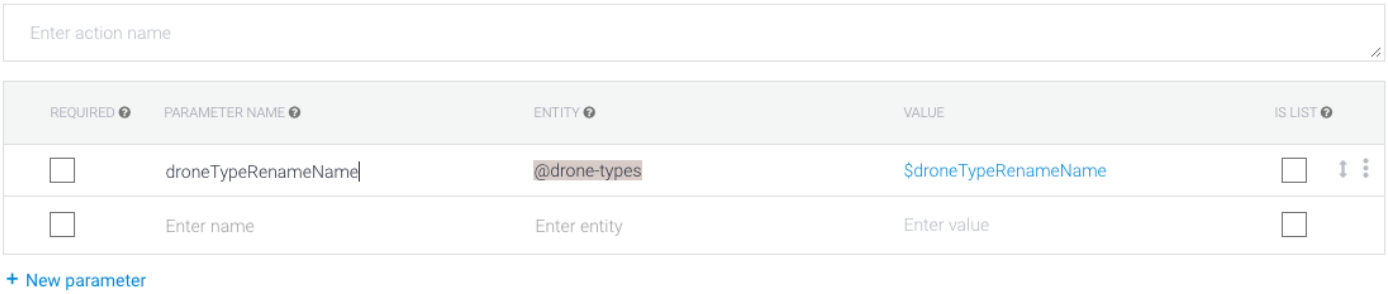
So far you saw that entities can be system entities, user-defined entities, and developer-defined entities. Now I'll show you how to get parameters from the input string with the help of entities.

We've created an entity called course with the list of all available drone types. Let's create a demo intent that will get the drone type from user input. I'll add a new intent. First, we need to name it. I'll call it simply want type of drone. Ok.

And now we can start with Training phrases. The first one I'll add is: I want to buy a drone for my child.



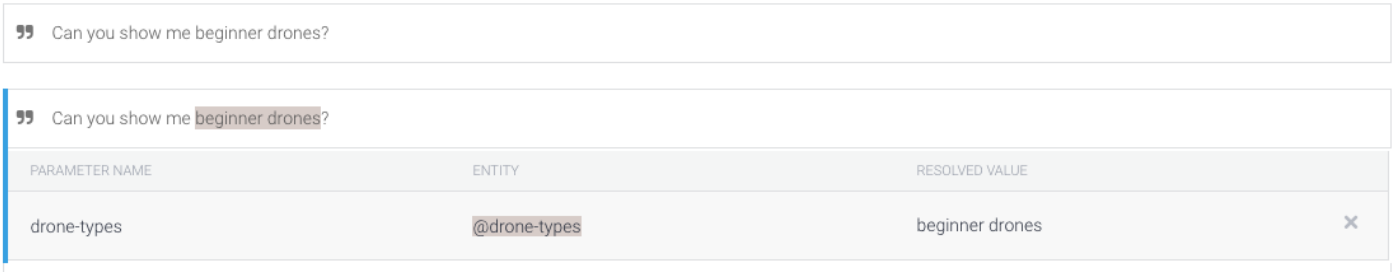
And woohooo. Did you see? DialogFlow recognizes that this was the entity. It suggested it for us. And it already named the parameter drone-type. We could rename the parameter name if we wanted. It does not have to be the same name as the entity is.



But I don't mind having the same name. So, I'll leave the parameter as is, that is drone-types

And you can see it is already here in the list of parameters. I'll add a few more Training Phrases. The next one will be: Can you show me beginner drones?

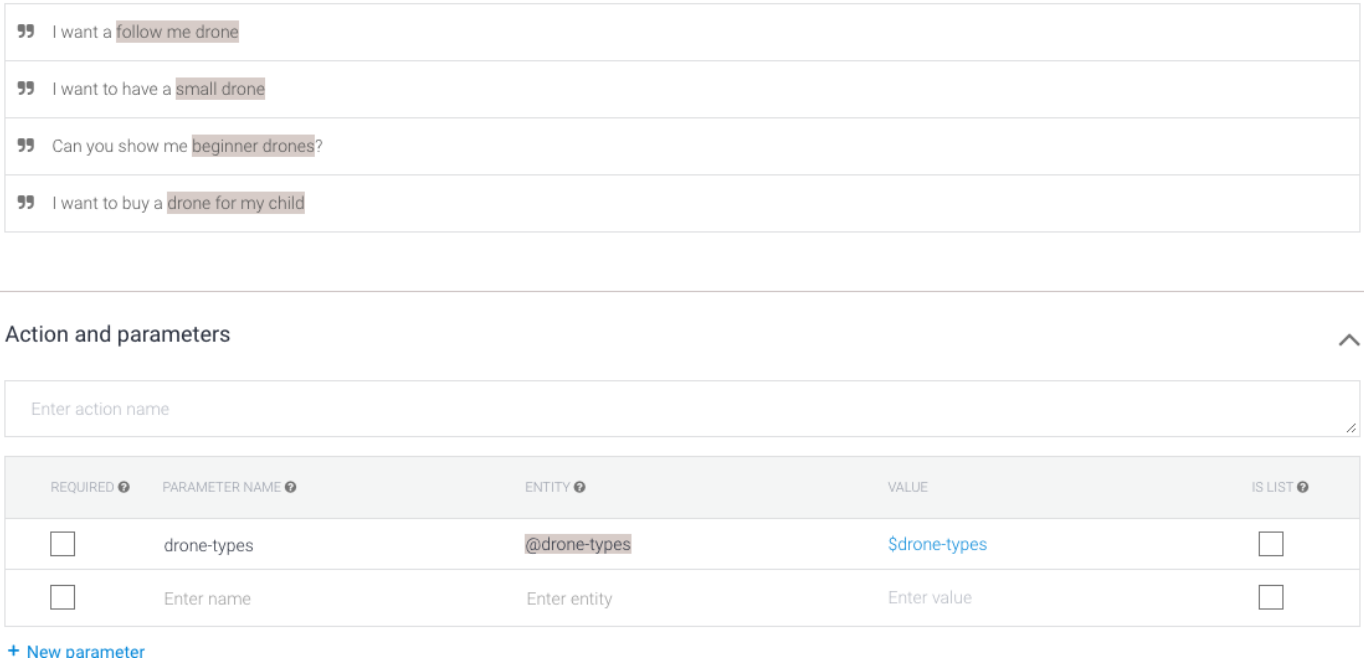
And again, Beginner drones was recognized.



And another one. I want to have a small drone. Ok, this time it recognized a mini drone.

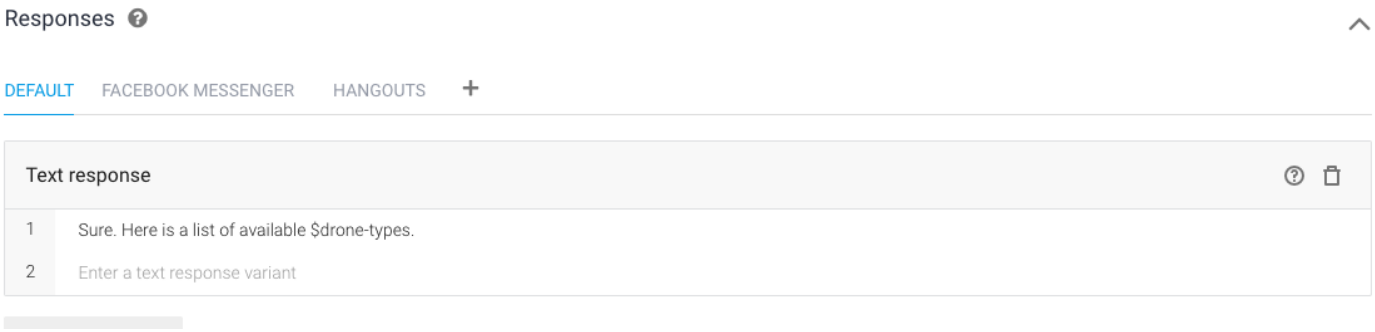
Just a few more. I want a follow-me drone. Like this. Now I have enough.

Ok. Here in parameters, we see that it will read the parameter's name from entity drone-types.



Now, all that I need to do is add a response. And I'll write. Sure. $.

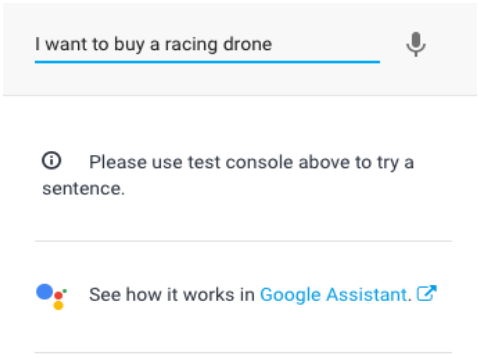
To access the parameter in the response, we can write $drone-types. We use $ sign and the name of the parameter.



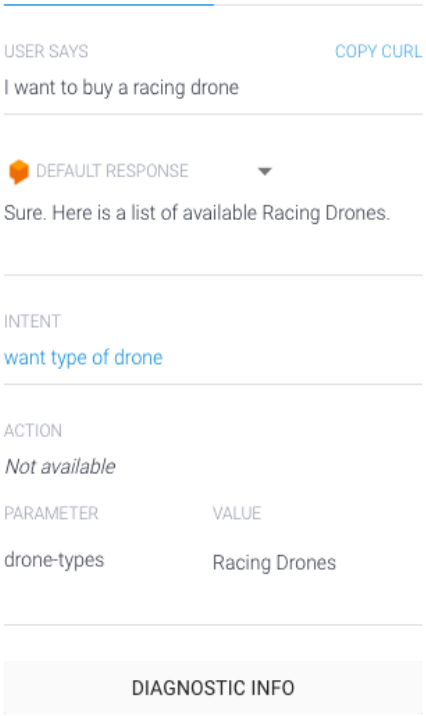
I have not added a list of available drones. Why? Well since we don't have a list of products here. And we don't want to hardcode them in the DialogFlow. I want this to be dynamic. Read from the same database and the website uses. O joj! What can we do now? How will we get it?

Don't worry! Remember I told you that we're going to build a backend app? Well, this is exactly why we need a backend app. Based on the course user will ask for, we'll search for the right drones in the backend app. But be patient. First, **save this intent.** Up here. Training will begin in the background. Ok!

Now we can go and try the intent. Up here in the try me now. Go and write I want to buy a racing drone.

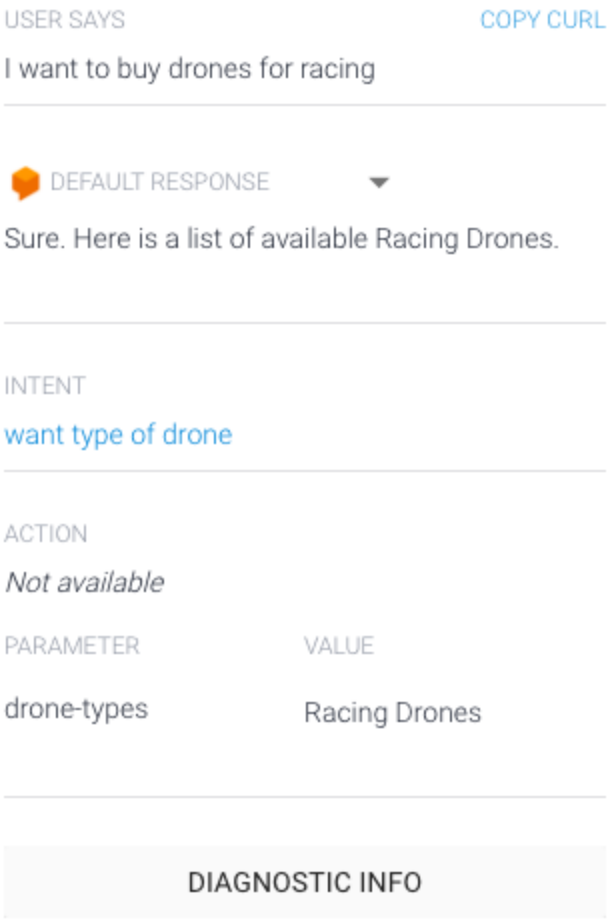


And woohooo. Here we have the response. And it also recognized the parameter. It is racing drones. That is cool.



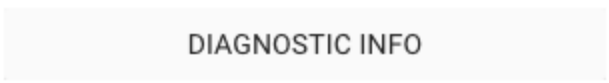
Now let me show you something. Now I'll write this to try it now: I want to buy drones for racing.

And enter. And again, the parameter name is again racing drones. That is because drones for racing is just a synonym of what we defined as racing drones. Cool, ha.



Our app will always get the same name as the drone type, so we don't have to handle all the synonyms.

What is being sent as DialogFlow response? Click the diagnostic info.



This is the JSON string DialogFlow would send now to the bot.

Diagnostic info

RAW API RESPONSE


```
{
  "responseId": "69fc1df4-f040-4ea0-a833-89df517bebbe-d2c275dc",
  "queryResult": {
    "queryText": "I want to buy drones for racing",
    "parameters": {
      "drone-types": "Racing Drones"
    },
    "allRequiredParamsPresent": true,
    "fulfillmentText": "Sure. Here is a list of available Racing Drones.",
    "fulfillmentMessages": [
      {
        "text": {
          "text": [
            "Sure. Here is a list of available Racing Drones."
          ]
        }
      }
    ],
    "intent": {
      "name": "projects/dialogflowtutorial-587b5/agent/intents/21af6fd3-8b1a-41a7-aacd-21200fcb3db9",
      "displayName": "want type of drone"
    },
```


CLOSE COPY RAW RESPONSE

As you can see under queryResult is the queryText, that is the input from the user. And right after that, we have parameters. This way our app will know what parameters have been collected.

```
{
   "responseId":"69fc1df4-f040-4ea0-a833-89df517bebbe-d2c275dc",
   "queryResult":{
      "queryText":"I want to buy drones for racing",
      "parameters":{
         "drone-types":"Racing Drones"
      },
      "allRequiredParamsPresent":true,
      "fulfillmentText":"Sure. Here is a list of available Racing Drones.",
      "fulfillmentMessages":[
         {
            "text":{
               "text":[
                  "Sure. Here is a list of available Racing Drones."
               ]
            }
         }
      ],
      "intent":{
         "name":"projects/dialogflowtutorial-587b5/agent/intents/21af6fd3-8b1a-41a7-aacd-21200fcb3db9",
         "displayName":"want type of drone"
      },
      "intentDetectionConfidence":1,
      "languageCode":"en"
   }
}
```


Then what is important here is also this property: allRequiredParamsPresent. If we set our parameters to be required, then this flag would tell us if the user provided all the parameters. We'll be doing this in one of the future articles, where we'll collect data from the user and save it to the database.

The next thing our app will use is fulfillment messages. Here all the messages will be listed. Remember, the bot can reply with more than one message. Ok?

And finally, if we would have added an action, that would also be seen here.

So, ok. Now you learned how to get parameters from user input. And you've also learned how the response from DialogFlow will look like.

Since we don't have all the drones listed in the DialogFlow we should read them from a database. We could do that in the fulfillment.

Rich messages

DialogFlow can be integrated into all sorts of different platforms. And **each of these platforms has a different kind of responses**. For example, we could be using DialogFlow to integrate it with Facebook. Facebook Messenger can display text, cards, galleries, images, quick responses. It would be a different scene if we integrate it with Google Assistant. Google Assistant can work on a phone or on a speaker. Therefore the responses must be different if we use a screen or just voice. Furthermore, if we use to screen the responses are different than on Google Messenger or Slack or Viber. Google Assistant can display Simple text responses or cards, a carousel of cards or lists or suggestion chips.

DialogFlow helps us to provide responses that would work with the platform we are integrating agent with.

We can integrate the agent with several platforms at the same time. DialogFlow can provide to add some basic rich media for all platforms.

What I would like to do to show you an example is add responses for Facebook Messenger, for Google Assistant, and for Slack.

And you can also with a little extra skill add anything a platform can provide. Here you can read more about possible messaging options in the documentation.

Here is Slack documentation.

https://api.slack.com/messaging/composing/layouts

And this is a nice interface that will help you interactively build the JSON needed for sending messages. Super nice to learn:

https://api.slack.com/tools/block-kit-builder?

Messenger's documentation is not as nice, but I do recommend going through it.

https://developers.facebook.com/docs/messenger-platform/send-messages/quick-replies

And here is google Assistants documentation with all possible responses described:

https://developers.google.com/actions/assistant/responses

So, we would need to set up the right responses for each platform. From a design point of view **using the same agent for different platforms might not always be the right answer**. That is because the conversation on a speaker or on the screen is much different. And in the same way, because the interaction is even different between platforms it would be wise to think about when to use the same intents for different platforms. But in case we decide so, we would be able to respond to each platform.

Let me show you an example. I will show you how to add responses for Facebook Messenger, for Google Assistant, and for Slack

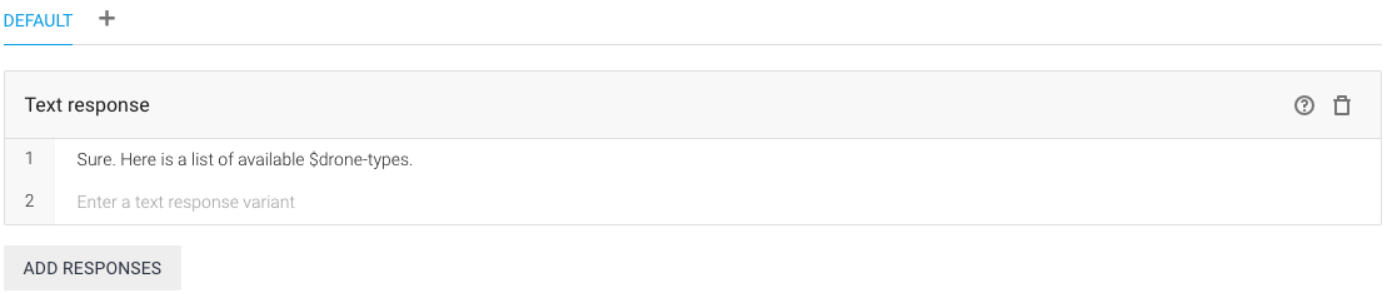
What we will do is add responses to “want the type of drone” intent. Since the user is asking for a specific type of drones we will return a gallery or a list of available drones. Because we are not using a backend application to get the right kind of drones, the reply will always be the same. Meaning the user will get a list of drones but it will be the same kind of drones for each request. Yes, we are making progress but are not there yet. I will also show you how to get the right type of drones in the near future. We will be adding backend code and reading the right kind of drones from a database. But first, we need to know how to add responses to different kind of platforms. Messenger will show a list of cards that will change into a gallery. Google Assistant will show a list the user can scroll and click. And Slack will show a block with image and link. It is not as complicated as it sounds. Let me show you.

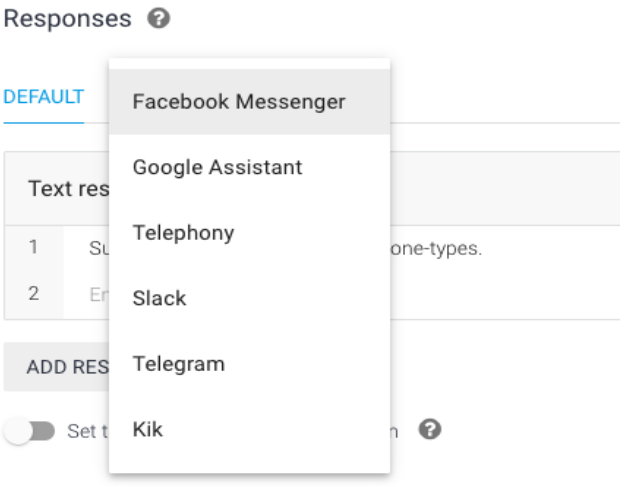
Let’s start with Facebook Messenger.

Facebook Messenger Rich messages

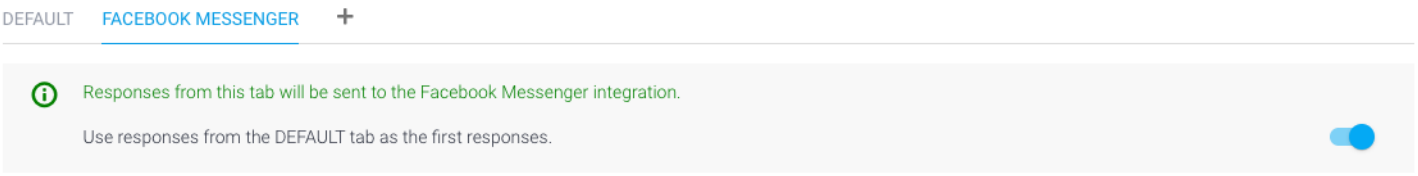
Go to “want type of drone” intent and scroll all the way down to responses. So far we have one response under the Default Tab and it says: Sure. Here is a list of available $drone-types. $drone-types variable will be replaced with the value of the parameter we extract from the user input text with the help of entity drone-types. Perfect!

Now let’s add a list of cards. Add the Facebook Messenger tab if you don’t have one. Add it by clicking on the plus sign.



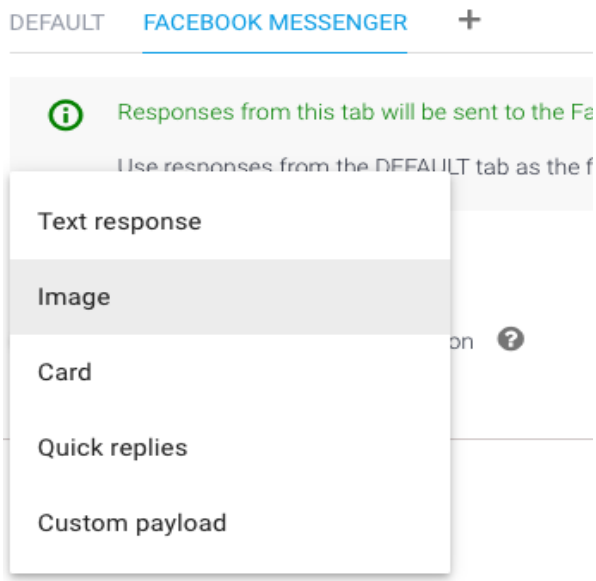


Here you will see a tab where you can include responses from a default tab to the platforms responses. Therefore, if this checkbox is ticked all the responses in the default tab will be included with the Messenger responses.



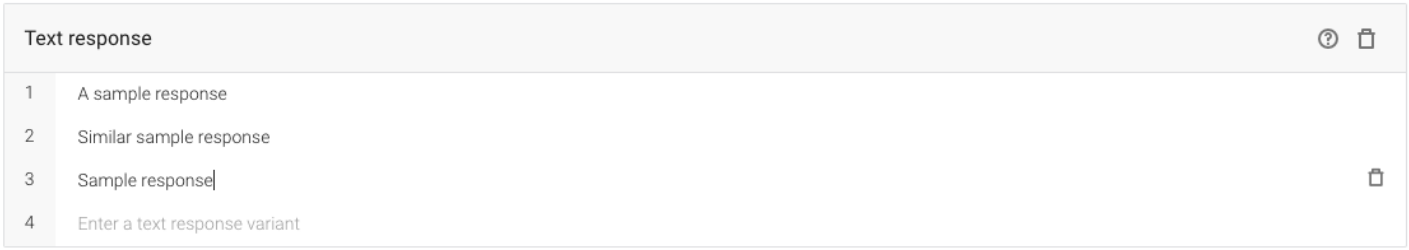
Since in our default tab we have the response: “Sure. Here is a list of available $drone-types.” I want to have this response included in the Messenger responses, so I will leave this option on.

Now let’s click Add responses button and see what DialogFlow offers out of the box.



Here we see text response, image, Card, Quick replies and Custom. It is true, this is not all the possible responses. But with a custom payload, you could add other kinds of responses. Let’s see the one available to us.

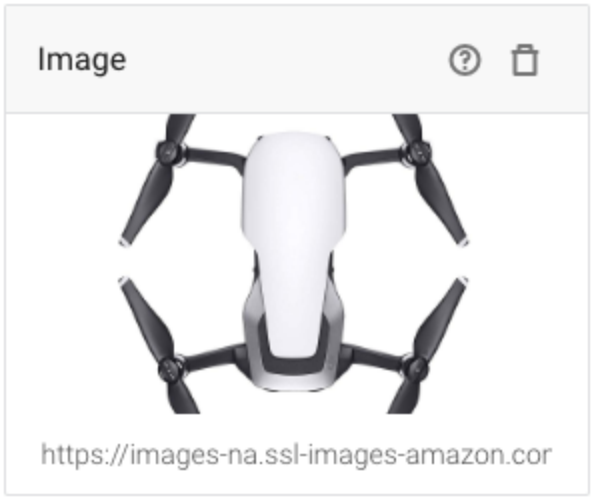
The text we already know. We can provide text variations so that responses look more natural



Remember: you can add more responses. Like in a natural conversation. We can send more messages at once.

An image

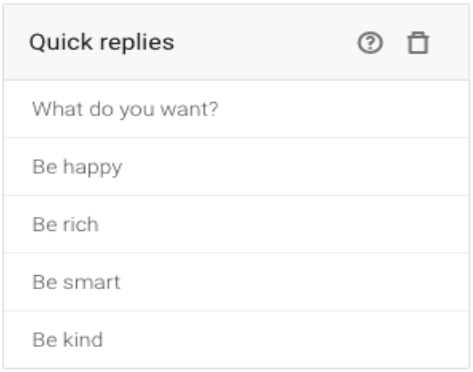
Next type of response is an image. In here you enter the image URL. So yes, the image must be uploaded to a server. Just surf the web for free image hosting if you have none.



Quick replies

A quick reply is a preset button the user can click. User will be able to click only one option. After one is selected the others will disappear. This way Facebook prevents the user to click more options. So it is one option-click only.

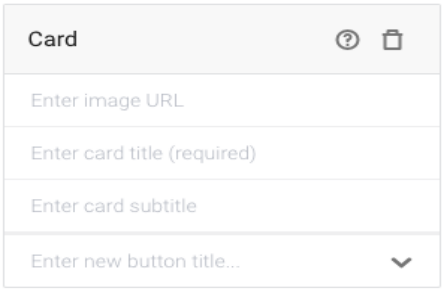
Tell me, what would you choose?



And a little warning. Quick replies should come last. If you add another response after the quick replies, they will not be shown.

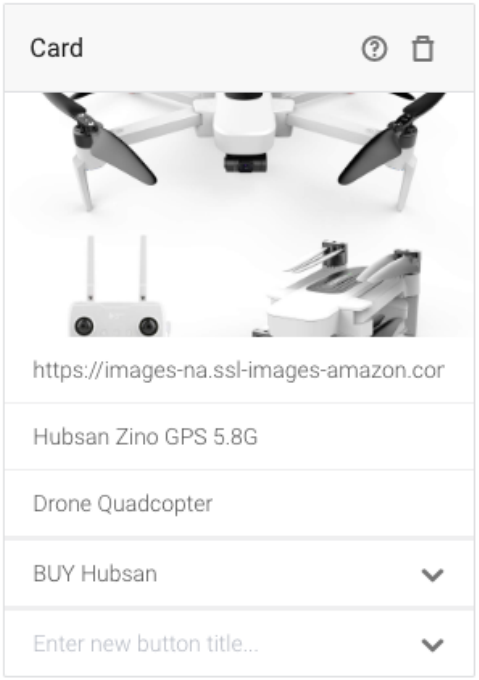
A card

Then we can add a card. A card contains image, title, subtitle and up to two buttons. Here is how an empty card looks like:



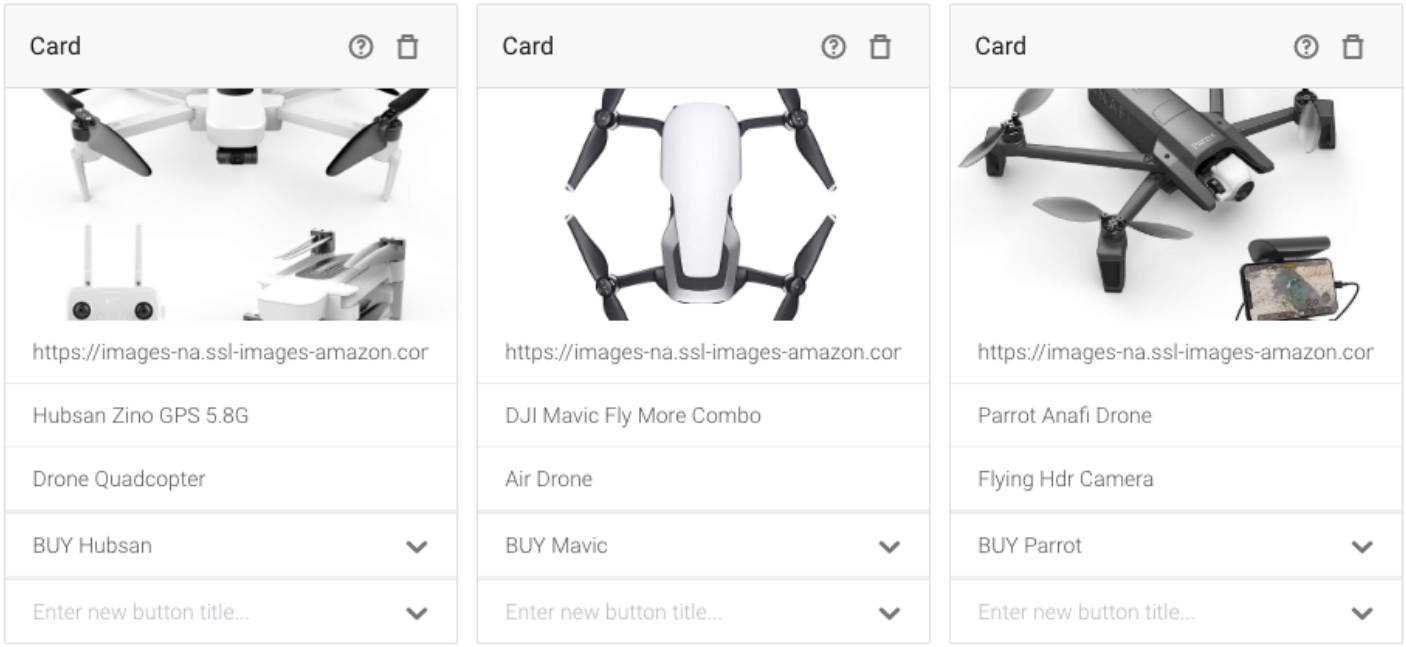
For now, I will only add one button with a link to buy the drone.

This is how a full card would look like:



A gallery of cards

If you want to have a gallery of cards you can simply add more cards to the responses. They but be in order. One after another. The ones that are together will form a gallery. You can add up to 10 cards into a gallery. Like this:



In DialogFlow we can only add two buttons although Facebook supports up to three buttons. If we wanted to add a card with three buttons we would need to use a Generic payload message. Meaning you would add a custom payload. In that case, we can format the JSON in a way that Facebook supports. Let me give you an example.

Here is a link to documentation about Generic payload:

https://developers.facebook.com/docs/messenger-platform/send-messages/template/generic/

As you can see we can add more elements. We can add up to 10 cards for example.

```
"payload": {
  "template_type":"generic",
  "elements":[
    <GENERIC_TEMPLATE>,
    <GENERIC_TEMPLATE>,
    ...
  ]
}
```

Default_action property defines the URL that will be opened in the Messenger webview when the card is clicked.

```
"payload": {
  "template_type":"generic",
  "elements":[
     {
      "title":"<TITLE_TEXT>",
      "image_url":"<IMAGE_URL_TO_DISPLAY>",
      "subtitle":"<SUBTITLE_TEXT>",
      "default_action": {
        "type": "web_url",
        "url": "<DEFAULT_URL_TO_OPEN>",
        "messenger_extensions": <TRUE | FALSE>,
        "webview_height_ratio": "<COMPACT | TALL | FULL>"
      },
      "buttons":[<BUTTON_OBJECT>, ...]
    },
    ...
  ]
}
```

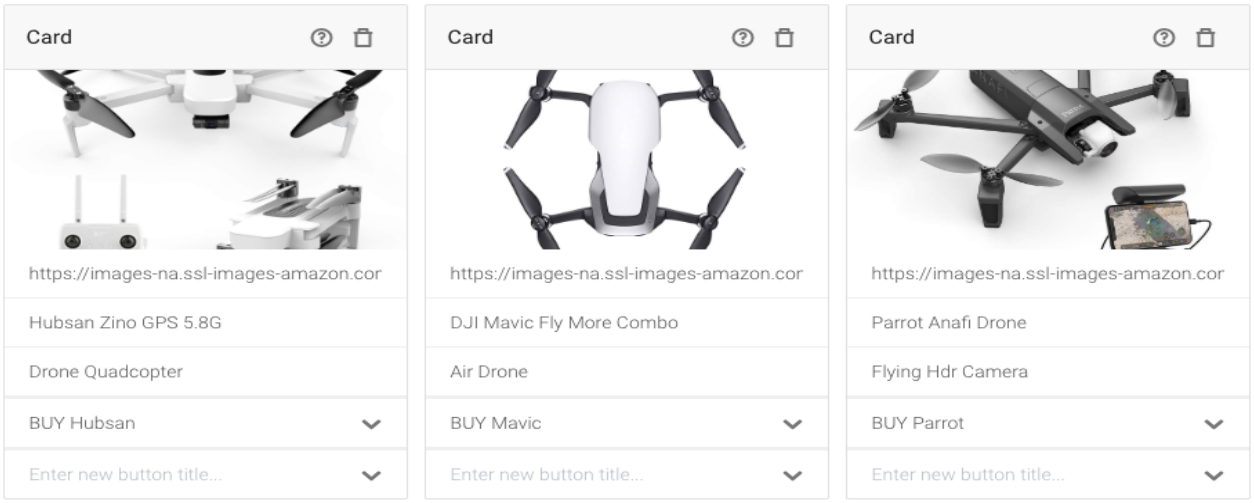
And you can also add up to 3 buttons to a card. So a little different than the functionality that is offered by adding cards through the DialogFlow interface.

I will show you how to add a card with three buttons. Here it goes:

Custom payload

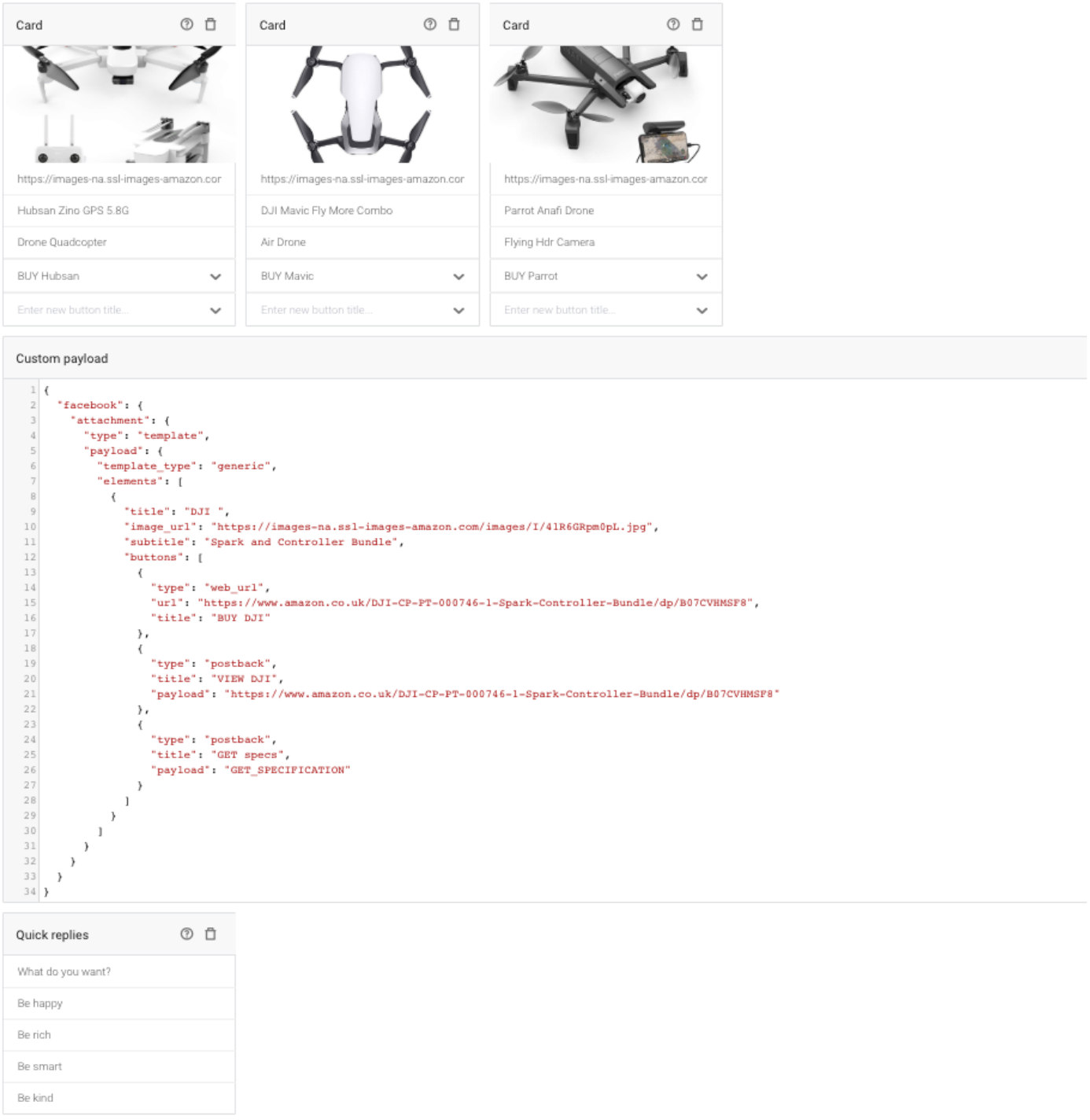
```
{
  "facebook": {
    "attachment": {
      "type": "template",
      "payload": {
        "template_type": "generic",
        "elements": [
          {
            "title": "DJI ",
            "image_url": "https://images-na.ssl-images-amazon.com/images/I/41R6GRpm0pL.jpg",
            "subtitle": "Spark and Controller Bundle",
            "buttons": [
              {
                "type": "web_url",
                "url": "https://www.amazon.co.uk/DJI-CP-PT-000746-1-Spark-Controller-Bundle/dp/B07CVHMSF8",
                "title": "BUY DJI"
              },
              {
                "type": "postback",
                "title": "VIEW DJI",
                "payload": "https://www.amazon.co.uk/DJI-CP-PT-000746-1-Spark-Controller-Bundle/dp/B07CVHMSF8"
              },
              {
                "type": "postback",
                "title": "GET specs",
                "payload": "GET_SPECIFICATION"
              }
            ]
          }
        ]
      }
    }
  }
}
```

If we add more than one card then the ones that are added together merge and form a gallery of cards. Let me add a few more cards with a drone in it. Like this:

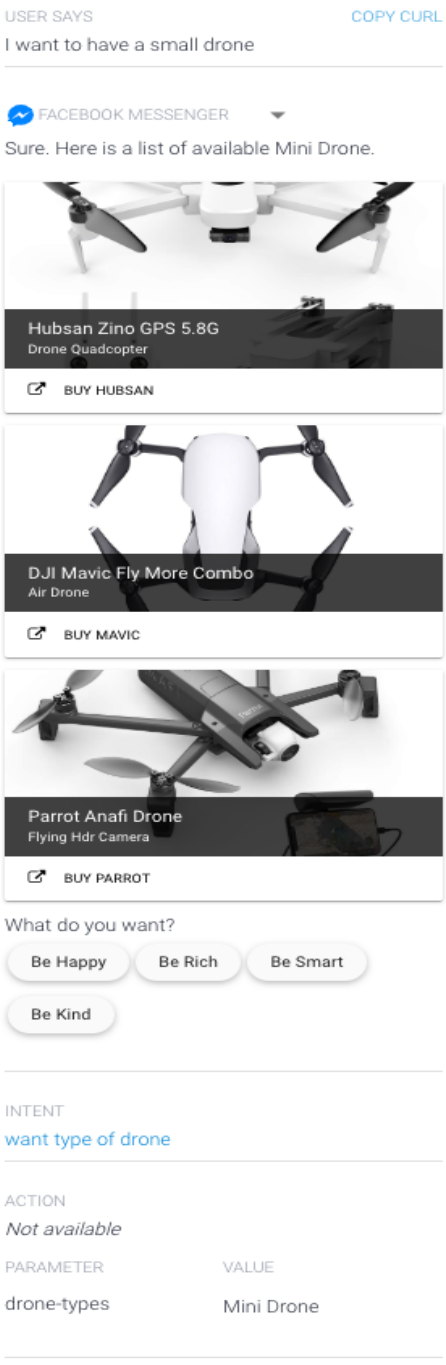


https://cloud.google.com/dialogflow-enterprise/docs/intents-rich-messages

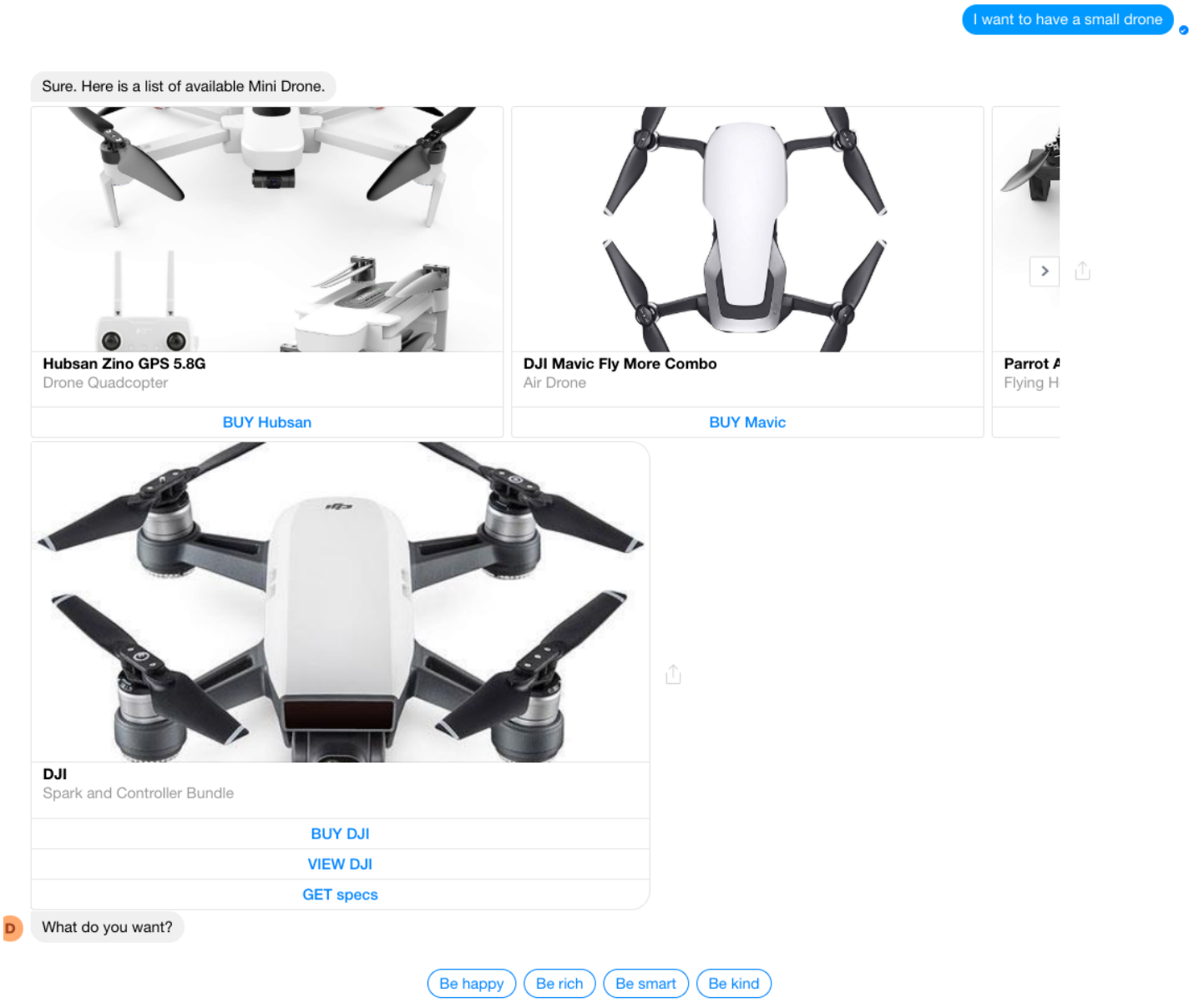
To show you an example I will keep 3 cards, a custom payload, and quick response buttons. Like this:



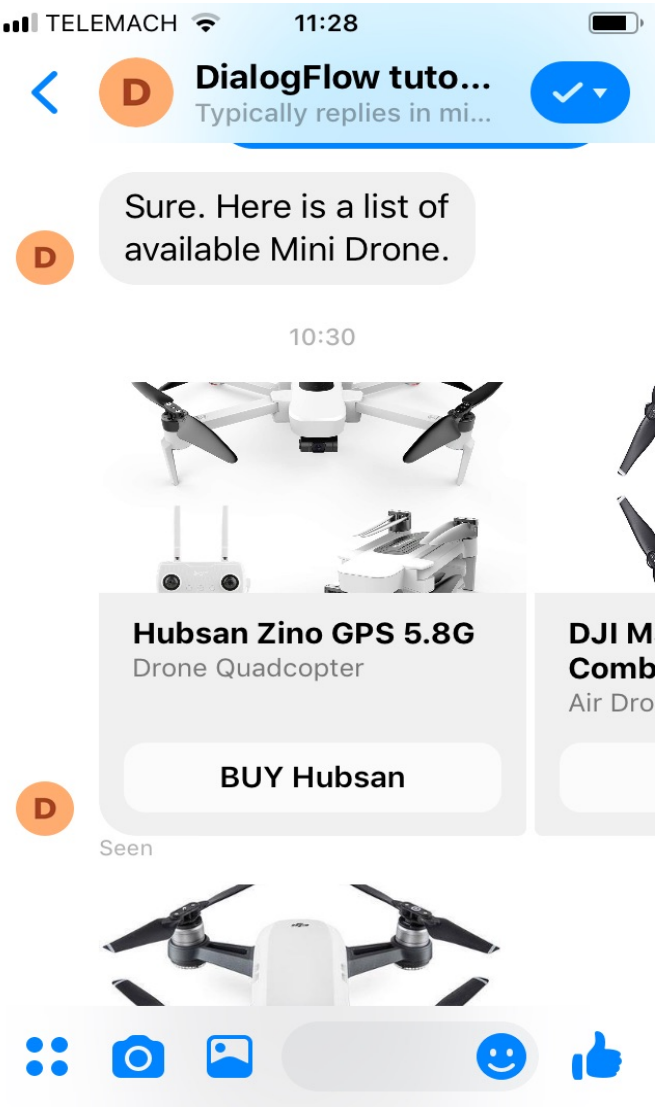
Don't forget to save the intent after you add responses. After the training, we can test the intent in the test area. I will write a query: "I want to have a small drone". And this is what I get when I select Facebook Messenger responses:



We can see here the text response coming from the default tab. And we can see the three cards. And the quick replies. But no custom payload. That is because DialogFlow does not know how to show it. If we link this agent with a Facebook Messenger then we will be able to see the responses. I will show you how to do this later in this little course. For now, I will just show you how the result looks like. This is on a web interface:



And this is on a mobile interface:



And as you can see here we have the cards gallery that we can list. Is a horizontal gallery. And we also can see the custom payload with three buttons.

DialogFlow gives us the tool to add any kind of rich messages to Messenger bots. For the more complex responses, we need to use custom payload.

Google Assistant - Rich messages

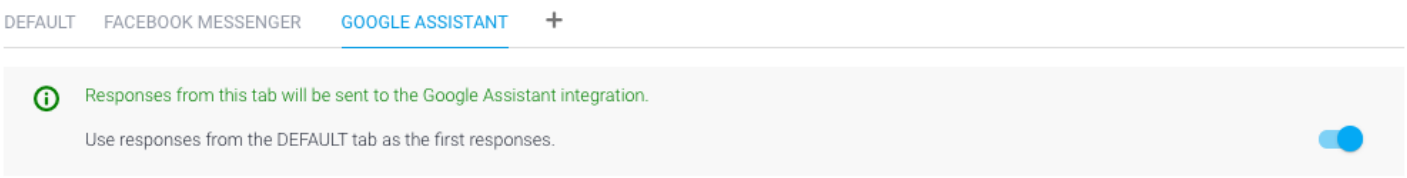
Ok, now we have seen how to add Messenger responses. Response for Google Assistant looks different. When we develop for Google Assistant we say we are building an action. Actions on Google lets you create Actions to extend the functionality of the Google Assistant

As you know Google Assistant can work on different kinds of interfaces. From a mobile screen to smartwatches, headphones, speakers, cars, TVs, and more.

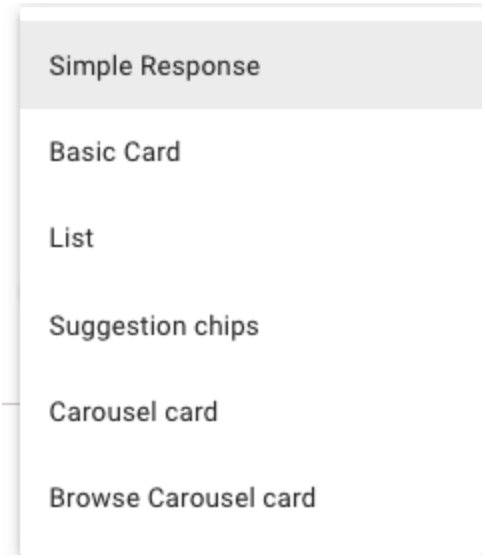
The design of the responses used where only voice is available should be quite different than on a device that has a screen. Also, both can be supported. We can decide and build an action that can support all kinds of devices or we can choose to support only screen devices or only speaker devices.

Here is a little reference where you can start if you have no prior knowledge of Actions on Google: https://developers.google.com/actions/overview.

If you want to add responses for Google Assistant you need to add a tab to responses. In the same way, as with Messenger can enable or disable using the responses from the Default tab.



And the responses available to us are: Simple response, Basic card, List, suggestion Chips, Carousel card and Browse Carousel card.



If we look in the documentation (https://developers.google.com/actions/assistant/responses) there are almost all responses available for the Assistant.

Simple response & SSML & Google Simulator

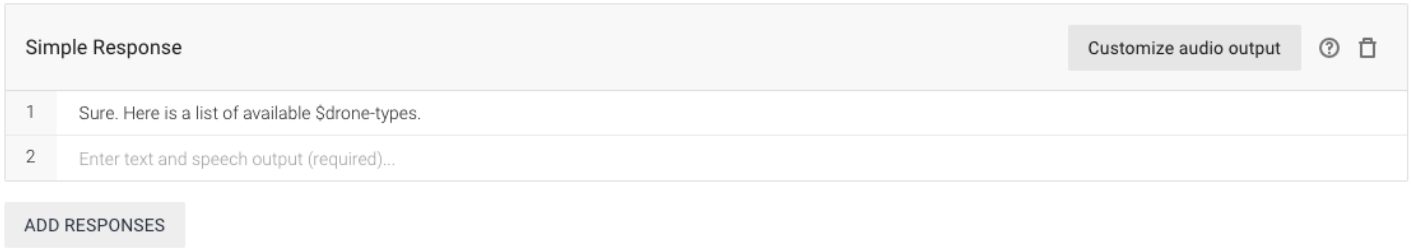
First, let's take a look at the Simple Response. As we have mentioned our action can work on devices that support screen interfaces or it can work on the voice-only devices. Or both.

With a simple response, we can define text responses. But not only that, we can design voice responses. With voice, we can do a lot of customization. We can add pauses, define how to format dates, numbers, we can add audio, customize the pitch, speaking rate and volume. Also, we can add emphasis to the words and more. This is available to use out of the box.

To format the audio we use SSML. SSML is a markup language to design voice. We don't use monotone voice, so SSML helps your bot/agent to sound more natural.

Here is a link to the documentation: https://developers.google.com/actions/reference/ssml. I will not show you all the examples since they are all listed in the documentation. Be sure to review documentation before we proceed. I will show you how to use SSML in the DialogFlow basic response for Google Assistant. Let me show you.

I will disable using the responses from Default tab and I will add text response here. To add text response we need to add a Simple response. Like this:



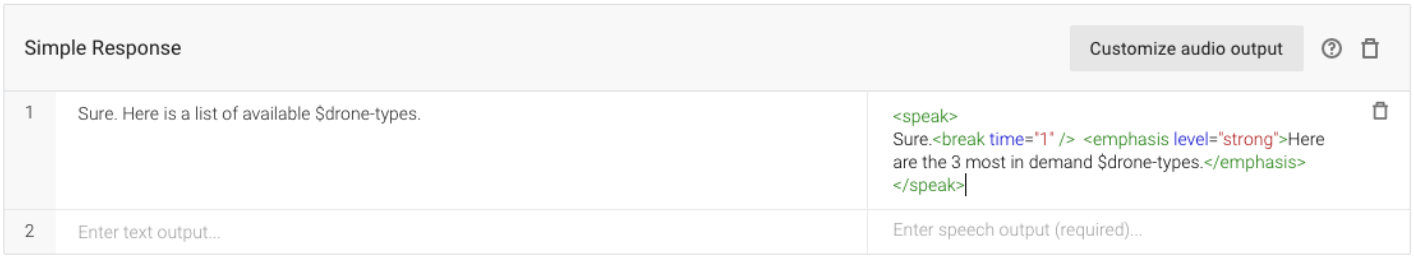
If you want to customize the audio you can click the link “Customize audio output”. In here we can add SSML code.

We start the SSML with <speak> tag. And we need to end it with </speak>. So, I will add a little break. And an emphasis on the second sentence. SSML I will add will look like this:

<speak>

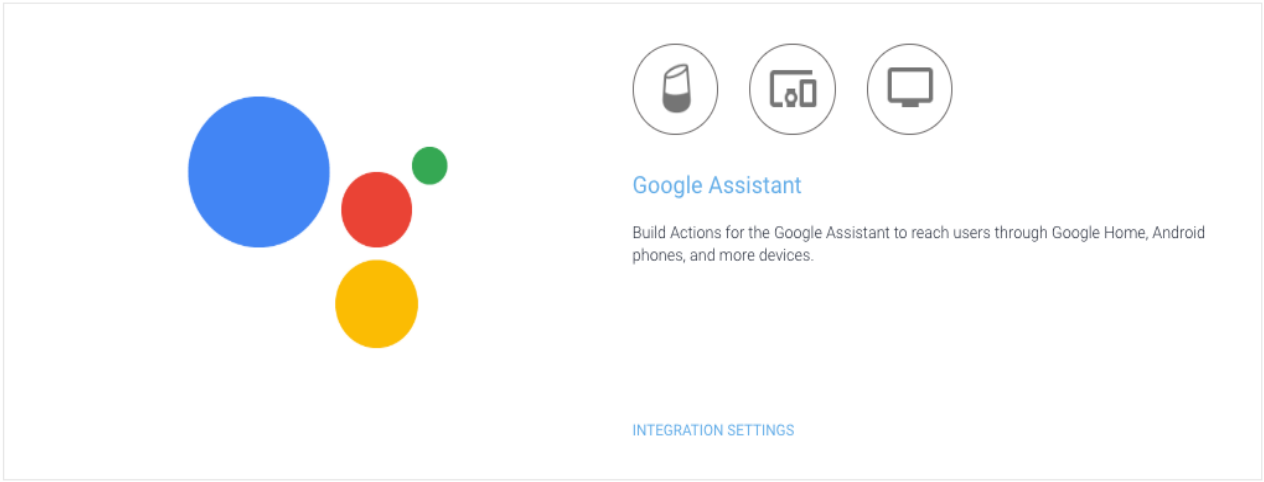
Sure.
<break time="1" />
<emphasis level="strong">Here are the 3 most in demand $drone-types.</emphasis>

</speak>



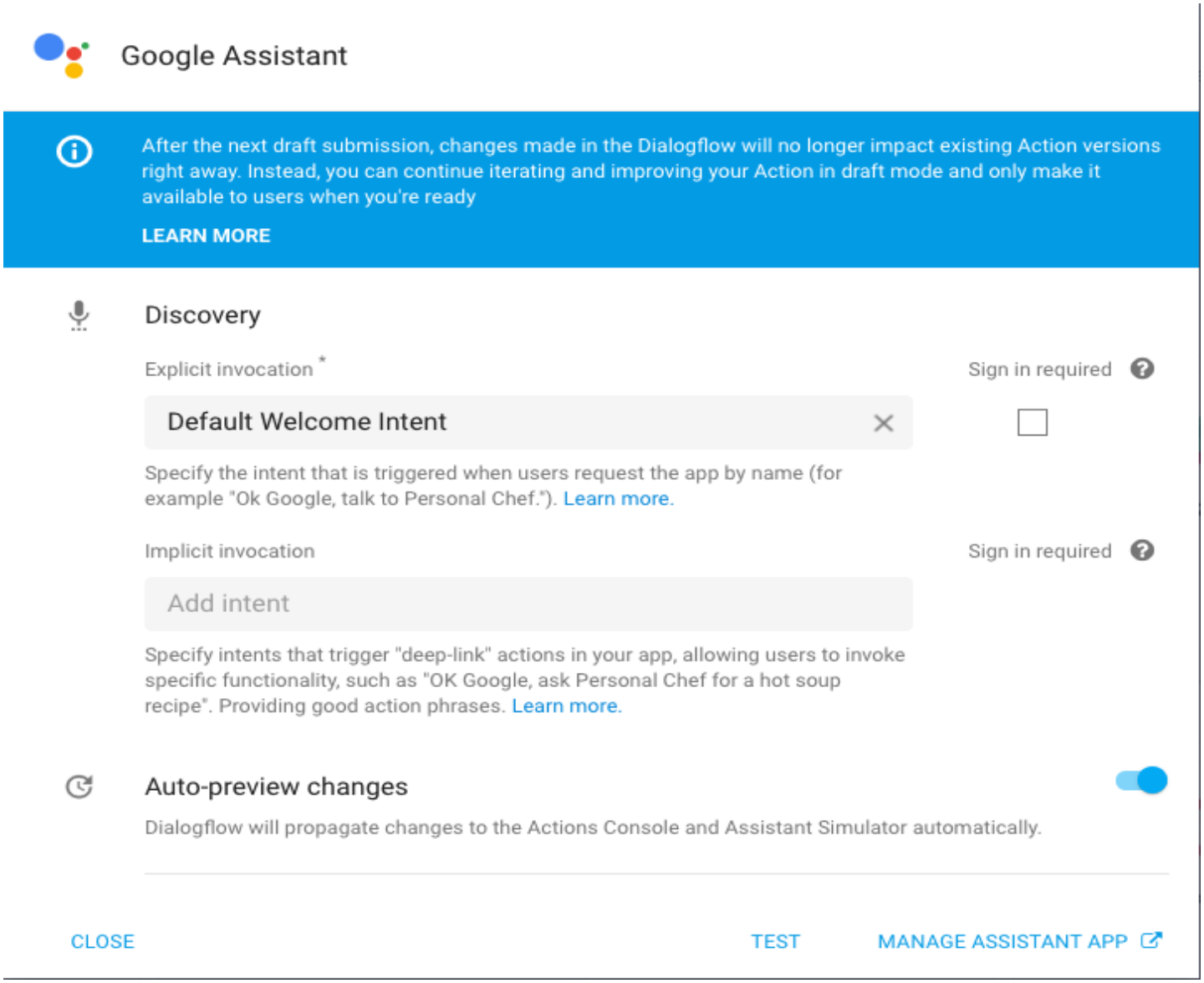
And now How can we test this? Well, pretty simple. Google has a simulator where you can test actions. It will show you what will be displayed on the screen and how the voice will sound.

To get to Simulator, go to Integrations tab and on top, you will see this Google Assistant banner.

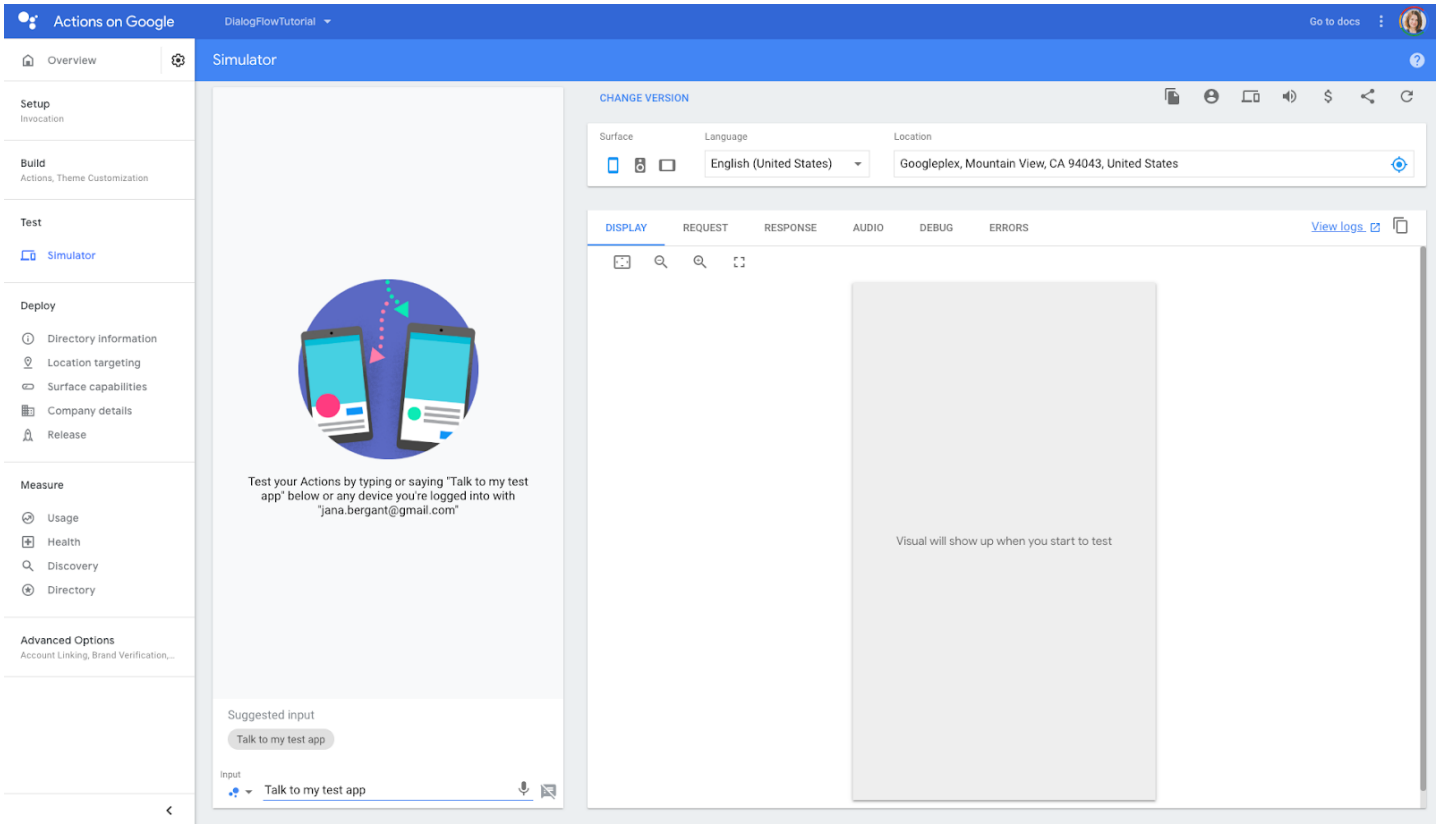


Click on the Integrations settings link. You will get this interface. First, we select the intent that will be triggered once the action on Google Assistant will be called. For us, we can leave the Default Welcome Intent. We can change this anytime we want.

But more important is to check the Auto-preview changes tab. This will save us a looot of time! We want each change in the DialogFlow to be reflected in the simulator and in the action. Do check it and click on TEST button on the bottom.

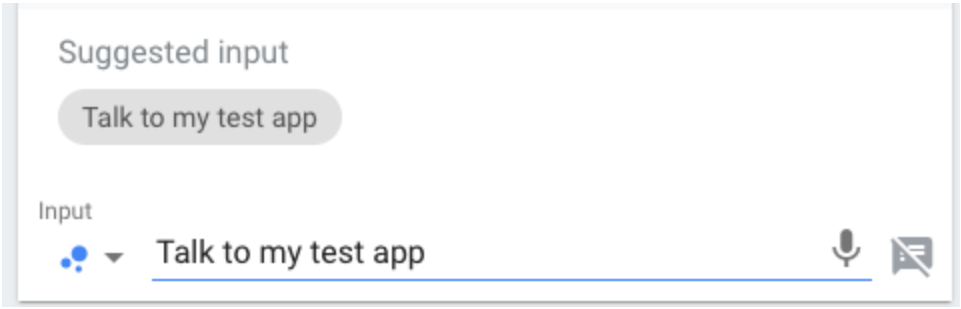


The Simulator will open. This is how it looks like. I will show you around in one of the upcoming articles.



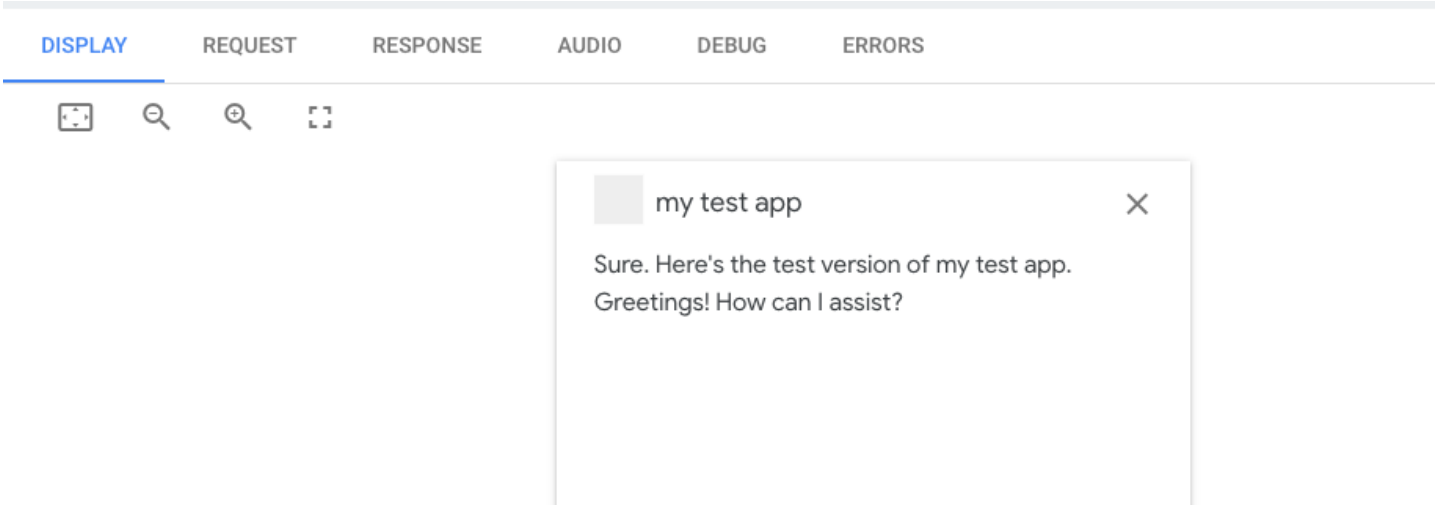
For now, let me just show you the basics.

Here is where you trigger your test action:



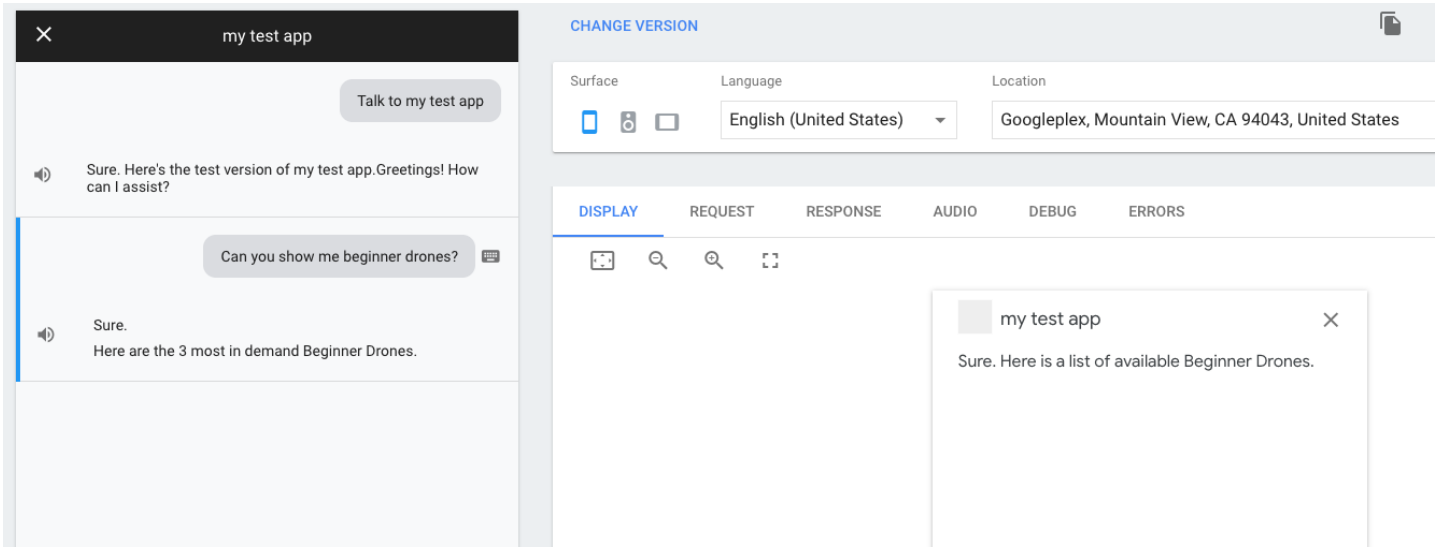
You can use voice or write text. Or just click the Suggested input.

You will see the display under the display tab and you will hear what the voice will sound like. Also, you can see written what will be said in the left panel.

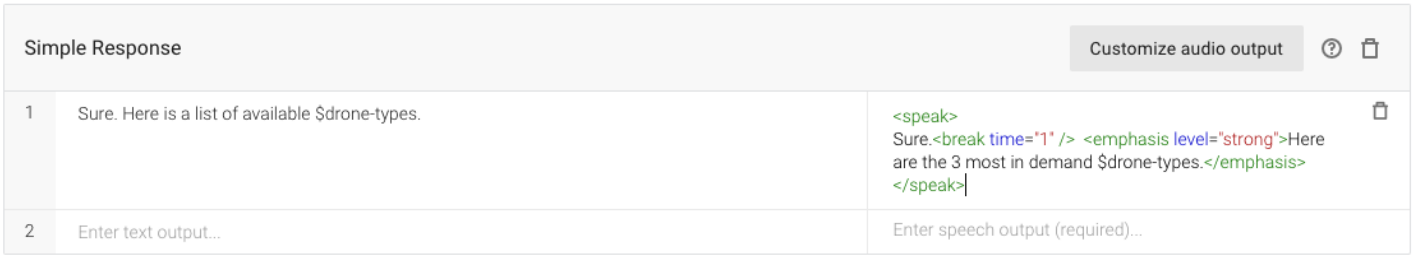


Now we can ask for drones. I will ask for beginner drones. You can type or use voice to do so. On the left you will see the voice response, also you will hear it. You can see there is a pause after sure. And that the second sentence sounds different. You can play with this a little more.

On the left, you will see what is to be displayed on the screen.



And that is what we have written in the Simple response.



Browsing carousel

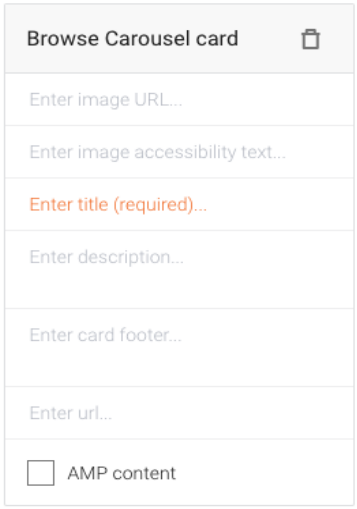
With browse carousel, users can browse vertically and select more than one item in the list. That is the difference with a List response that Google Assistant also offers. With a list, users can select only one option before the screen changes. With a browsing carousel, each item will onclick open a web browser with the content.

Therefore that is exactly what we need. We want to show drones and give a link to the drone where the user can continue shopping. And of course, we want him or her to be able to select more than one drone since they are still in the decision-making process.

To add a browsing carousel you need to add at least two Browse Carousel cards.

Browser carousel can have minimum 2 and maximum of 10 items.

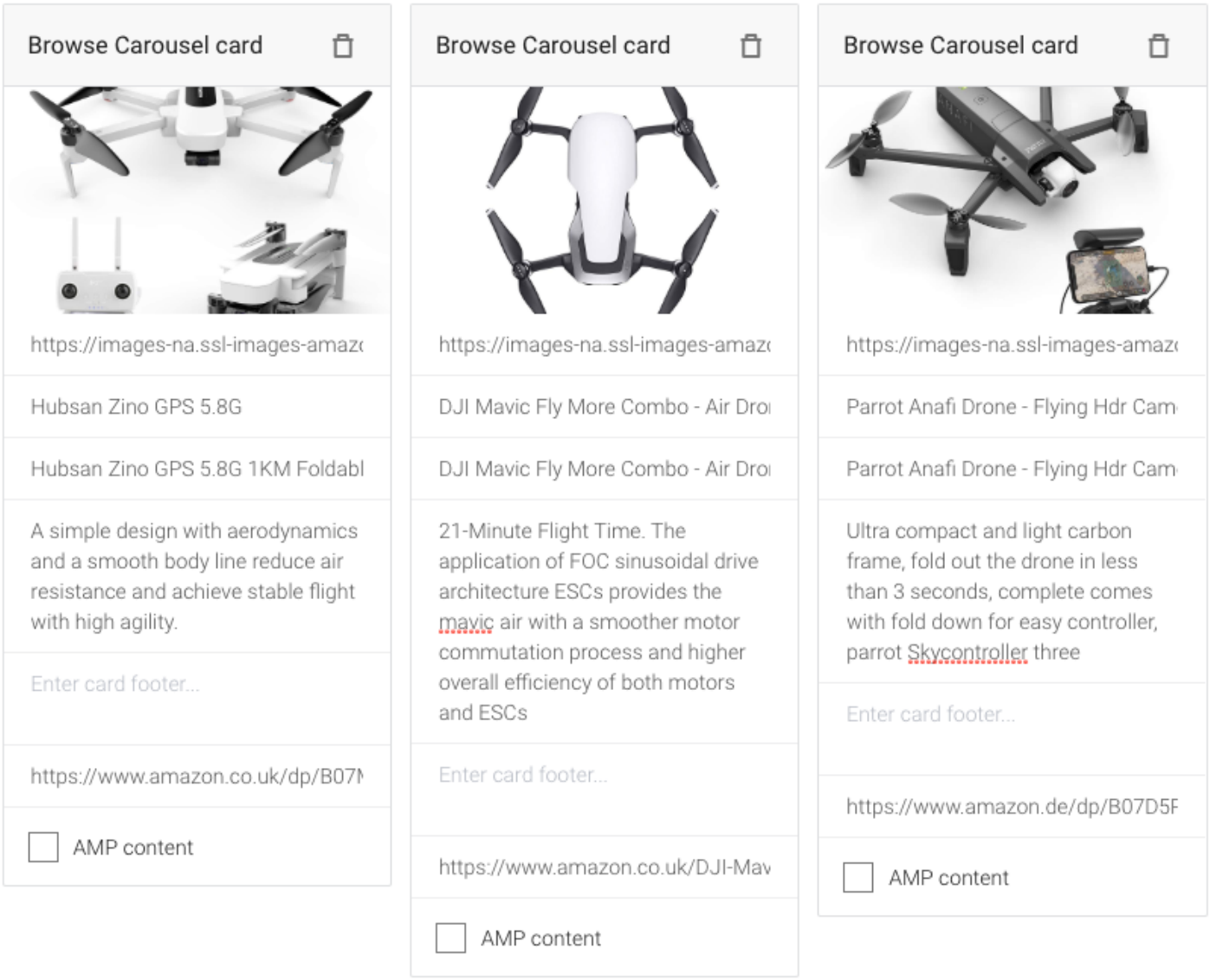
An empty Browse carousel card looks like this:



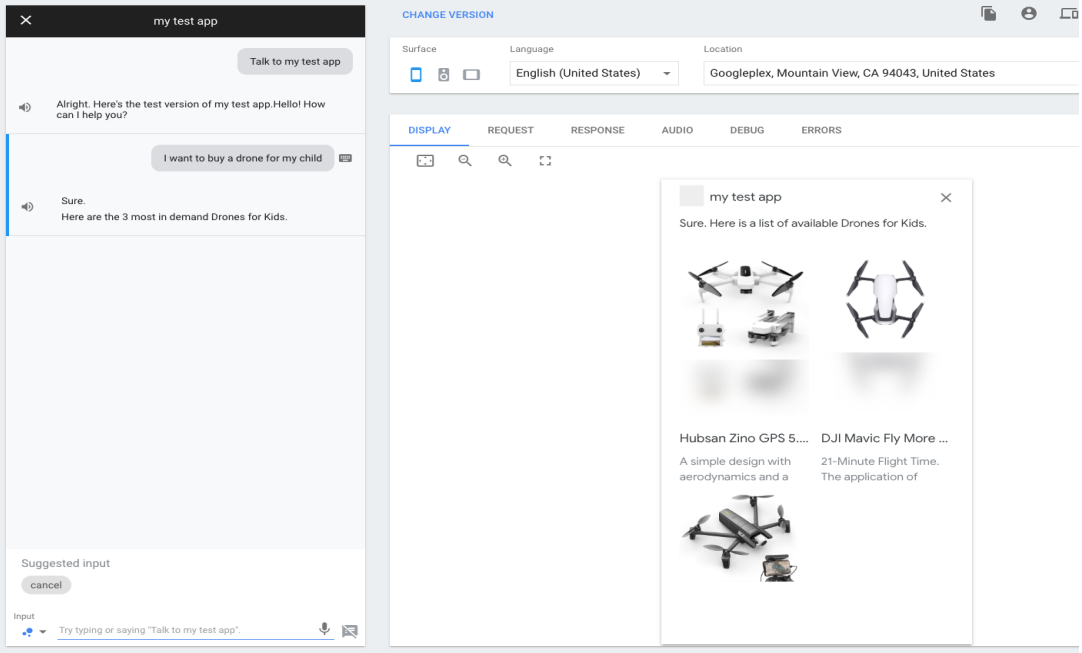
I will fill in the same data as for the Messenger. This is a static content that we will display for the user. So here is how a Browsing carousel card would look like.

If the link would show to Accelerated mobile page, then we would check the AMP content. AMP is a Google project with which they want to promote websites that work super fast because the use of super-simple structure and no extra javascript to slow the page down. You can read more about AMP here: https://amp.dev/about/how-amp-works. My page supports AMP :)

And I will add three drones, the same as for the Messenger.

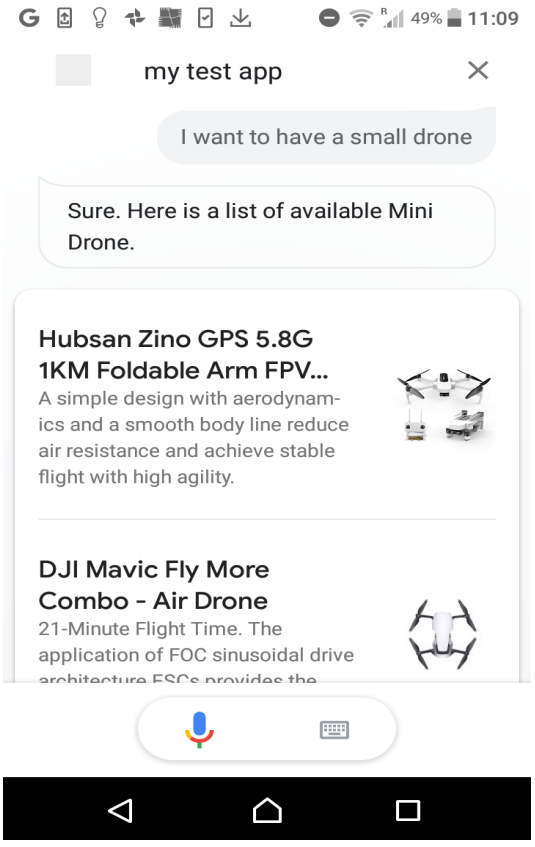


When we test the Browsing carousel in the simulator the display of the content will be a bit broken. Like this:



But the good news is that you can test the action in all devices that are connected with the developer email you use in DialogFlow and Google project. When I talk to my test app on my phone and ask for available drones, the response looks like this:

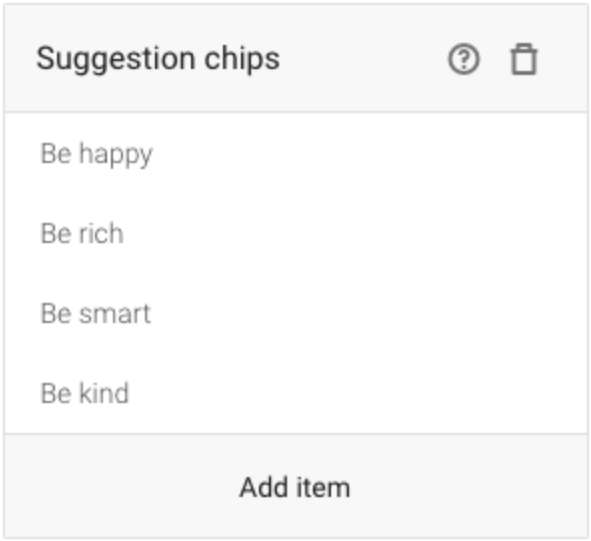
s214-Google_Assistant-Rich_messages/Google-assistant_drones_gallery.png



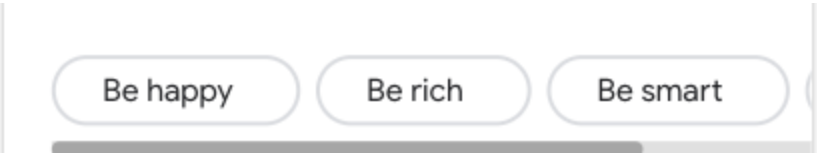
Suggestion chips

Suggestion chips are similar to Quick replies in Messenger. They offer users a way to quickly respond. Only one can be selected. And they disappear after one is clicked. Therefore it is basically the same.

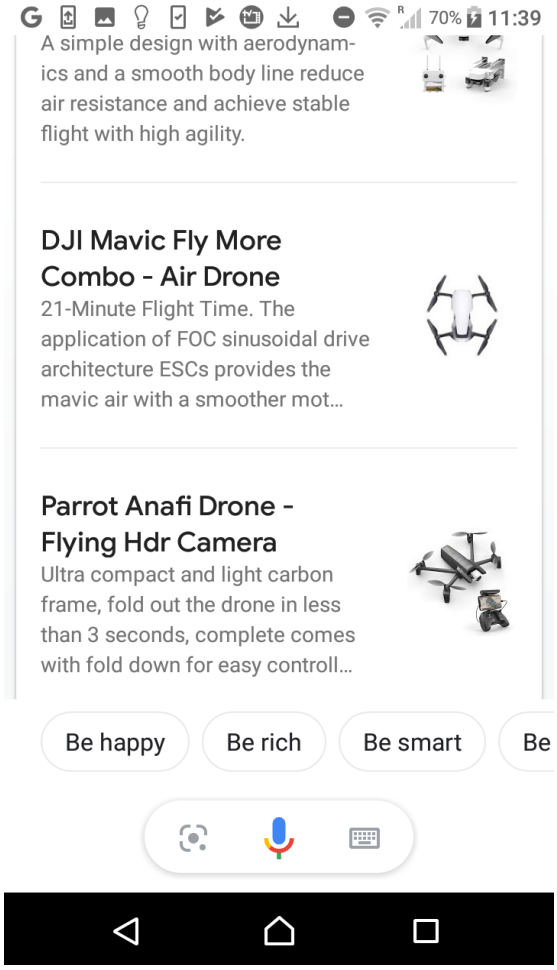
Add them to responses and add items. Again this question for you. What would you choose if presented with this option?



This is how the response would look in the Simulator:



And here is how they look on my device:



There is more that you could do. You could detect what kind of interface the user has and then provide the right kind of responses. If the user has only voice interface then you need to present drones in a different way. You could do this with a backend app. And that is exactly what I teach in one of my courses. What you could also do is detect what kind of interfaces user has available around and redirect them there. But that is outside the scope of this tutorial.

Let's see what we can do in Slack.

Slack - Rich messages

In Slack, you can add text, images, links, date pickers, fields, buttons and more. Messages are grouped by blocks and you can send several blocks in a message.

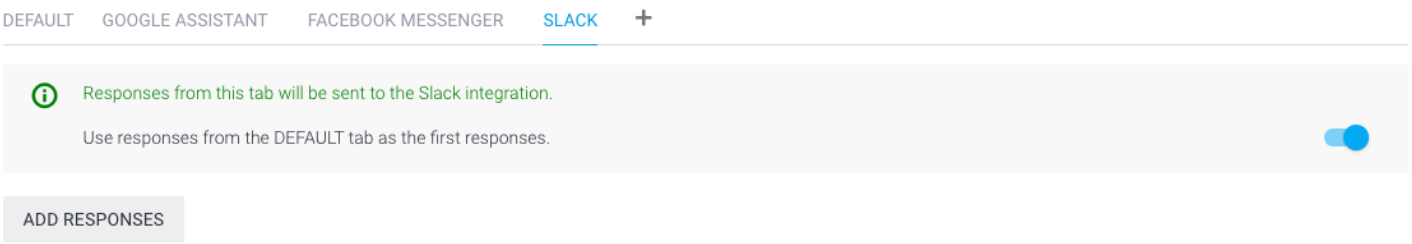
The documentation: https://api.slack.com/messaging/composing/layouts they offer is quite good.

But what I love the most is their interactive block kit builder:

https://api.slack.com/tools/block-kit-builder

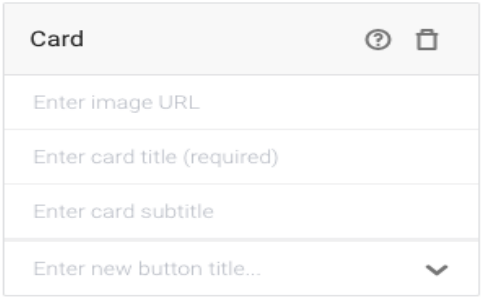
With this, you can play, add components, blocks, elements. And what it will do for you is to generate the JSON that needs to be sent as a message. Perfect! Job done easy.

Let's see what comes with DialogFlow out of the box. I will add a Slack tab to responses. Like this:

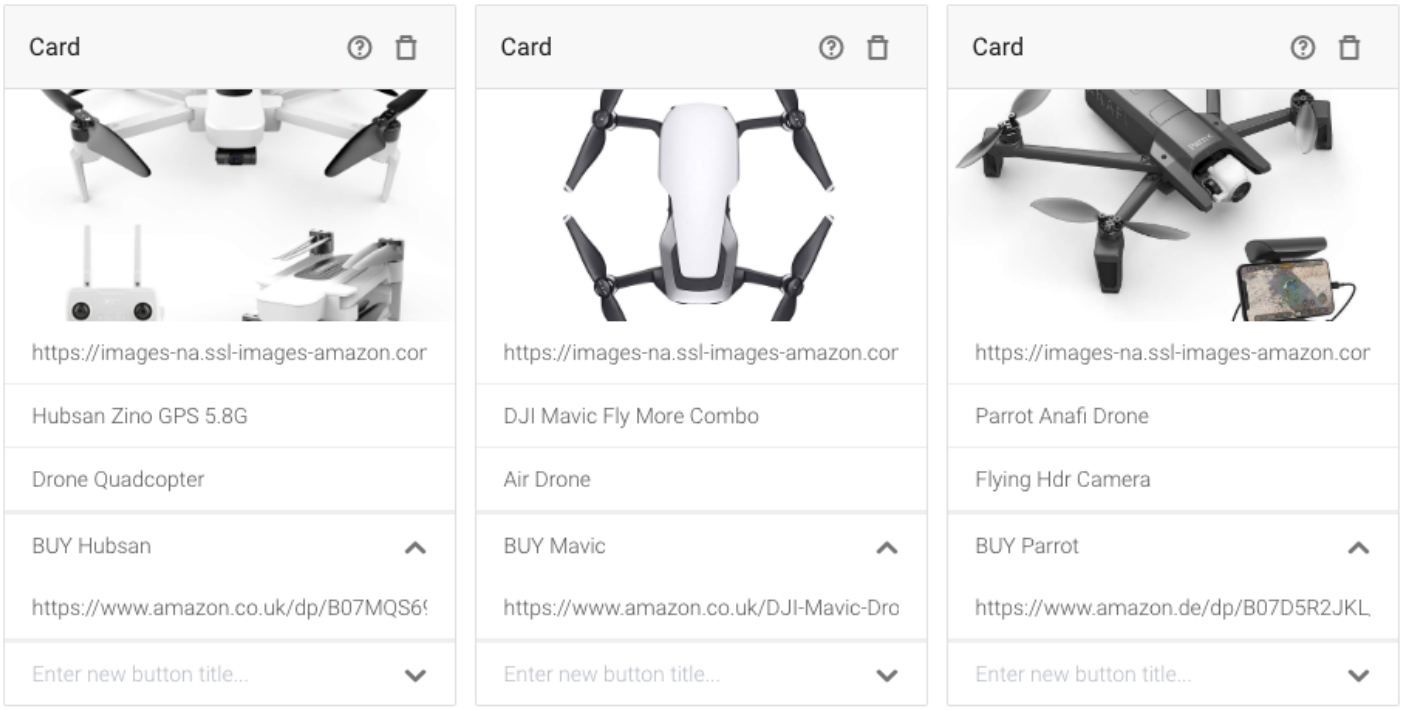


I will leave the default response from the DEFAULT tab.

To display drones I will use cards provided by DialogFlow:

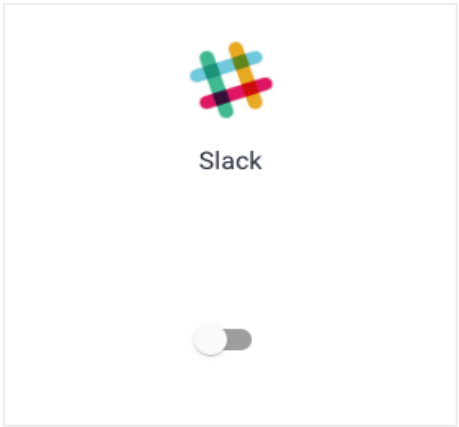


And I will add the three drones we also used with other platforms.



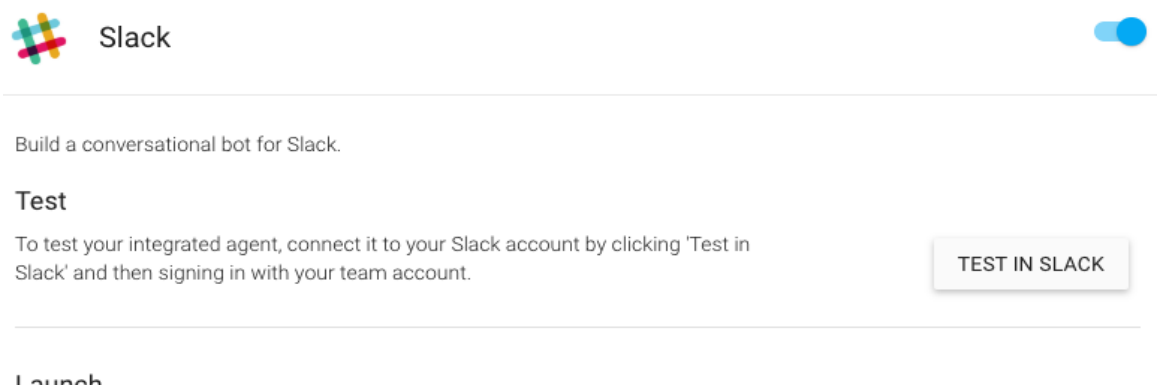
Now how will we test this on Slack? Pretty simple. Are you using slack and have a workspace? Then that is all you need. You will add a test Sack bot to that workspace where you can test the bot. And it will be done with one click. Super simple.

Go to Integrations tab and find Slack and click Enable

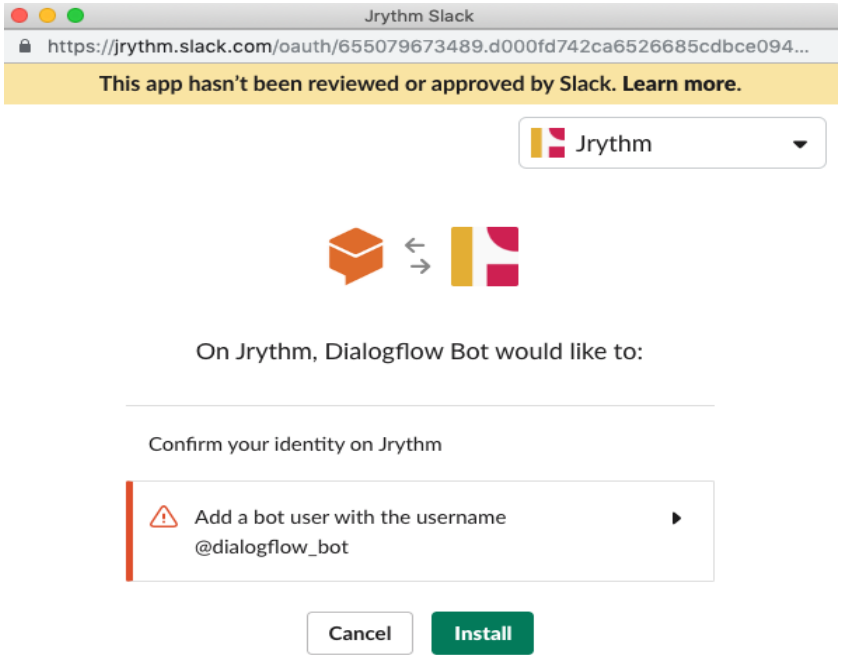


After you enable the Slack Integration, you will have an option to create an app and connect it to the workspace. What we will do is connect it just for the purpose of testing directly to DialogFlow. And we'll use the one-click step.

To do this just click button TEST IN SLACK. This one.

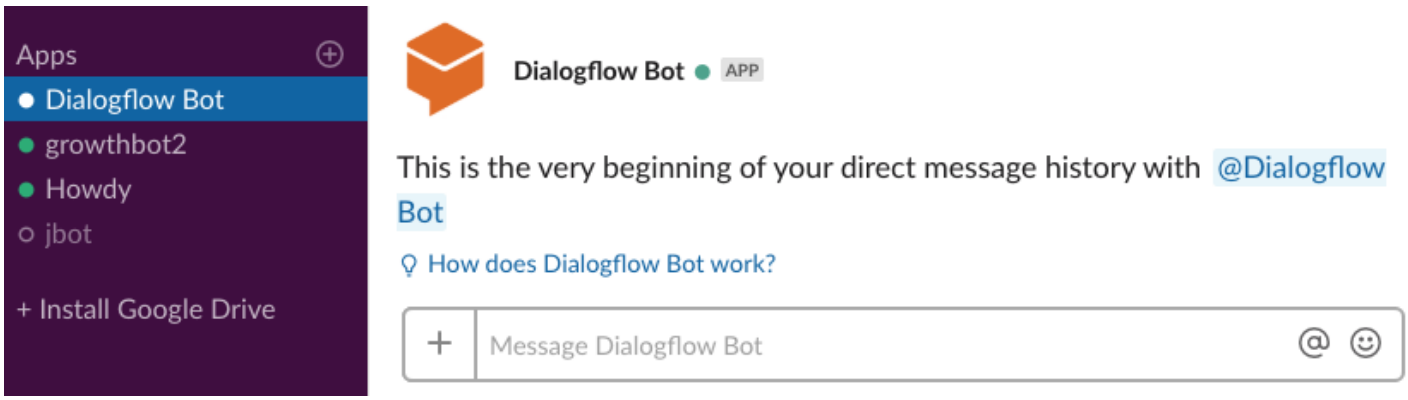


Now another interface will open. In here you must choose the workspace you want to connect the agent with. You choose a workspace in a select up here:

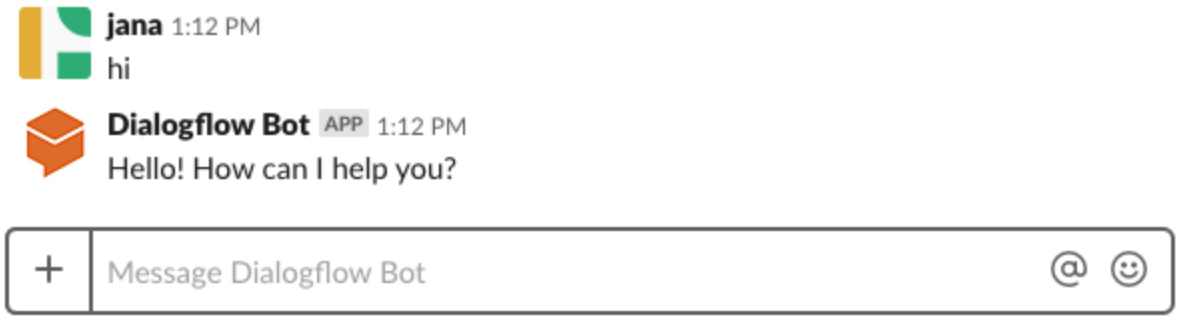


For me that is Jrythm. Now click install. And this is it!

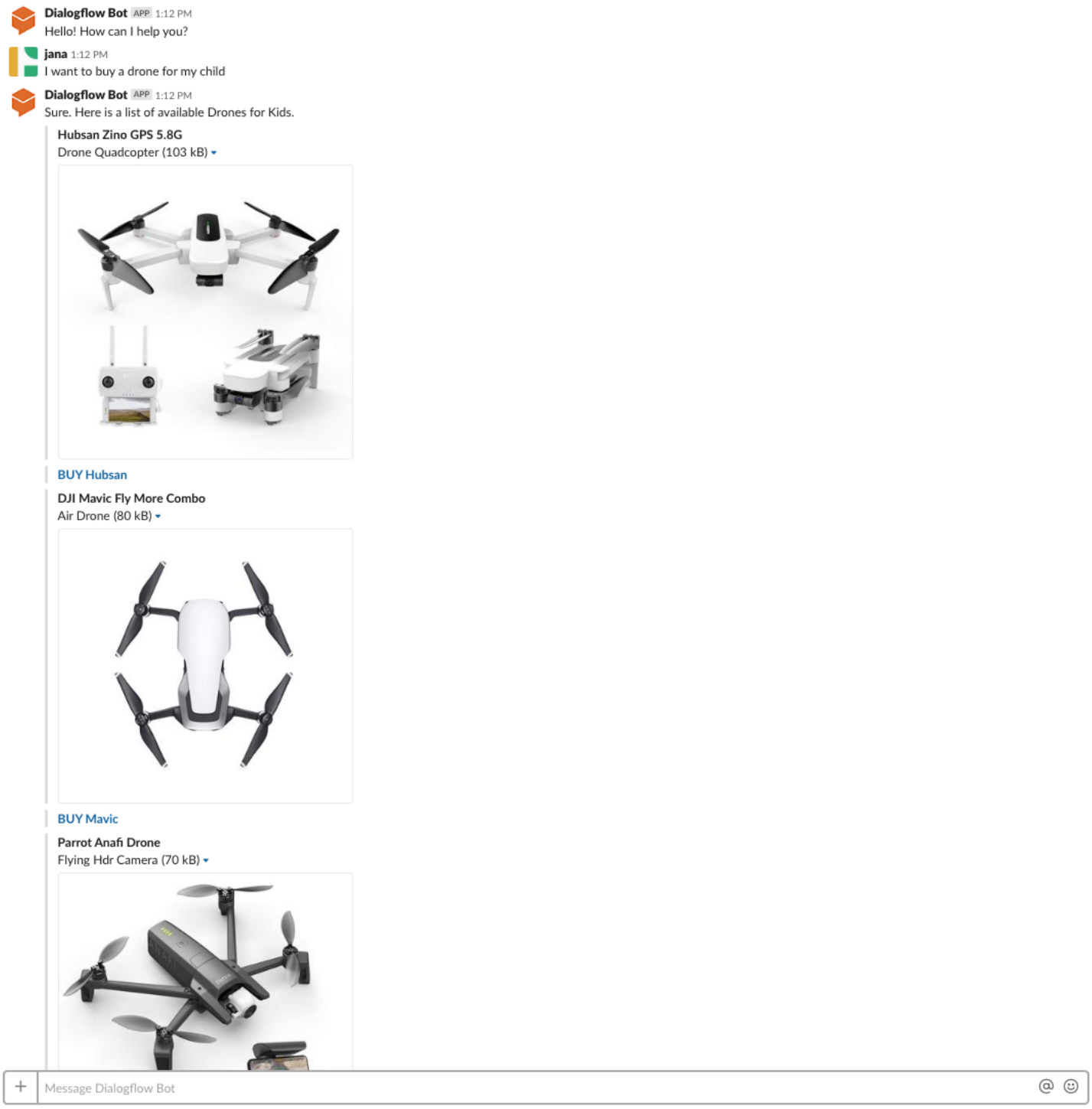
You should now see the bot under apps. It will be named Dialogflow Bot. And when you click on it, it will look like this:



We can greet it, to see if welcome intent will be triggered and all is connected as it should be:



And now we can call our drone action by looking for drones for kids. Here they go.



The bad news is that in the time of writing this tutorial, DialogFlow does not yet support sending blocks. It supports Message attachments. Message attachments are outdated. But I guess DialogFlow would make the transition soon. Till then here is a link to Message attachments. You would use them when sending the custom payload.

https://api.slack.com/docs/message-attachments

So far we have been working with static response Messages. All the drones have been added to DialogFlow. If the drone is no longer available we would have to come back to DialogFlow and update it. Not reasonable at all.

It would make much more sense to have a database or a service that all of your apps are connected to. Therefore if the drone is out of stock the same change is seen in the bot, on the website, and in the mobile app. Changes should not have to be duplicated.

And how do we do this? Well, we would somehow need to connect DialogFlow to the database or the web service. This can be done with the backend app. Or with what DialogFlow provides, that is Fulfilment. Let's see what fulfillment is next.

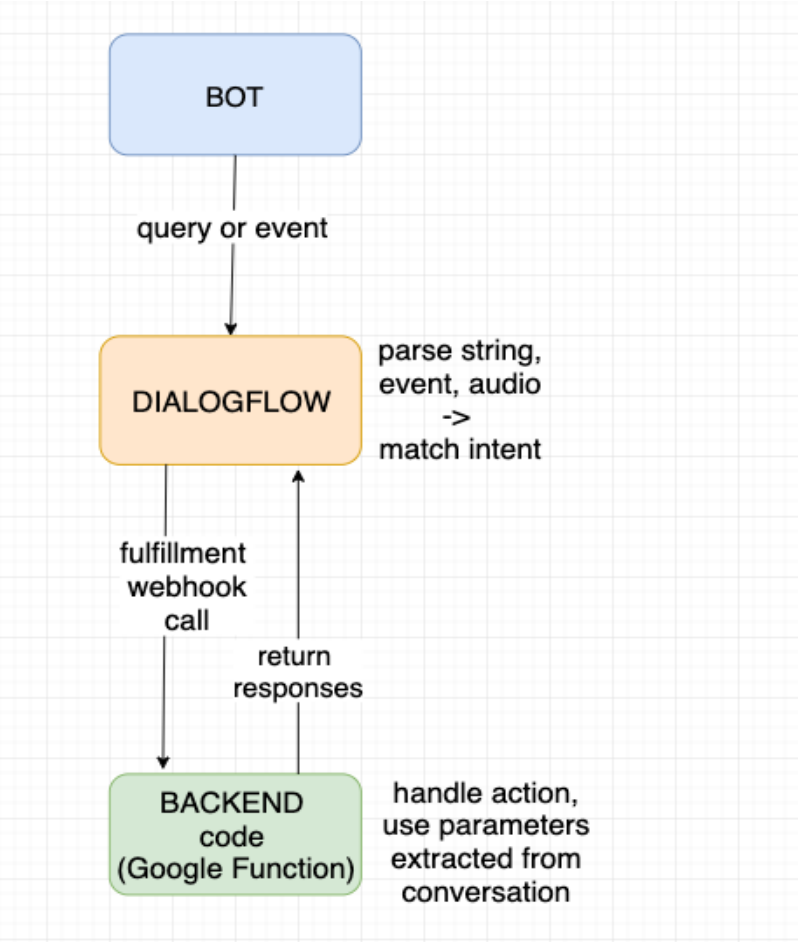
Fulfillment

We can enable a webhook that can be triggered when the specific intent is triggered. If we enable fulfillment call in the intent then this extra webhook will be triggered. DialogFlow calls this webhook call the fulfillment. To do this we need to:

- Have a deployed code that is reachable via the internet (default would be Google Function code)
- We need to set up the endpoint of this code as a fulfillment webhook
- Enable fulfillment call in the intent where we want this extra code to be called.

With other words, fulfillment is code that's your agent can call to execute business logic on an intent-by-intent basis. In the fulfillment code, you to use the information extracted by Dialogflow's natural language processing and you can generate dynamic responses or trigger actions on your back-end (in the webhook code).

And let me put this in an even more simple description:



To use fulfillment, you need to set up a webhook. A webhook is a web server endpoint that you create and host. In the DialogFlow we set this endpoint as the fulfillment. You can configure fulfillment for your agent from the **Fulfillment** page.

When the query is sent to DialogFlow, DialogFlow will parse the string, event or audio to match the intent. When the intent is matched it will see if the fulfillment is checked in that intent.

When it's matched, DialogFlow will make an HTTP POST request to your webhook with a JSON object containing information about the matched intent.

In other words, it will make a webhook call to the backend code we set as a fulfillment. By default, this code can be hosted on Google Function. But it can also be hosted anywhere else since this is a simple HTTP endpoint call.

The webhook can perform any required tasks. In this backend code, we can then handle the action of the intent that triggered the call, we can use the parameters that we extracted from the conversation or we can trigger some other business logic. For example, the webhook might use information from the request to look up a product in a database or place an order.

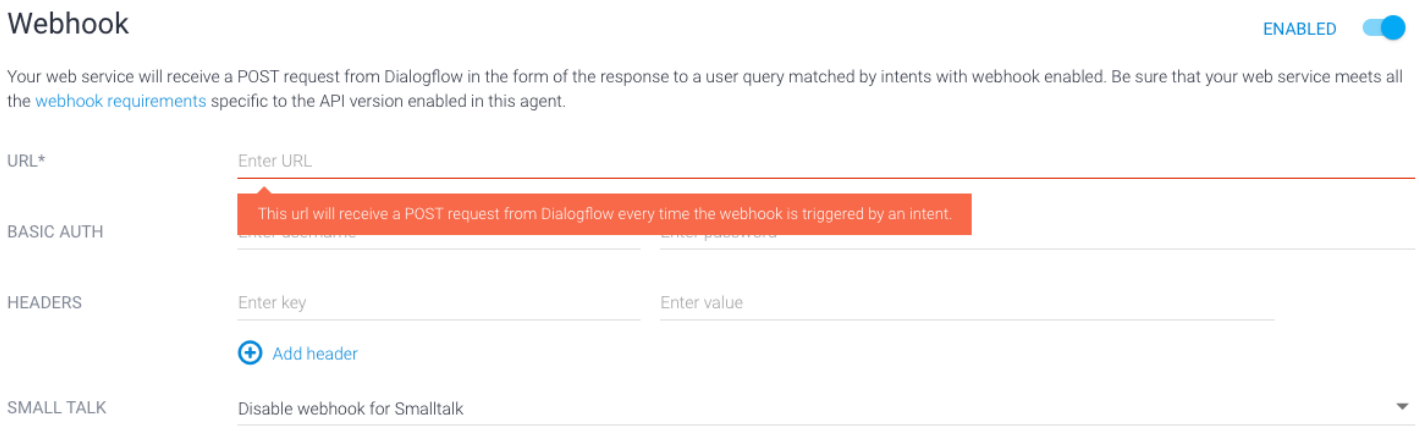
Finally, your webhook should respond back with instructions for what DialogFlow should do next.

You can enable fulfillment for any of your agent's intents.

Setting up fulfillment.

To set up a fulfillment go to Fulfillment tab.

You can host your Fulfillment code on google or you can host it anywhere else. If you host it anywhere else then the endpoint needs to meet all the webhook requirements. When you have your endpoint ready you can enable webhook and enter the authentication to use the API.



Authentication

It's important to secure your webhook to prevent unwanted, potentially malicious calls. Dialogflow supports two mechanisms for authentication:

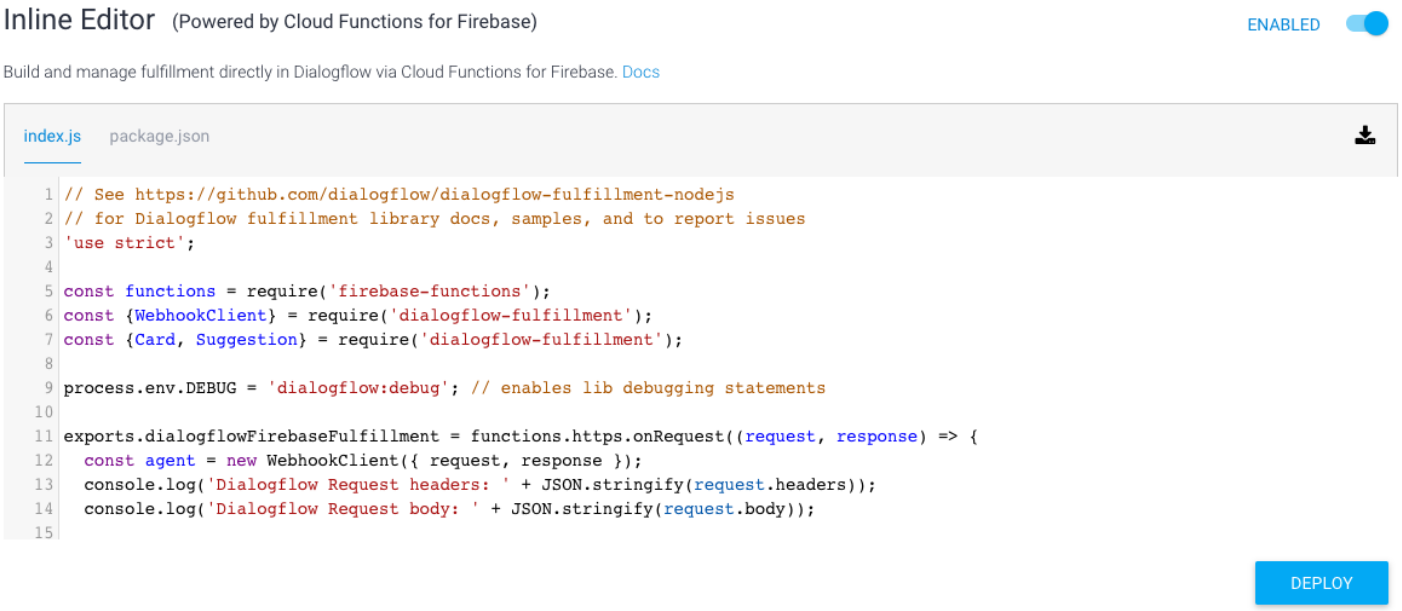
- Basic authentication with a login and password.
- Authentication with additional authentication headers.

If your webhook doesn't require any authentication, leave the authentication fields blank.

Note that your webhook must use HTTPS, and the URL must be publicly accessible.

Inline editor

You can build and deploy a webhook using your preferred development environment, or use the inline code editor. Inline editor if only for the code that is hosted on a Google Function.



The inline editor has all sorts of limitations. These are the annoying ones:

- You don't have all the benefits of a local IDE. It will be harder to debug and your efficiency will go down. I recommend using inline editor just for test and look round.
- The function must be named "dialogflowFirebaseFulfillment" if you want to deploy it via DialogFlow. Name of the function can be changed if you download the code, change the function name, and deploy through Firebase's CLI. But if you are not just testing things out, I strongly encourage you to do this anyway.
- If you modify your function outside of inline code editor, you can no longer use the editor to modify the code. Once you are out of the editor you are out.
- The inline editor only supports two files: index.js and package.json (modifying package.json will install any dependencies you specify upon deployment). But you cannot split your code into modules.
- You need to deploy the code before you can save or download it.

One other thing you need to do if you decide to host your fulfillment code on Google function. You will not be able to call any other web services besides Google's if you don't have payment enabled. If you will want to call web services outside Google you need to have payment for the Google project enabled.

In fulfillment, you can also handle missing parameters. That is called slot filling by DialogFlow.

Let's have a quick look at the libraries that are being used in the default code provided by Google.

The first one is firebase-functions: https://www.npmjs.com/package/firebase-functions

This is the library that lets you write code for Google functions hosted on Firebase.

A little intro on Google functions if you are not familiar with them. A very simplified introduction.

With Google Cloud Functions you can run backend code in response to events triggered by Firebase features and HTTPS requests. Your code is stored in Google's cloud and runs in a managed environment. There's no need to manage and scale your own servers. That is the beauty of serverless computing. Serverless means that Google is responsible for executing a piece of code and will dynamically allocate the needed resources, with the developer focusing just on the code and not on the servers. Perfect. Ok.

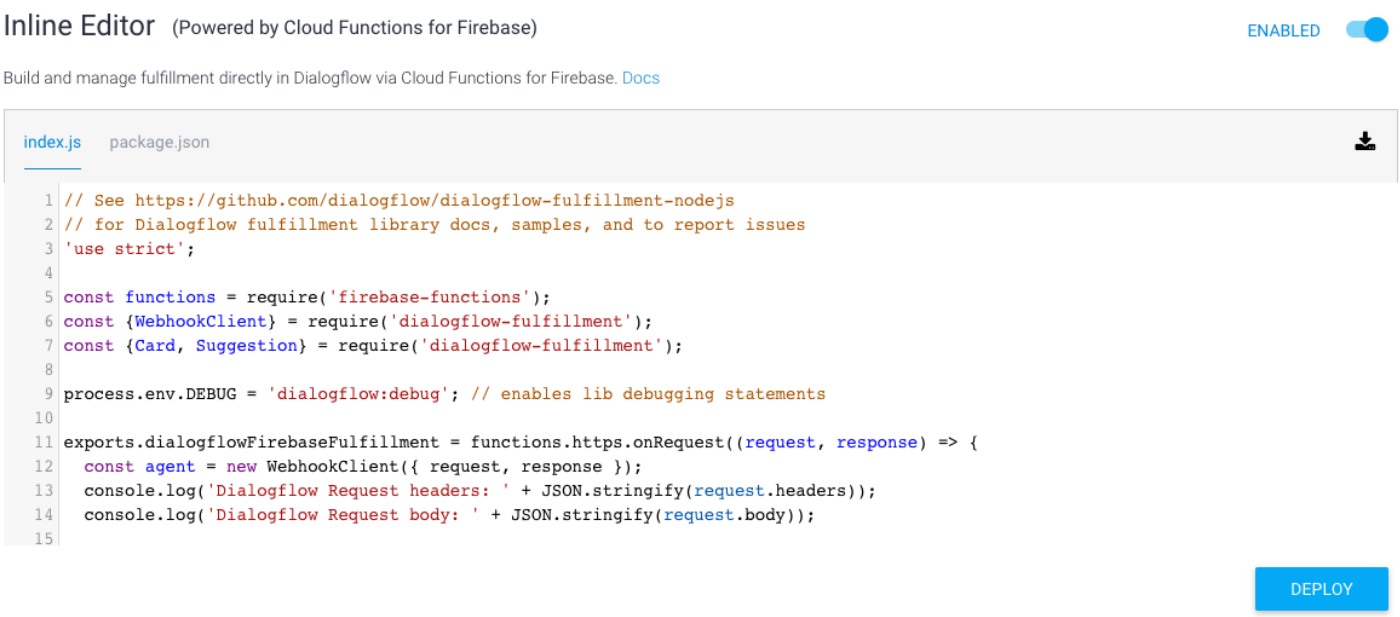
The next library used in the fulfillment code is dialogflow-fullfillment: https://www.npmjs.com/package/dialogflow-fulfillment

This is the library that will help you write code for the fulfillment. If you are creating an action for Google Assistant then this library will provide classes for the responses like Simple response, cards, images, suggestion chips, and payloads.

WebhookClient https://dialogflow.com/docs/reference/fulfillment-library/webhook-client#webhookclientoriginalrequest_object is a class that is responsible for communicating with DialogFlow. It has methods for sending responses, setting contexts, clearing contexts, ending a conversation. You can also access the properties of the recognized intent, like its name, parameters, contexts, action, responses defined in the DialogFlow interface, session, and query.

If you are creating an action for Google Assistant then these are the classes I strongly suggest you use when building a fulfillment code. Otherwise its better to create your own service that will provide JSON responses for DialogFlow. I will show you how we do this.

First, let's take a peek on how the code for Google Assistant looks like.



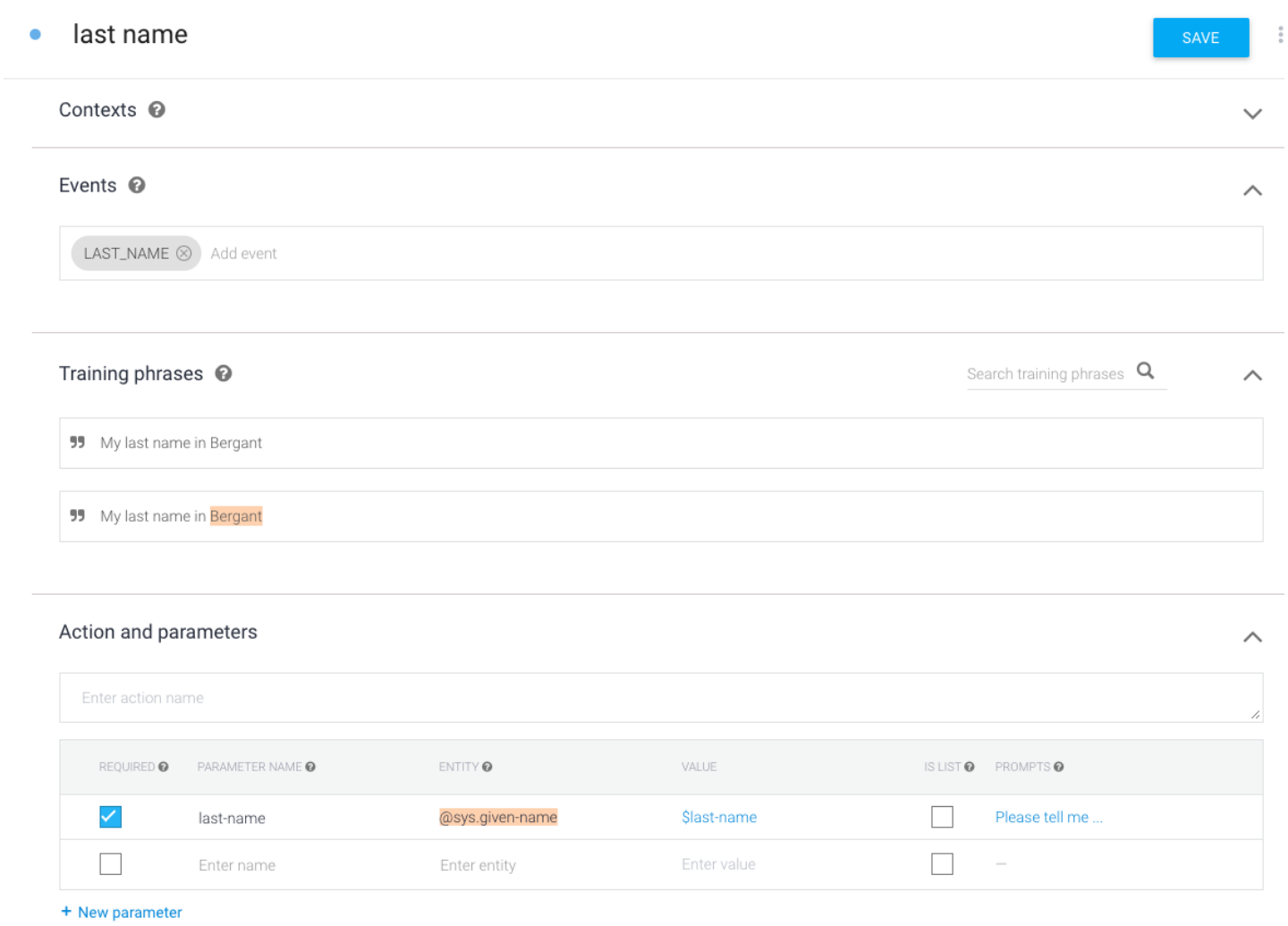
Upon receiving a request we initialize WebhookClient. WebhookClient receives the request and a response object. The new agent object now represents the intent that was recognized by the intent. We can access all other parameters describing the recognized intent like its name, parameters, contexts, messages defined.

If you scroll down you can see a intentMap that maps the function handler for the intent. It is matched with the intent's name.

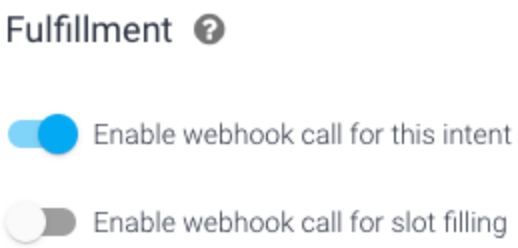
```
// // for a complete Dialogflow fulfillment library Actions on Google client library v2 integration samp.


// Run the proper function handler based on the matched Dialogflow intent name
let intentMap = new Map();
intentMap.set('Default Welcome Intent', welcome);
intentMap.set('Default Fallback Intent', fallback);
// intentMap.set('your intent name here', yourFunctionHandler);
// intentMap.set('your intent name here', googleAssistantHandler);
agent.handleRequest(intentMap);
});
```

What does that mean? Simple! When we create a new intent and enable fulfillment for it. Then we can in the code add a new line here, specifying which method will handle the request that will come from DialogFlow after the intent will be matched. In the method we will run the code, maybe write to a database, do some calculations, then return a response, that will go back to DialogFlow. And DialogFlow will return that response to the client that sends a query to it.

For example. We can add a new intent called the last name. This intent can be triggered by query or by the event. And it will collect a parameter last-name from the user like this:



And we will enable the fulfillment for this specific intent.



Now we can catch this intent in the fulfillment code by adding this line to it. The method name could be different. You can name it whatever you like. But!! The name of intent must be the same. And it is case sensitive. 'Last name' is not the same as 'last name'. Therefore be careful.

```
let intentMap = new Map();
  intentMap.set('Default Welcome Intent', welcome);
  intentMap.set('Default Fallback Intent', fallback);
  intentMap.set('last name', getLastName);

  agent.handleRequest(intentMap);
});
```

And now we can add a method and in the method give a response back to DialogFlow.

```
function getLastName(agent) {
  agent.add(`Thank you`);
}
```

What we can also do is read the parameter we got in the intent:

```
function getLastName(agent) {
  console.log(agent.contexts[0].parameters['last-name']);
  agent.add(`Thank you`);
}
```

What we could do is save this information to a database. Like this:

```
function getLastName(agent) {
  console.log();
  let new_user = {
      last_name: agent.contexts[0].parameters['last-name']
  };
  let sessionData = db.collection('users').add(new_user).then(ref => {
    return {
      id: ref.id,
      user: new_user
    };

  });
  agent.add(`Thank you`);
}
```

If you want to know more about Google Assistant actions development I would invite you to see my course on Udemy.

Fulfillment example with dynamic responses from a database

In this article I show you how to read information from a database and use it to dynamically generate responses for DialogFlow fulfillment.

Follow-up intents

Hello and welcome back. We already know a little bit about intents. We know how we use intents to recognize what the user wants when he or she says or text something. With intents, we map user input into action, parameters and responses. In this article, I want to introduce you to DialogFlow's feature called follow up intents. Follow up intents help us create natural dialogs faster. It's like a short cut to many responses we use in our everyday speech. Wanna know more? Read the article!

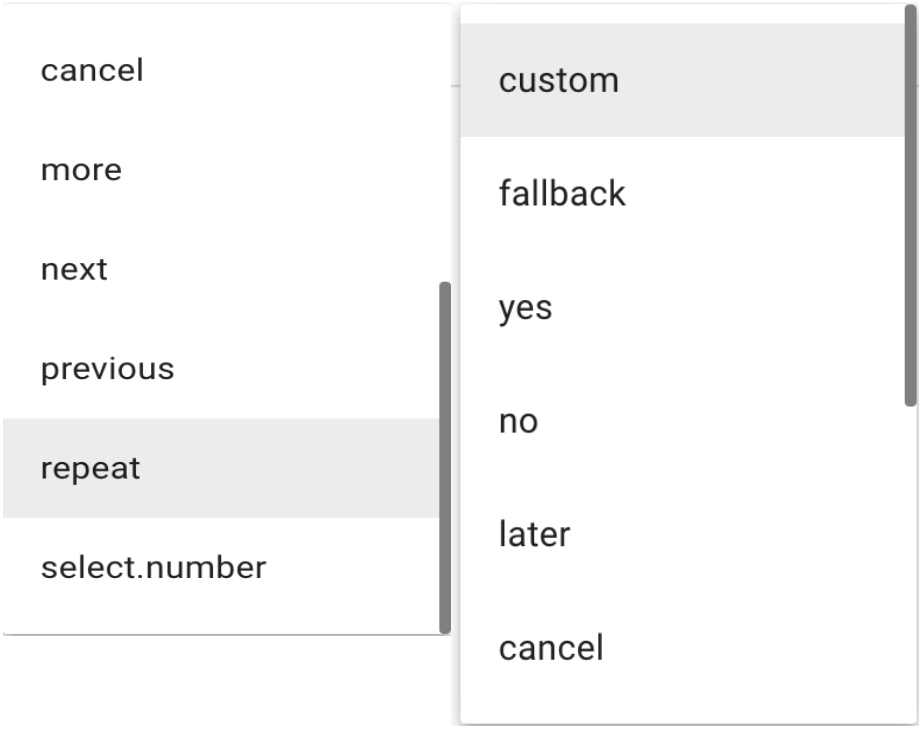
Well, DialogFlow is not just a natural language understanding tool. It's the whole **system that helps you build a natural conversation**. And in the natural conversation, we often have a situation when we reply to a question or a sentence something like yes or no or later one tell me more, cancel, next, repeat... And because this is so common DialogFlow introduced a feature called follow up intents.

Well, for example, because no can be said in many different ways. Like no, never, not really, not interested… and more. So many short phrases that basically mean the same thing. And without follow up intents we would need to think of them by ourselves. DialogFlow enables us to easily create intents that are common responses in a natural language. Responses like yes, no, next, previous, later, repeat and so on… Well yes, DialogFlow made it easy for us to do it. Let me show you how. Go to DialogFlow. And open the list of intents. If you hover over intents you'll see three options. The first one is the one we're interested in. The add follow up intent.

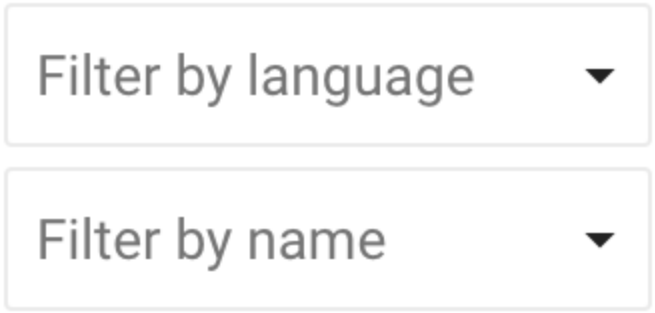
And hover over 'want type of drone' intent. Here is where we'll add a follow-up intent.

And now you'll be presented with a different type of follow up intents. From custom, fallback, yes, no, later, cancel, then more, next, previous, repeat and select number.

Here I have a page explaining all the follow-up intents: . Here you see all the follow-up intents.



You can filter them by language and name.



Let’s go through them. I’ll show you the English version.

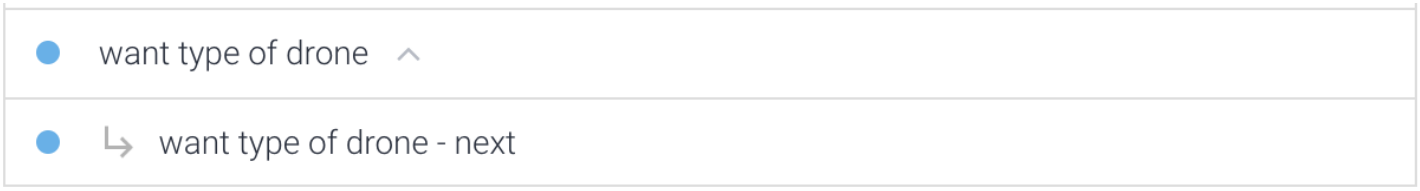
Language	Name	Examples
English (en)	yes	"yes" "do it" "sure" "exactly" "confirm" "of course" "sounds good" "that's correct" "I don't mind" "I agree"
English (en)	no	"no" "don't do it" "definitely not" "not really" "thanks but no" "not interested" "I don't think so" "I disagree" "I don't want that"
English (en)	later	"later" "not yet" "ask me later" "not " "next time" "I said later" "some other time" "do it later" "not at the moment" "not at this time"
English (en)	cancel	"cancel" "stop" "dismiss" "skip" "forget that" "stop it" "never mind" "do nothing" "just forget about it" "cancel that"
English (en)	more	"more" "more results" "anything else" "other results" "what else" "are there more" "tell me more" "show me more" "more information"
English (en)	next	"next" "next page" "go forward" "show me next" "following result" "switch to the next" "go to the next page" "read the next results" "show me the next page" "show me the following results"
English (en)	previous	"back" "previous" "go back" "previous page" "previous results" "return to the previous" "return to previous page" "go to the previous page" "go back to previous results" "read out the previous results"
English (en)	repeat	"repeat" "repeat it" "come again" "do it again" "say it again" "say the same again" "please repeat that" "repeat these results" "could you repeat that" "repeat what you've just said"
English (en)	select.number	"second" "I choose number 3" "see 1 2 3 4 and 5" "number 4 5 7 and 8" "select the first one" "show me the second one" "choose 1st and 2nd one" "choose numbers 1 2 3 and 5" "the 1st one and the 2nd one" "I select number 2 and number 5"

First one is custom. This one you would need to define yourself. You would define all possible answers, but the DialogFlow would set the right context. I'll introduce the concept of contexts soon. For now, let me just tell you that contexts glue intents together. Like in a conversation between humans, we have a context that we speak about. A line of thoughts we follow. So, context tells what intents belong together. If we say yes, we want to know what we've said yes to. Right? We'll talk more about this in the next article.

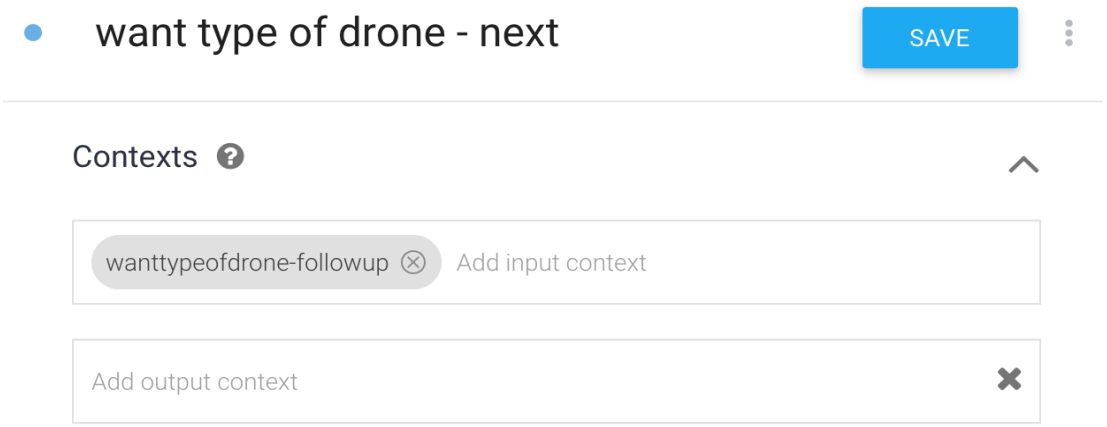
Ok, the next type of follow up intents is fallback. This is the intent that gets triggered if no other follow-up expression is triggered. Then a yes. This is the one we'll use. You can see on the right all the possible Training Phrases, that is user queries that trigger this intent. Yes, this can be everything from yes, do it, sure, exactly and so on. Ok.

Next one is no. The same as with yes, no can be said in many different ways, like no, don't do it and so on. The same thing with later, cancel, more, next, previous and repeat. The last one is for selecting a number, one or many. Like second, see 1,2, 3 and so on. Ok.

Let's see this in action. Go back to DialogFlow. And choose next. A new intent will be created. The intent for the next three suggested drones.

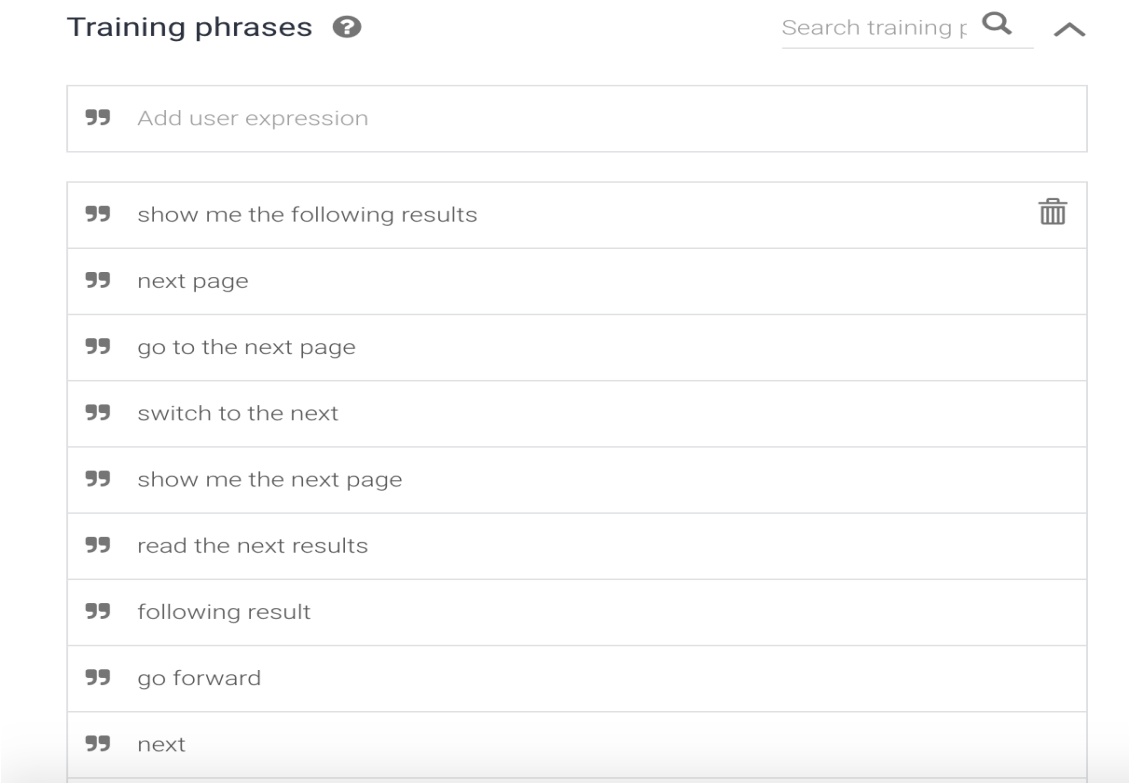


And this intent will have a context predefined for us. It has the input context. This same context will be defined as an output context in the wanttypeofdrone intent.

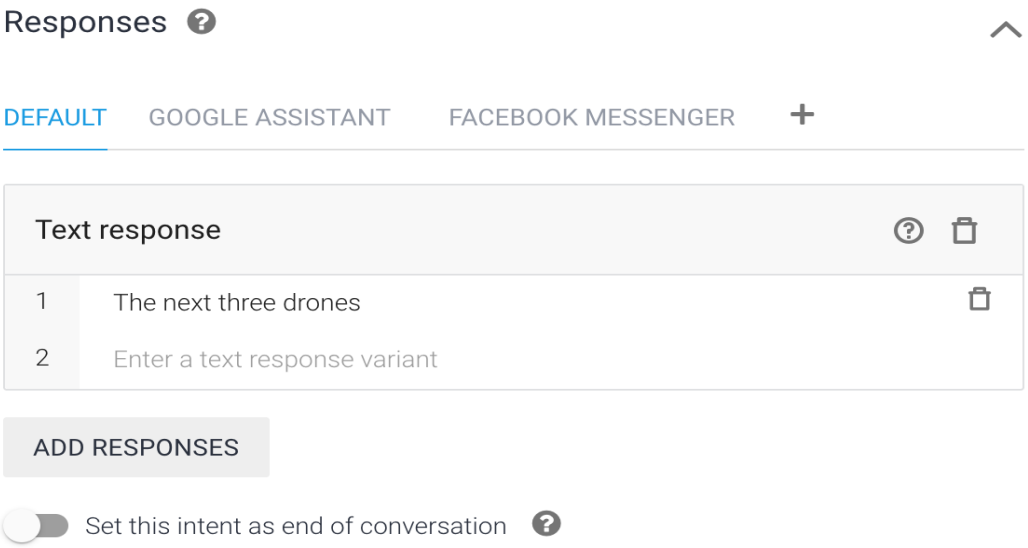


This way the intents are glued together. Via the output and input context. The previous intent has an output context and the next intent has the same context as an input context.

Ok, what we also have is predefined Training Phrases.



What I'll do is add a response to this intent. And for now, I'll just say The next three drones.



We can now save the intent. We have now the yes follow up intent. In the same manner, we can add a no follow up intent. Go to all intents.

So what have we learned about follow up intents?

Follow-up intents provide a simple way to shape a conversation without having to create and manage contexts manually. These special intents are nested under their parent intent and are designed to handle preset replies from the user, like "yes", "no", "cancel", or "next". You can also customize follow-up intents by choosing the **Custom** option.

When you create a follow-up intent, an output context is added to the parent intent and an input context of the same name is added to the child intent. This means that the follow-up intent is matched only when the parent intent is matched in the previous conversational turn.

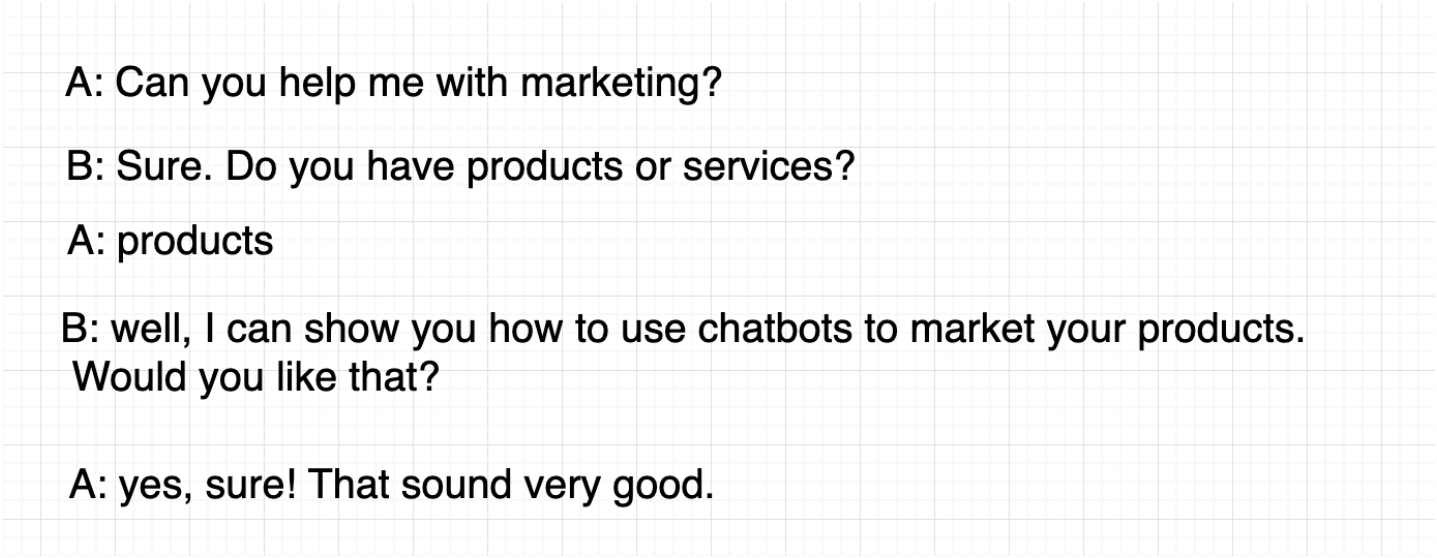
So this is it.

In this article, we have added two follow up intents. Follow up intents are predefined intents DialogFlow offers. Follow up intent are some commonly used responses people use in natural conversation. Responses like yes, no, cancel, more and so on. In the next article, you'll get to know more about contexts in intents.

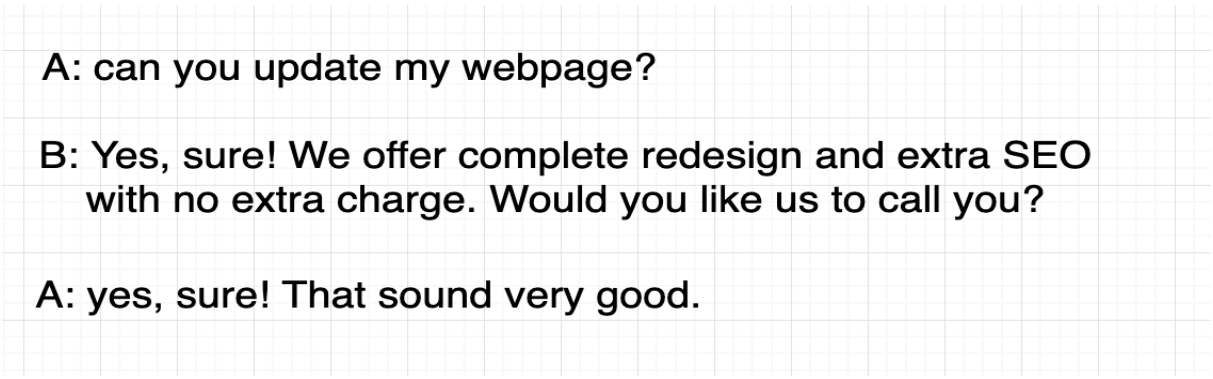
Contexts

Hello and welcome back. In the previous article, you've learned about follow up intents. Follow up intents are intents for common responses in a conversation like for example: show me more, next, yes, no, cancel and so on. These intents are predefined for us and we can use them with intents. But DialogFlow needs to know somehow that one intent is supposed to follow another intent. Let me show you how.

Here we have two conversations. These are both a conversation with a bot from a marketing agency. The first conversation is about marketing products. Client A needs help with marketing products and bot B offers them to build a chatbot for them.



And in the second conversation: client A needs help with updating a webpage. Bot B offers extra SEO services along with webpage update.



And now, let's look at something. Reply from client A looks exactly the same in both conversations.

And if the intent does not know what the previous conversation was about, then the agency would not know what the client wants. That could cause a lot of trouble!

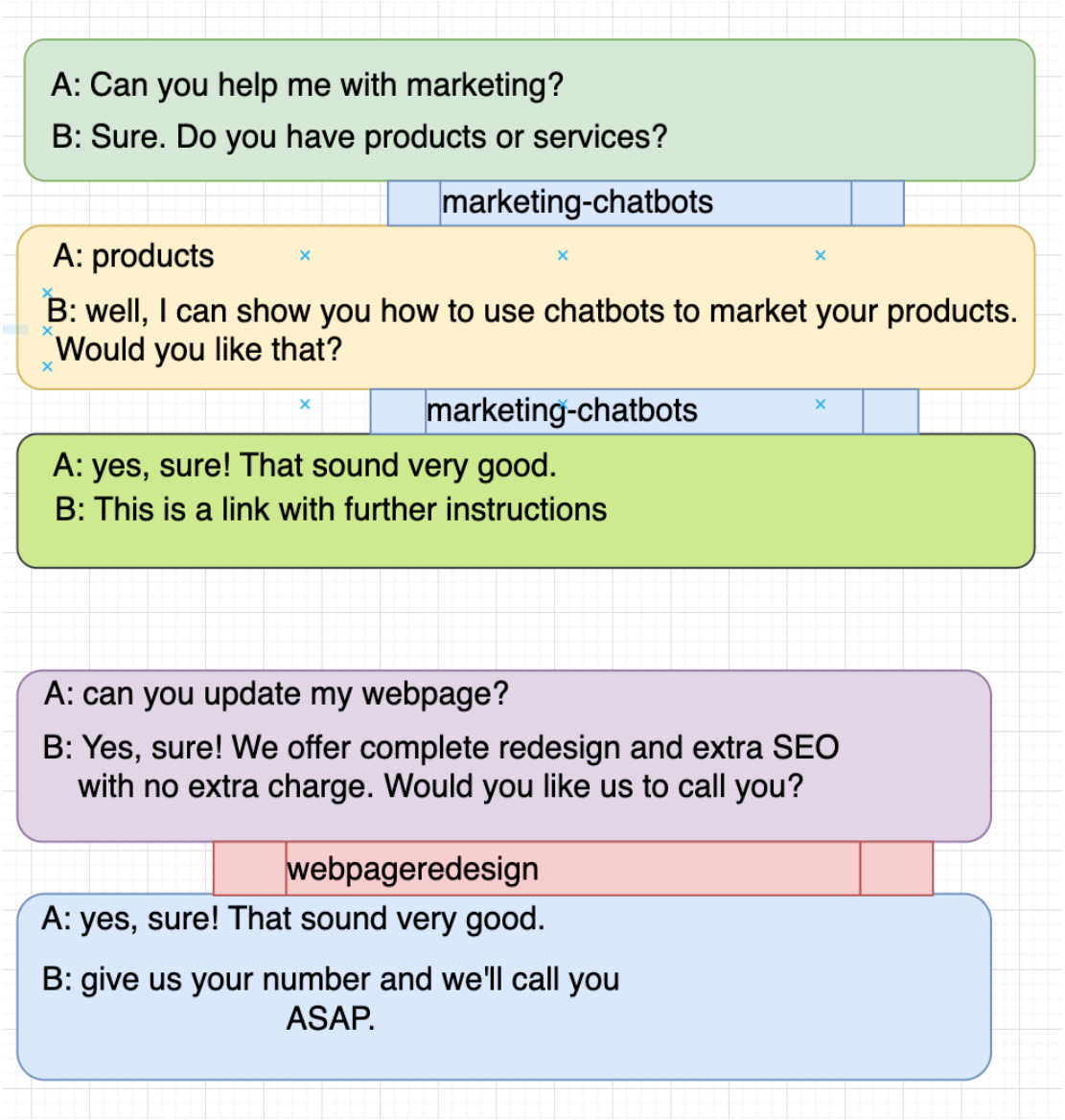
That is why DialogFlow uses contexts.

Contexts tells us in what context the intent can be used.

To each intent, we can add an input context and an output context.

Input context is about the previous conversation.

And the **output context is what comes after this intent**. That is the context that all the following intents use as an input context. In the example we looked at, we would set it like this.



In the first conversation, we would have three intents. And the first one would have marketing-chatbots output context. The second would have marketing-chatbots as an input context and also as an output context. Then the last one would have marketing-chatbots as an input context. In the second conversation, we would add webpage redesign context as an output and then input context. Then, in this case, we would know exactly what the user wanted when he or she said Yes, sure! That sounds very good.

So, what are contexts?

> With contexts, we can carry information from an intent to an intent. They also show the current state of the conversation.

Simply put, we remember parameters from previous intent and we know what we were talking about.

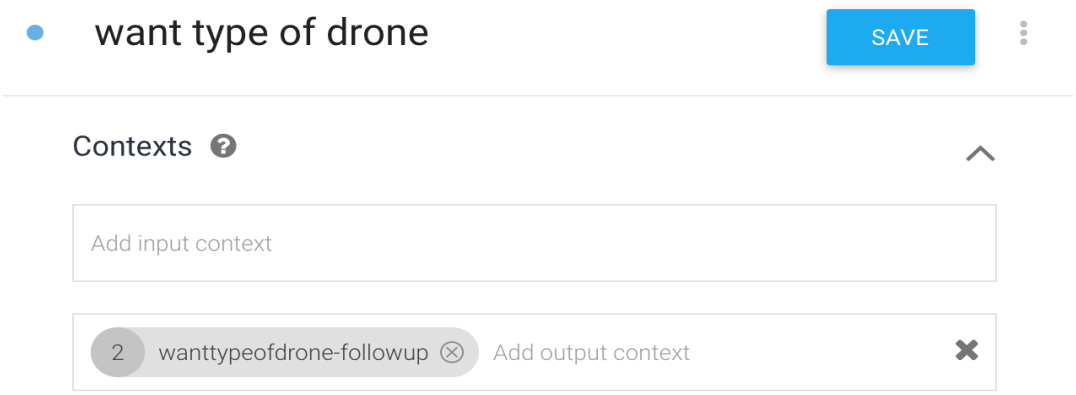
You can use combinations of input and output contexts to control the conversational path the user takes through your dialog.

Contexts let you control conversation flows by letting you define specific states that a conversation must be in for an intent to match. For example, to end the purchase we must know we are talking about buying, we must know what items user selected, we must gather all the delivery and payment data. We cannot just trigger the purchase successful intent out of the blue. That is why we add to the intent of the input contexts. And if the dialog has not these contexts then the intent cannot be triggered. So, DialogFlow will only match that intent if the context is active in a conversation.

With contexts, we can create dialogues, make branches, just by adding the right input and output contexts.

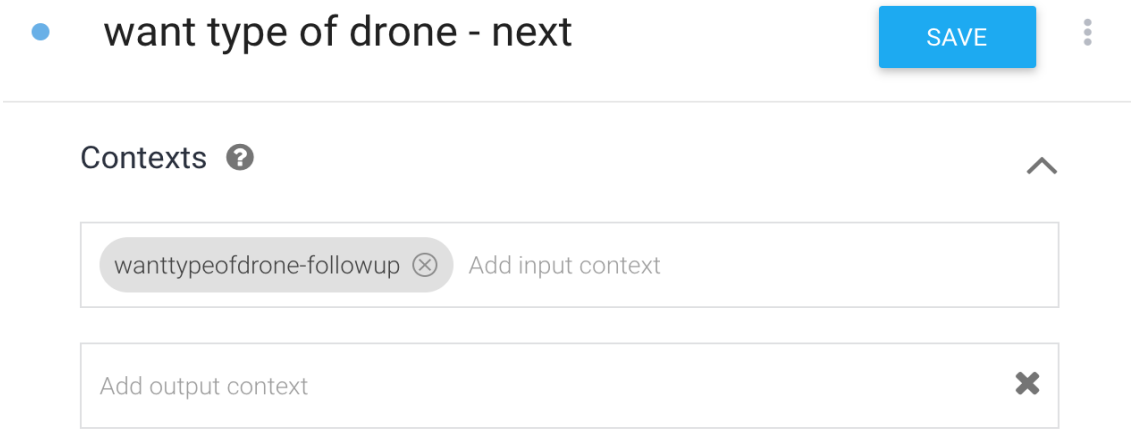
When we created follow-up intents, contexts were already set up for us.

Let's look at DialogFlow. Go to intents and open 'want type of drone'. Here in the 'want type of drone' an output context was added.



Yes? Go back to all intents. Ok

And open a follow-up intent. As you can see here also a context was added, but this time it is set as an input context.



This is how the intents are glued together. Yes, contexts glue intents together. Go back to all intents. Ok, again.

You don't have to put them in a hierarchy like this. You could add a new intent and just set up the right input context and it would know that this intent is after the intent that has output context. So, they don't have to be in a tree view like this.

Ok.

When you create a new context you should:

- use alphanumeric names.
- use "-" or "_" instead of spaces.
- Know that context names are not case sensitive. It's the same if you say buyin_drones or if you say Buying_Drones. Is not case sensitive.

With input and output context you can activate and deactivate contexts and can control the flow of your conversation.

Input context

> By setting an input context you tell DialogFlow to match the intent only if the user text is a close match and if the context is active in the conversation.

Important!

1. If we want the intent to me matched when having an input context then the request should have that context live. Of course, request can also have other contexts living but for the intent to be matched it should also have the one set up as an input context of the intent.
2. If the intent has more then one input context then the request must have all the context that the intent has in order for the intent to be matched.
3. Queries without any contexts can match any intent. Even the ones that have output contexts.

Output contexts

> Output context tells DialogFlow that only intents with this specific input context can be activated after a context is activated.

And when you add an output intent you tell DialogFlow to activate a context if it is not already active or to maintain it after the intent is matched.

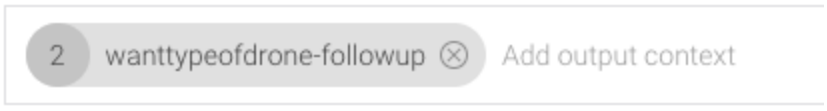
For example, say you have two intents that have the training phrase "Show me a latest offer", but one displays woman shoes and the other one man's shoes. You want to control which intent is matched depending on the user information. That is in another intent we ask the user if they look for woman or man shoes and activates a "woman shoes" or "man shoes" output context. Each "Show me latest offer" intent has the right input context so that DialogFlow matches the right intent.

Passing parameters with contexts

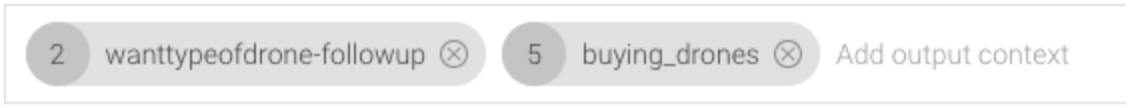
Contexts can also be used when you want to pass information captured from the user to future responses. And in our case, the information about what kind of shoes the user is looking for can be passed along in the output context. And you can add more than one output context to an intent to create branches.

The lifespan of a context

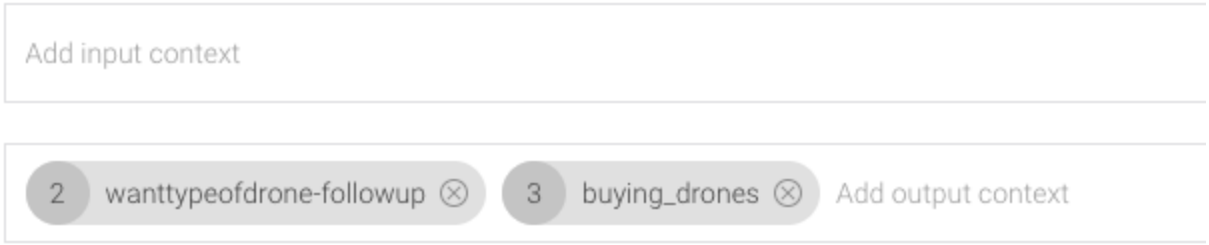
Output context won't stay active for all the rest of the conversation. By default, output contexts expire after either five requests or 20 minutes after its corresponding intent is matched. If the same output context is included in another intent, the context resets the counter and clock to five requests and 20 minutes. If you want the context to expire after a certain amount of the request you can change it's lifespan. Lifespan is the number of requests to be made to DialogFlow before one context is expired. And it is this number.



In the follow-up intents the default value would be two. But when you create a new output context then the default value will be 5. Like this:



This can be of course changed. Like this:



If the "woman shoes" context has a lifespan of two and the context's lifespan isn't reset by its related output, the agent could only observe "woman shoes" for two conversational turns, as shown in the following example:

- **"Are you looking for woman shoes?"** - Context is attached to this intent.
- "Yes, I do." - Context is set and activated.
- **"Do you want to see the latest offers?"** - Turn 1.
- "Sure!" - Turn 1.
- **"Here's a list of new arrivals."** - Turn 2 and context is removed.

Let's recap. What are the contexts?

Contexts let you control conversation flows. Contexts represent the current state of the user's request and allow your agent to carry information from one intent to another.

With contexts you can:

- create dialogues,
- make branches
- Control order of intent matching.
- Creating different outcomes for intents with the same training phrases.

Read more about the contexts:

https://cloud.google.com/dialogflow-enterprise/docs/contexts-overview

DialogFlow events

> Events allow you to invoke intents based on something that has happened instead of what a user communicates with text.

Events allow you to invoke intents based on something that has happened instead of what a user communicates. Dialogflow supports events from several platforms (like Google Assistant, Slack, Facebook and more) based on actions users take on those platforms.

DialogFlow predefines some events for us. And we can also create custom events.

Let's first take a look at some predefined events: https://dialogflow.com/docs/events/platform-events

Welcome events are triggered when a user starts a conversation with your action, bot, skill, or interface. When you create a new agent, a Default Welcome Intent is automatically added. Such intents have a pre-defined WELCOME event and text responses.

A WELCOME event is a generic event for supported one-click integrations. It's a short way to set all welcome events. For example, if we want to send a welcome message when the user click on the get started button when first interacting with your Messenger bot then you should use FACEBOOK_WELCOME event. FACEBOOK_WELCOME is triggered when a user starts a conversation with your Facebook Messenger bot. That is clicks on the Get started button. Let me show you how.

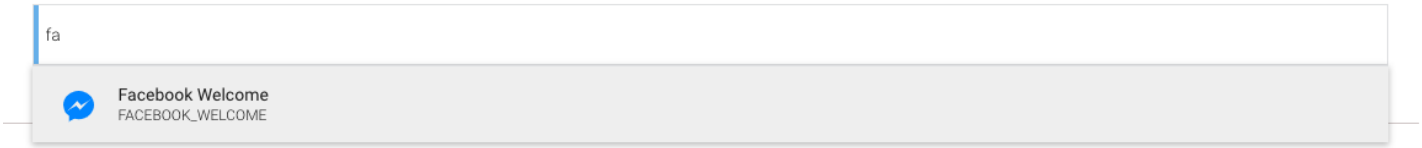
Facebook Welcome event

For this to work Facebook Messenger should already be integrated with DialogFlow. This is the article showing you how to do the integration.

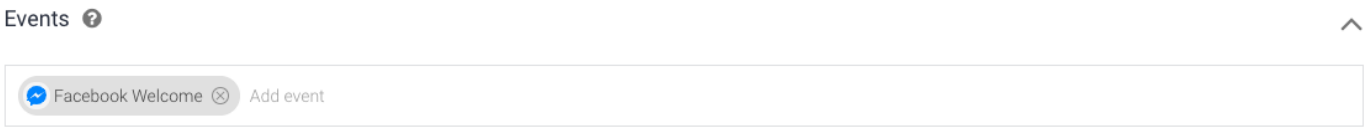
I will create a new intent that will handle greetings just for users that are starting a conversation on Messenger. I could also use the Default Welcome intent that already has a Generic Welcome event that works for any platform. But I want the greeting for Messenger users to be different.

Therefore I've created a new intent called Facebook Welcome. Like this.

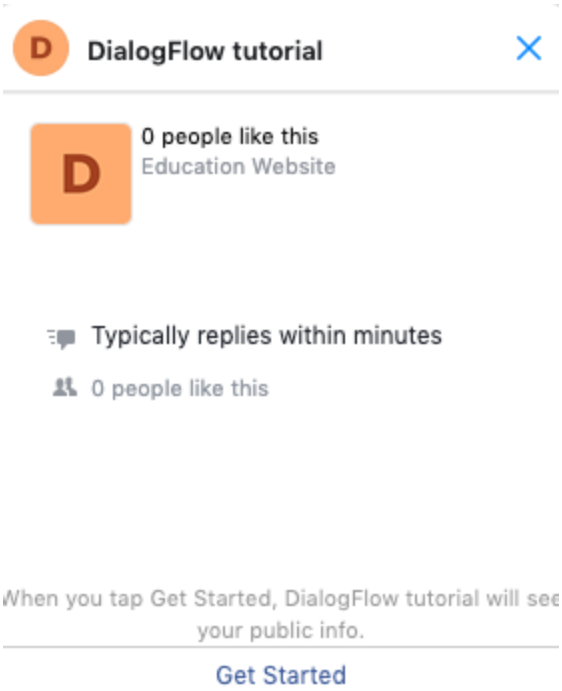
I will add a facebook welcome event. Like this:



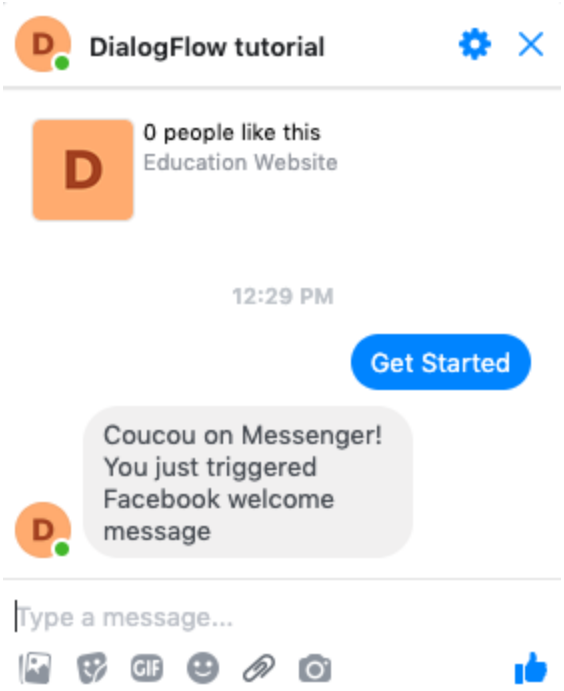
When added it should look like this:



And now I can only add a simple greeting to be sent when the user clicks the Get started button:



After the button is clicked, the event in DialogFlow will be triggered and response will be sent from DialogFlow responses. Like this:



Super simple, was it not?

Custom events

We can add custom events to intents. We can name them any way we want. We would use them when some events might happen that are not text and voice input.

Like for example a button click, or actions in a game by other players or no response from the user after some time and so on.

These events can be triggered through Dialogflow fulfillment or the detect intent API. We have talked about fulfillment or DialogFlow API here.

Integrations

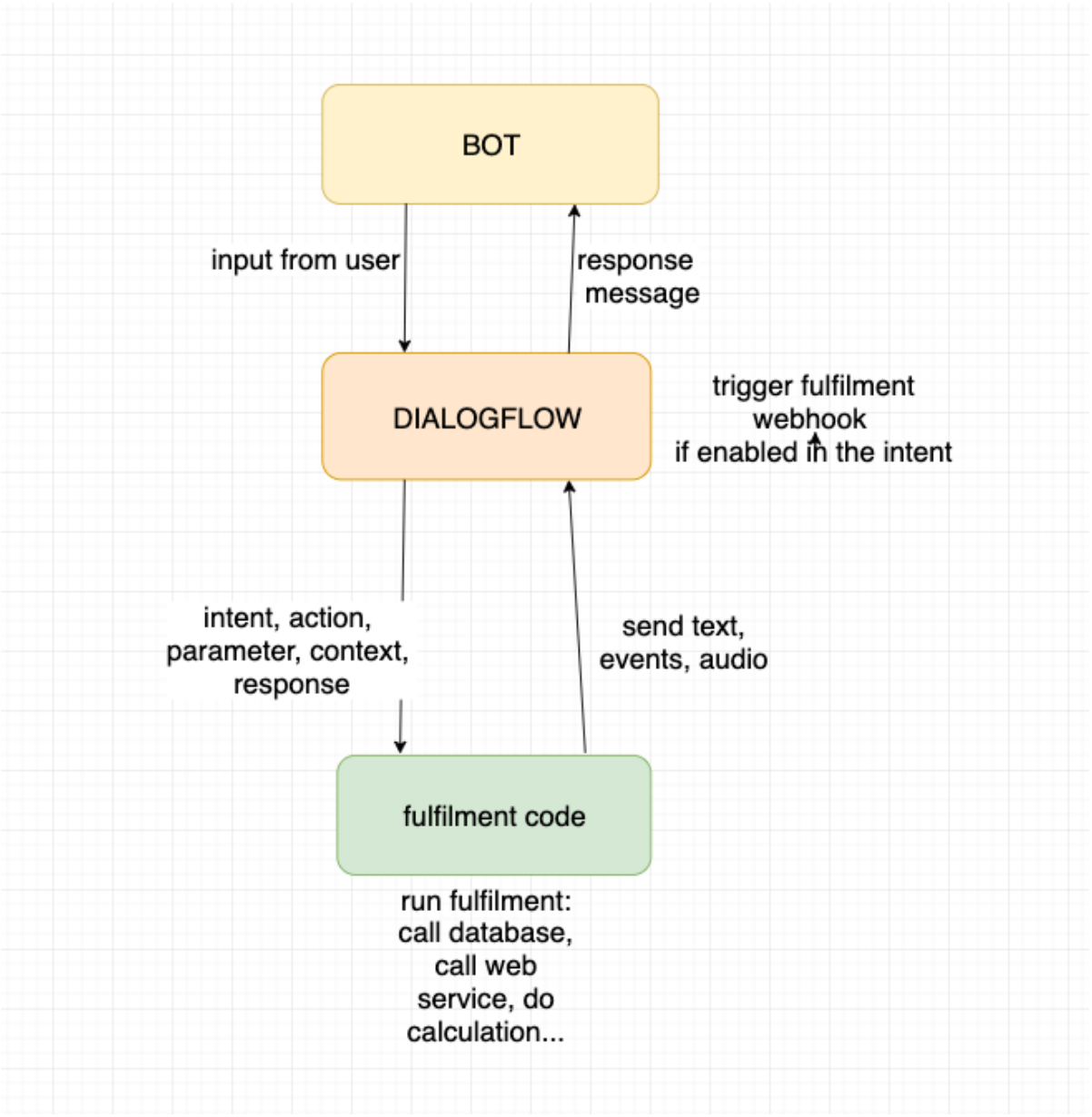
Integrating DialogFlow to other platforms is super easy. Especially if using the integration tools provided by DialogFlow. You can integrate your agent with the Google Assistant via Actions on Google with one click of a button. And you can test the action in a simulator, to see how it will respond in speakers and on screen.

One-click integration is also possible for other platforms like Facebook Messenger, Slack, Viber, Skype, Telegram, Kik, LINE, and Twitter.

Dialogflow also allows an agent to be easily exported to or imported from, other natural language understanding platforms such as Amazon Alexa and Microsoft Cortana.

When integrating with platforms like Messenger, Kik, Slack... one-way integration comes handy when you are using fulfillment as a backend app. That is when you connect the bot directly to the DialogFlow.

Like this:

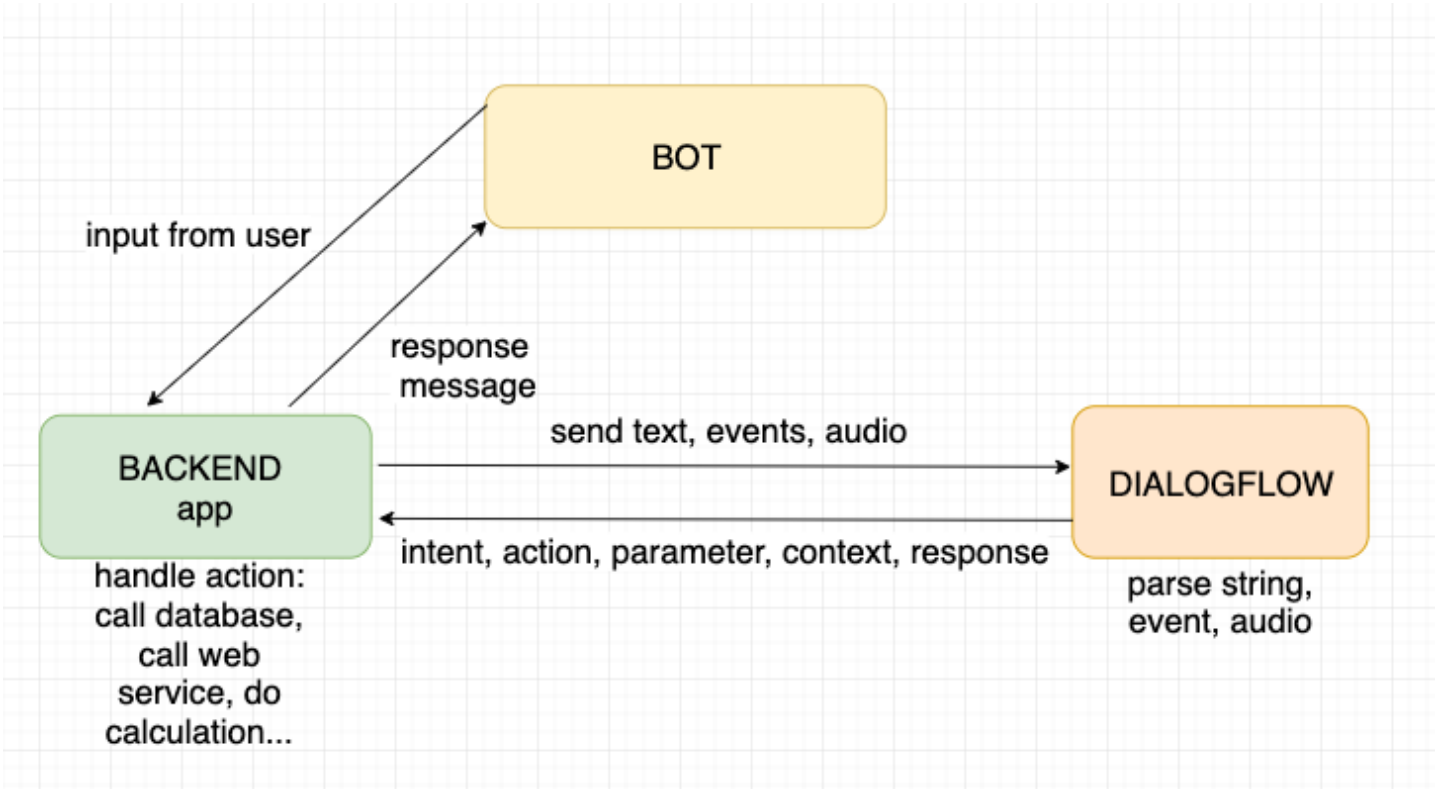


One way integration handles the connection between the bot (an app on the platform like Messenger or Slack) and the DialogFlow.

The other way to connect bot with the DialogFlow is to connect it to Backed app first and then use DialogFlows API to send queries to be parsed by DialogFlow.

There are pros and cons to both approaches. You can read about them here.

This is how the building blocks are connected in this case:



For this, we cannot use the one-click integration.

Do not fear I will show you how to integrate both ways in this tutorial.

But first, let's start with Google Assistant.

Integration with Google Assistant

When integrating with Google Assistant you'd be working with Actions on Google. I have a whole course dedicated to building apps for Google Assistant.

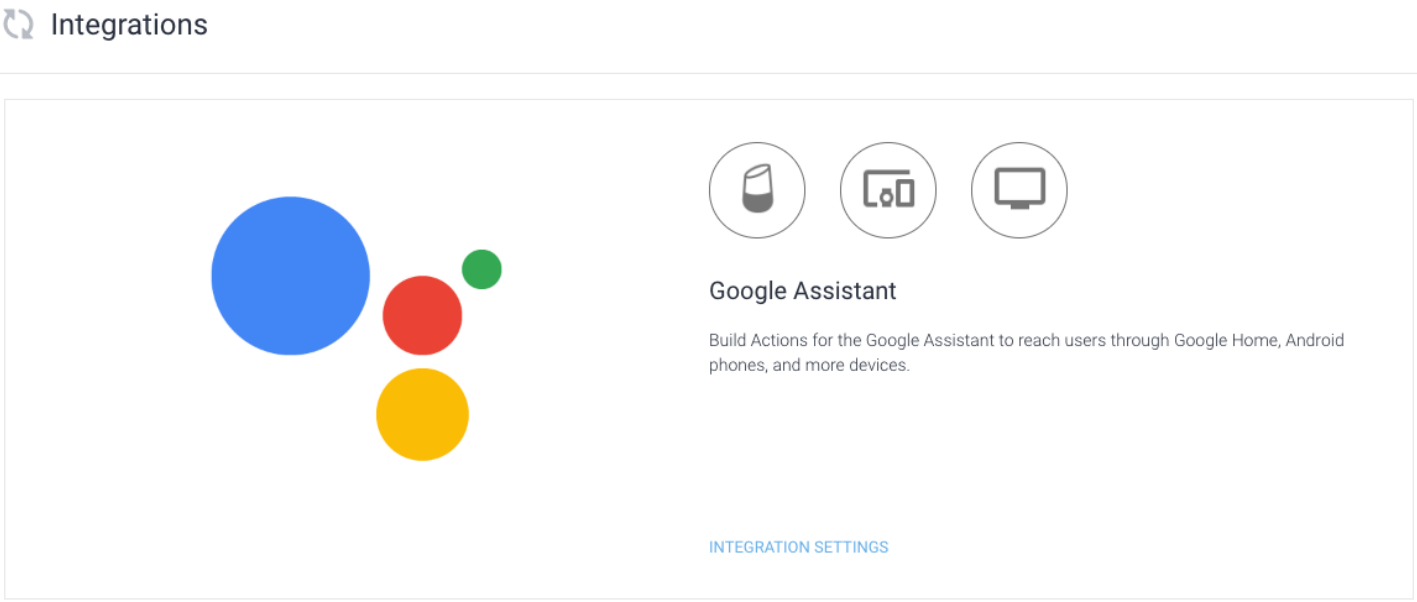
Integrating your Dialogflow agent with Actions on Google enables your agent to reach users on over 500 million devices that support the Google Assistant, like smart speakers, phones, cars, TVs, headphones, watches, and more.

Integrating DialogFlow with Google Assistant is simple. What we will do is:

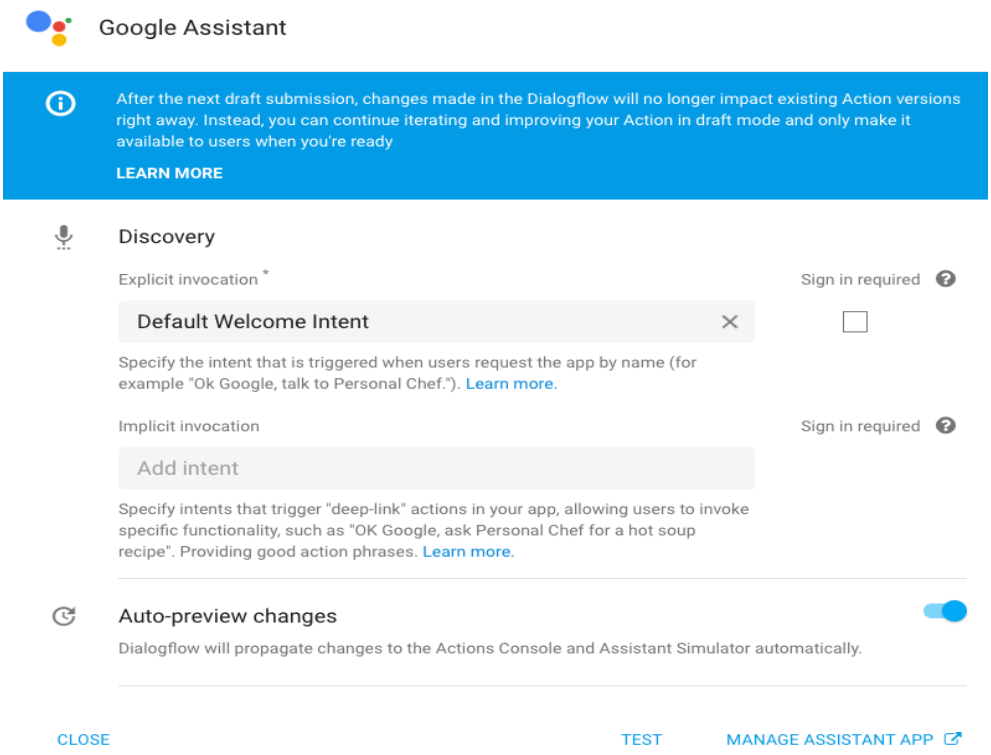
- Enable integration
- Choose the display name of your action
- Chose the voice of your app
- Add descriptions, images, extra information
- Test it in the Google Assistant Simulator

Enable integration

In the Integrations tab, you will find Google Assistant right on top.



The first thing to do is to go to Integration settings:



Explicit invocations

When you click on Integration settings you will be able to select the intent that should be triggered upon your action activation. So, when the user on Google Assistant calls this action, the intent you chose in the Explicit Invocation will be called. When a user would say Talk to 'this and this action' the intent defined here will be called. And yes, usually that is Default Welcome intent. So, explicit invocation occurs when a user tells the Google Assistant to use your Action by name. Optionally, the user can include an invocation phrase at the end of their invocation that will take them directly to the function they're requesting. Like for example, OK Google, ask Dialogflow tutorial what is an intent.

Implicit invocations

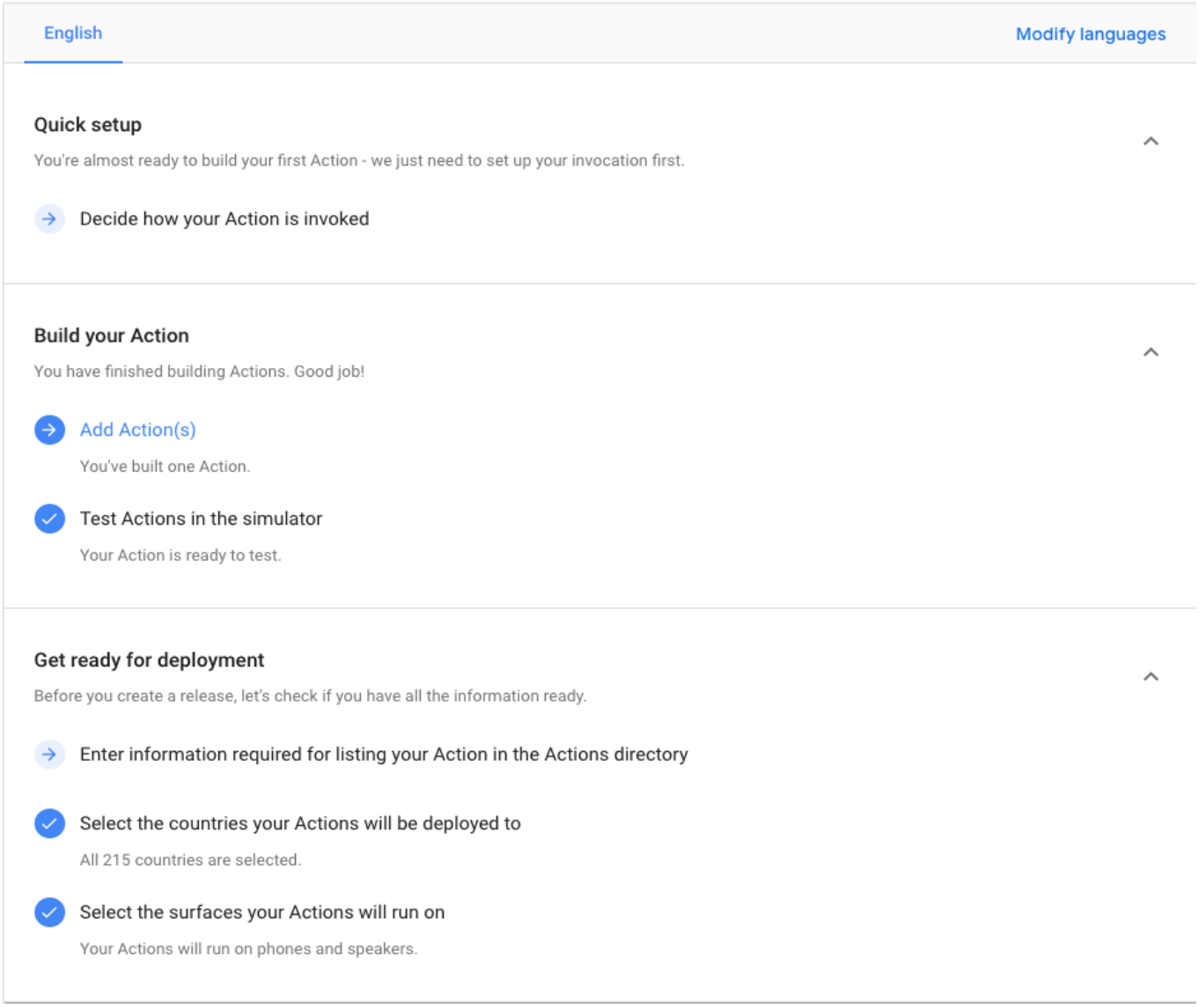
Action can be also called without the user specifying the name of your action. That is called implicit invocation. In that case, the user makes a request to perform some task without invoking an Action by name. The Google Assistant will try to help the user by finding the appropriate Action or search result, or a mobile app. Therefore adding implicit invocations helps Google Assistant to figure out what your action can also do. This is a way to help Google Assistant promote your action better.

Auto propagate changes

If you want the changes you make to be seen in the Simulator and in the action, set this to true. As I do.

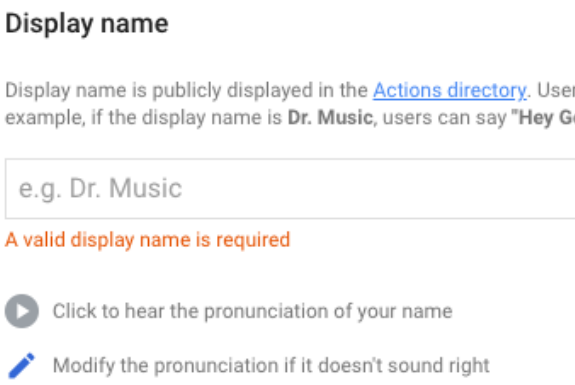
Before we go to check the simulator, let's manage the actions settings. We will set the name of the action, add images and other settings of the action.

You will see an overview of the app when you go to Manage Assistant app

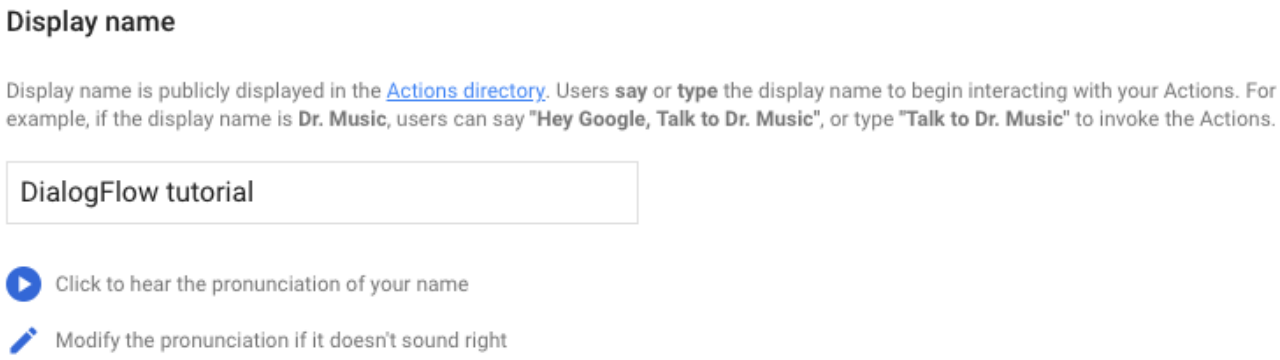


As you can see we have not yet set the name of the app. And our action is built for us since we have been using dialogFlow. It is ready to be tested. And it will be available in 215 countries. Running on speakers and phones. These are the settings we can still change.

Let's start with the action name.

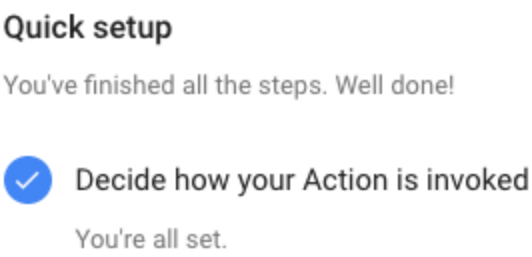


Action name should imply what the action can do. I have set up my intents now to have a little DialogFLow tutorial guiding users step by step through DialogFlow development.



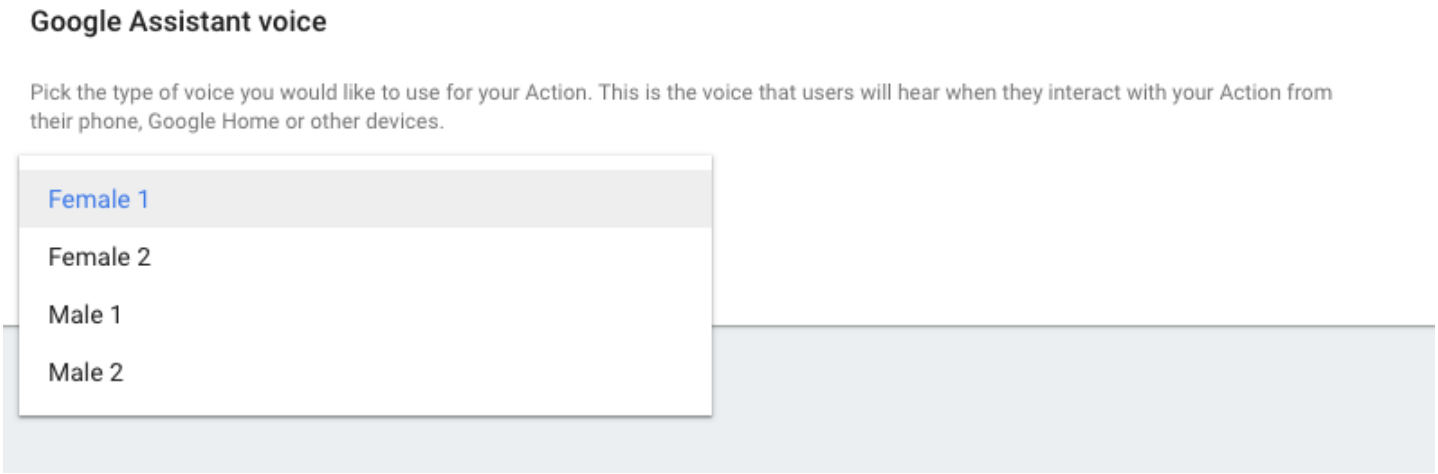
That is why I will call my action DialogFlow tutorial. Simple.

And after I save this page I can see that in the Action overview tab, the change was saved and that my action now has a name.

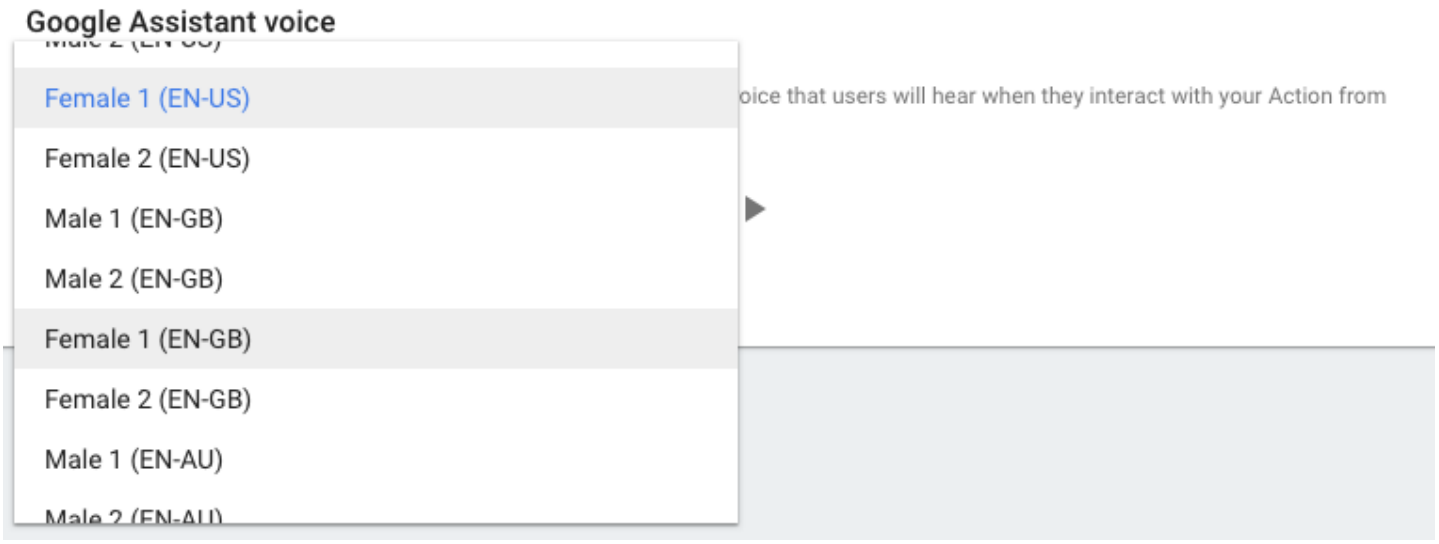


Now we'll be able to call it in the simulator and out devices with the name we choose and no longer test app.

For the action, we can choose between 4 voices, two-woman voices, and two male. We would choose this if we want to match the user's language settings so that the accent would be the same as users. Not for example American English for British users.



But if we wanted to use for example British accent for all users, then we would uncheck 'Match user's language settings' and we could choose a British accent now.



Add other settings here: actions,

A little theme customization is available for you to make your action fit your brand better by changing the colors, the typography and the background image of the action.

Account linking is also available for your action. You can use account linking to connect your users' Google accounts with user accounts in your authentication system. This allows you to build richer experiences for your users; for example, you can save the user's preferences, making a more personalized experience. You could save their shopping choices, past transactions, and other preferences that are useful for interacting with the user in order to accomplish the task better.

If you have apps on mobile and web platforms, you can use account linking to securely make users' preferences available to all platforms, which ensures a consistent cross-platform experience.
https://developers.google.com/actions/identity/

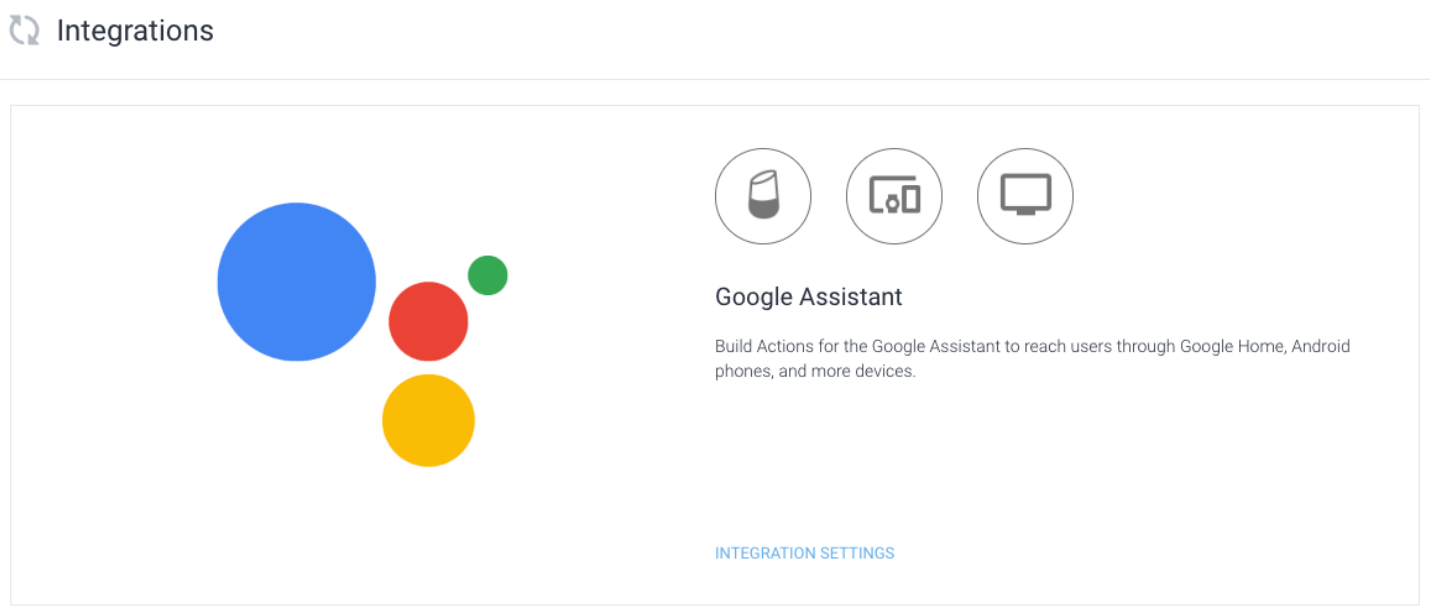
Upon your request, I can make a little tutorial on how to set up account linking.

Google Assistant simulator

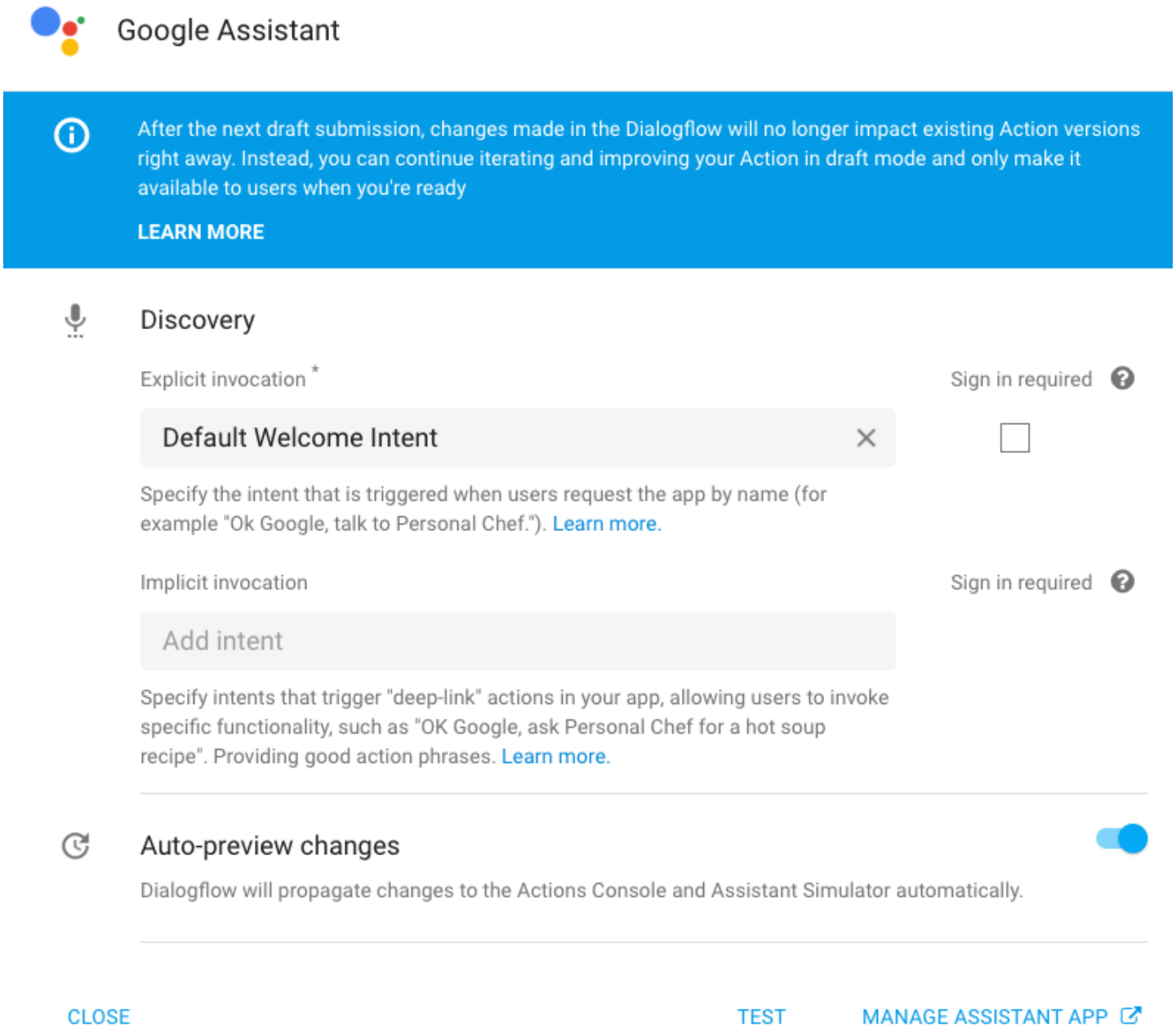
Google Assistant simulator will give you a chance to test your app and it's behavior on the speaker, phone or Smart Display without having a device. Although you can also test your action on devices that are connected to the Google Assistant user that is the developer of the action.

You can access the simulator by DialogFlow. Like we did so far. Or you can go to Actions On Google console: https://console.actions.google.com/, click on the project you are working on and go under tab Test. You will come to the same address as if you would access it through DialogFlow.

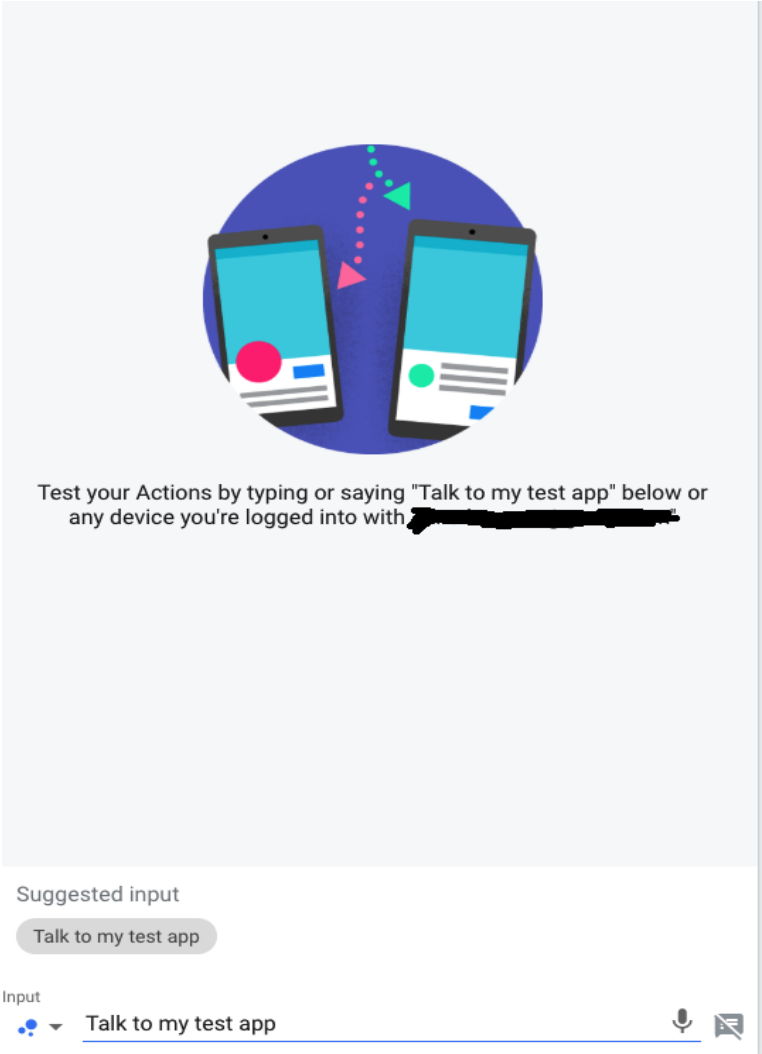
In DialogFlow go to Integrations and click Integrations settings



And then click test



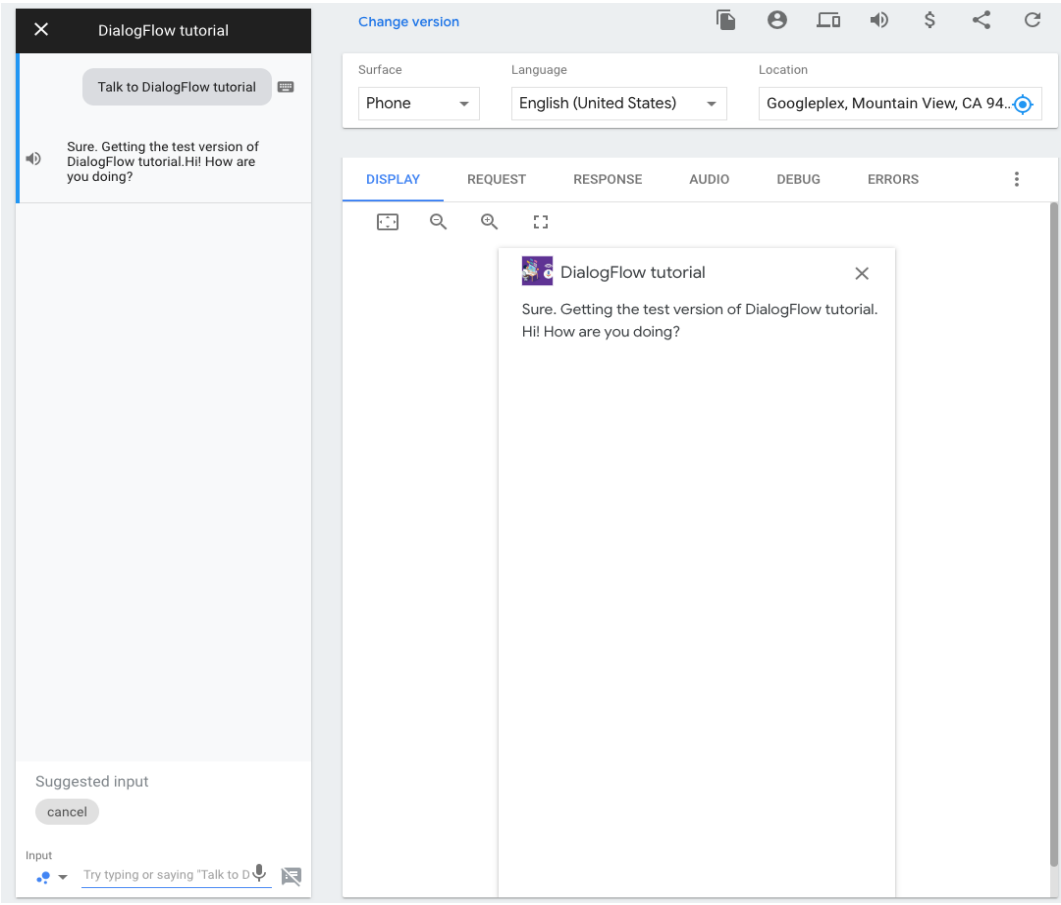
And here we are in the Simulator.



If you have not yet added the name of the action you will invoke the action Talk to my test app. It is called a test app until you give it a name. You can use the microphone, write a text or use the suggested input, that is Talk to my test app at the beginning.

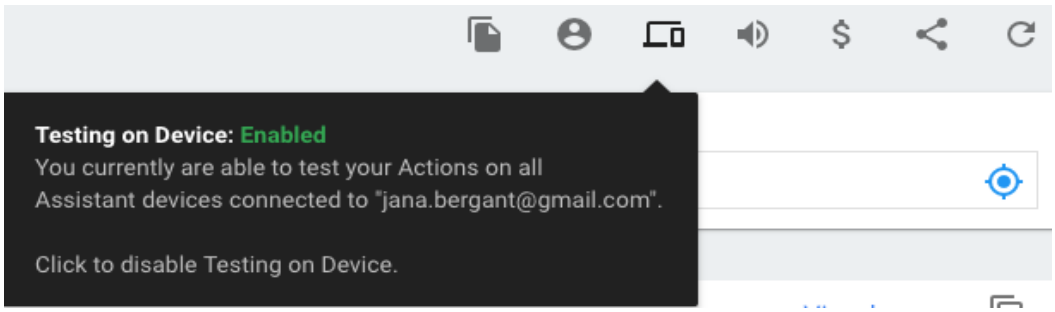
After you name your action as we did in the last article, you can call it by its name. And since I've called it DialogFlow tutorial I can invoke it now with Talk to DialogFlow tutorial. Like this

The response will be audio and on the selected device simulator.

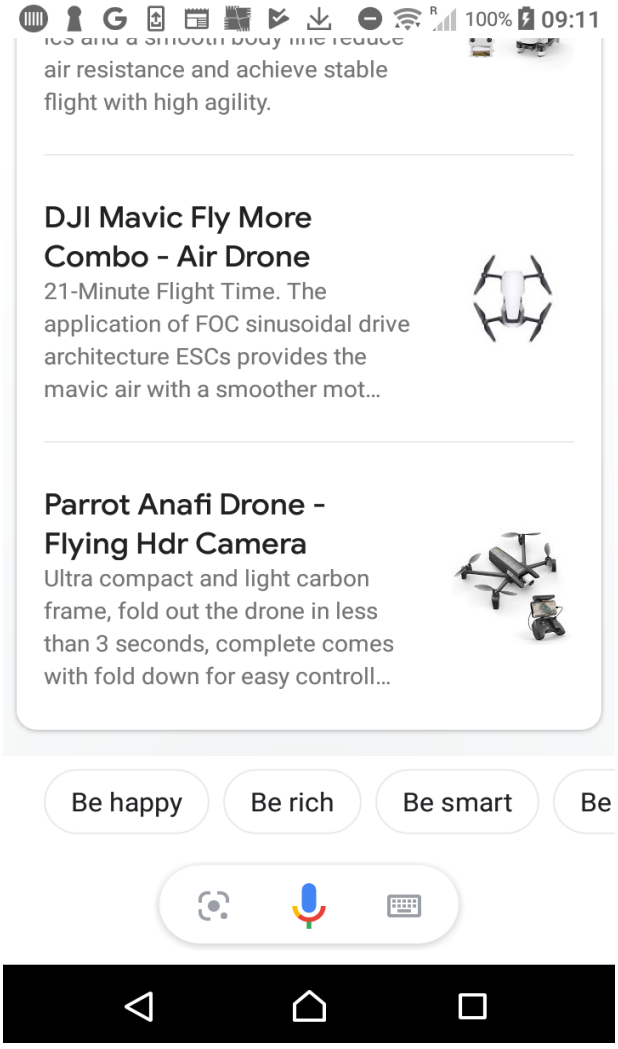


If you want to test on other devices you can switch between phone, speaker and smart screen. But you need to terminate the current conversation.

If you want to test on other devices you need to have the enable on-device option enabled. And the device needs to use the same account as the developer account of the action.



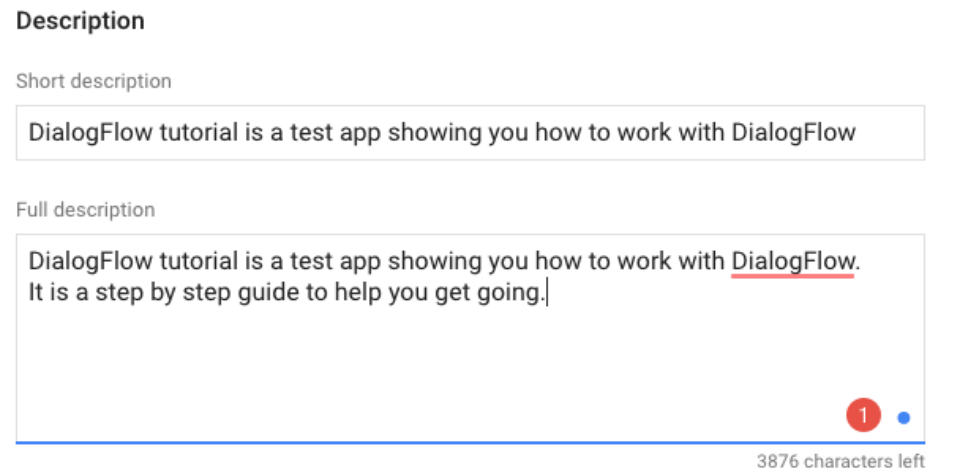
We can test our action on our device even before it is published. You can do this on all devices that are connected with the action developer's email. Here is how it looks like on my phone:

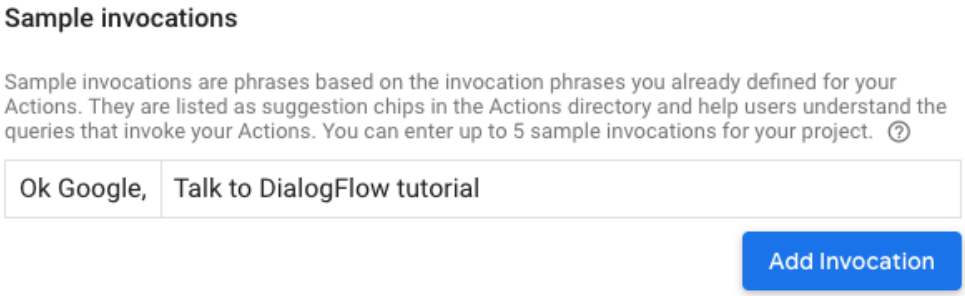


Deployment of Google Assistant action

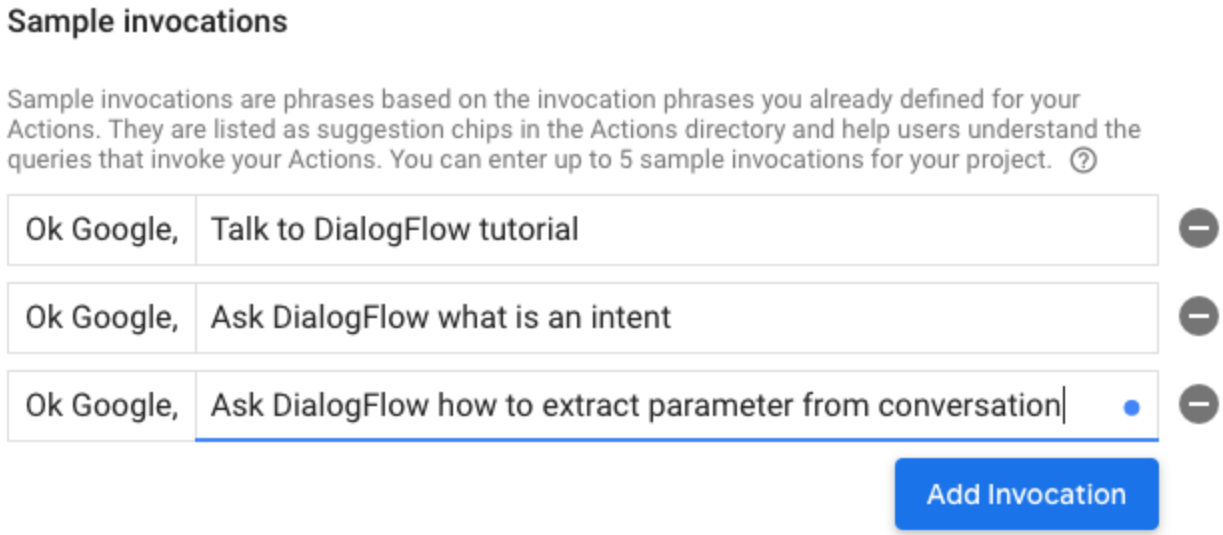
When you want to deploy you need to go to actions on google console to your app and go to Deploy tab.

In the Directory information, you will add a short and full description.

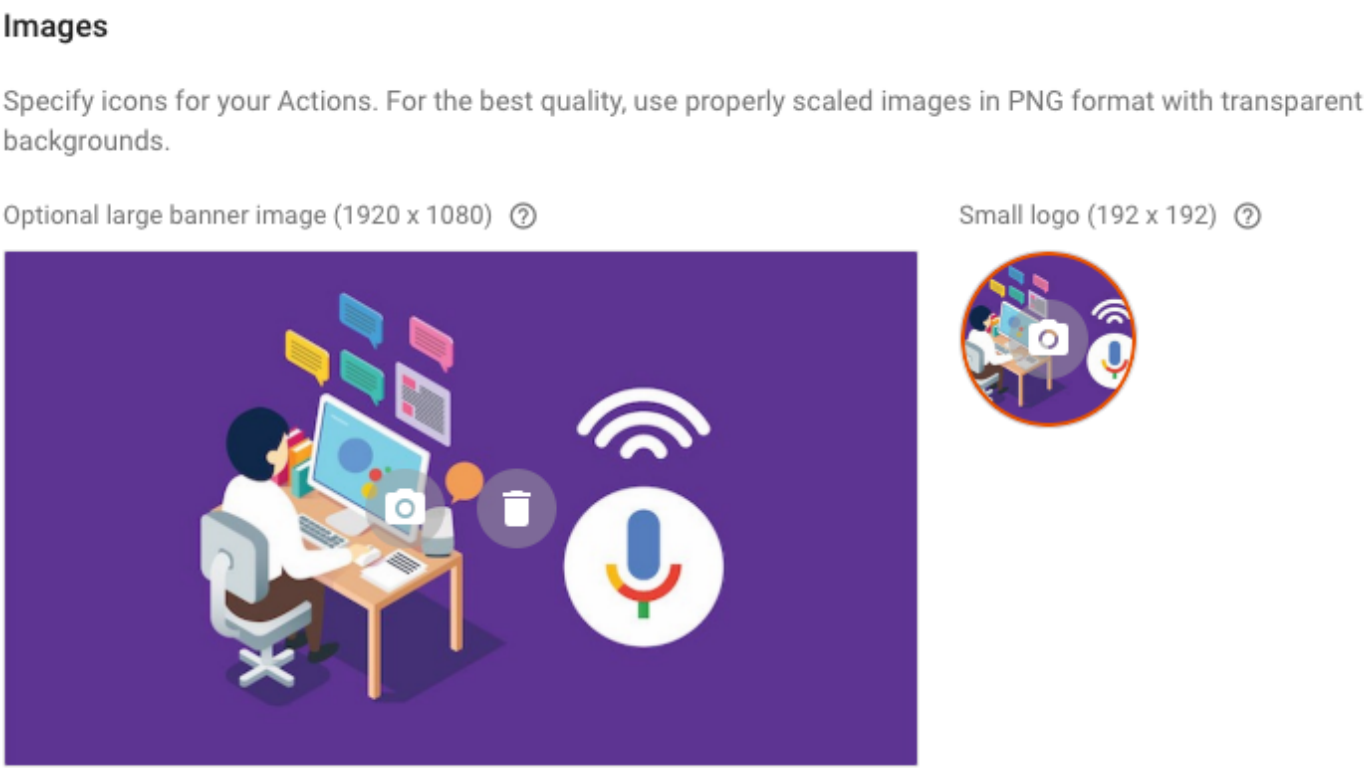




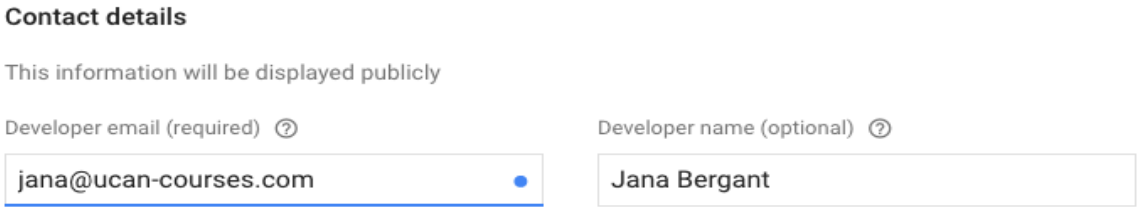
You will also add sample invocations of your app. You can add invocations of specific intent. And with that tell Google what queries invoke your action.



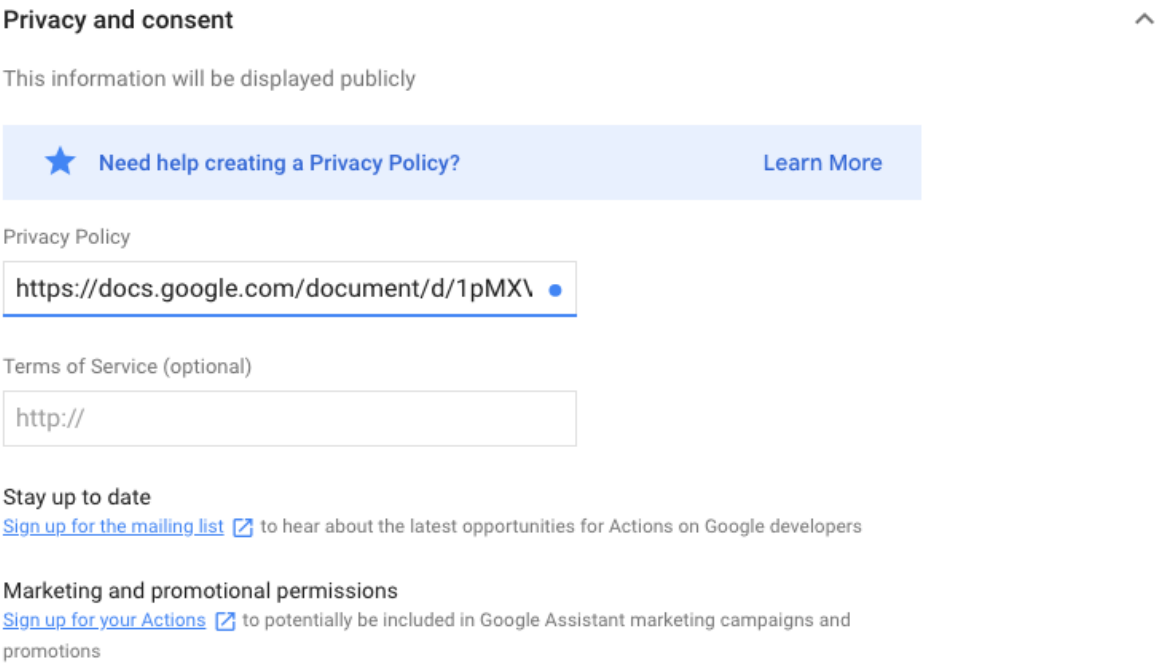
Next thing to do is add images of your action. The large banner should be 1920*1080px and the small one 192*192 pixels.



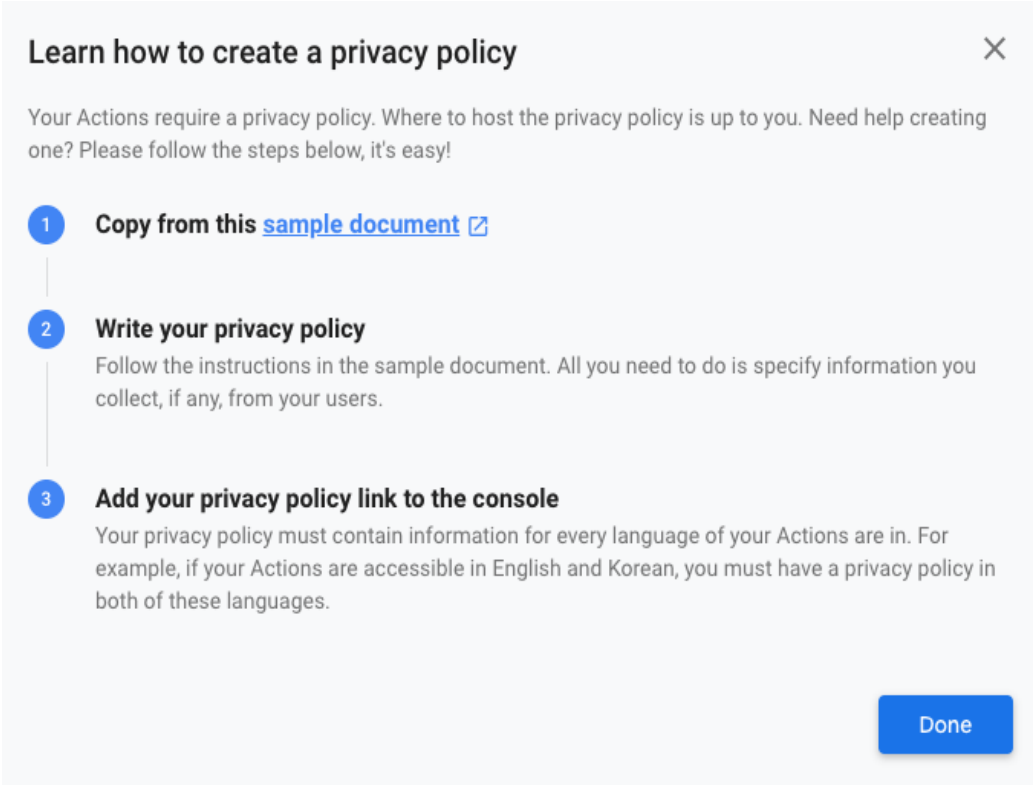
And now we can add public contact details.



Now we will add a privacy policy.



For the privacy policy, you can use Google's template. Access it by clicking Learn More link. A popup will open with the instructions on how to create a privacy policy document. Simply follow the instructions.



The privacy policy should look like this. With your information inserted into it. Get a public link and paste it to Actions on the Google console.

DialogFlow tutorial Privacy Policy

The content for DialogFlow tutorial has been developed by Jana Bergant. However, DialogFlow tutorial runs on systems provided by Google, and Jana Bergant does not have full access to the app. In particular, Jana Bergant has no ability to access the messages you send to DialogFlow tutorial or the responses that it sends back to you.

Jana Bergant may receive from Google non-personally identifying information about the use of DialogFlow tutorial. For instance, Jana Bergant can access usage information regarding how many users are using the app, which geographical regions they are located in, and basic data including users' language, device type, and length and frequency of use. None of the information Jana Bergant receives from Google identifies you, nor does it reveal to Jana Bergant what information you sent to the app or what specific responses the app sent to you.

When you use DialogFlow tutorial, you are also using the Google Assistant, and the Google Privacy Policy describes how Google collects and uses data about your use of the app. You may wish to consult Google's documentation regarding what information is shared with your Google Assistant and how to delete your Google Assistant activity.

Jana Bergant may, in the future, modify DialogFlow tutorial to run on infrastructure which Jana Bergant has full access to, which would allow Jana Bergant access to additional information about your interaction with the app. Should that occur, Jana Bergant will comply with its obligations under applicable privacy law, including updating this Privacy Policy as required.

Additional information will include information about the category of your app, whether it is for families, does it include alcohol and tobacco, transactions, testing instructions.

Additional Information

Category

Education & reference

The category that best describes your Actions. This will help users discover your Actions.

For Families

Are children under the age of 13 one of the intended audiences of your Actions?

If yes, you must join the Actions for Families program. The Actions for Families program allows developers to designate that their Actions are family-friendly, so parents and kids can find trusted, high-quality content more easily on the Google Assistant.

- ○ Yes
- ◉ No

Alcohol and Tobacco

Do your Actions contain alcohol or tobacco-related content?

If yes, you must include age verification at the beginning of the conversation. If your Actions mainly sell alcohol or tobacco, you must implement account linking and verify that the user meets legal age requirements.

- ○ Yes
- ◉ No

Testing Instructions (optional)

Provide any additional information needed to test your Actions. If your Actions require account linking or login information, you must provide a username and password for a test account. Please make sure that any provided accounts are not real user accounts. This information will only be used by the review team, and will not be visible to users.

Transactions

Do your Actions use the Transactions API to perform transactions of physical goods?

- ☐ Yes

Do your Actions use the Digital Purchase API to perform transactions of digital goods?

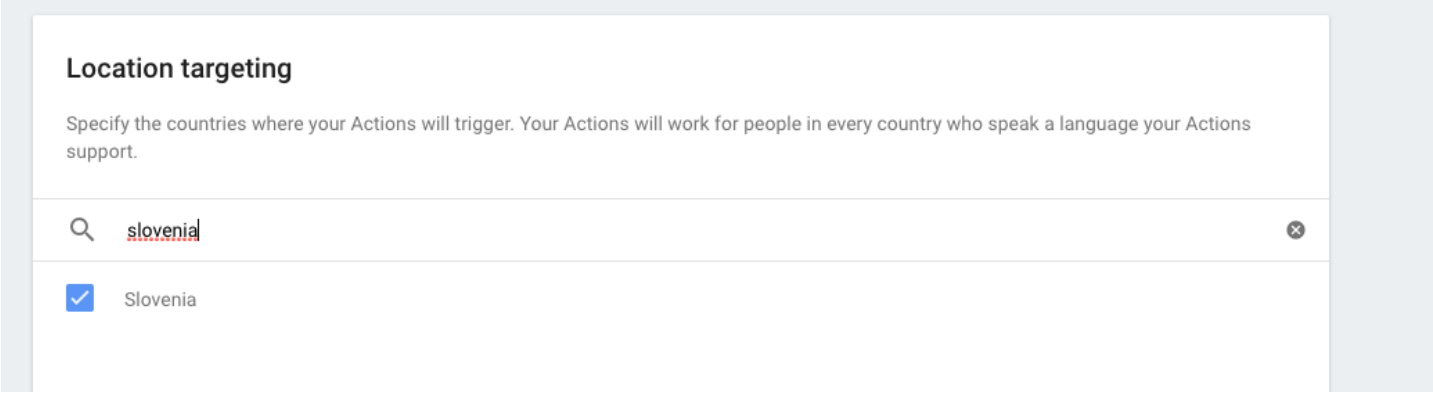
- ☐ Yes

Mic Policy

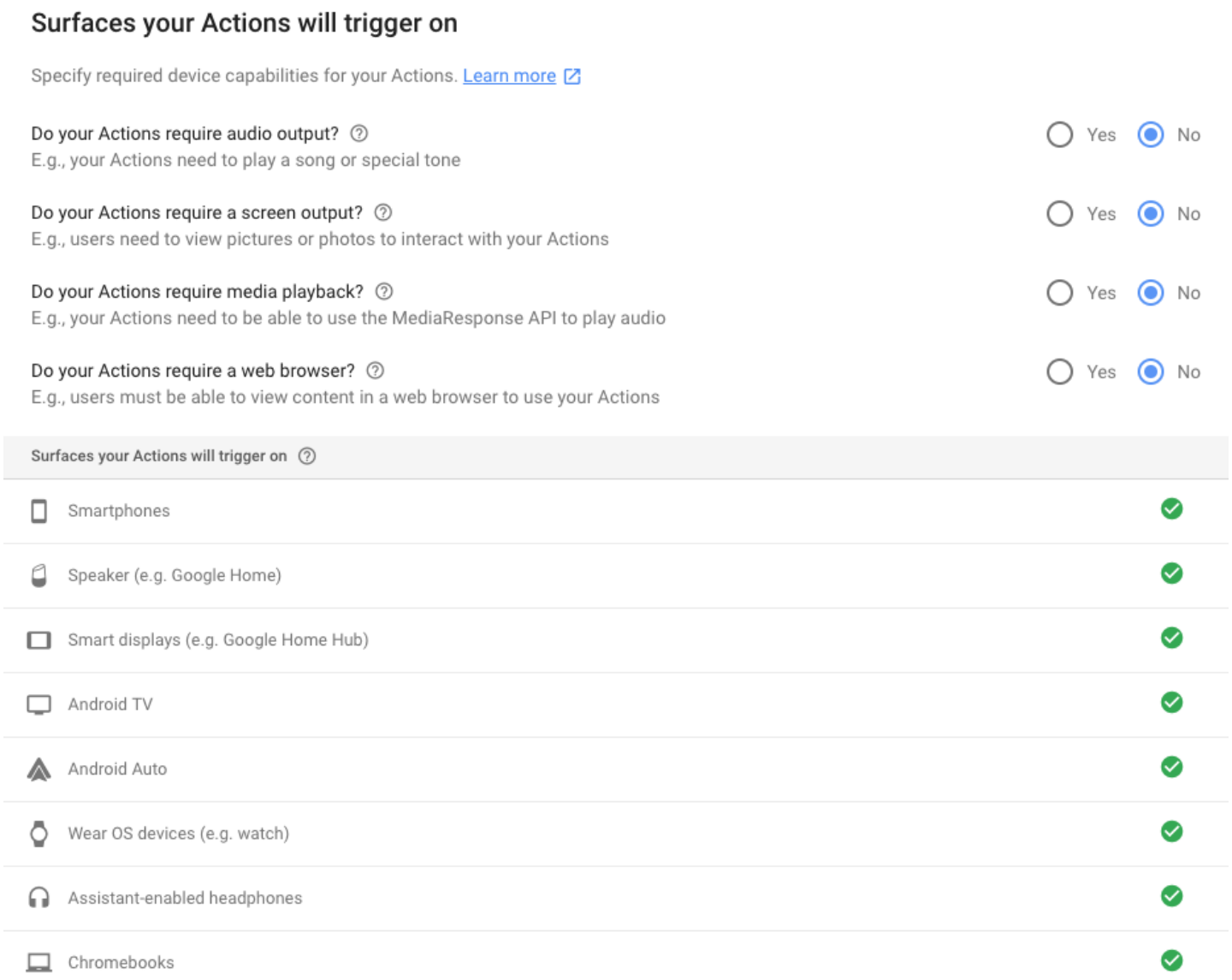
Do your Actions, at any point of interaction with users, leave the mic open without prompting them to respond?

- ☐ Yes

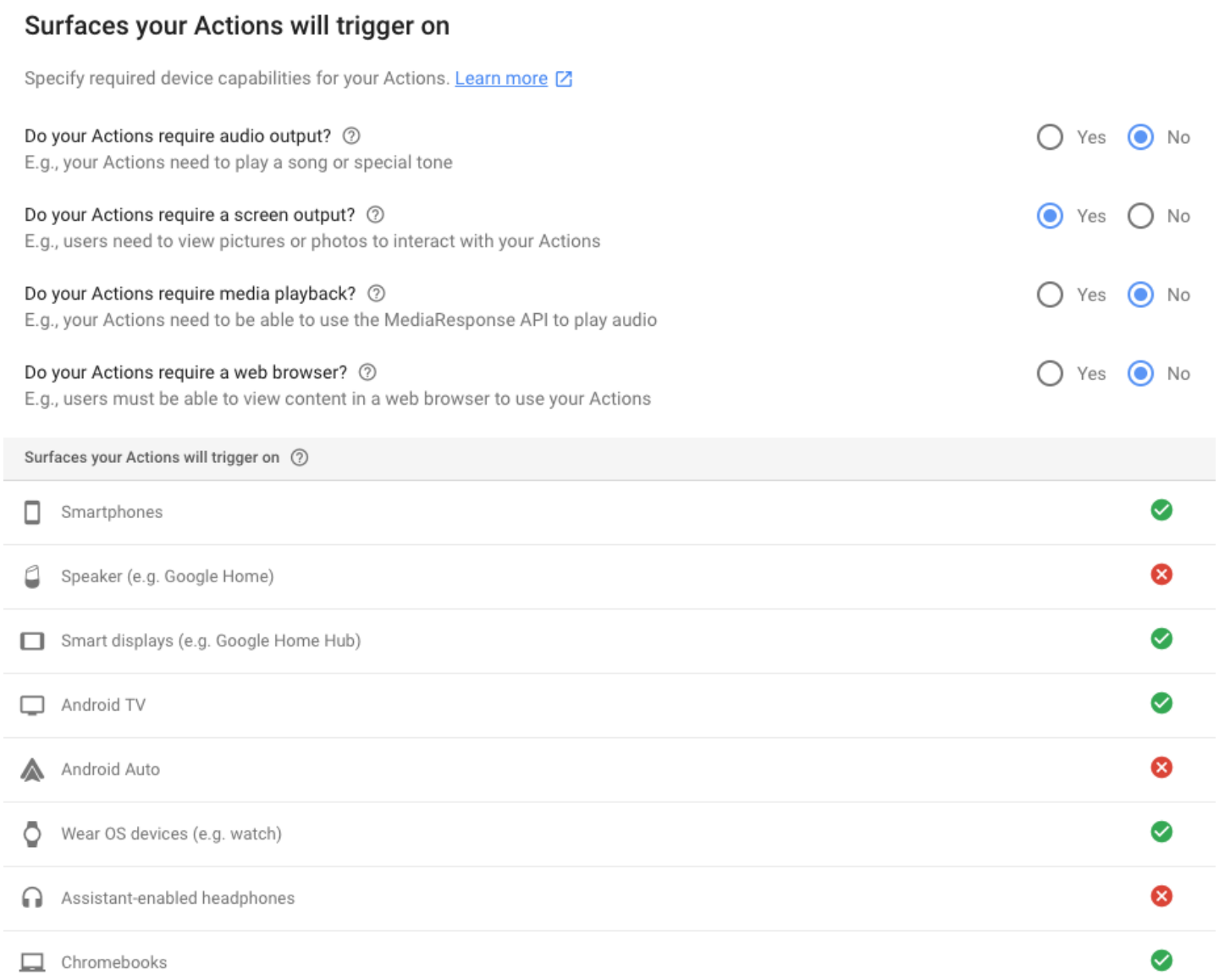
Your action will by default be enabled in 215 countries of the world. But will only work in those who speak the language of your action. You can filter the countries you want to support. This is my country:



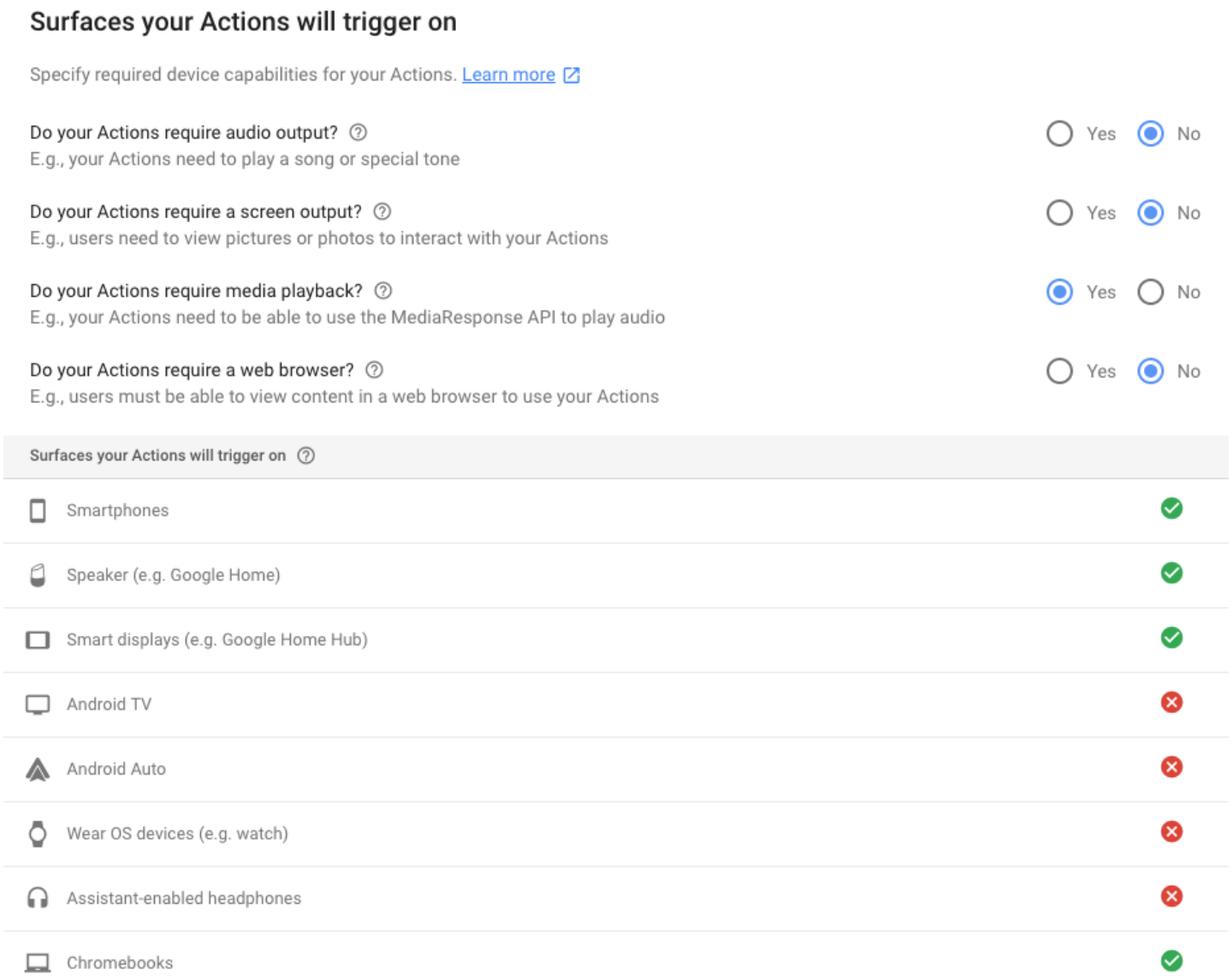
In surface capabilities you tell if you action need an audio or screen output, if it requires playback or a web browser. Based on what it supports Google will know to what devices he can offer you action for.



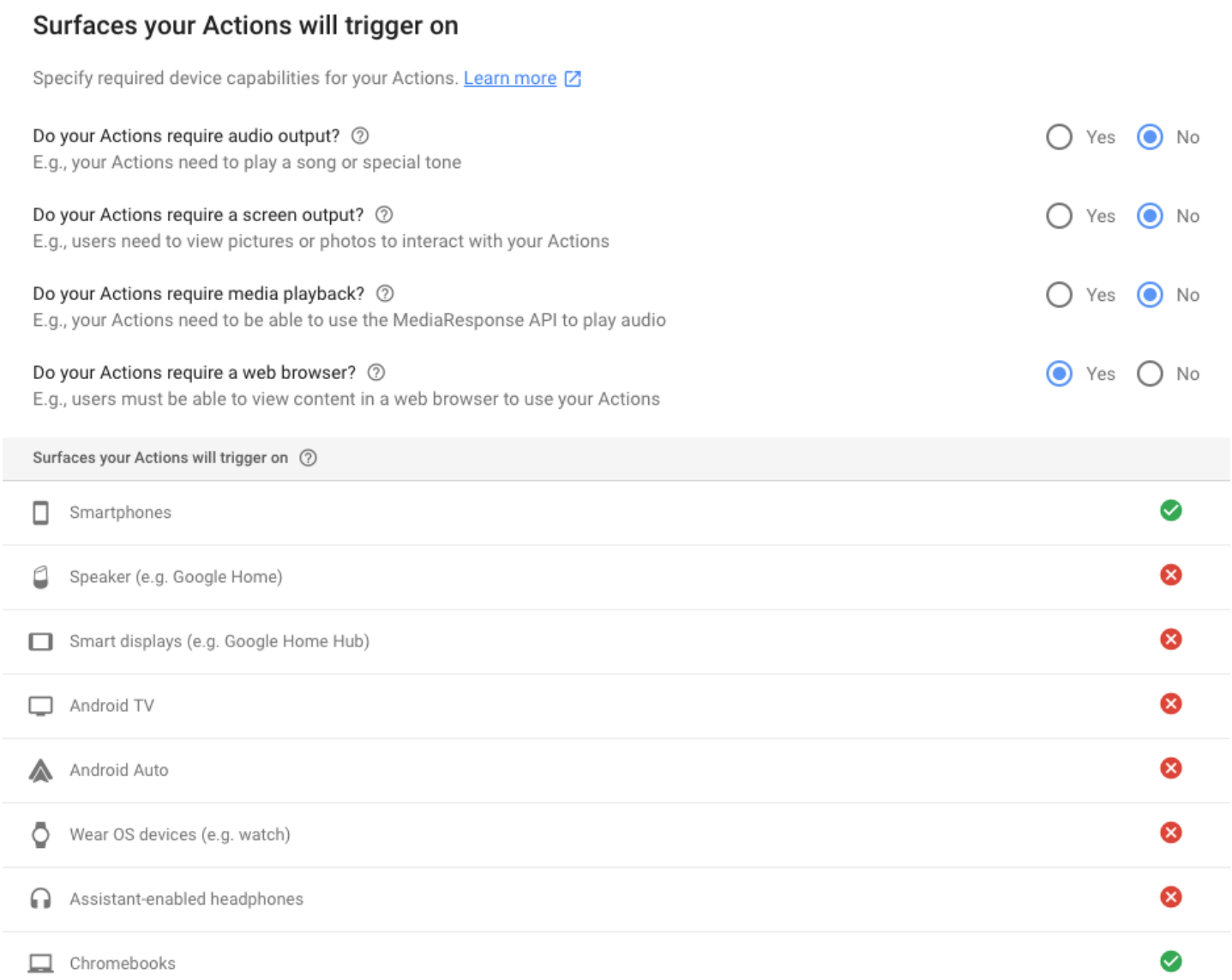
For example if our action only supports screen output, then our action cannot work on speaker or in a car or in the headphones. That is because we don't support voice output.



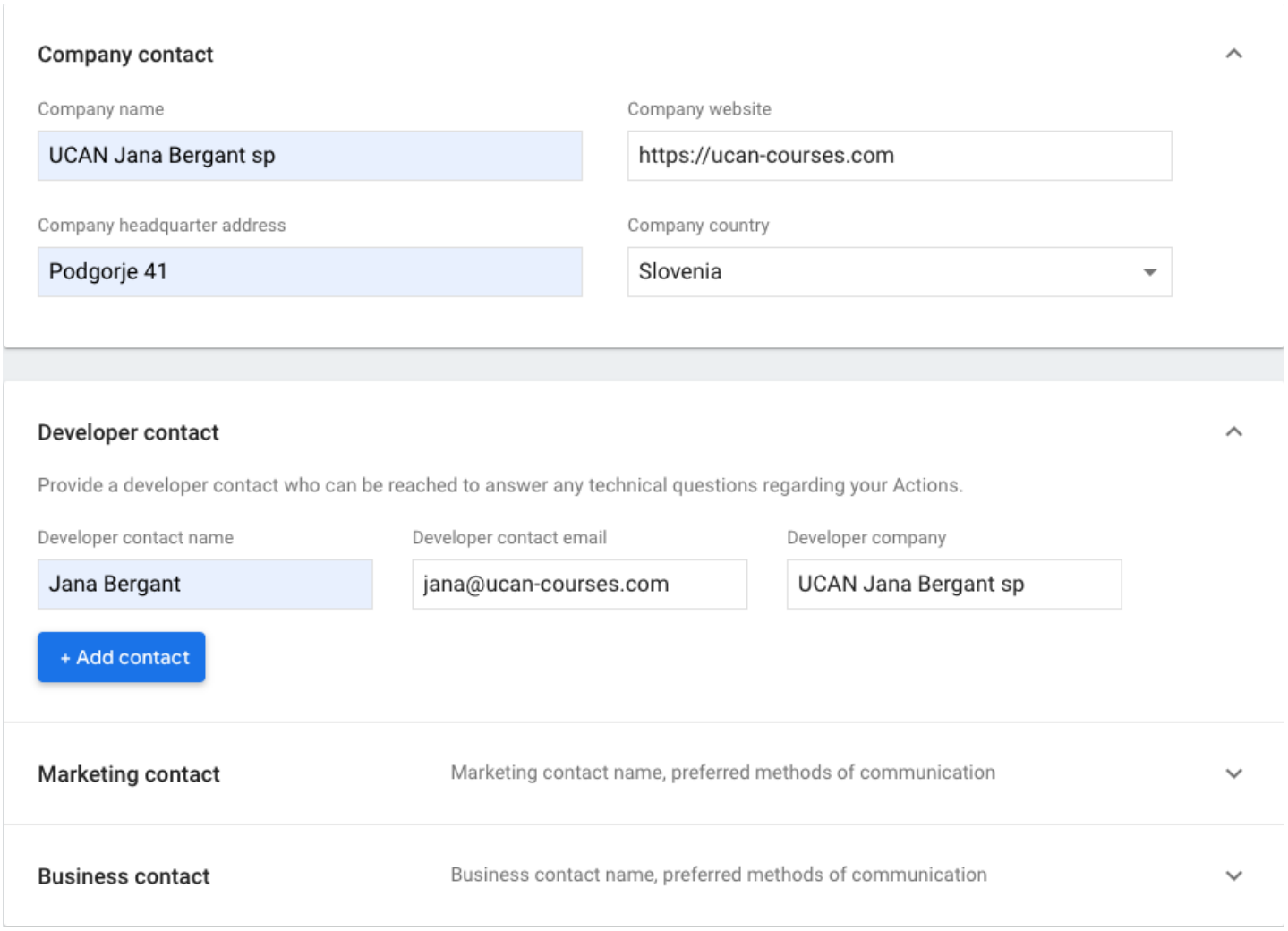
The same goes for media playback. If we only support that, then our reach is limited.



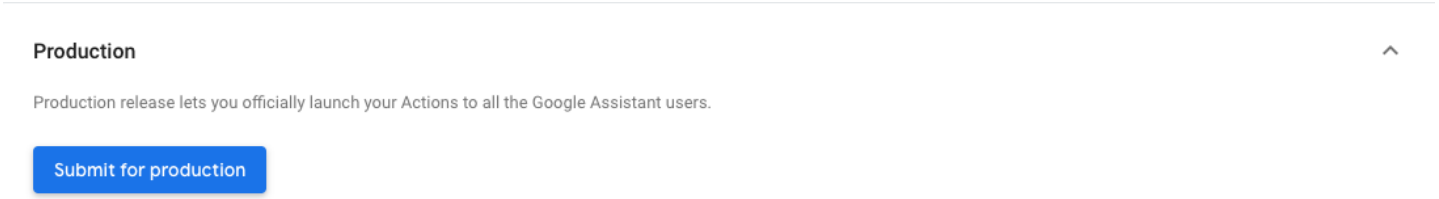
If we only support browser output, then we only reach smartphones and chromebooks.



Next we need to add company info and contact information.



And when we are finally through all the settings we can submit for production.



Integration with Messenger with One-click integration

You can integrate with Messenger in two ways. You can use one-click integration and have Dialogflow connected to Messenger. You can use DialogFLlw's fulfillment to execute extra code. For example, read products from a database or do the payment transaction.

Or you can connect Messenger to the backend app and have that backend app calling DialogFlow API.

We have already talked about the pros of each approach.

And the pros of the first approach:

- the code will run on a serverless function and it will be triggered only when fulfillment is called
- we only call the code in the intents that have fulfillment enabled, not for all intents

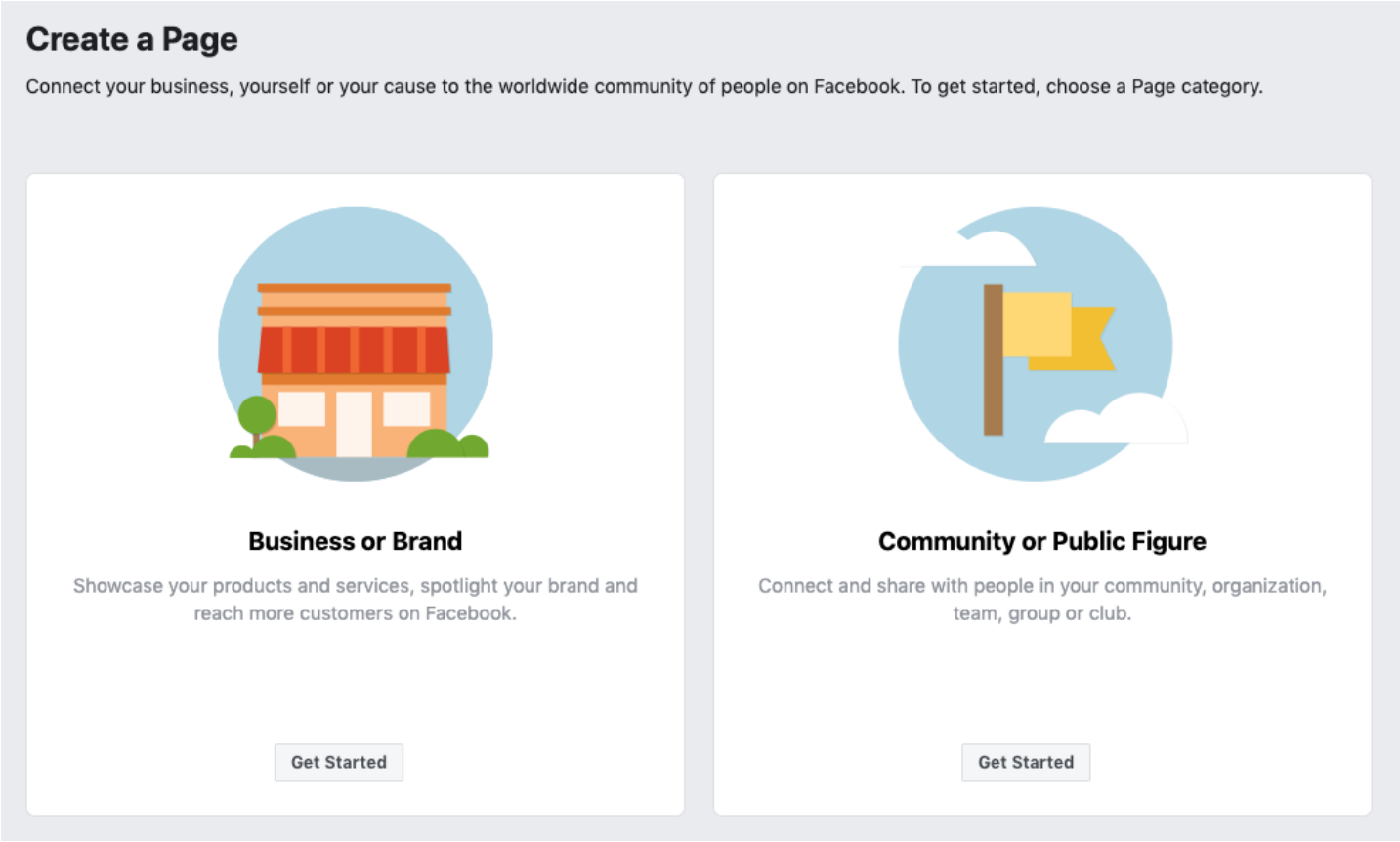
And what are the benefits of having a backend app as the middleman between DialogFlow and Messenger:

- we can choose if we want to send queries to DialogFlow or not. If the request is text, then we forward it to DialogFlow. But if the user clicked a button or sent an image, then we don't want to send this to DialogFlow
- we can send messages to different agents. For example, we have a different conversation path in two different languages, so we want to send a text to the right agent
- we don't want our request to timeout

With both ways, you first need a Facebook page. The bot will be answering messages on the Facebook page. Therefore bot is connected to a Facebook page.

To create a Facebook page go here and follow the instructions:

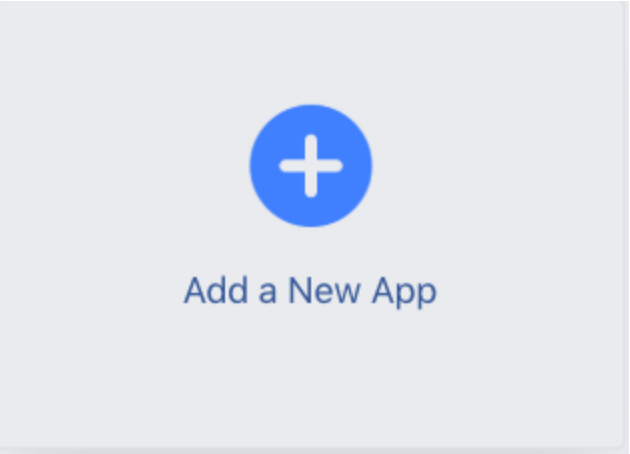
https://www.facebook.com/pages/creation/



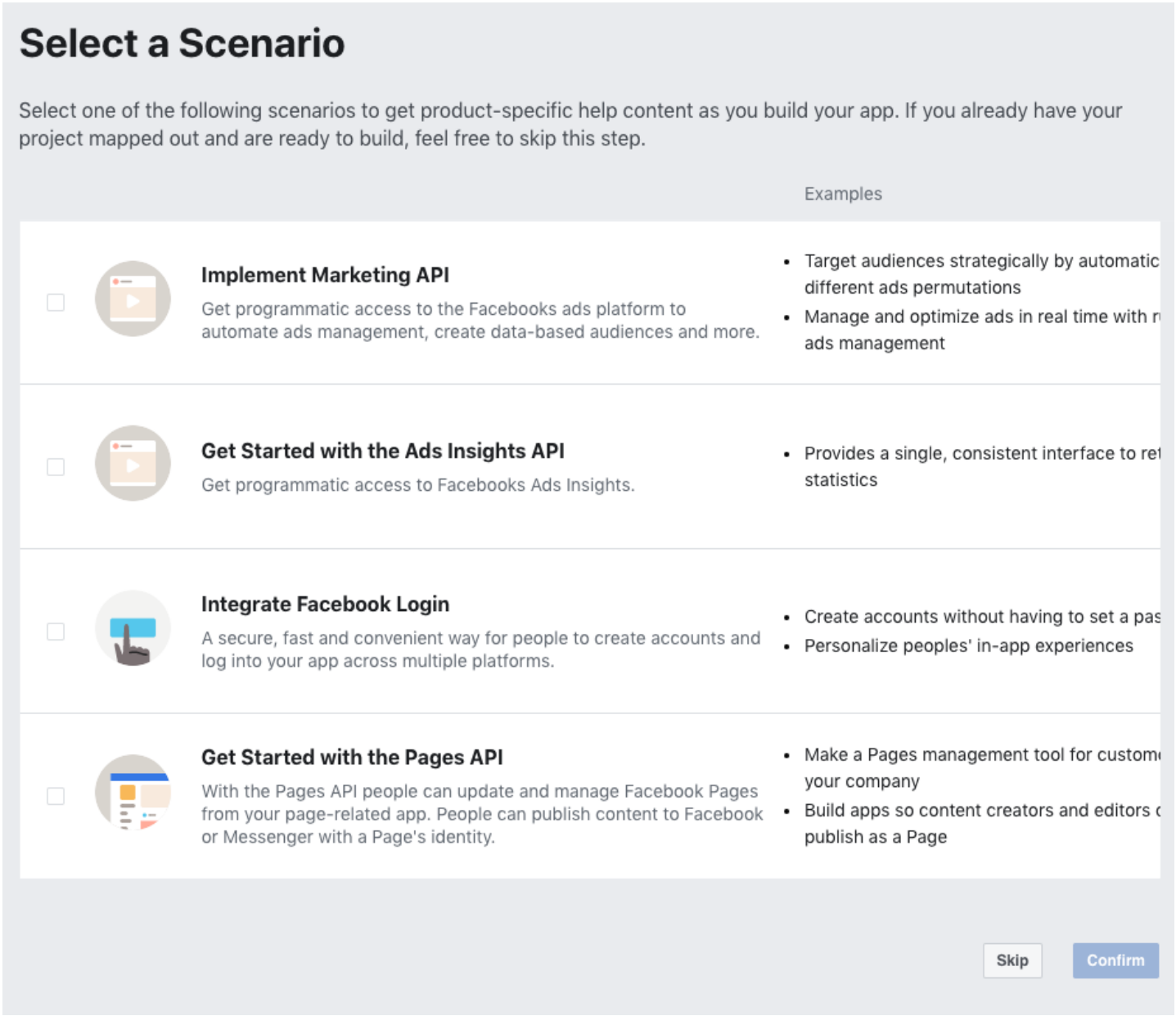
I will not guide you through Facebook page creation since many of you already have it. Here are some instructions for you if you are creating a business page and also need tips on how to publish, promote and improve your Facebook page: https://www.facebook.com/business/pages/set-up.

And since a bot is nothing but an app we need to create a Facebook app. To do so you need to register as a Facebook developer. Here: https://developers.facebook.com/

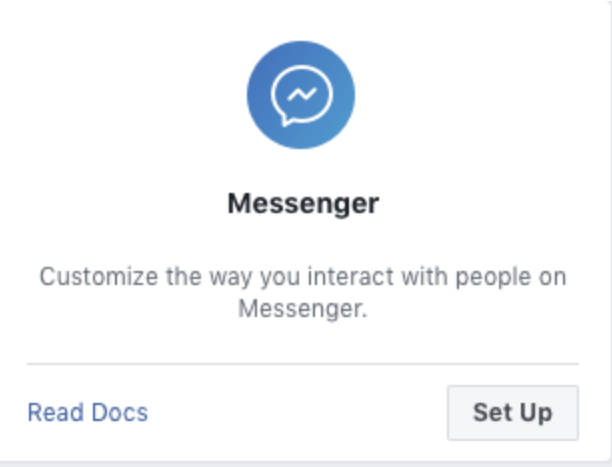
And then you can add a new app.



You can skip the first step since we won't be implementing Marketing API and Ads Insights API, Facebook Login and Pages API.



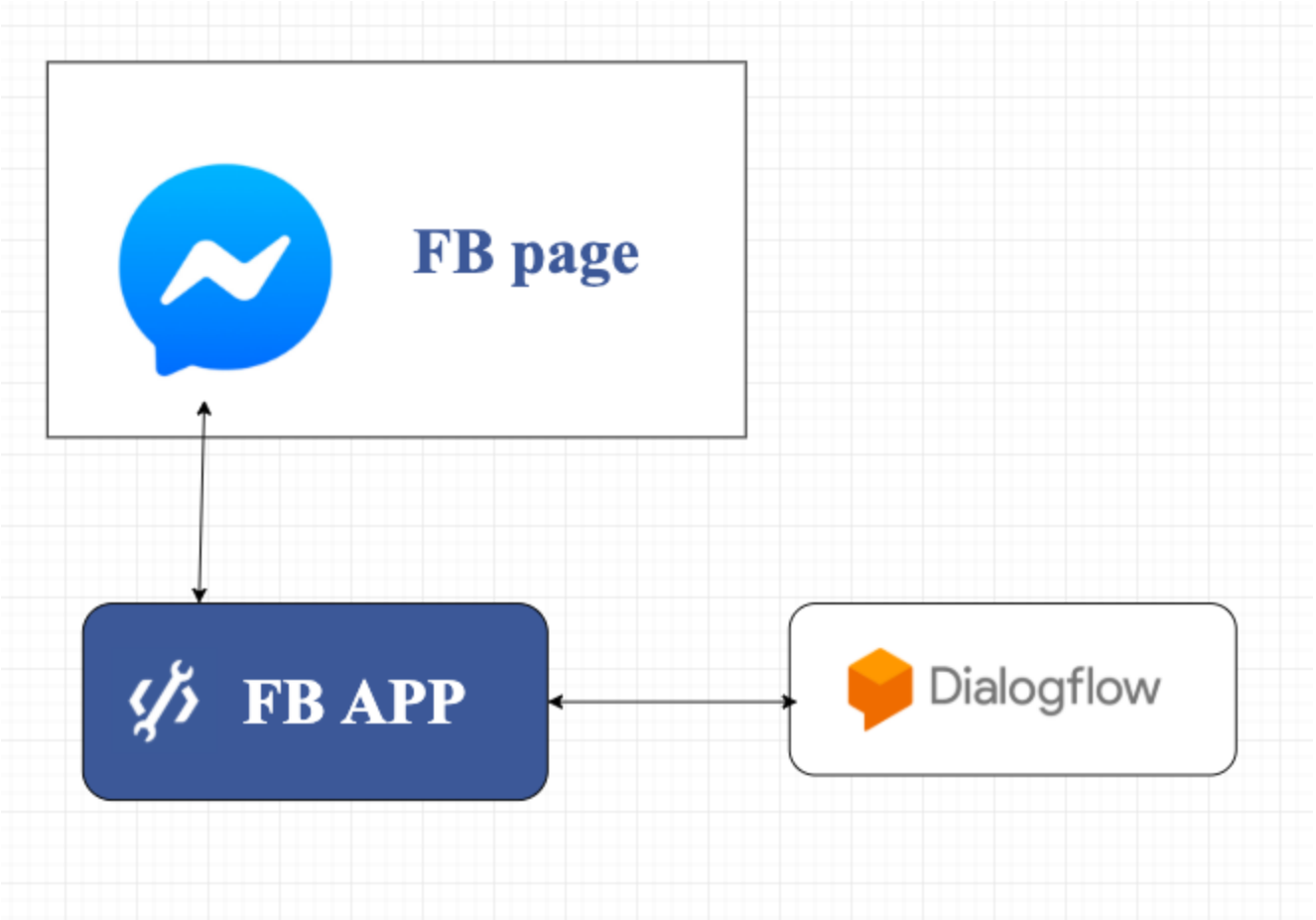
Now you need to add Messenger to this app. We will use this app to communicate with people on Messenger. On the first page you will see this banner:



Click the setup button.

In one-click integration, we'll connect DialogFlow, Facebook app, and Facebook page.

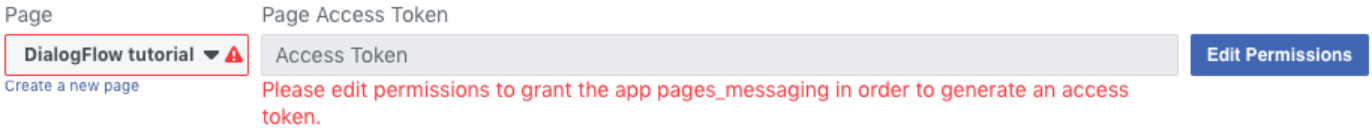
Facebook app will communicate with DialogFlow to get language parsed and get responses and it will display these responses/messages on Messenger of the Facebook page you have created. Like this:



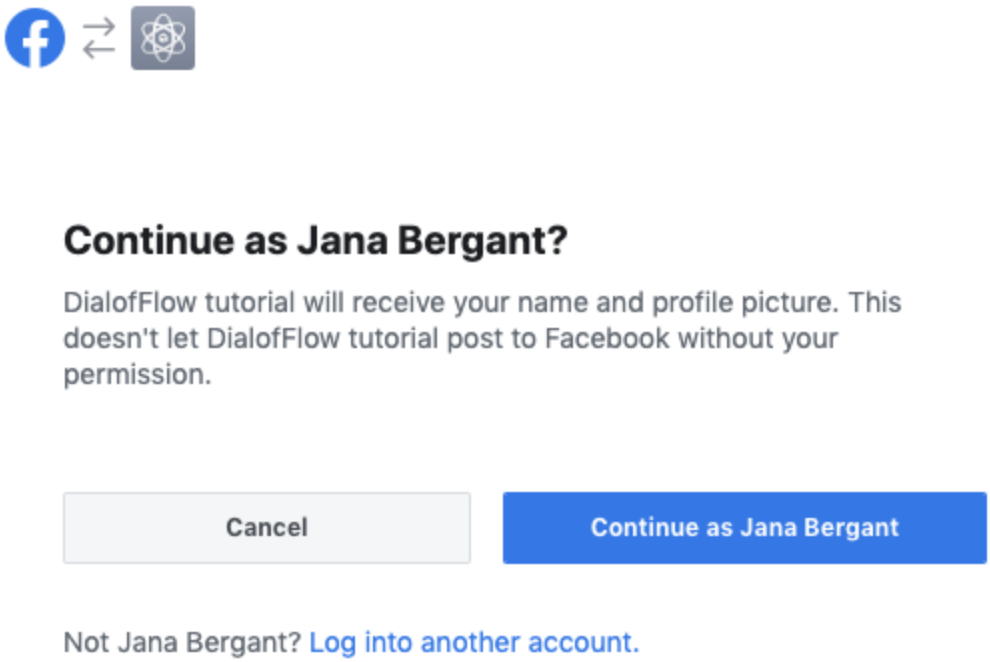
To do so, you need to:

- Generate an access token with the permissions,
- Set up webhook on Dialogflow
- Setup webhook in Facebook app

We will now generate the access token to connect DialogFlow straight to the Facebook app. In the Facebook developers console under Messenger Settings go to Access token. You need to select the Facebook page, this app will be connected to.

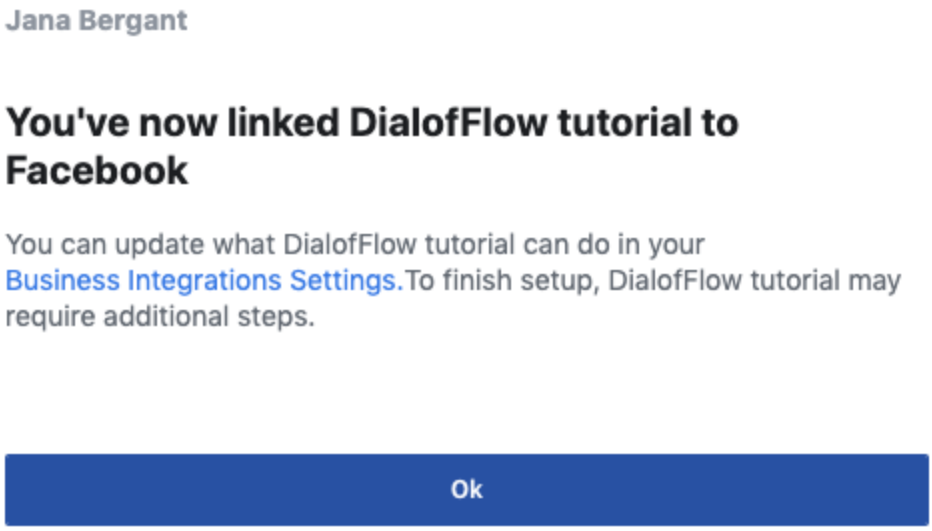


After you select the page you will see this warning. Don’t panic! Nothing is wrong. You need to obtain the right to send messages to Messenger of the selected page. That is the pages_messaging right. You will click the Edit permissions button to do so.

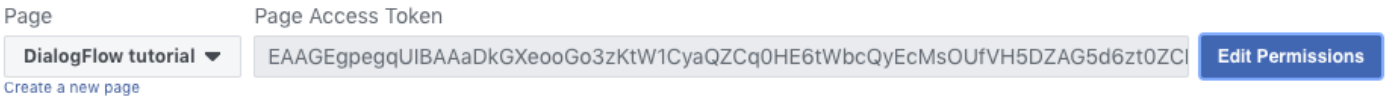


Follow the steps the popup will take you through.

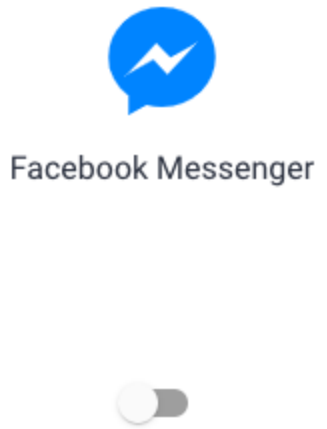
Until you have obtained the right.



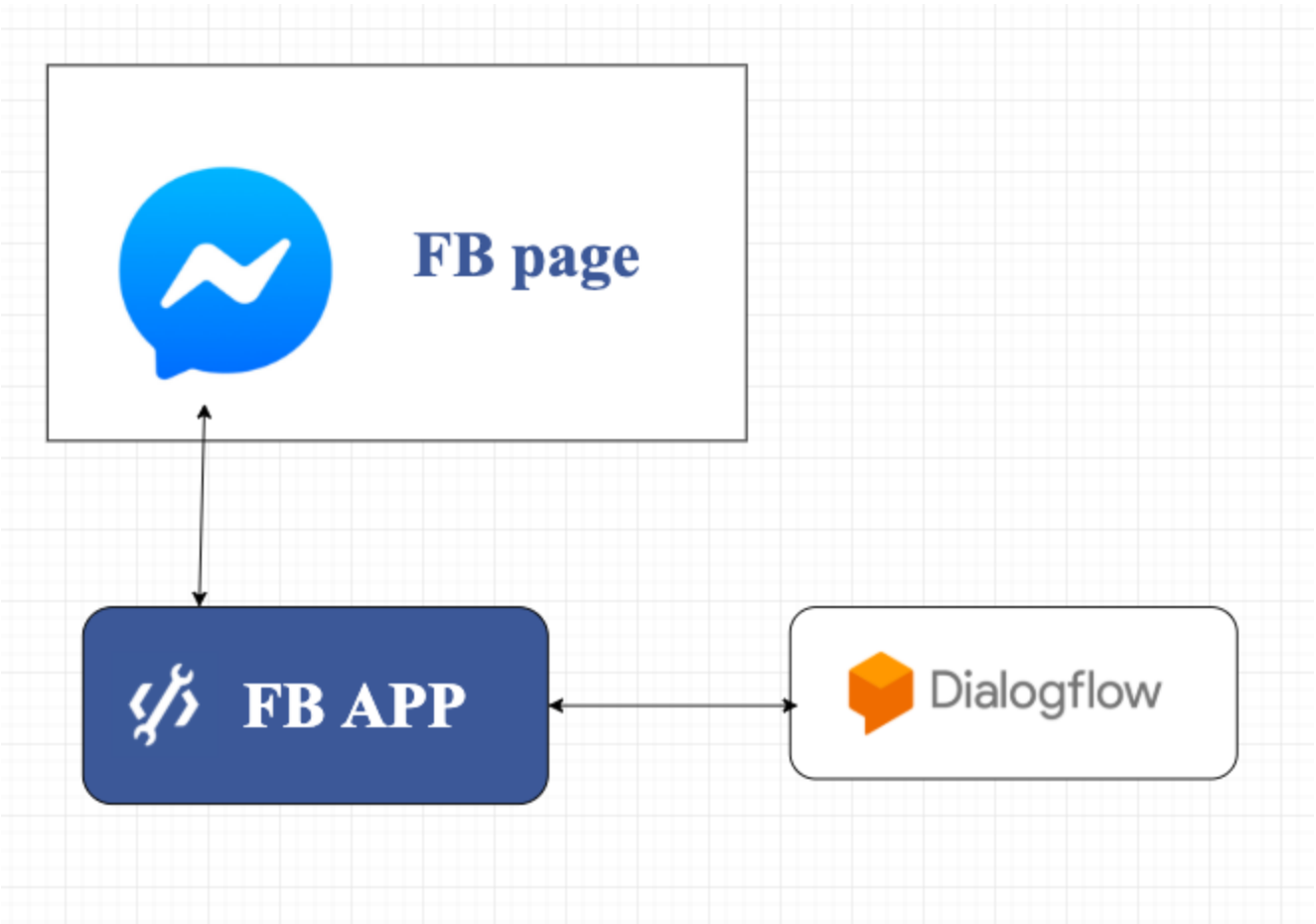
Until you get an access token



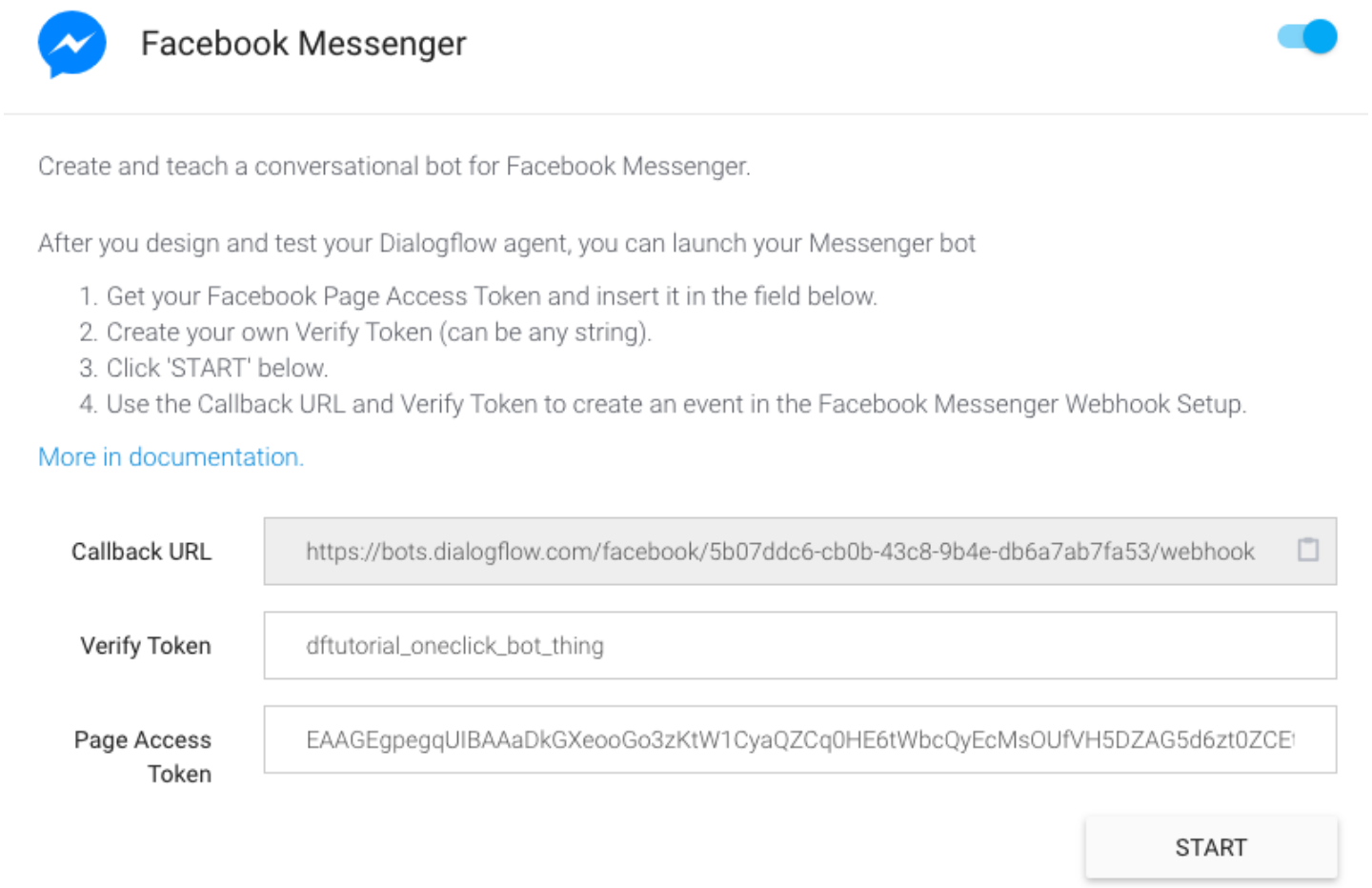
Now you can go to DialogFlow to Integrations and enable Facebook Messenger.



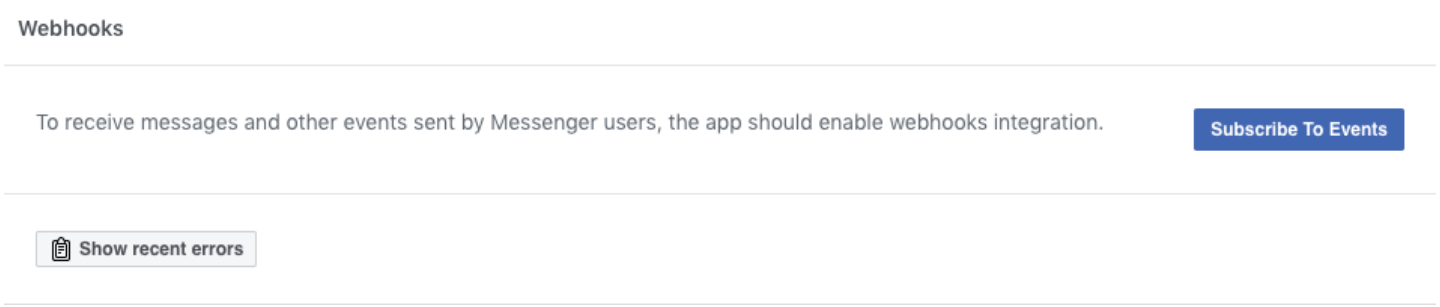
A popup will appear.



In here you will paste the access token you have generated in the Facebook app. And verify token will be a string you will make up. A token that will be used in the authentication. Make something up. Not so simple is better.

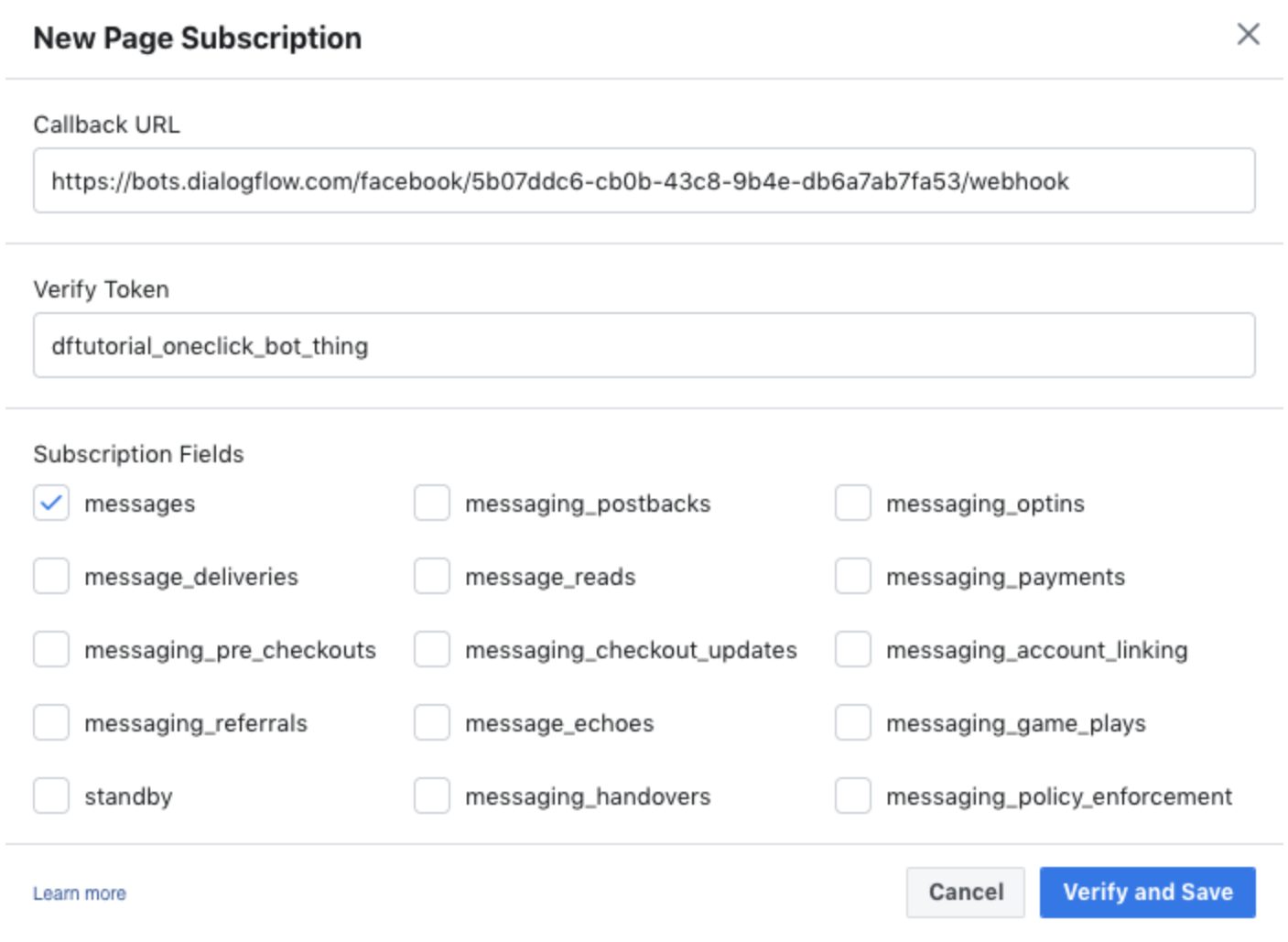


When you have done so, click on the Start button. With this, you enable the webhook on DialogFlow. Remember the verification token and the callback URL. You will need it now. We will go back to the Facebook app and set up a Webhook. Click the Subscribe to events besides the Webhook. That is under the Access token you generated recently.

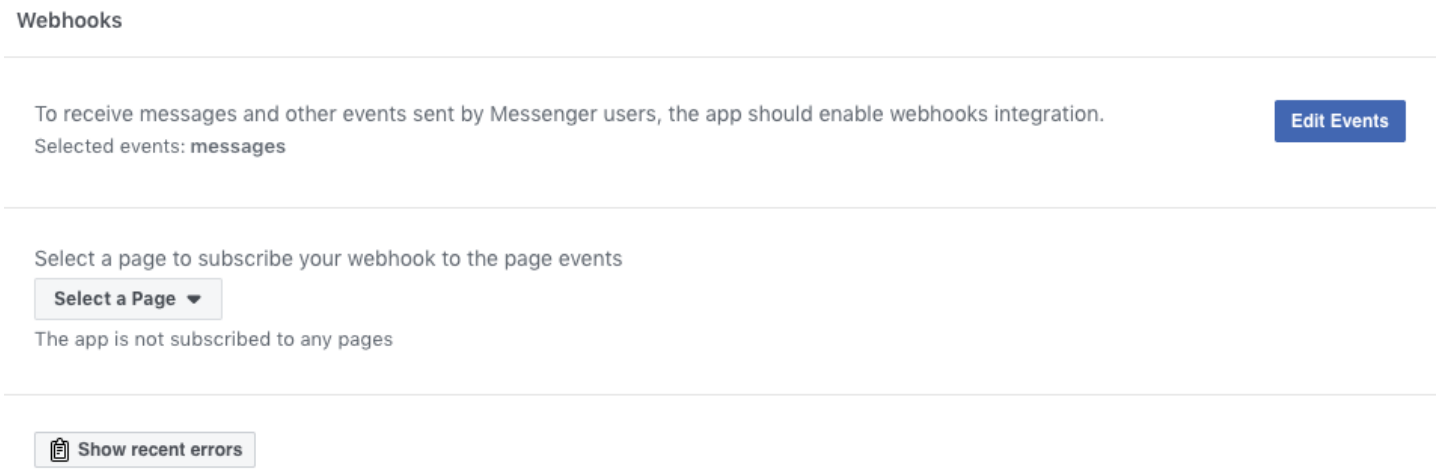


Now we will copy-paste here the Callback URL and the verify token. We want to send all the subscribed events to DialogFlow. And what we will send now is just the messages. For example, a postback is events like click on the button, optins is a optin from a page, then we have payments, account_linking, handover protocol. All sorts of events that would need to use the backend app to handle them. And this is exactly what my Udey course teaches you to do. For now, let's just send messages to DialogFlow to be parsed and answered.

After you enter callback URL, verify token and subscribe to messages then you can click the Verify and Save button.

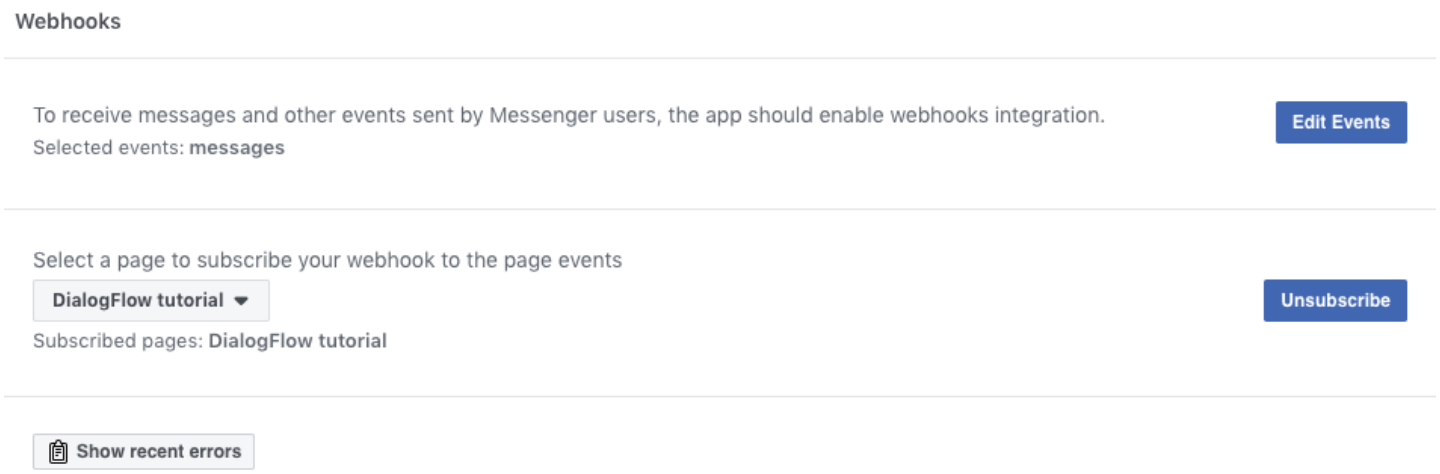


Now just connect the Facebook page.



Now just connect the Facebook page.

When you have subscribed, the subscription should look like this:



Integration with Messenger with a backend app

Integrating with a backend app as the middle man has several advantages:

1) One of them is we can now handle events that we cannot handle when we connect Facebook app straight to DialogFlow. What we can do is:

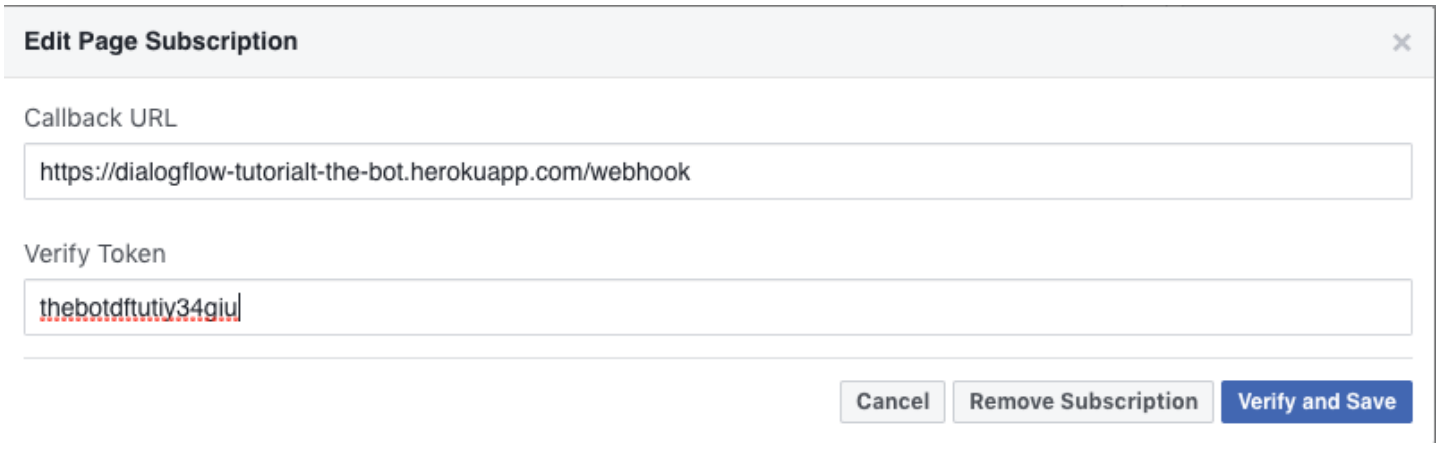
- Handle postbacks (click on the buttons)
- Do a hand off to Live agent (we can route conversation to page inbox to ba answered by a human when the bot does not know how)
- Handle payments, optins and other events
- Send subscription messages to subscribed users

The other benefit is that with a backend app we don't have any timeout as we do with fulfillment.

Let me show you how to connect a backend app with Messenger. In this tutorial, I assume you already have a backend app. If you want to learn more about how to make a backend app that would know how to do all the above, I recommend taking my course on creating chatbots for Messenger on Udemy. To all the readers I also give a discount.

You backend app can be hosted anywhere on the Web and be written in any programming language, but it must know how to communicate with the Facebook app. Since backend app is nothing but a web service that uses JSON for sending information this is possible. In my case, I will use node.js as a programming language and Heroku for hosting the app. In the app, I must have a webhook authentication and handling of sent events working. Detailed instructions are in my course.

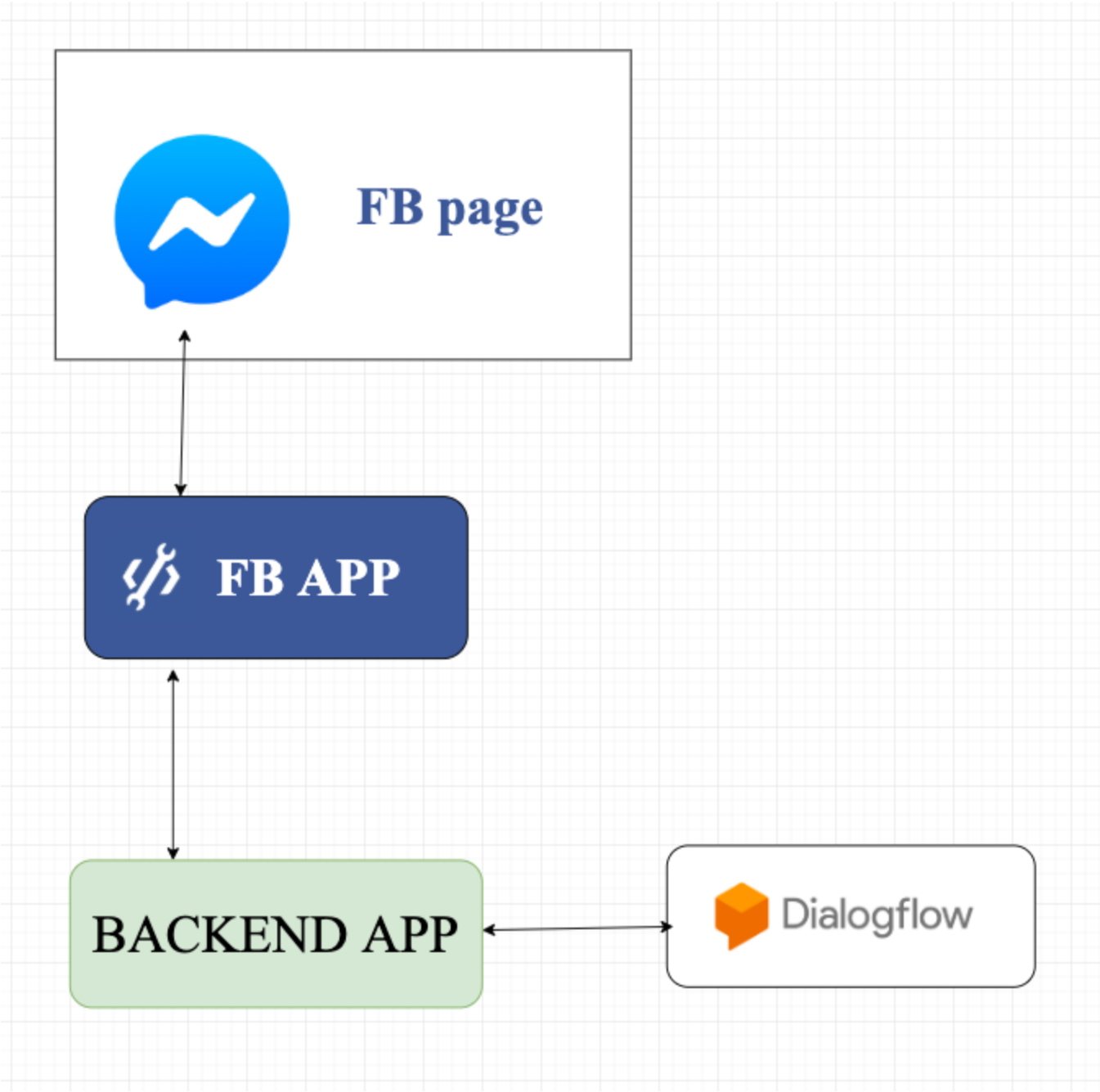
Webhook verification would look like this:



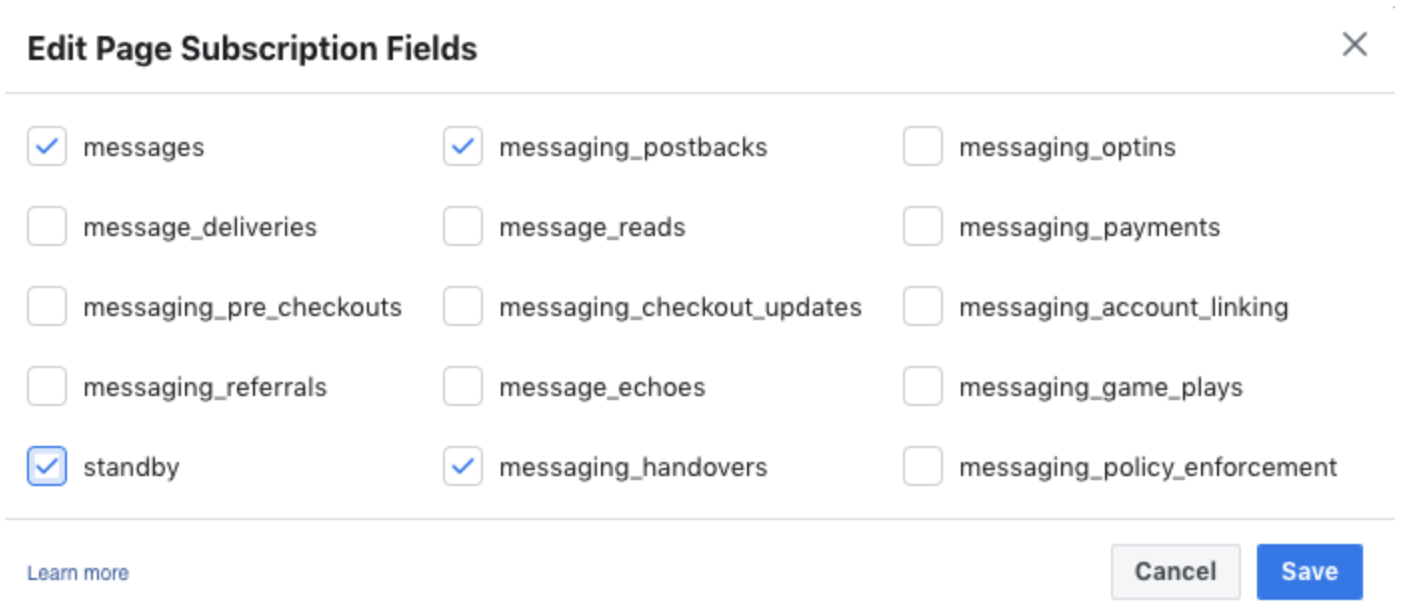
And this is how the webhook for handling events would look like:

```
app.post('/webhook/', function (req, res) {
    var data = req.body;
    console.log(JSON.stringify(data));

    // Make sure this is a page subscription
    if (data.object == 'page') {
        // Iterate over each entry
        // There may be multiple if batched
        data.entry.forEach(function (pageEntry) {
            var pageID = pageEntry.id;
            var timeOfEvent = pageEntry.time;

            // Secondary Receiver is in control - listen on standby channel
            if (pageEntry.standby) {
                // iterate webhook events from standby channel
                pageEntry.standby.forEach(event => {
                    const psid = event.sender.id;
                    const message = event.message;
                    console.log('message from: ', psid);
                    console.log('message to inbox: ', message);
                });
            }

            // Bot is in control - listen for messages
            if (pageEntry.messaging) {
                // Iterate over each messaging event
                pageEntry.messaging.forEach(function (messagingEvent) {
                    if (messagingEvent.optin) {
                        fbService.receivedAuthentication(messagingEvent);
                    } else if (messagingEvent.message) {
                        receivedMessage(messagingEvent);
                    } else if (messagingEvent.delivery) {
                        fbService.receivedDeliveryConfirmation(messagingEvent);
                    } else if (messagingEvent.postback) {
                        receivedPostback(messagingEvent);
                    } else if (messagingEvent.read) {
                        fbService.receivedMessageRead(messagingEvent);
                    } else if (messagingEvent.account_linking) {
                        fbService.receivedAccountLink(messagingEvent);
                    } else if (messagingEvent.pass_thread_control) {
                        // do something with the metadata: messagingEvent.pass_thread_control.metadata
                    } else {
                        console.log("Webhook received unknown messagingEvent: ", messagingEvent);
                    }
                });
            }
        });

        // Assume all went well.
        // You must send back a 200, within 20 seconds
        res.sendStatus(200);
    }
});
```

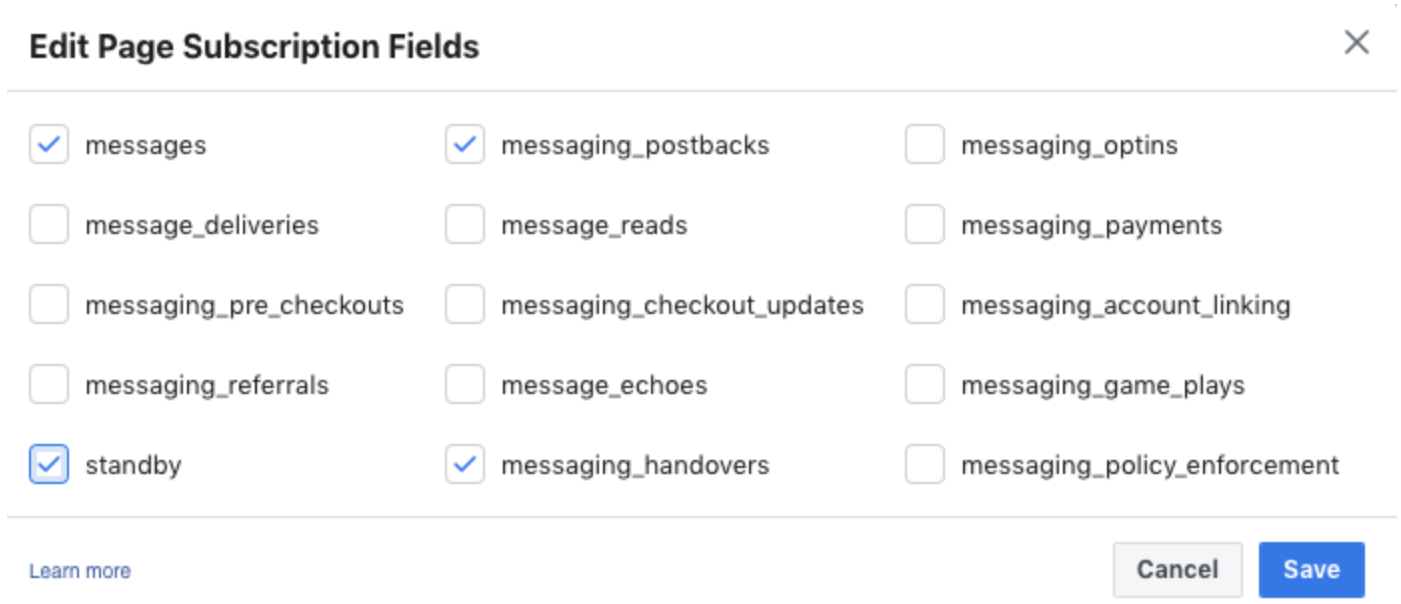
Now we will reroute the connection from DialogFlow to Backend app. And backend app will be calling DialogFlow's detect Intent API to parse natural language. The flow will be like this:



When your app is deployed or ready you can go to Facebook developer console to Products -> Webhooks and select Edit subscription. In here we will now add a link to endpoint in our app that handles messages and authentication. Verify token must also be used in the app to handle authentication, as we can see in the code for Webhook verification.

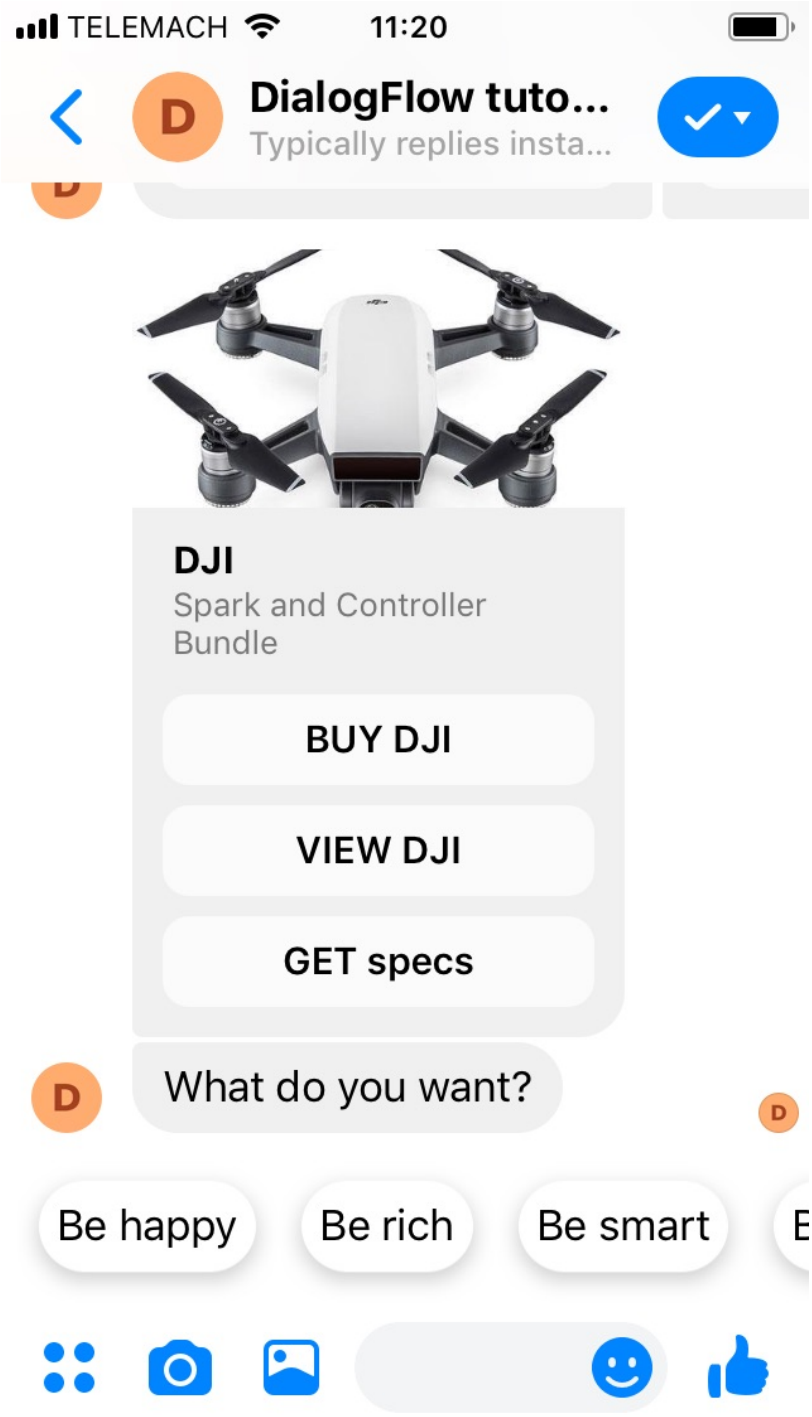


And now, since we have an app that knows how to handle postbacks, handovers, account linking... we can now subscribe the webhook to these events. Therefore go to Webhook and click Edit Events button and subscribe to events you want your app to watch for and handle.



And this is it. To learn how to create this backend app I invite you to take my course.

Here is how the bot on Messenger looks like:



Now let's see how to integrate to other platforms too.

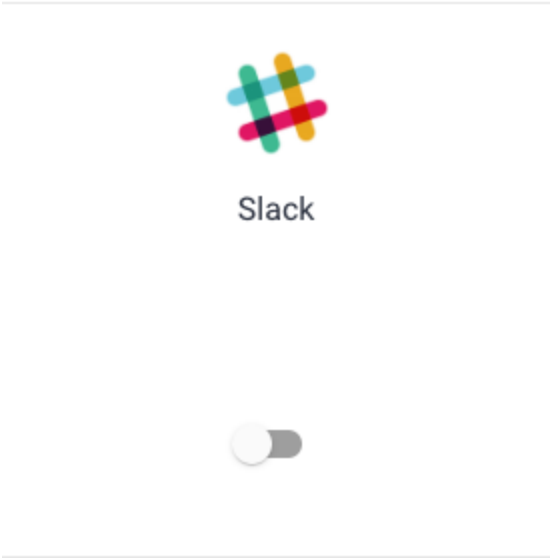
One-click integration with Slack

Integrating with Slack is quite simple. First, let's do a one-click Slack integration. We are going to create a Slack app and connect it straight to DialogFlow.

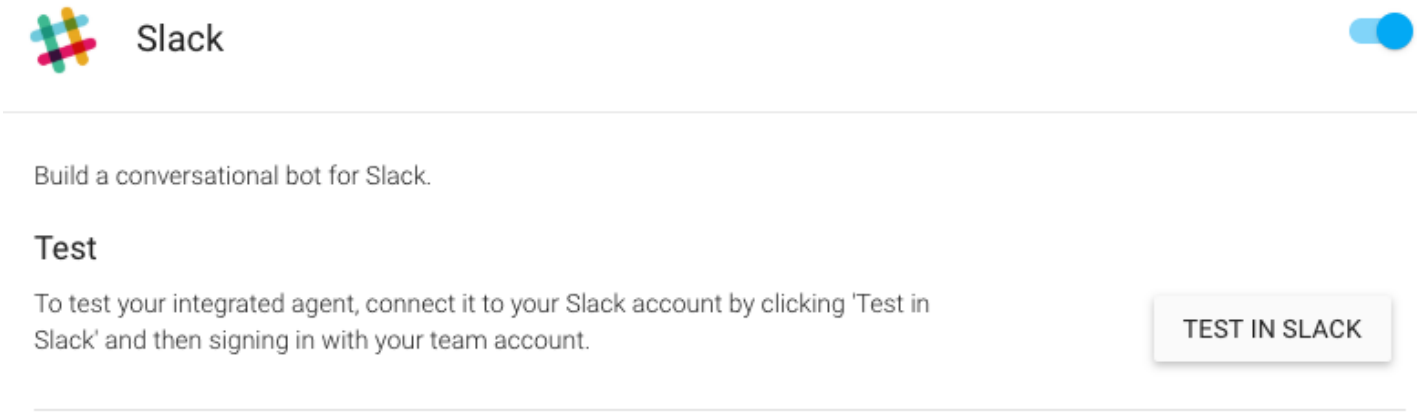
To do so we'll need to do these steps:

- Create an app
- Add bot users
- Enable webhook
- Add events subscription
- Manage distribution of a Clack app/bot

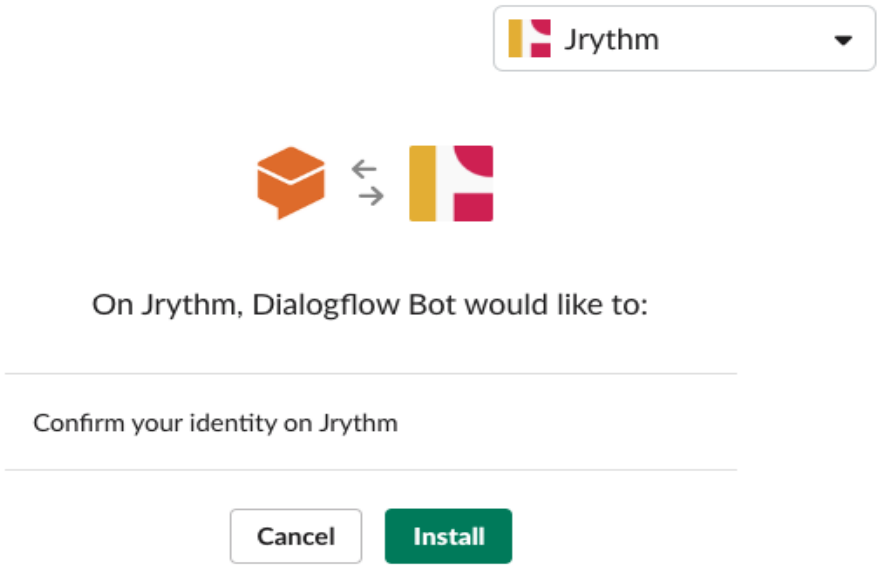
If you go to Integrations of DialogFlow you will see Slack as one of the possible integrations.



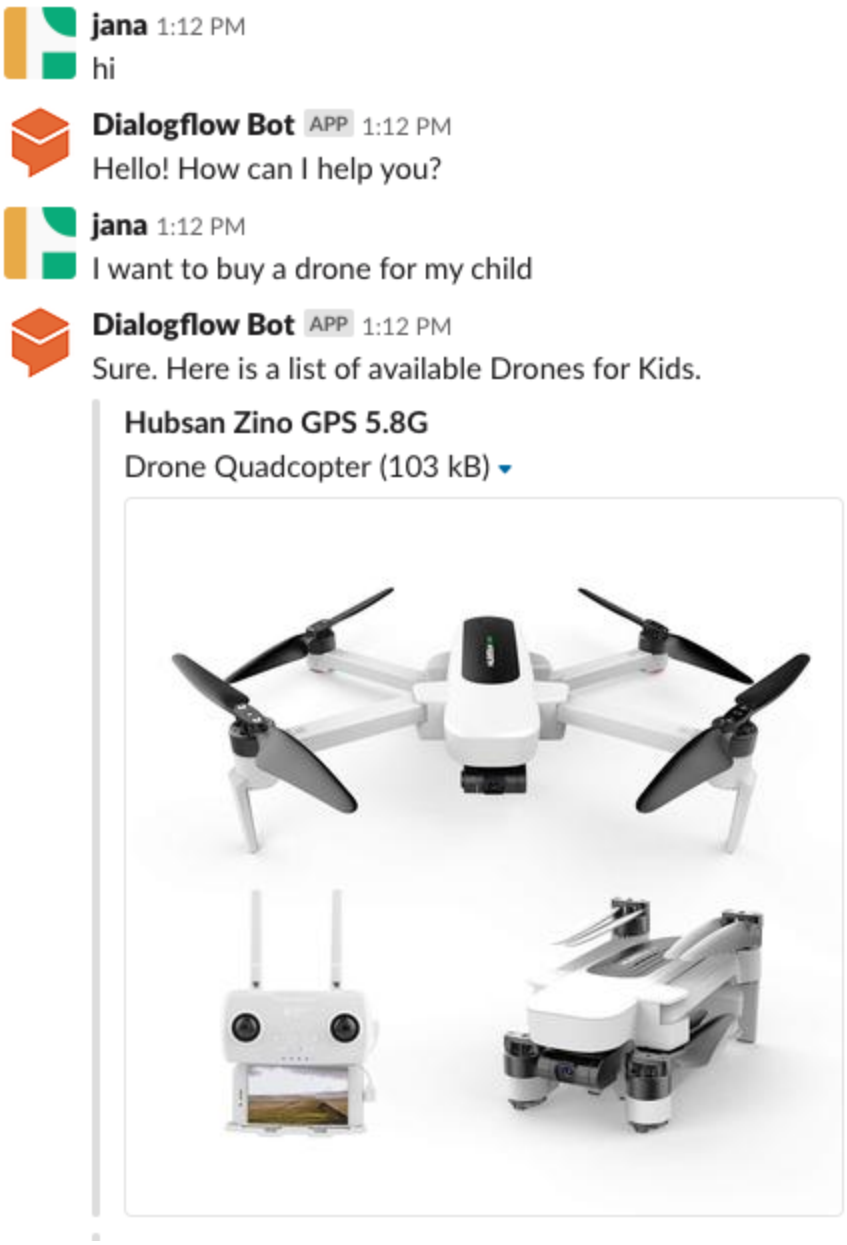
DialogFlow has very good instructions that you can simply follow. What is also cool is that for testing a Slack bot, you only need to click the Test in Slack button:



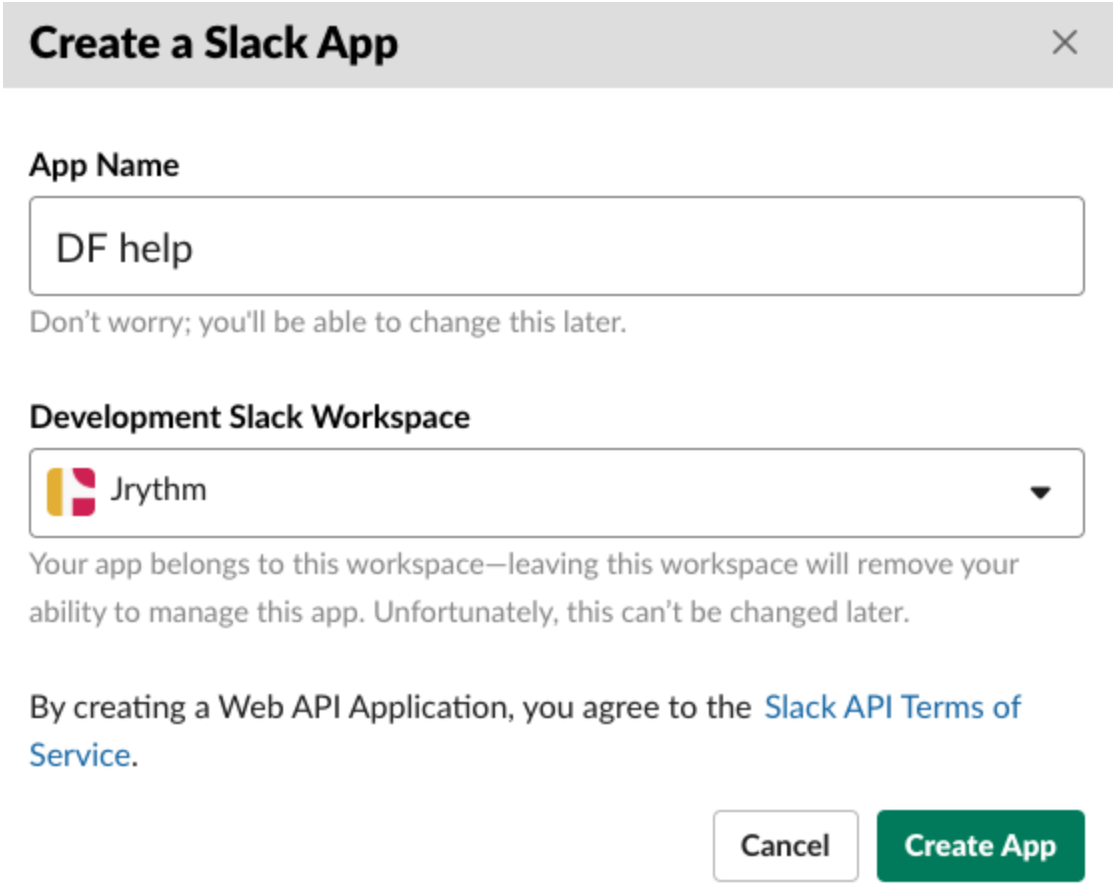
Make sure you select the right workspace to install your slack bot to:



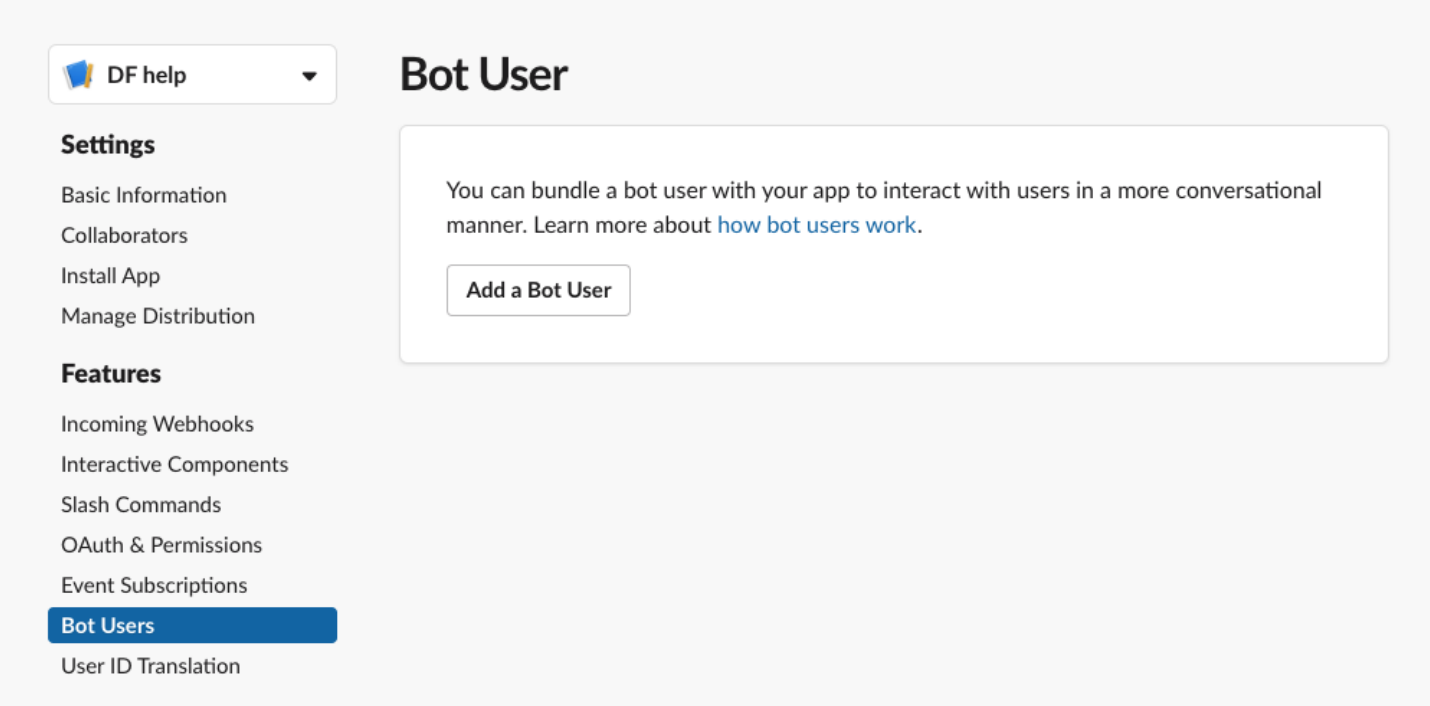
And we can now communicate with the bot:



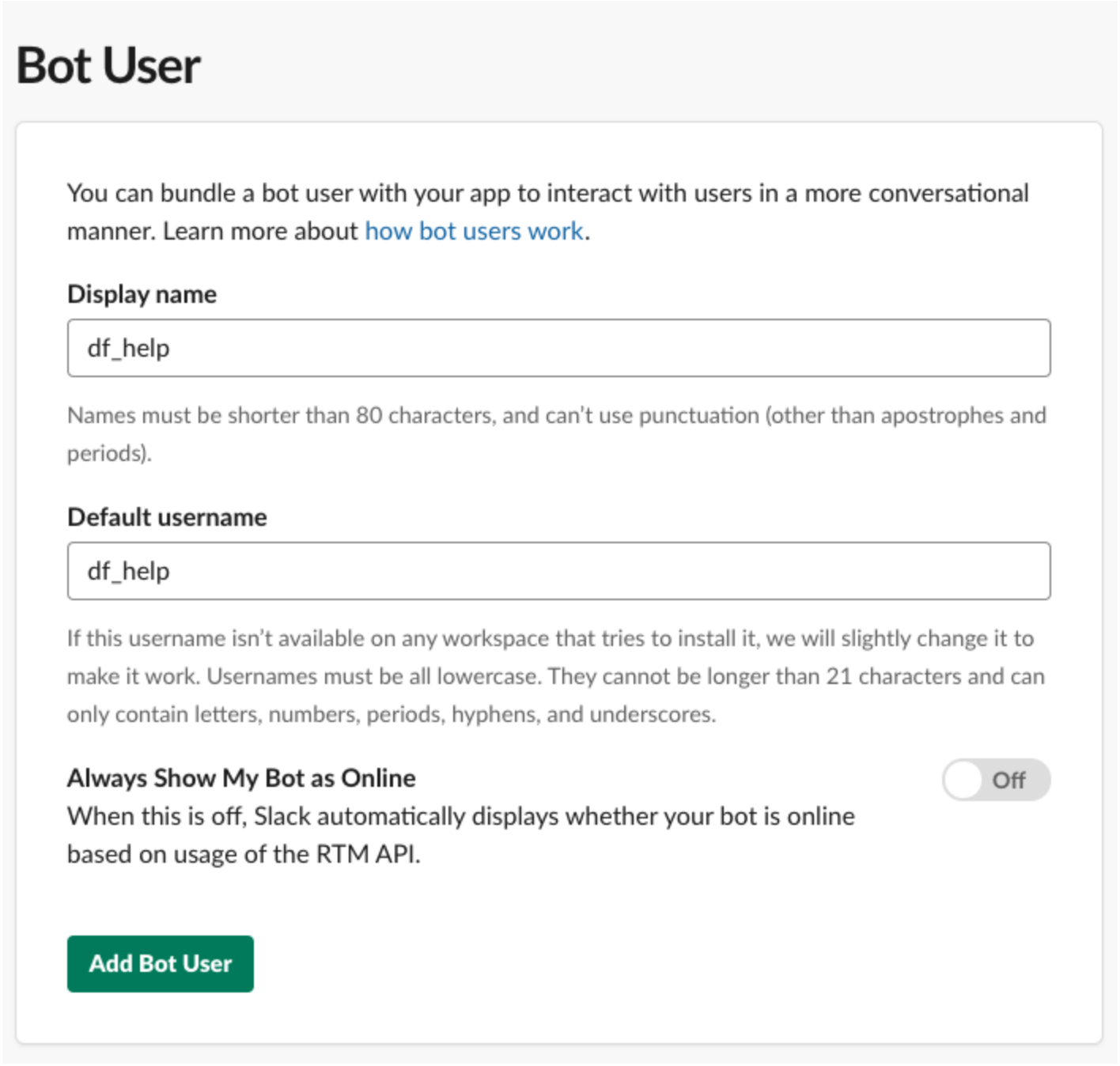
When we are ready to publish it we can create a new app for Slack. Choose app's name and the workspace:



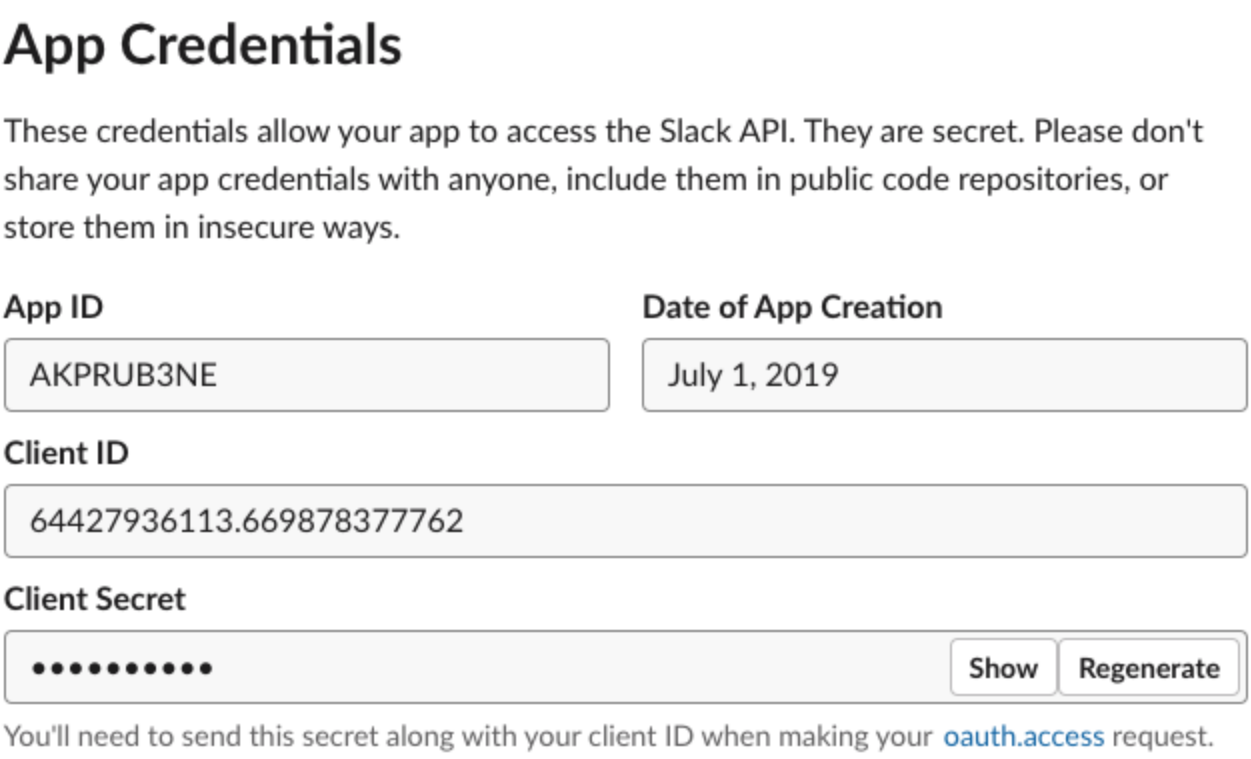
In the bot users you can now add a user.



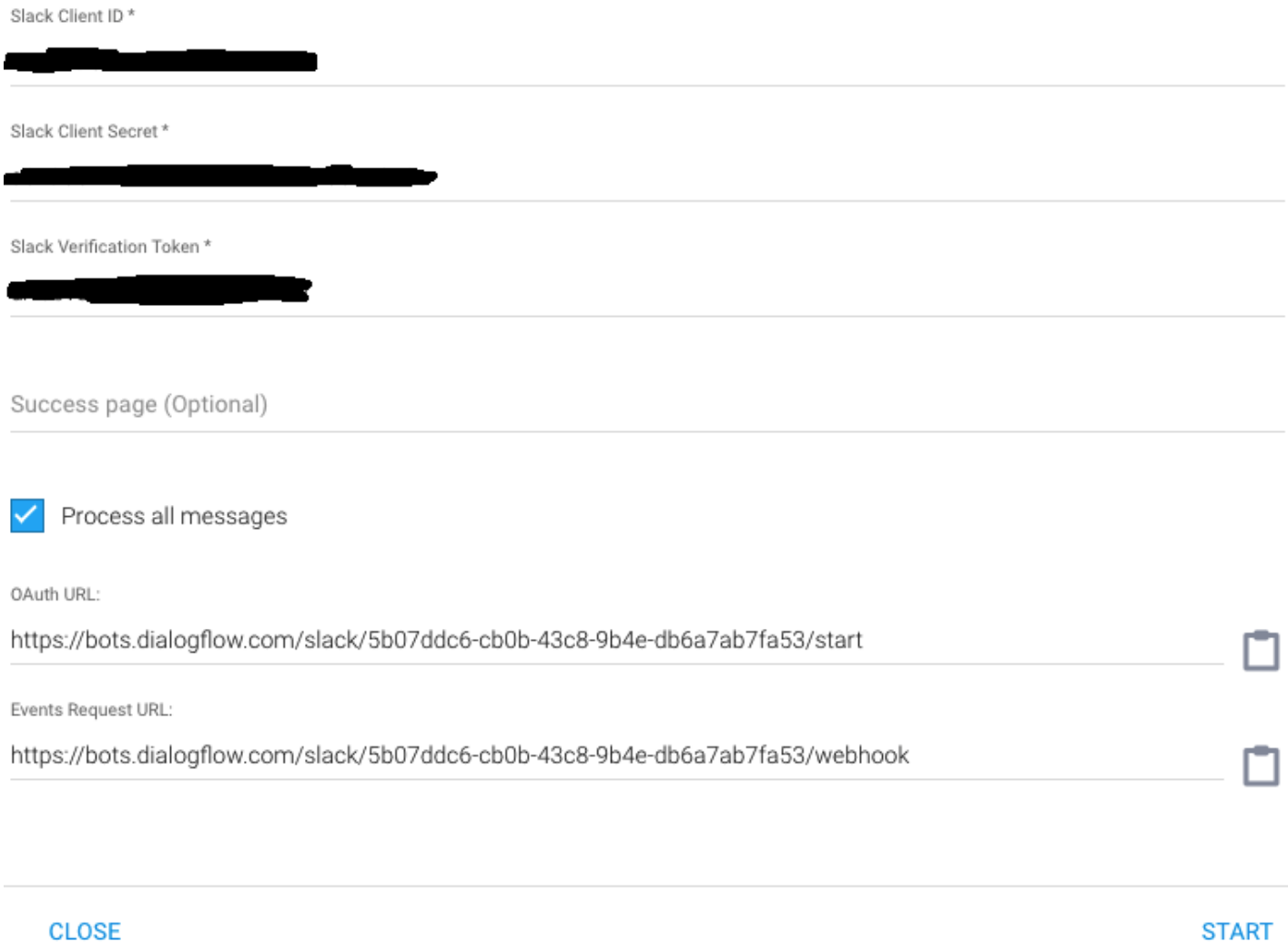
Here you can change the bot's display name. You can change it after if you change your mind. Click the Add Bot User button.



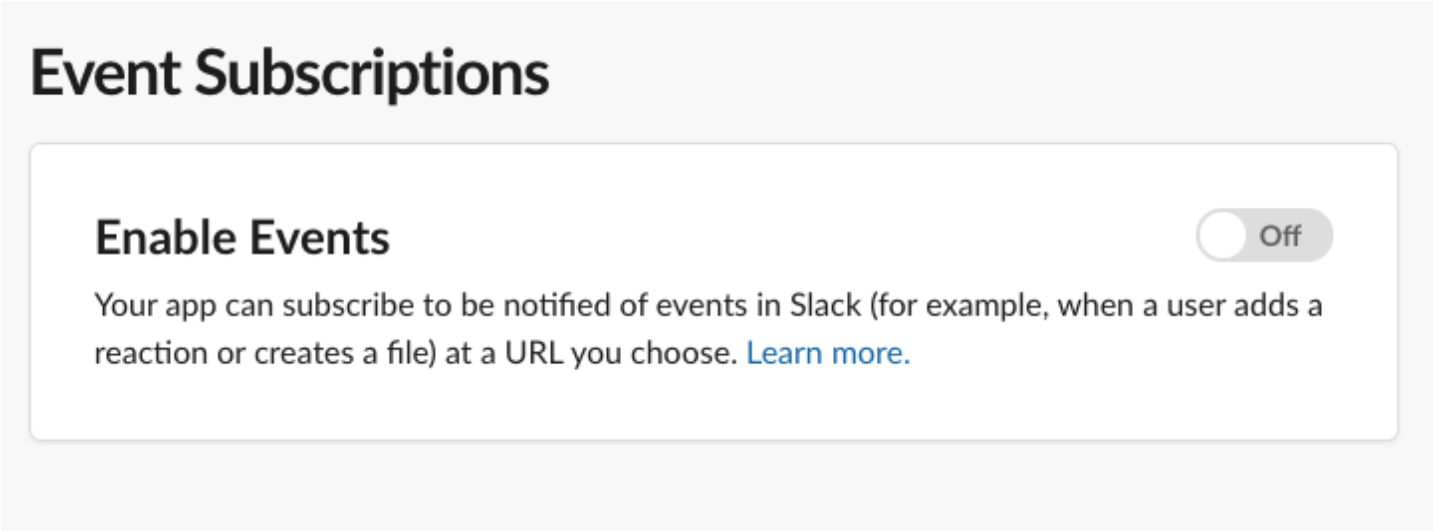
In the Slack app 'Basic Information' section, copy the 'Client ID' and 'Client Secret' and paste their values into DialogFlow's interface.



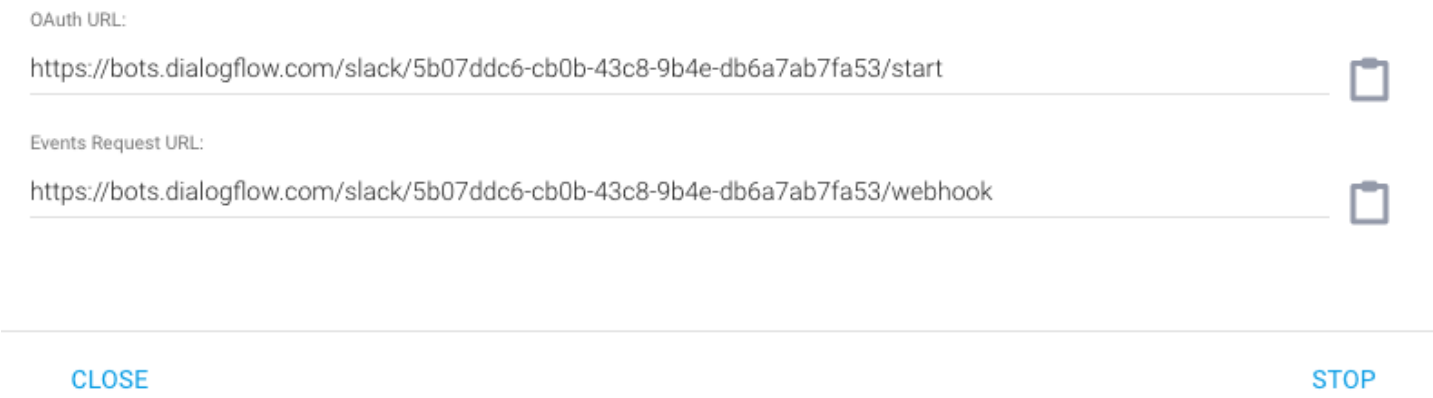
To the same with verification token and click the start button.



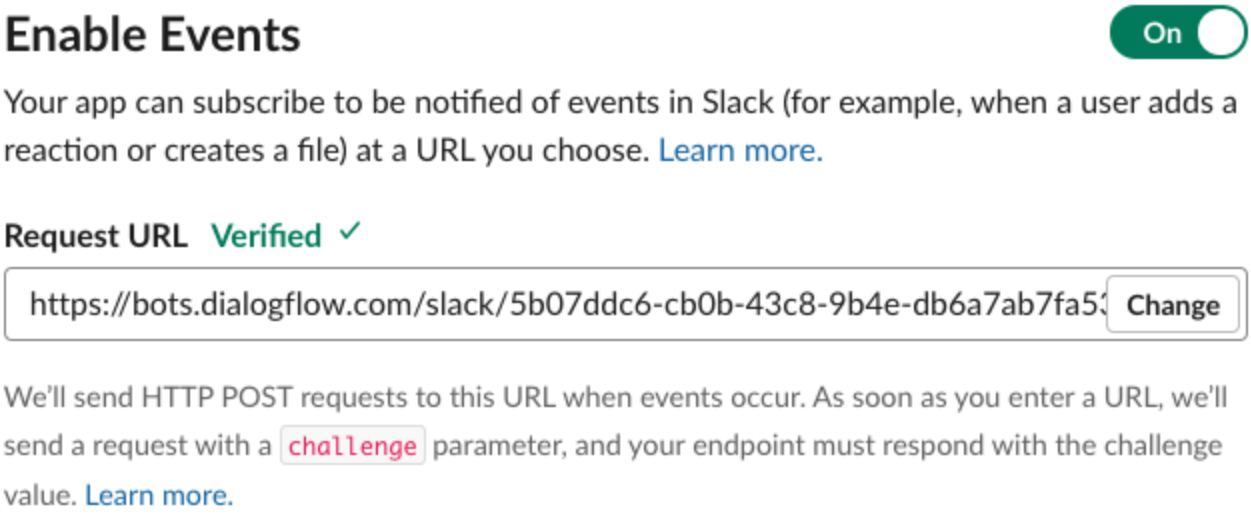
Now go to Events Subscription and enable Events. What we will do is subscribe the app to specific events like when someone starts talking to the bot :)



And now you can copy the Event's request URL from DialogFlows interface.



to Slack subscription. When you paste it there the URL should be verified. If you see this green Verified sign, you have a winner.



A bot can respond to many events that happen in Slack collaboration. Here is a list of all possible events:

https://api.slack.com/events/api

The bot can respond to a message posted in a private channel- Or when you send a direct message to the bot or when somebody mentions your bot. In the Subscribe to Bot events, you can choose the events you want the bot to respond to.

For example, like this:

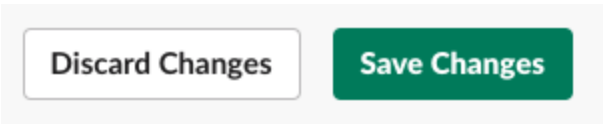
Subscribe to Bot Events

Bot users can subscribe to events related to the channels and conversations they're part of.

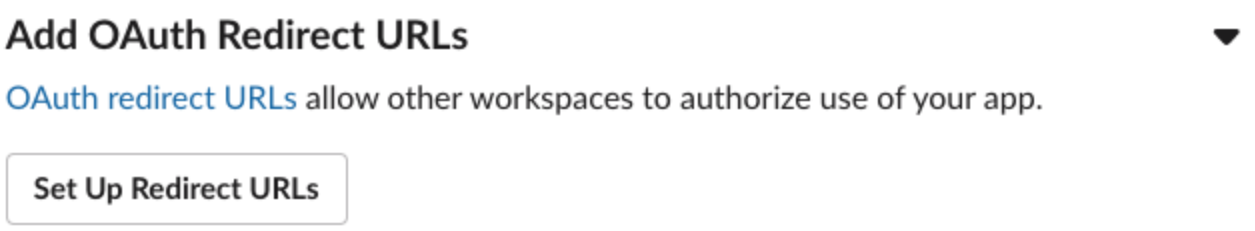
Event Name	Description
app_mention	Subscribe to only the message events that mention your app or bot
im_created	A DM was created
im_open	You opened a DM
im_close	You closed a DM
message.im	A message was posted in a direct message channel
message.groups	A message was posted to a private channel
message.channels	A message was posted to a channel

Add Bot User Event

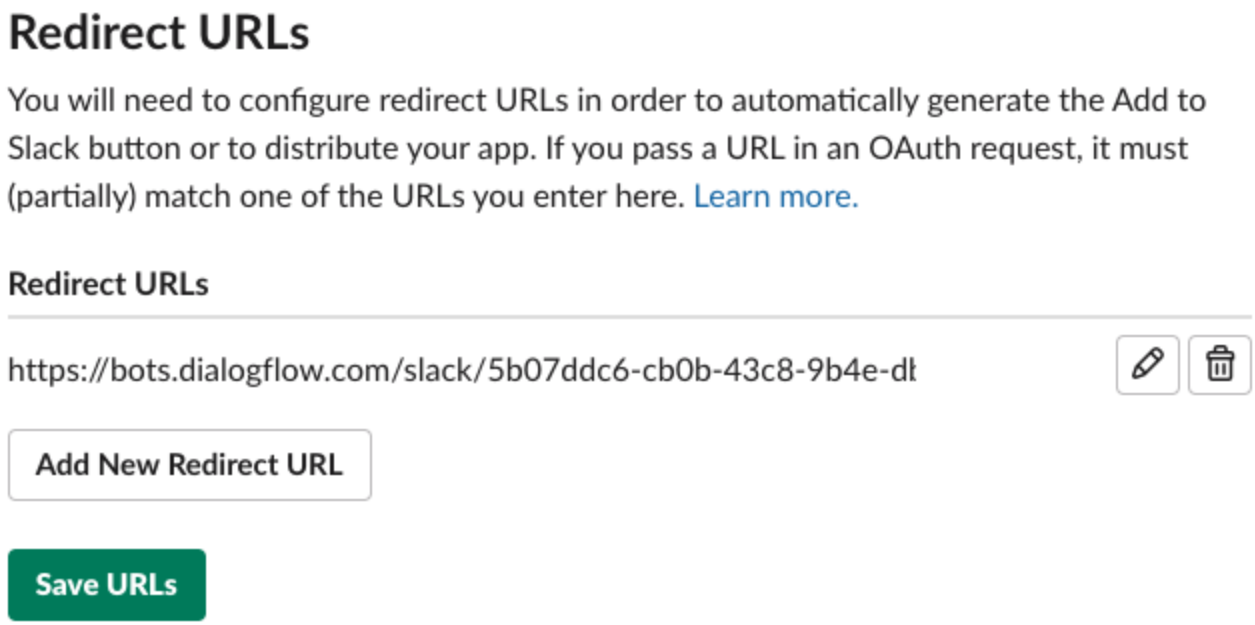
Be sure to put them under Bot Events and not Workspace Events. And do click on Save changes. It is well hidden on the bottom of the page :)



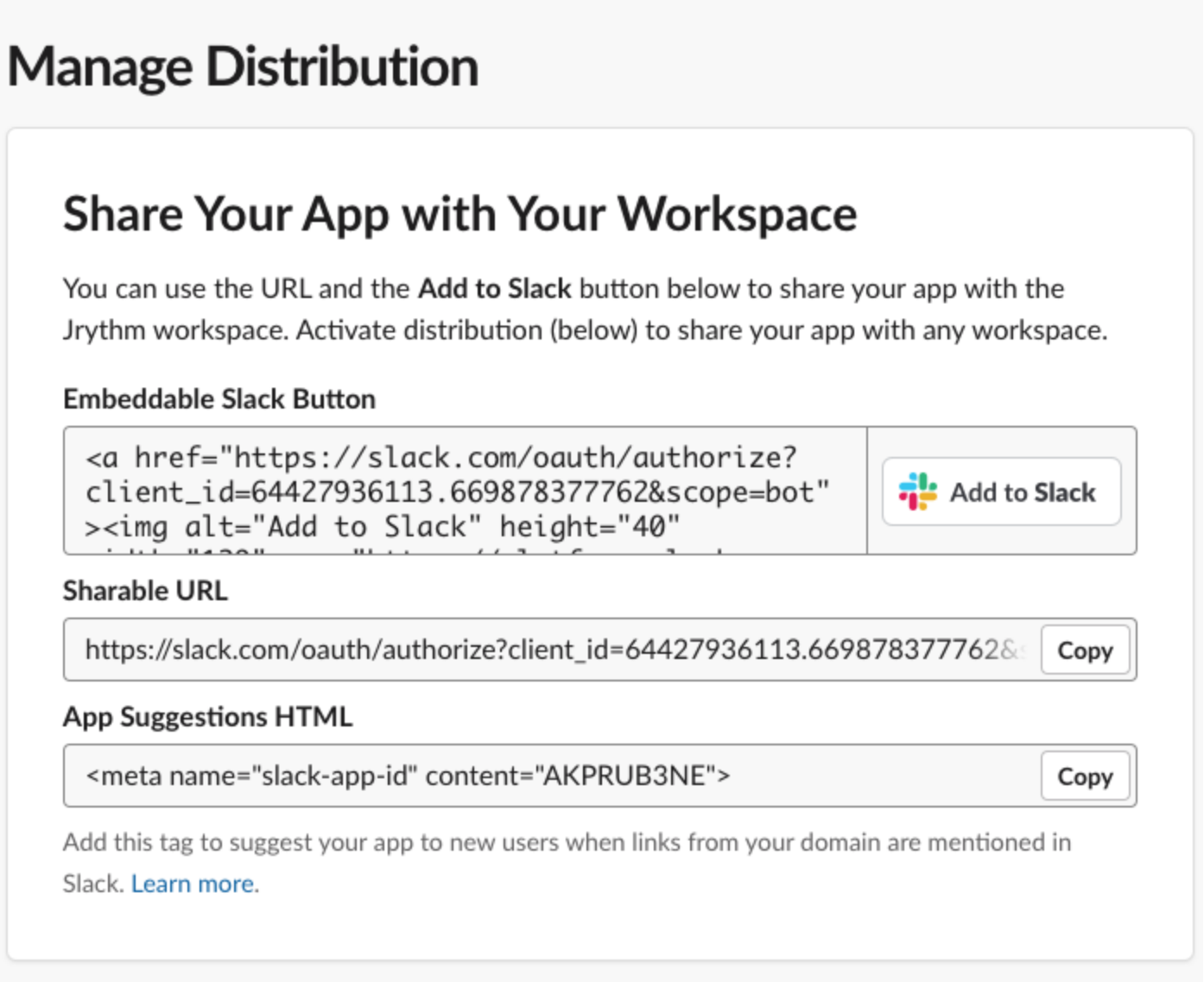
What we need to do now is enable OAuth. Go to DialogFlow and copy the OAuth URL. You will add it under the OAuth & Permissions



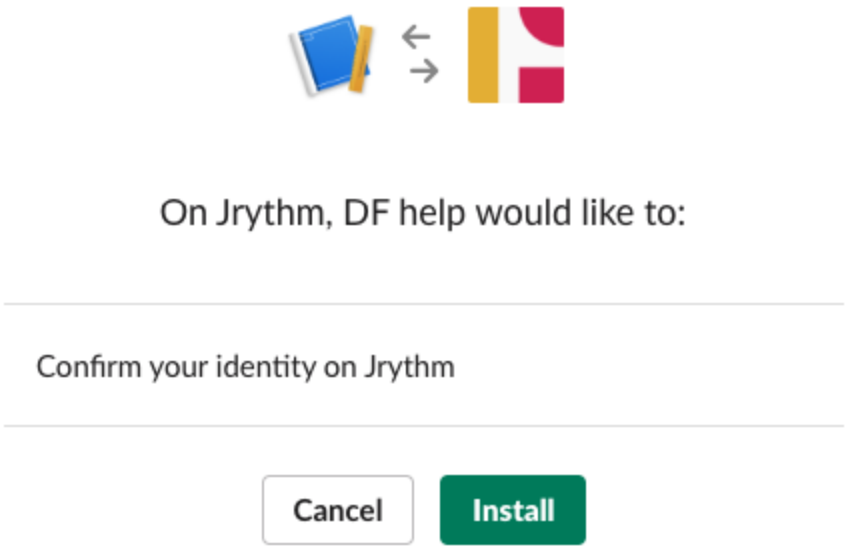
Like this:



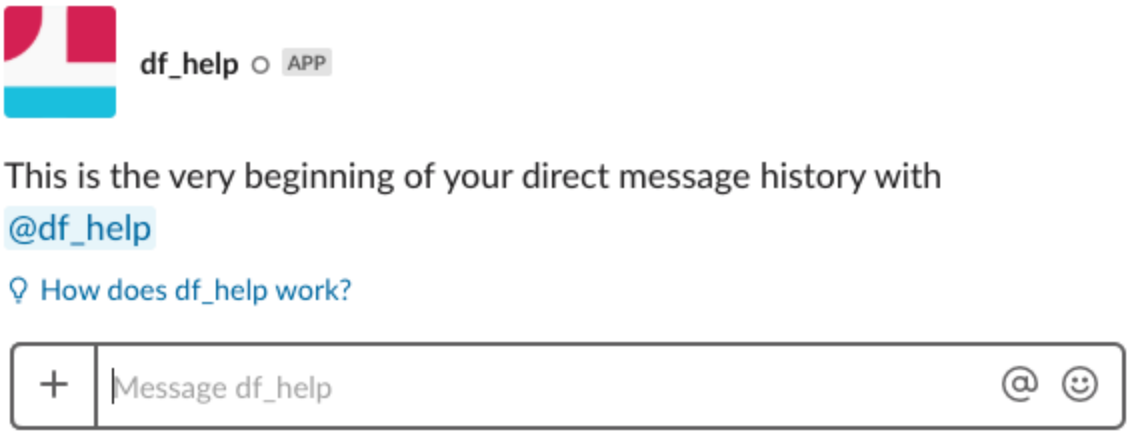
You can now distribute the app to your team. Go to manage distribution. Here you will see an add to Slack button.



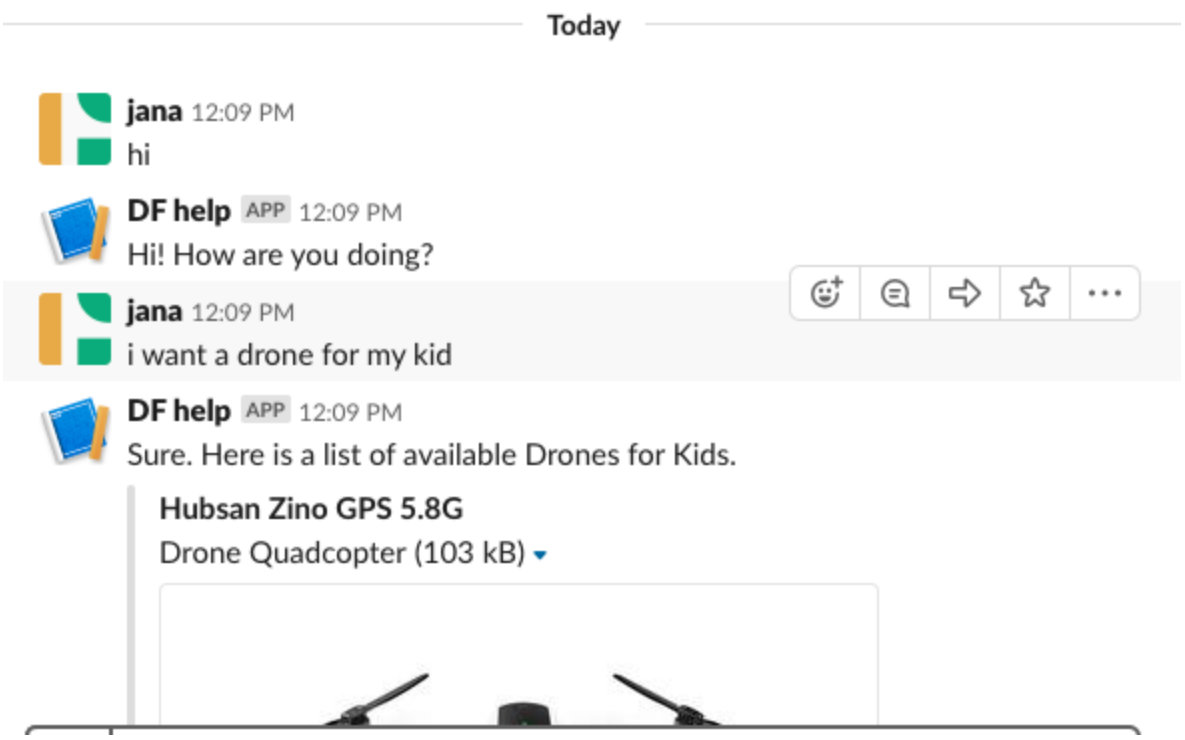
All you need now is to confirm.



And you can start communicating with your bot



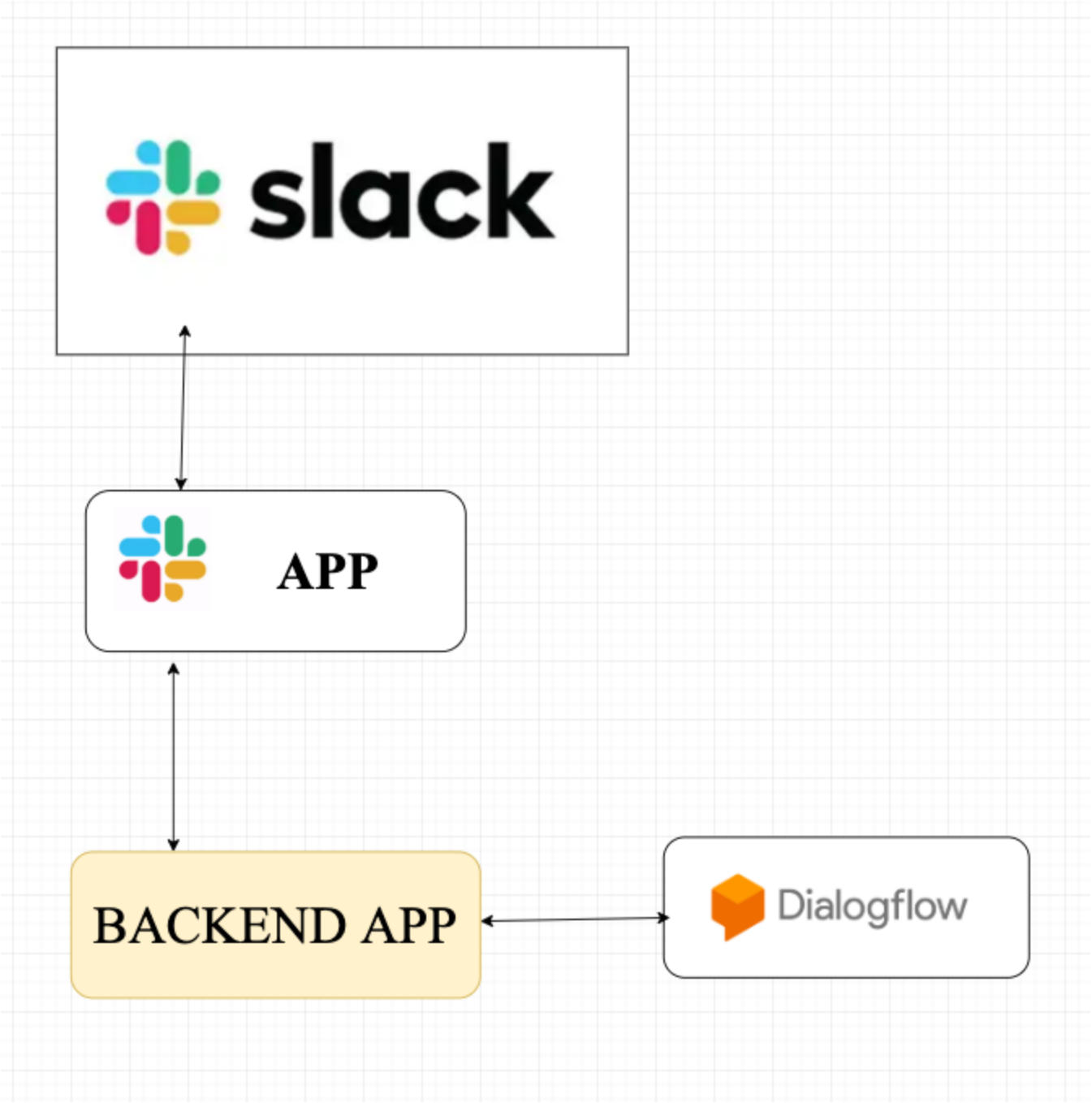
And wooohooo! The bot works on Slack.



Slack integration with a backend app

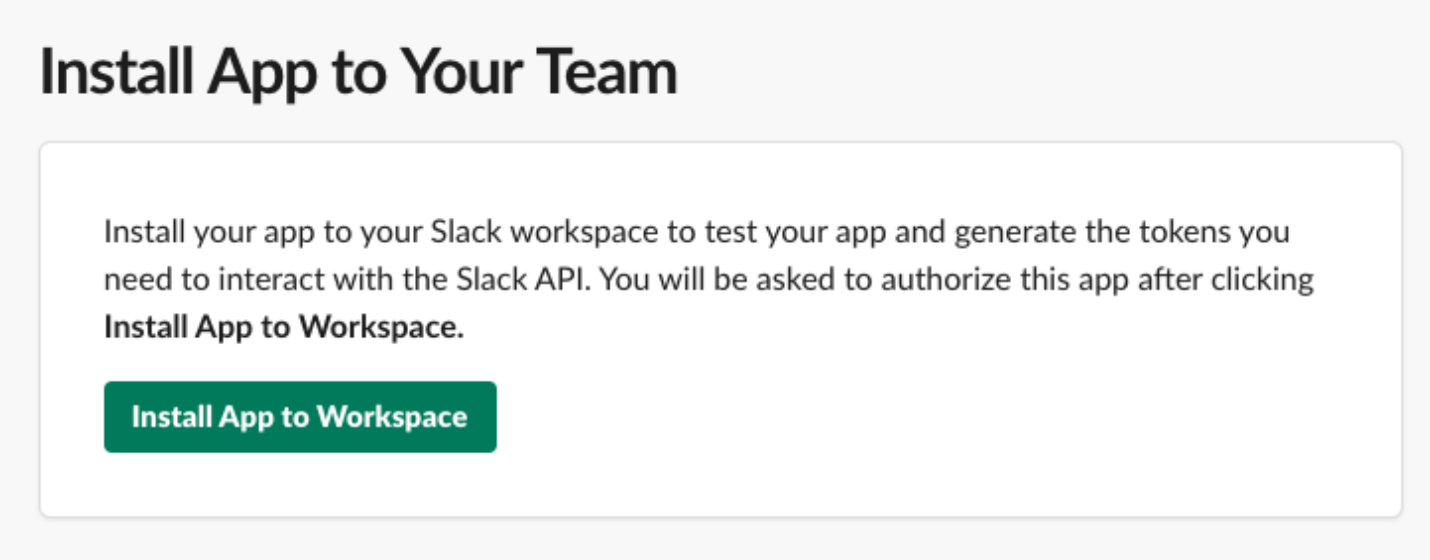
In this article, we'll add a backend app as a middle man between Slack app and DialogFlow. In this way, we can use the power of a backend app without having the timeout and other limitations of the fulfillment code. And also we can handle Slack events that do not send messages.

Here is how our integration will look like:

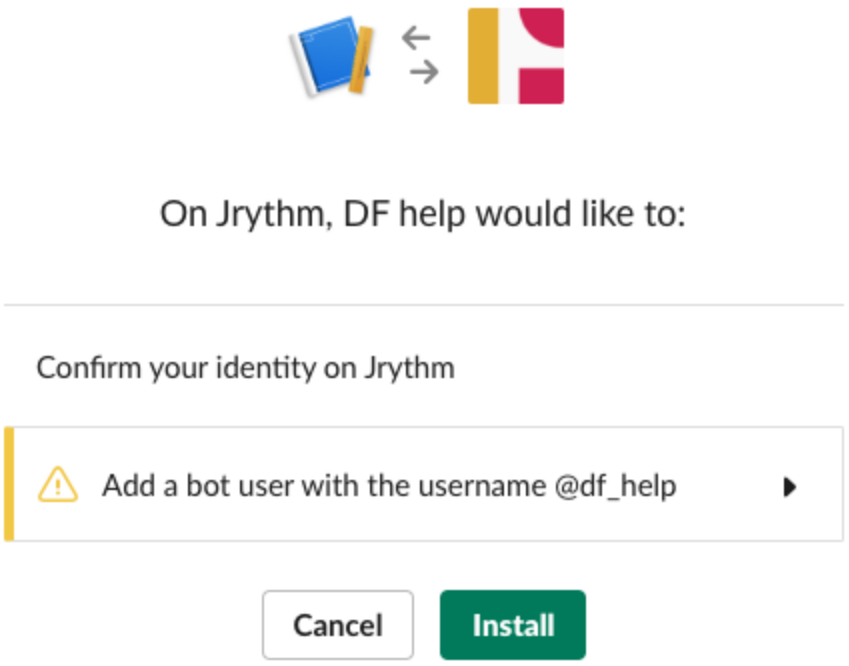


So, we have a backend app that receives events from Slack and forwards them, if the event has a text to DialogFlor. Or it can trigger an intent with event call. Or it can log it into a database. Or call a webservice. If DialogFlow is called then it waits for a response and handles extra action if needed, and returns the response back to Slack app/bot.

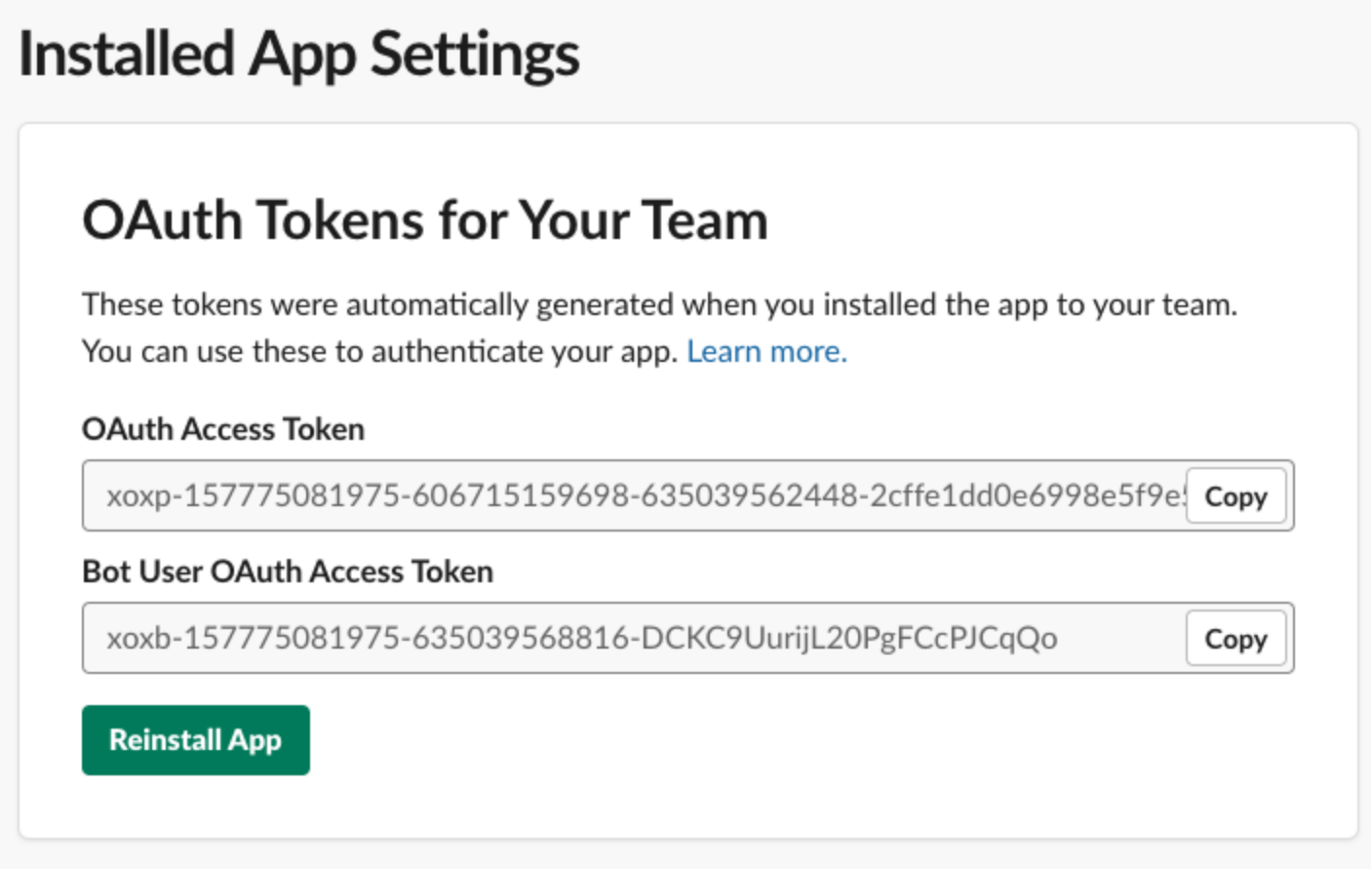
And what I need to do is write a code that will handle the authentication. For authentication, I will need to use Bot User Oauth Access Token. This token will be generated once you install the app to your Workspace. Go to Install app tab and install it.



Proceed with installation:



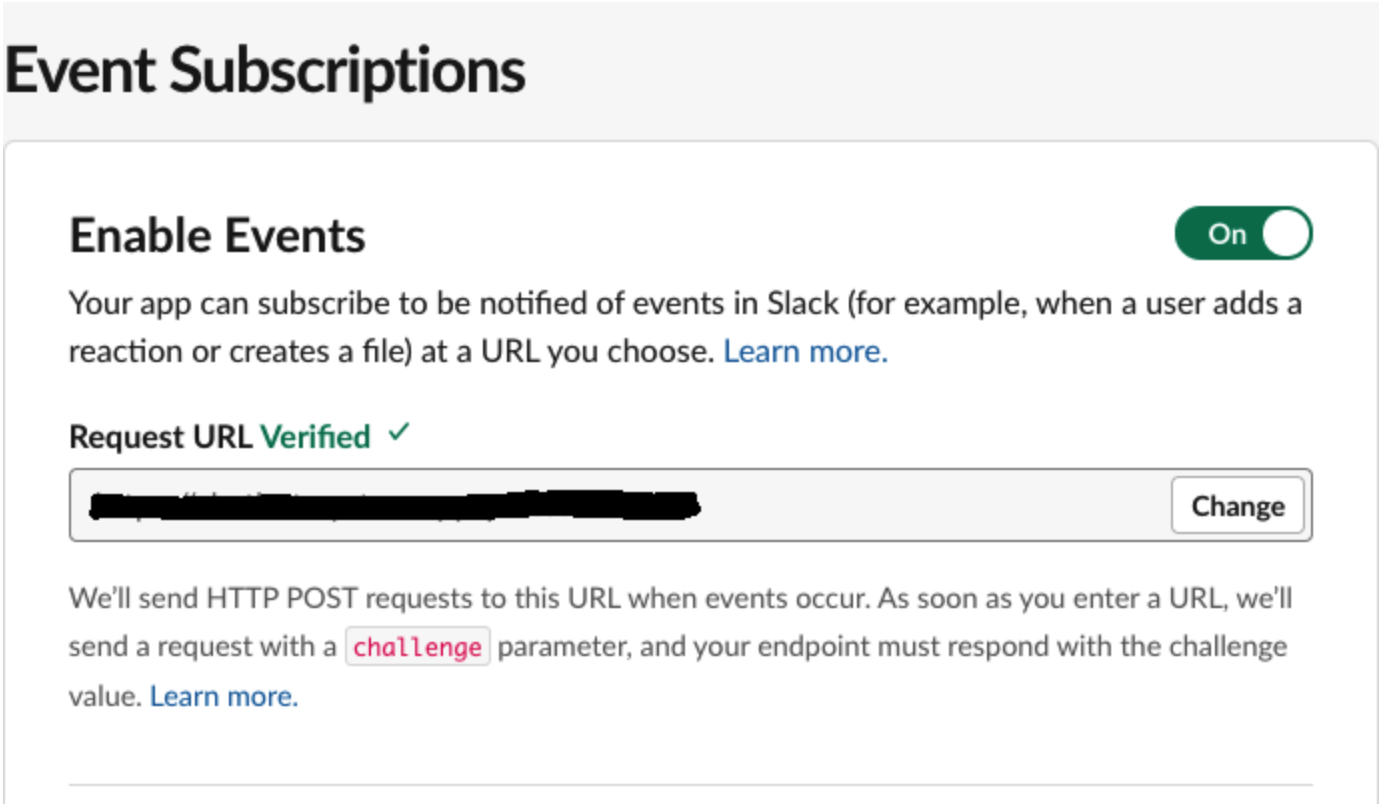
What you will need to use in your app is Bot user Oauth Access token



I am creating my app in node.js. My authentication code looks like this:

```
  /**
   * slack url validation
   */
app.post('/slack/', function (req, res) {
      let noSlackToken = (req.body.token === undefined || req.body.token === '');
      let noSlackChallenge = (req.body.challenge === undefined || req.body.challenge === '');
      let noSlackType = (req.body.type === undefined || req.body.type === '');
      if (!noSlackToken && !noSlackChallenge && !noSlackType && req.body.type === 'url_verification') {
          if ( req.body.token === config.SLACK_VERIFICATION_TOKEN ) {
              res.send({ "challenge": req.body.challenge });
          } else {
              throw new Error('Couldn\'t validate the Slack verification.');
          }
      } else {
          throw new Error('Couldn\'t validate the Slack verification.');
      }
  });
```

Once your app is up and running you can enter the URL to Events subscription. The endpoint should be Verified.



And this is it. The bot is again working. This time with a backend app.

DF help APP 2:37 PM
Good day! What can I do for you today?

jana 8:43 PM
I want to buy a mini drone

DF help APP 8:43 PM
Sure. Here is a list of available Mini Drone.

Hubsan Zino GPS 5.8G
Drone Quadcopter (103 kB)

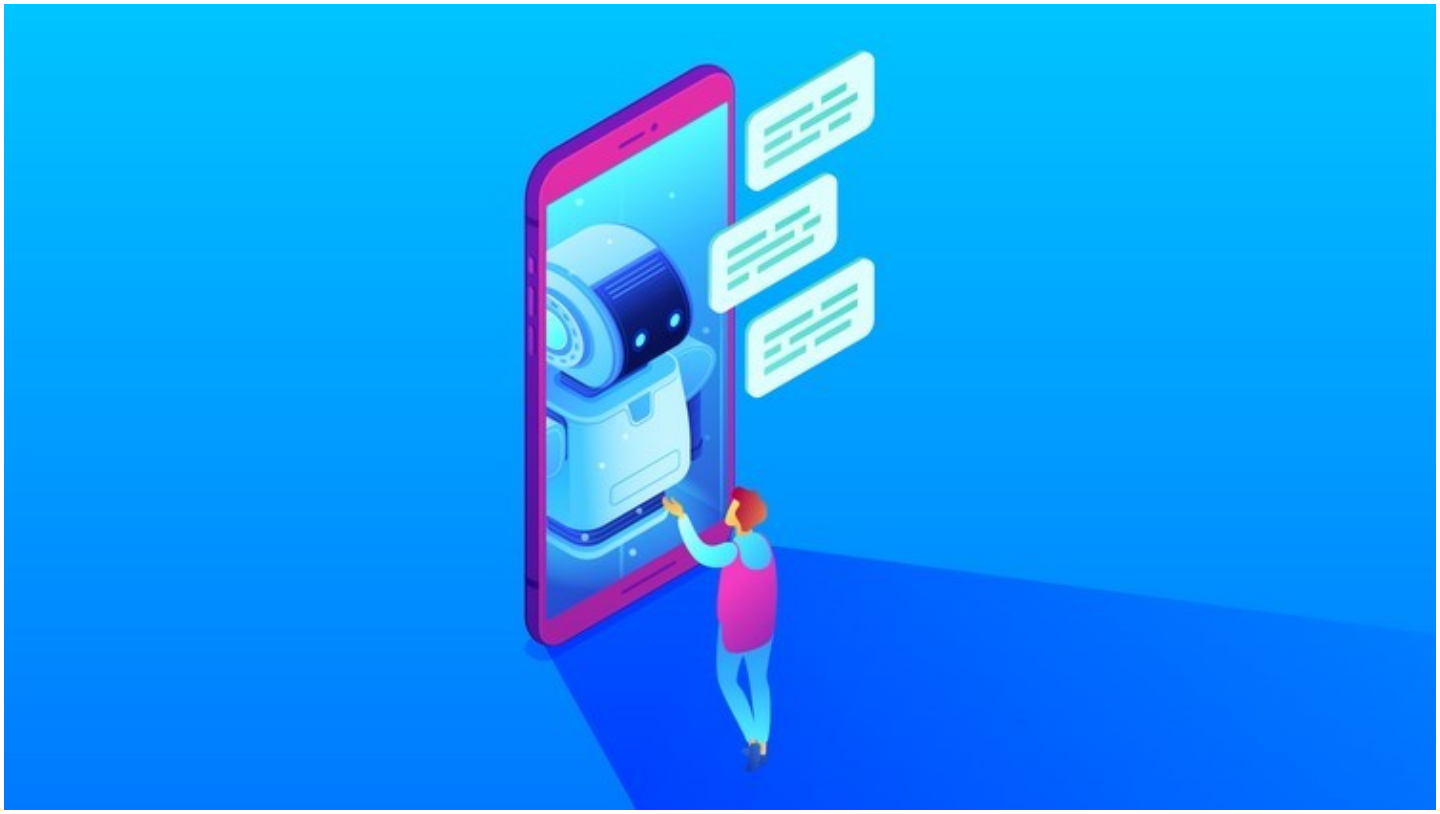
Get in touch

Subscribe to my news by talking to my chatbot.

If you liked my DialogFlow tutorial, you could connect with me on LinkedIn.

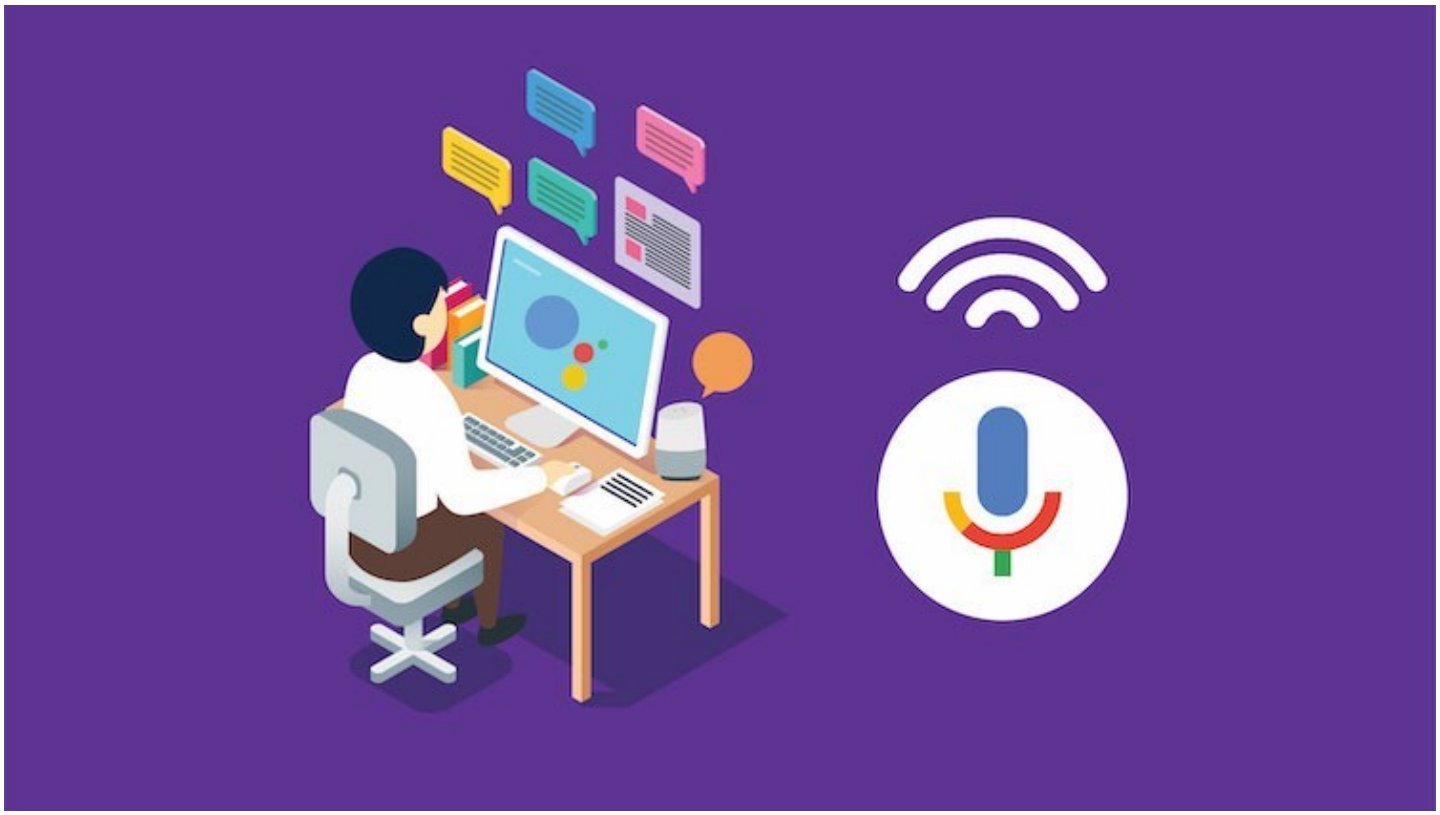
Or you can follow me on Medium.

You are welcome to sign up for my Messenger chatbot course. In the course, you will learn all there is to know about creating chatbots for Messenger, including a section about WebViews, Handoff protocol, and many more Messenger related topics not discussed in this tutorial. Here is a link with a 93% DISCOUNT.

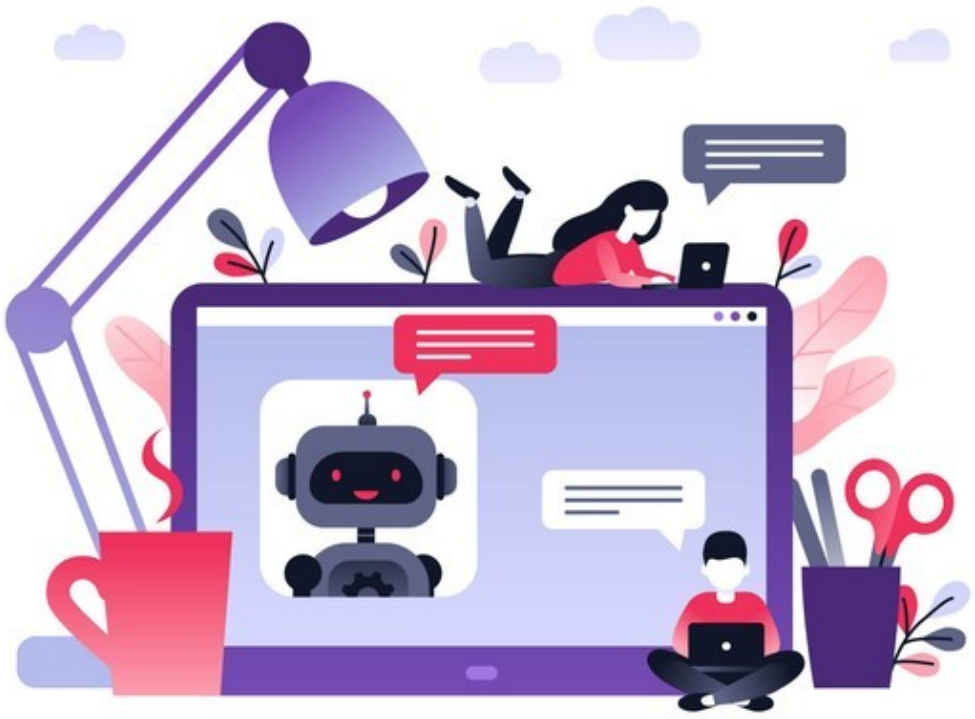


In the course "Introduction to voice interface design" you'll learn all about designing conversations for chatbots and voice bots.

I will guide you through the most common misconceptions of voice and chatbot design. And I'll introduce SOLUTIONS. Here is a link with a 93% DISCOUNT.



In the course Chatbots for a website, you will build a chatbot for a Webpage. We'll use Node.js and React for programming and GIT for deploying and version control. The bot will is on Heroku, but you can host it anywhere else where they support Node.js. We'll use DialogFlow to process natural language. Here is a link with a 93% DISCOUNT.



Actions on Google course will show you how to create an app for the Google Assistant and Google Home in Node.js. You'll learn about Google Assistant development from scratch. Here is a link with a 93% DISCOUNT

For all of you who are not familiar with Javascript programming here is a simple introductory course on programming in Javascript. Here is a link with a 93% DISCOUNT.