iNeuron

# Building Chatbot using Google Dialogflow

iNeuron

iNeuron

## Contents

iNeuron

## About This Guide

This Guide makes you familiar with the concept of chatbots. We would discuss what chatbot is, how it works, and what exactly is the need for it in today's era? It focuses on creating a bot using Google Dialogflow, and getting the bot deployed on social media channels like Facebook, twitter, telegram, slack for live chatting. This Guide will train you on how to create a chatbot using Google's Dialogflow and test the bot in Dialogflow console. It provides information on how to enable the webhooks in Dialogflow Fulfilment and return the response from the bot to Dialogflow intents as a fulfillment response.

Commented [vS1]: Remove plagiarism. Currently it's about 40% which is unacceptable.

iNeuron

# 1.Chatbot Concept

## 1.1 What are chatbots?

A Chatbot is a software or an agent or a service which simulates human conversation in natural language through messaging applications, websites, mobile phones, or telephone. They can be programmed to respond to simple keywords or prompts to complex discussions. A Chatbot has two different tasks at the core: analyze the request (identifying the intent of the user) and providing the response.

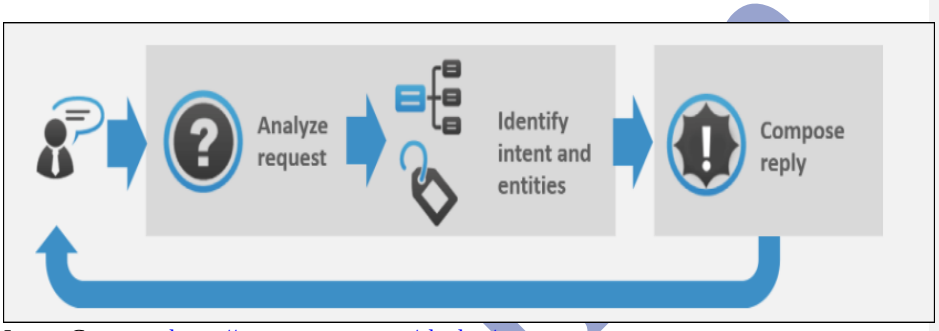



Image Courtesy: https://expertsystem.com/chatbot/

Chatbots are of two types: one functions on a set of rules, and the other more advanced one uses artificial intelligence.

1. **Chatbots that function on rules**: It is easy to build and can get basics tasks done with limitations to specific commands. This bot performance would depend on programming skills.

2. **Chatbots that function on machine learning**: It understands languages and not just commands, it provides relevant responses from the previous experience and training. It communicates through speech or text, relying on AI technologies like machine learning and NLP.

## 1.2 Use cases

Following are some of the chatbot applications, out of the infinite possibilities:

1. A chatbot for ordering food that allows customers to order from their office or home.
2. A chatbot that answers customer service questions
3. A chatbot that allows the customer to book a flight and receive relevant information.
4. A chatbot that helps a customer to purchase in eCommerce.
5. A chatbot that asks questions to the customer for a marketing campaign.
6. A health bot that provides services for booking a Doctor consultation and getting remedies for different symptoms.
7. A chatbot that books movie tickets and also provided the reviews.

## 1.3 Advantages of Chatbots

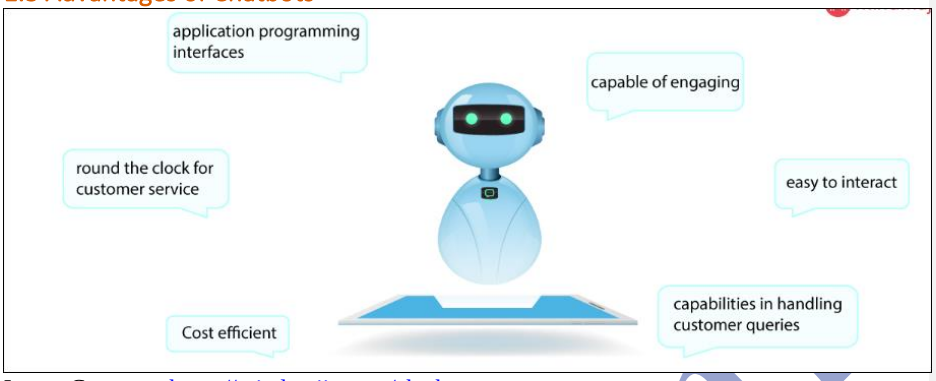



Image Courtesy: https://mindmajix.com/chatbot

## 1.4 Models for a Chatbot

When creating a chatbot, the goal is to automate the process altogether and to reduce human intervention. The first step involves getting all the existing interactions between customers and customer service representatives and use it to teach the machine which words/phrases are sensitive to the business. Then the next step is to identify the kind of chatbot we would like to implement.

There mainly are two models:

1. **Retrieval-based model**- These models are much easy to build and provide more predictable responses. They make use of context in the conversation for selecting the best answer from a predefined list of messages that they got trained. It includes all the previous discussions and the saved variables.

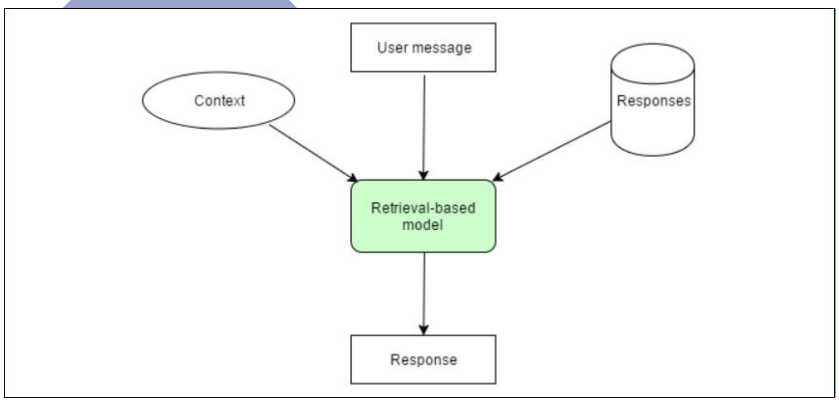



Image Courtesy: https://mindmajix.com/chatbot

Commented [vS2]: Try to use your own diagrams wherever possible or give the URL of the diagram source.

2. **Generative based model**: A generative model chatbot doesn't use any predefined repository. This kind of chatbot model is more advanced because it learns from scrape using "Deep Learning."

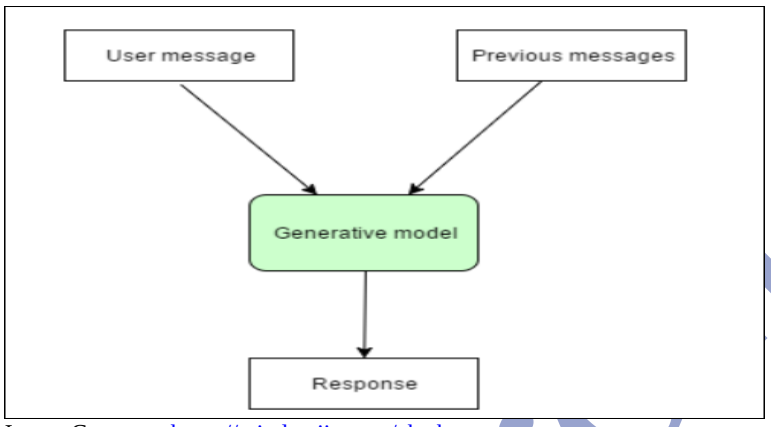



Image Courtesy: https://mindmajix.com/chatbot

## 1.5 Architecture of AI chatbot

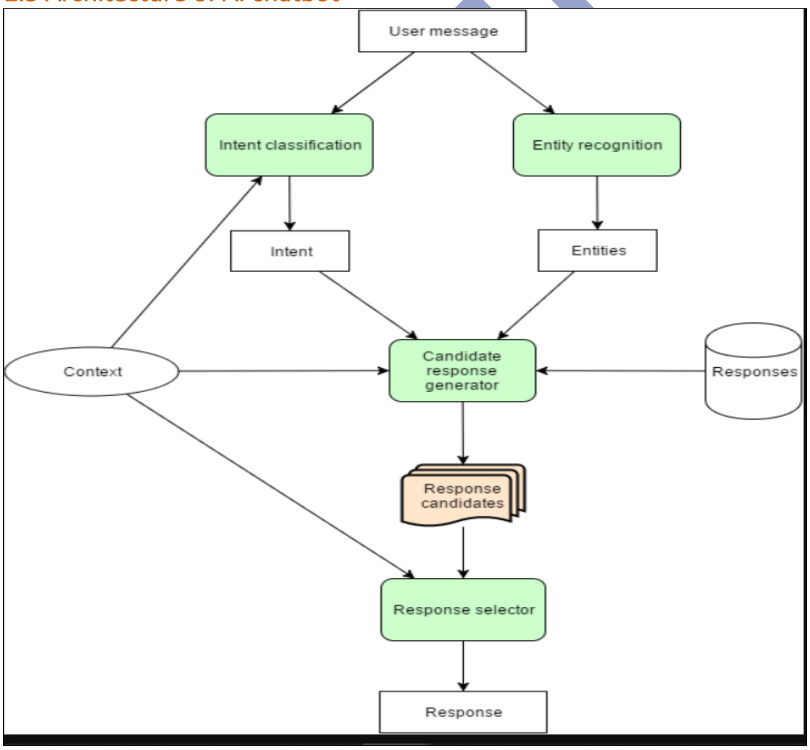



Image Courtesy: https://mindmajix.com/chatbot

Message processing begins with understanding what the user's intentions are.

1. An intent classification module identifies the intent of the user messages. Typically, it is a selection of one out of many predefined intents, though more sophisticated bots can identify multiple intents from one communication.

2. An entity recognition module extracts structured bits of information from the message. The weather bot, we can obtain the location and date to determine the weather.

3. The candidate response generator performs all the domain-specific calculations to process the user request. It uses different algorithms, initiates a call to external APIs, or connect to the human representative to help with responses. The result is a list of response candidates.

4. The response selector checks all the response candidate and selects a response which should work better for the user.

## 1.6 Chatbot Frameworks

Following are some of the chatbot frameworks:

1. **Pandorabots** – A Free & Open-source website to develop and deploy chatbots. Pandorabots is an open-source framework used for chatbot development with many AI rich features.

2. **Microsoft Bot Framework** – Microsoft Bot Framework is an excellent platform that helps you develop, connect, publish, and manage chatbots. The chatbots built on this framework come with agile learning and are interactive and smart to deliver the best user experience.

3. **Botpress-** Botpress is an enterprise-grade solution to develop, deploy, manage, and scale the chatbots. Though it is intuitive and straightforward, it is as flexible as a bot

4. **ChatterBot** –It's a free Cloud-based AI Chatbot Platform to Build Bots. Node.js drives chatterbot. It is a Python library that acts as a conversational dialog engine and uses machine learning. Moreover, it is language-independent, and this feature enables the usage of any language of choice to train the chatbot.

5. **Dialogflow (Google-owned)** – It is an End-to-end, Build-once Deploy-everywhere Development Suite. As Google's machine learning algorithm powers Dialogflow, we can build voice or text-based conversational interfaces for apps and chatbots. Also, we can connect to users on Mobile apps, Alexa, Messenger, Google Assistant, websites, etc. using Dialogflow.

6. **RASA Stack** – It is a machine learning-based open-source chatbot development framework. The two significant integrants of the RASA Stack framework are Rasa Core and Rasa NLU.

**7. Wit.ai** – it's a free Text or Voice-based Bot Building Tool by Facebook. With Wit.ai, we can develop a voice interface for smartphone apps, enable automation in wearables, etc. and it is not limited only to the development of bots.

8. **Botkit** – Botkit is one of the leading bot developer tools. Botkit.ai helps to build a bot with the help of a visual conversation builder and allows you to add plugins based on our needs. It works on a natural language processing engine from LUIS.ai plus includes open-source libraries.

# 2 Google Dialog Flow

## 2.1 Introduction

Google Dialogflow is a Google-owned developer of human-computer interaction technologies based on natural language conversations. It gives users a new option to interact with the product by building voice apps and chatbots powered by AI.

## 2.2 Why choose Google Dialogflow

- Delivers natural and productive conversations:
  It has built-in natural-language processing features and teaches artificial intelligence (AI) to the chatbot, thereby enabling it to process conversation naturally.
- Understands what users are saying:
  Machine learning makes Dialogflow is intelligent enough to predict the hidden intention expressed in the input language. A Dialogflow chatbot map's the user query with the database available with its backend server.
- Works with many platforms
  Chatbot developed using Dialogflow supports many platforms, and thus Google Chatbot development, businesses can target a wider audience with the least effort.
  Support for various devices
  Dialogflow helps with creating a device-antagonistic chatbot. Thus, it engages with users on wearables, phones, cars, speakers, and other smart devices. It helps to connect the businesses with their prospects or customers anywhere, anytime.
- Helps chatbots to speak 14+ languages
  Dialogflow supports many languages like Brazilian Portuguese, Chinese, English, Dutch, French, German, Italian, Japanese, Korean, Portuguese, Russian, Spanish, Ukrainian.
- Performance tracking with an analytics tool
  Similar to mobile app analytics, we can track the performance of the chatbots. The integrated analytics tool can read usage patterns, latency issues, and high- and low-performing intents

## 2.3 How do Chatbots work?

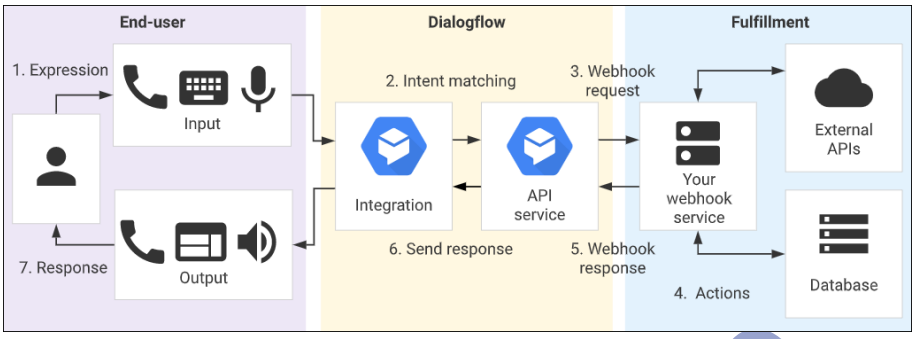



Image Courtesy: https://cloud.google.com/dialogflow/docs/basics

Steps:

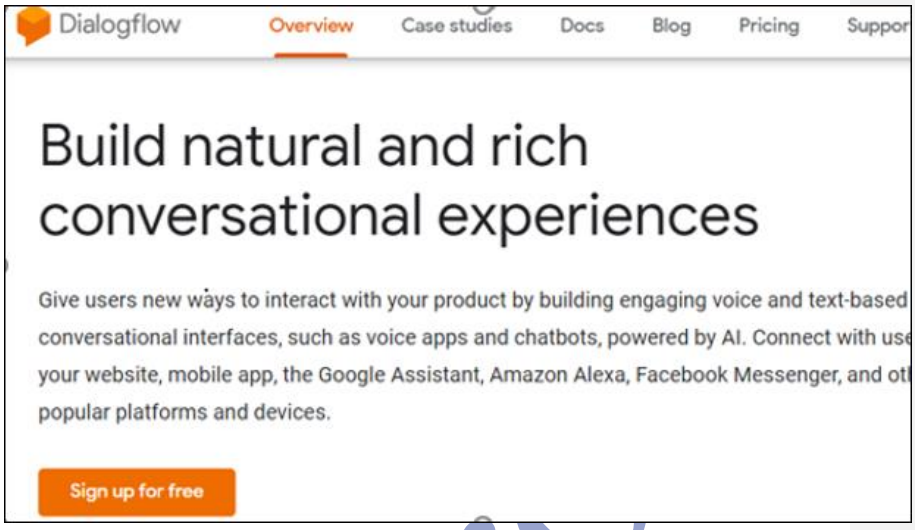
1. The user sends a text or voice message from a device or an App
2. The App or the Device transfers the data to Dialogflow
3. The message is categorized and matched to a corresponding intent
4. We define actions for each intent in fulfillment (Webhook).
5. When Dialogflow finds a specific intent, the webhook will use external APIs to find a response from external databases.
6. The external databases send back the required information to the webhook.
7. Webhook sends a formatted response to the intent.
8. Intent generates actionable data according to different channels.
9. Data go to output Apps or Devices attached
10. The user would get a text/image/voice as a response.

## 2.4 Build your first chatbot

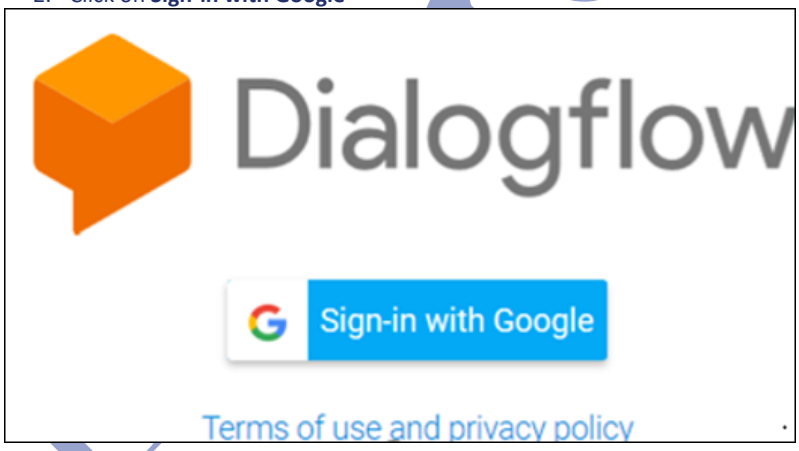
### 2.4.1 Signup for Dialogflow account

Prerequisite: A Google account is required for connecting to Google Dialogflow.

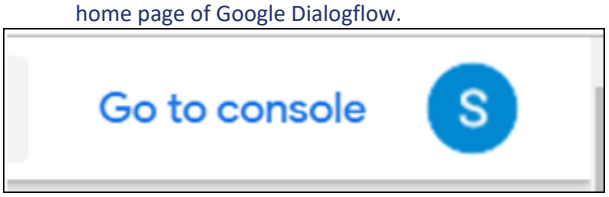
1. Create our Dialogflow account by using the link (https://dialogflow.com/). Click on **Sign for Free** and proceed with account creation.




2. Click on **Sign-in with Google**



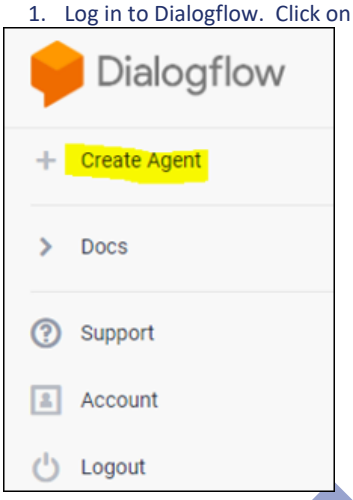

3. Select your google account, and it will direct to the Google Dialogflow home page. Click on **Go to console** in the upper right corner to navigate to the home page of Google Dialogflow.



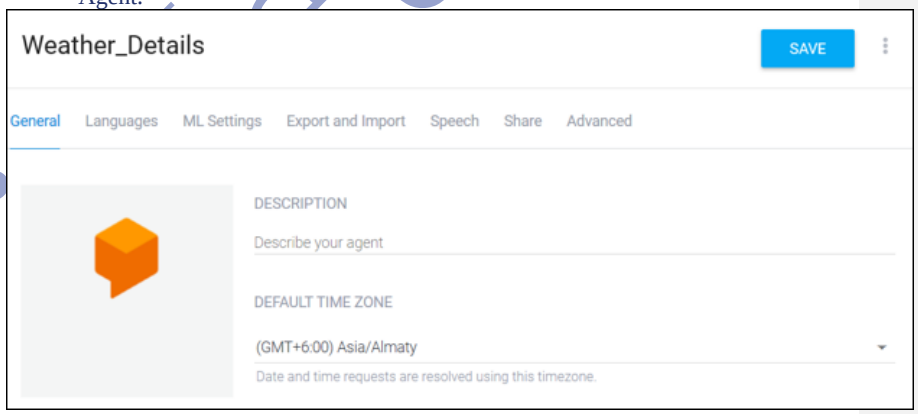
## 2.4.2 Creating an Agent

An agent is a virtual agent or bot that handles the conversation with end-users. We can design or build a Dialogflow agent to handle the different types of communications required by the system. These can be included in any app, product, or service and transform natural user requests into actionable data

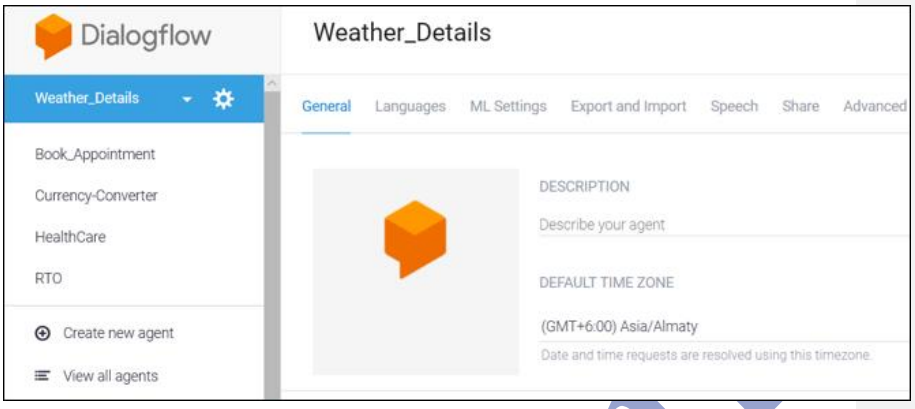
1. Log in to Dialogflow. Click on **Create Agent** from the left menu.



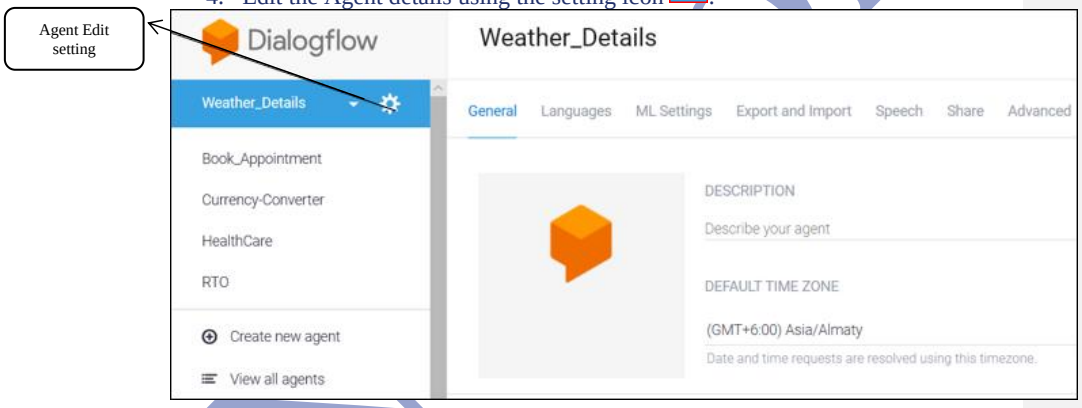

2. Provide the name of the agent and click on the **SAVE** button to create the Agent.




3. The Agent gets created and gets listed below the Dialogflow icon. If there are multiple agents, use the dropdown down button to select the Agent for editing or adding new data.



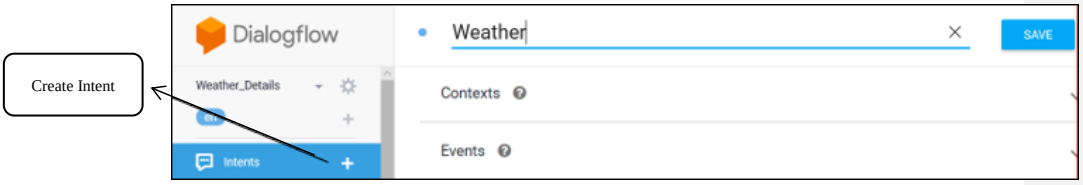

4. Edit the Agent details using the setting icon .




### 2.4.3 Creating an Intent

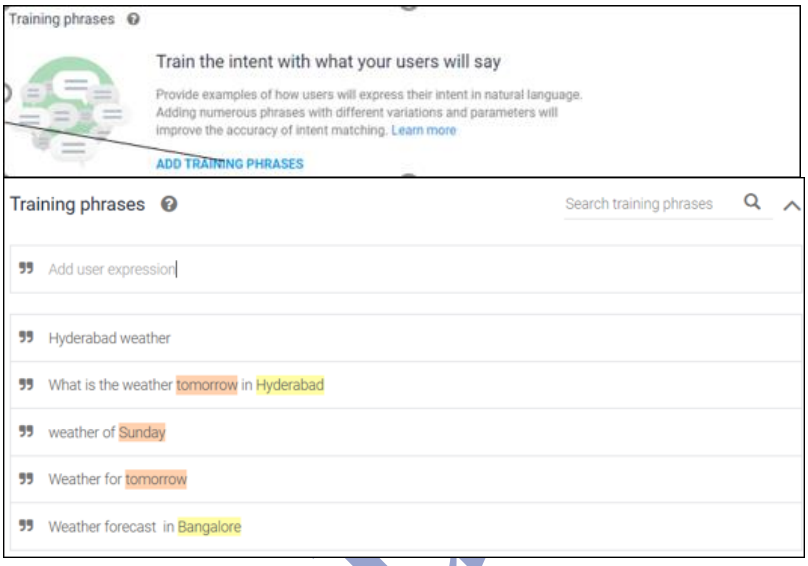
Intents are mappings between a user's queries and actions fulfilled by our software.

1. Click on the plus icon as specified in the below image to create a new intent, which allows you to map what our user says to what our agent responds. Provide a name as Weather and click on the SAVE button.

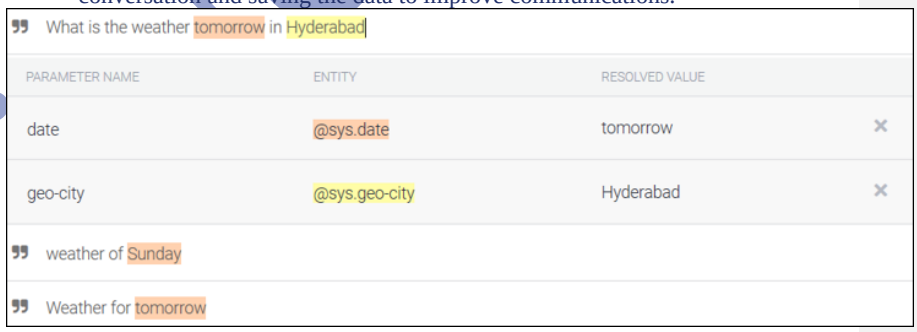


2. Click on **ADD TRAINING PHRASE** and then add **Training phrases**. Training phrases are examples of what users can say to match a particular intent. Adding numerous phrases with different variations and parameters will improve the accuracy of intent matching.

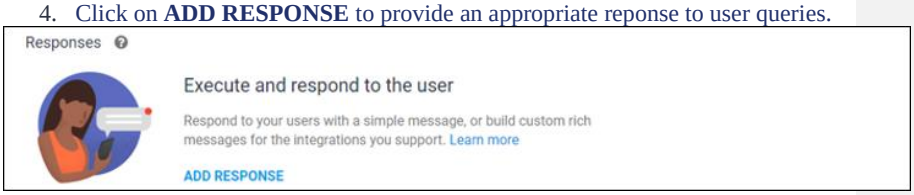
Click to Add the training phrases



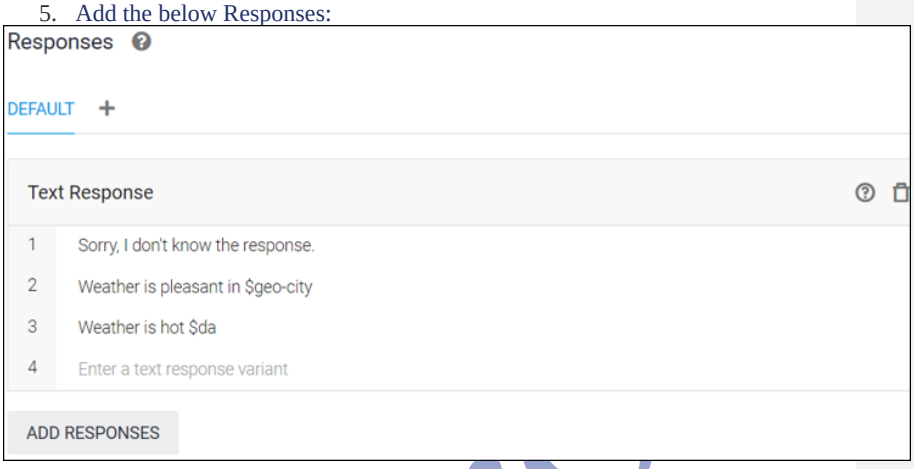

3. Highlight the dates in each phrase and right-click to select the system entity like @sys.date to capture a day. Similarly, select the system entity @sys.geo-city for the city. This step involves identifying the purpose of the user conversation and saving the data to improve communications.



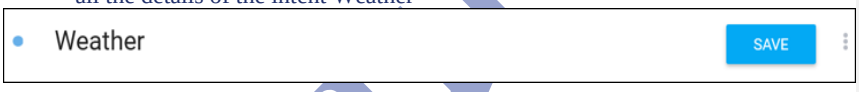

4. Click on **ADD RESPONSE** to provide an appropriate reponse to user queries.



5. Add the below Responses:

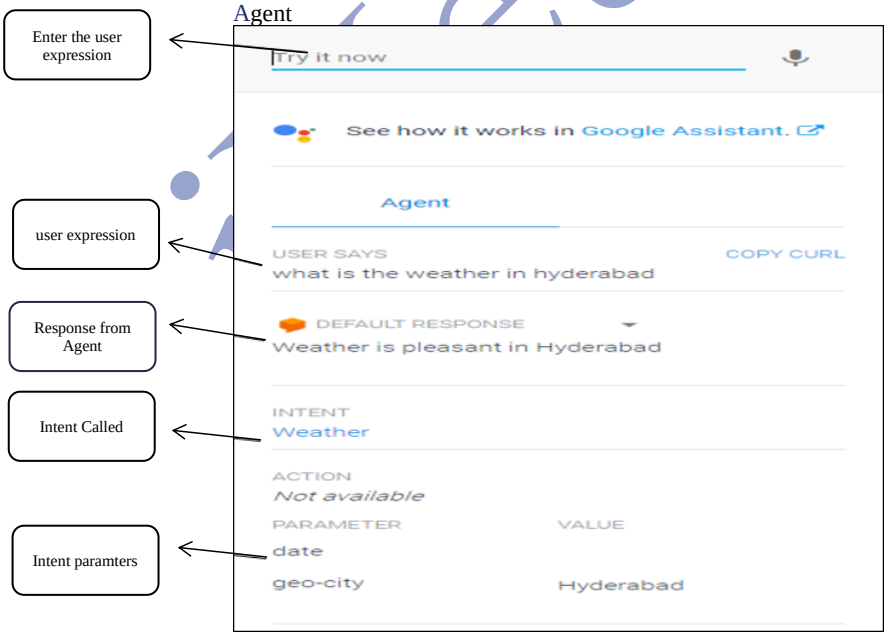



6. Click on the **SAVE** button which is available in the upper right corner to save all the details of the intent Weather




### 2.4.4 Test it Now

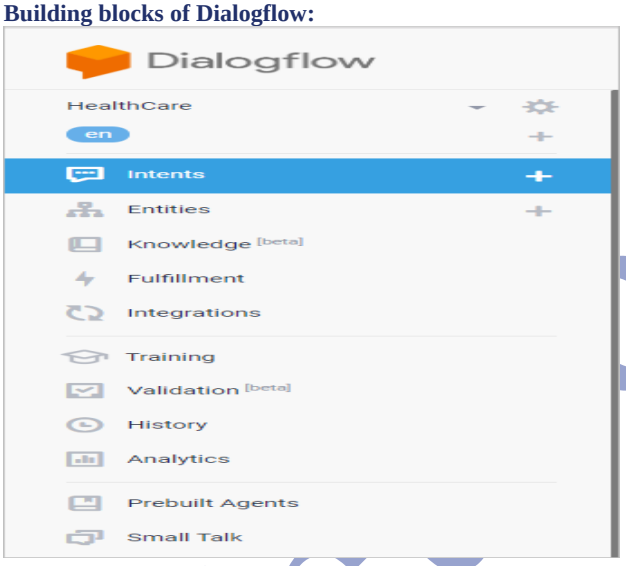
We can use the test console to test our requests and check the responses from Agent



## 2.5 Understanding the building blocks of Dialogflow

In the last section, we have created a simple Weather Agent(bot) utilizing some of the features of Dialogflow. In this section, we would be giving a brief of all the features of Dialogflow, which would help to understand the next segments.

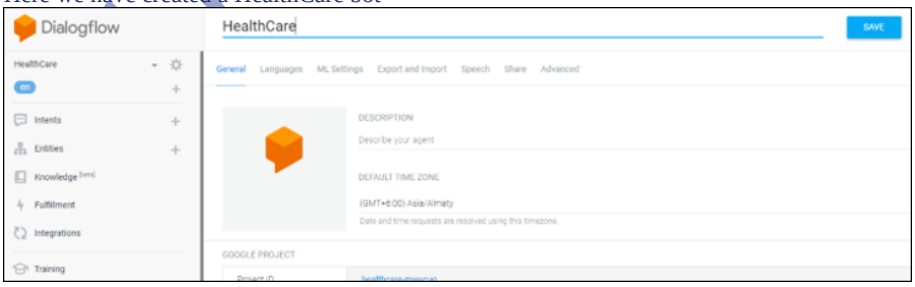
**Building blocks of Dialogflow:**




### 2.5.1 Agent:

An agent is a virtual agent or bot that handles the conversation with end-users. It incorporates Natural Language Processing to understand what the user meant and to figure out what "action" has to be carried out. Agents succeed in conversations with the user through intent, entities, contexts, and other building blocks.

Here we have created a HealthCare bot

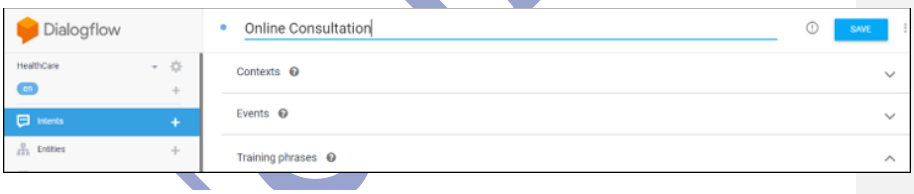



### 2.5.2 Intent

Intents determine the action to be taken by the code. It is a mapping of what the user says and what our software should do with the user utterance.


Two types of intent:
**Default intent**: Dialogflow provides two default intents, namely default Welcome Intent (for greetings) and Default Fallback Intent (Default fall through intent when no other intents match).
**Custom intent**: Dialogflow provides options to create customized Intents based on business requirements.

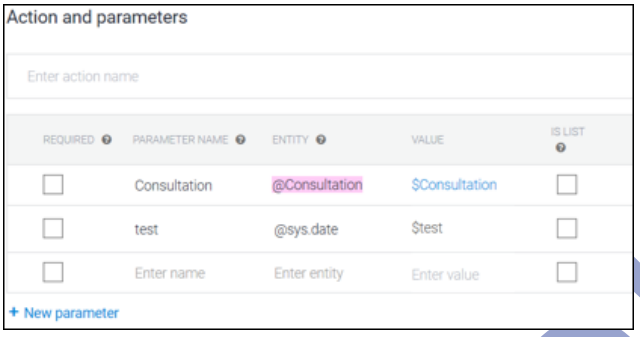
a. **Intent Name**: Create a new intent with the name Online consultation and save it.



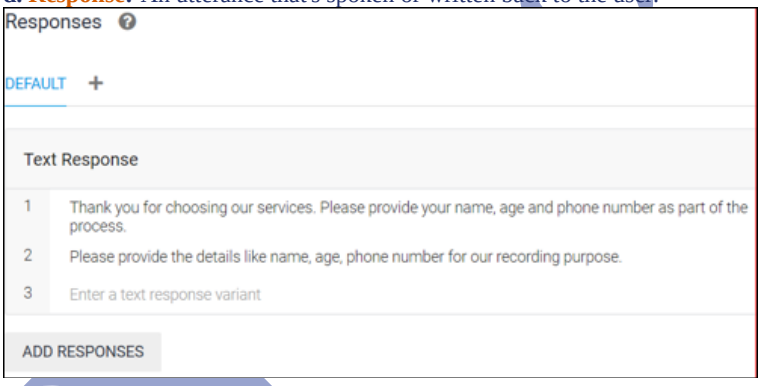

b. **Training phrase**: The phrases you can expect from the user that will trigger the intent. Examples of what the user says to match a particular intent. Adding numerous phrases with different variations and parameters will improve the accuracy of intent matching.

**c. Action and Parameters:**

The inputs we need from the user to act on the user request. e.g., consultation. Examples are dates, times, names, places, and more.

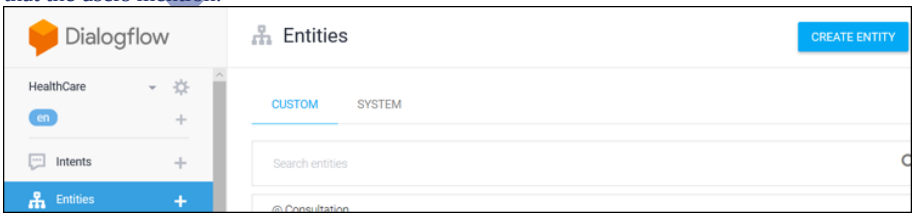



**d. Response:** An utterance that's spoken or written back to the user.



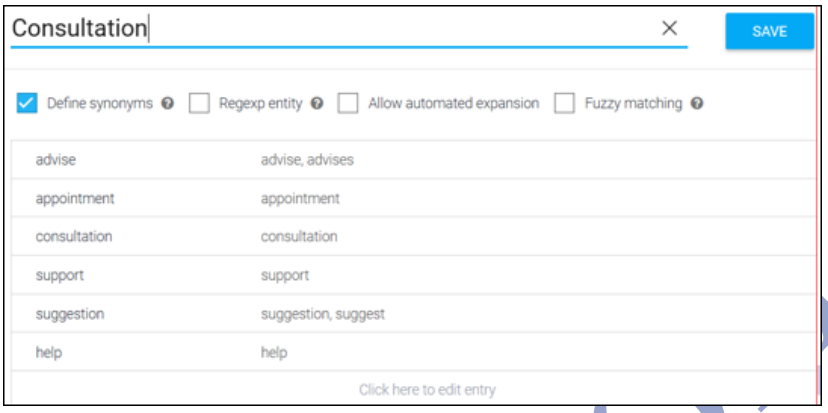

### 2.5.3 Entities

Entities identify and extract useful data from user's inputs. While intents help to understand the motivation behind a particular user input, entities pick out specific pieces of information that the users mention.




Two types of entities;
System entities: In-built entities provided by Google Dialogflow.

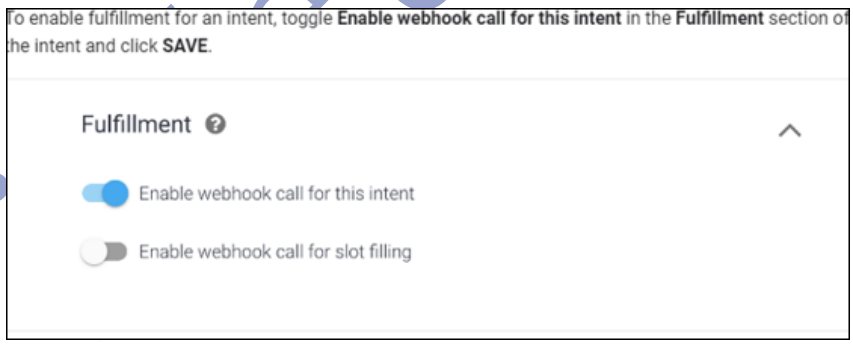
Customer entities: The developer creates customized entities. Here, we have created an Entity "Consultation," provided the synonyms, and saved it.




### 2.5.4 Fulfilment:

Fulfillment helps to use the information extracted by Dialogflow's natural language processing to generate responses dynamically or to trigger actions in the back-end.

a. Configure fulfillment:
   i. Enable fulfillment by sliding **Enable webhook call for this intent** to the intents, which involve some logic building or to trigger actions.




   ii. To enable the inline editor, Click on Fulfillment in the left menu. Click the switch for Inline Editor.

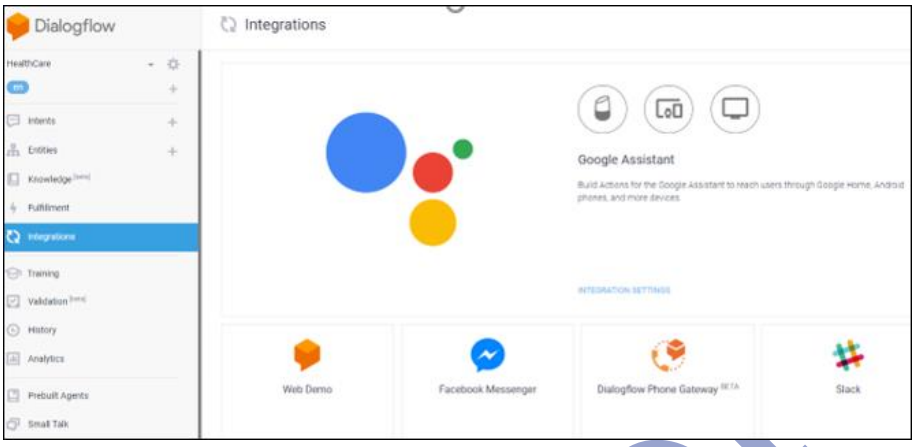

iii. To deploy our fulfillment, click **Deploy** under the code editor.


iv. To export our code: Once you're ready to move our code out of the **Fulfillment** page, you can use the **Download** button to get a '.ZIP' file of it.


### 2.5.5 Integrations

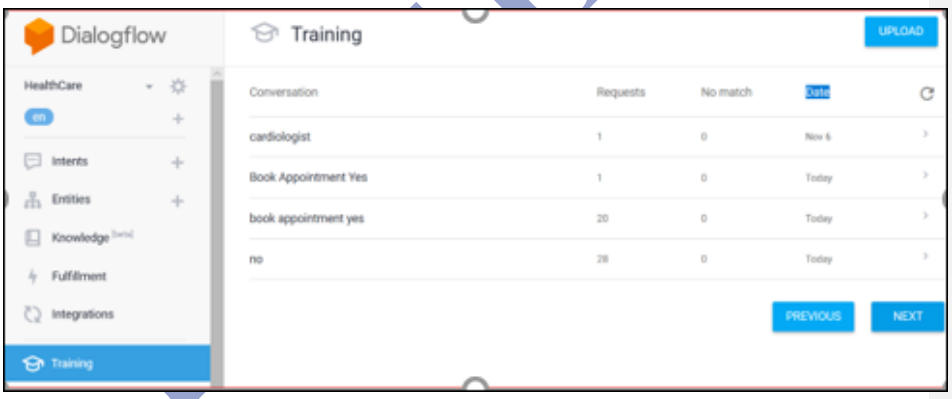
Dialogflow integrates with many platforms like Slack, Google Assistant, and Facebook Messenger. These integrations provide platform-specific features for building productive responses.

Note -This feature discussed in detail in the subsequent sections.



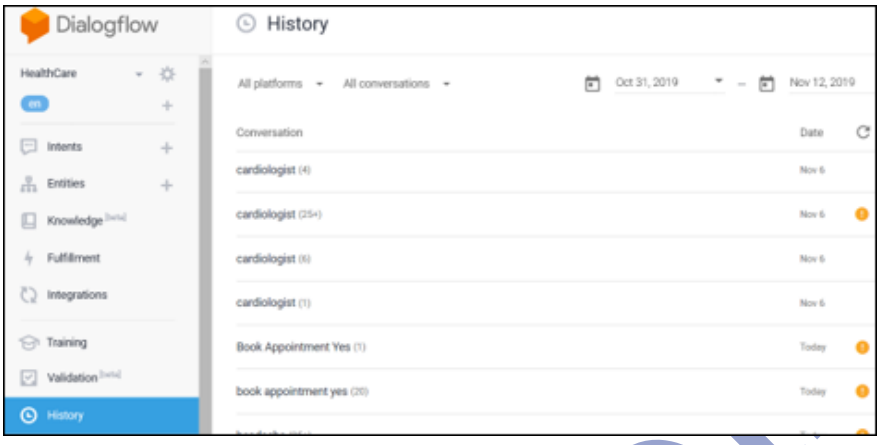

### 2.5.6 Training:

Dialogflow's natural language processing built on machine learning. We can add training data that the agent learns from and uses to improve its performance. Dialogflow's training feature provides an interface for incorporating both external and internal customer interaction logs into an agent's training phrases.



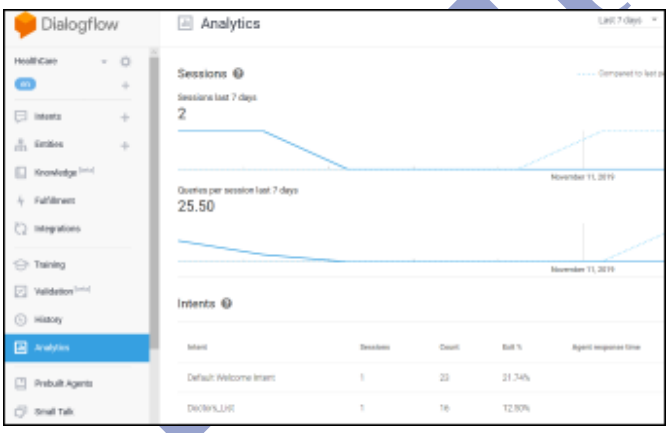

### 2.5.6 History

Dialogflow History section displays a simplified version of conversations the agent has engaged. The records provide an overview of how users interact with the agent.



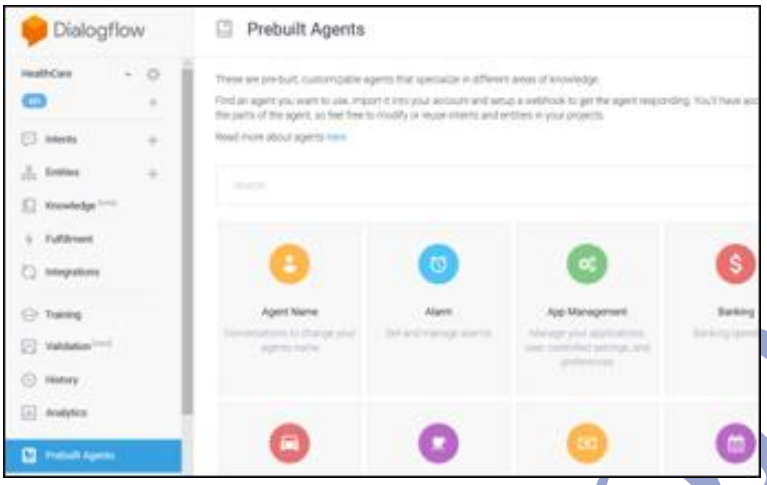

### 2.5.7 Analytics

The Analytics screen of Dialogflow gives understanding into how well the agent is performing so that we can work on to improve further the user experience you're providing.




### 2.5.8 Pre-built Agents:

Pre-built Agents are the agents provided by google Dialogflow which can be imported as agents and utilized based on the requirement.




## 3. Dialogflow: Linear and Non-linear

We can create Dialogflow to retain the flow of conversations. Dialogs can be linear or non-linear.

**Linear Dialogs**

Linear Dialogs can span single or multiple intents. The primary reason for Linear Dialogs is to capture information from the user to complete actions. Context permits information sharing to simulate a more natural conversation

**Non-Linear Dialogs**

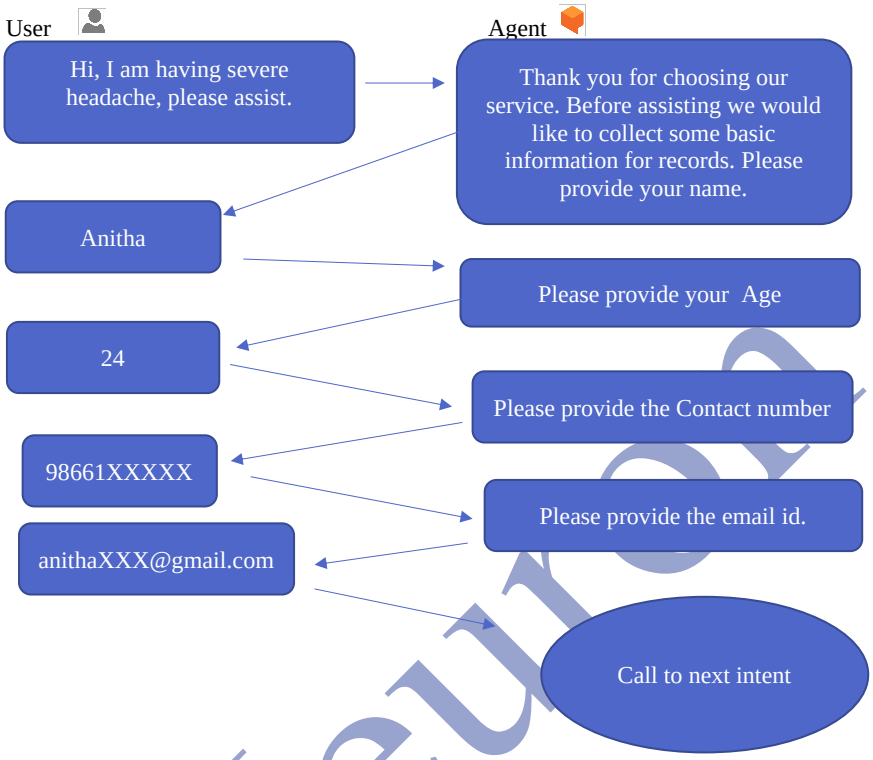
Non-linear dialogs help to branch to corresponding intents based on user responses.

### 3.1 Linear Dialogflow

When the flow dialogs between the user and Agent is linear, then it is called Linear Dialogflow. Information from the user replies is collected, and the next intent gets invoked. Linear Dialogflow is developed using a single intent or multiple intents.

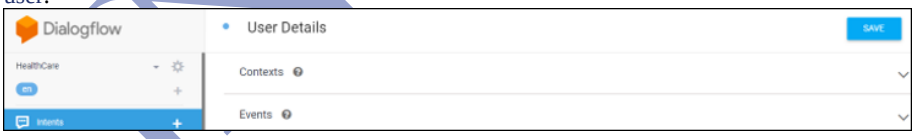
#### 3.1.1 Linear Dialogflow using Single intent:

We can collect the information from the user in a single intent and capture that information in parameters in a linear form, using the options in the "Action and Parameters" section while creating an intent. We are discussing the same via an example below.
Let us consider the flow below where the Agent is requesting for user details.



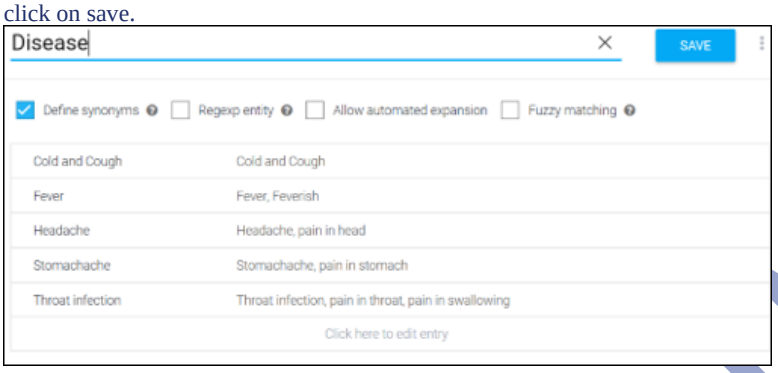

**Steps to create Linear Dialogflow with a single intent**:

1. Login to Google Dialogflow and connect to the Healthcare Agent created in the previous section.
2. Create an intent with the name **User Details** as we would be collecting the Details from the user.

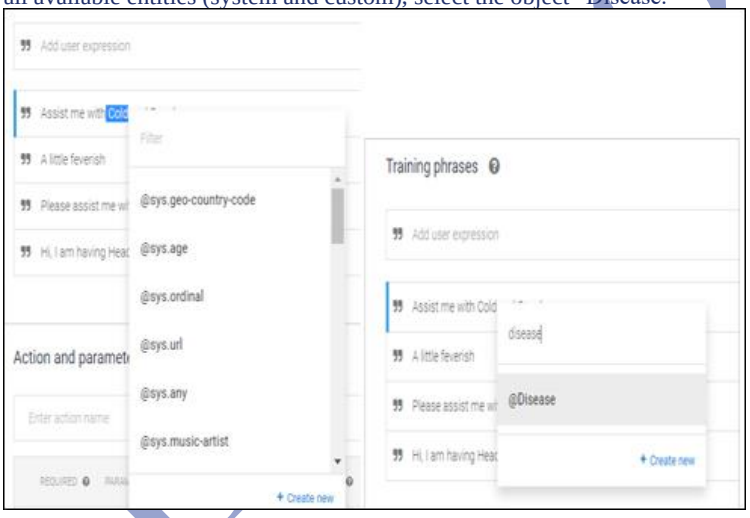



3. Provide the **Training phrases,** as shown below.

4. Capture the disease in a parameter. So, we would create a custom entity. **Save** the intent and Click on **Entity** to create an object with the name Disease. Provide the data as below and click on save.



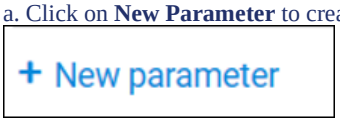

5. Go to the intent **User Details**. Highlight cold, a small window will be displaying a list of all available entities (system and custom), select the object "Disease."




6. Selecting the entity would associate the parameter with the import information in the training data. Please follow step 5 for the highlighted data in the Training phrases.

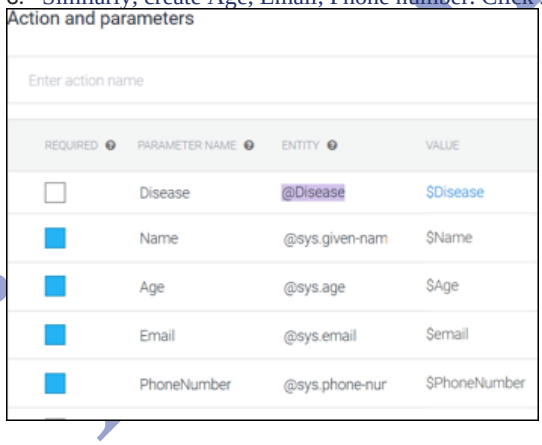
7. Go to **Action and Parameters**. Disease parameter is listed, we can add other parameters like Name, Age, Email, Phone-number to collect information from the user.

a. Click on **New Parameter** to create a new parameter

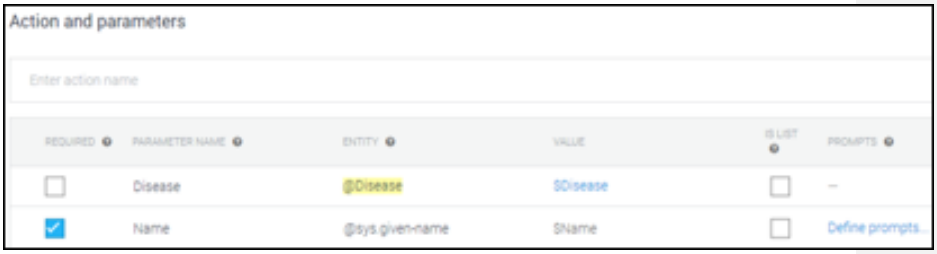



b. Enter Name under Parameter Name, @sys.givenname as Entity, $Name as Value.

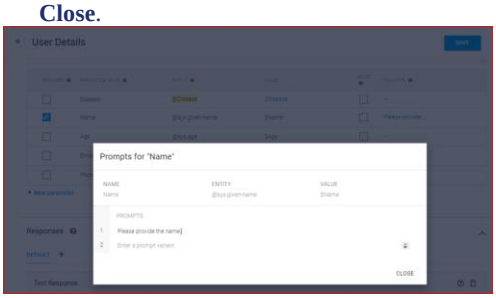

c. Similarly, create Age, Email, Phone number. Click save to save the data.




d. Select the **PARAMETER NAME** as required by checking the Required option, as shown below.

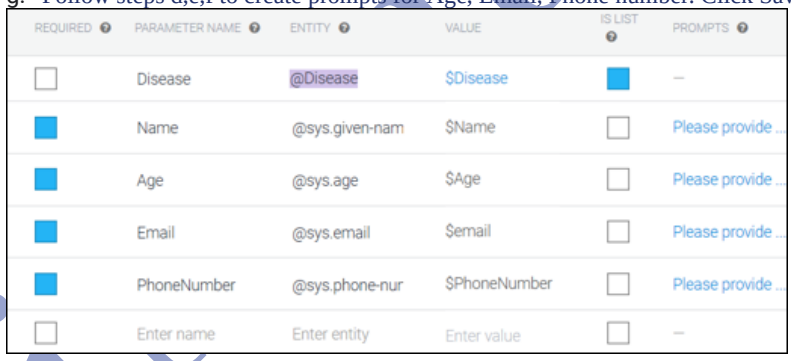


e. Under the prompts section, click on **Define prompts** to provide the Bot response to collect the user name. A pop-up window would be displayed requesting to provide the prompts. Add a prompt **Please give the name** as shown below. Click **Close**.




f. The prompt would be displayed in the parameter section for the selected parameter.


g. Follow steps d,e,f to create prompts for Age, Email, Phone number. Click Save.




h. Provide the Response to invoke the next intent. Here we can utilize the parameter value of Disease, which was captured by prefixing it with $ symbol .e.g. $Disease. Click the SAVE button.


i. Test the flow using the "Try it now" option discussed in the earlier section.

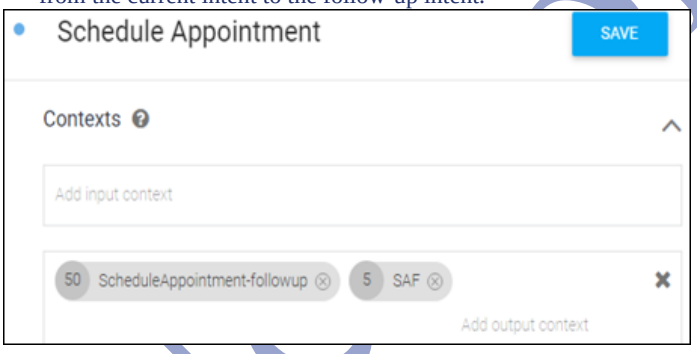
### 3.1.2 Linear Dialogflow using multiple intents:

Here, the linear dialogs flow to numerous intents. The main objective of the linear dialog is to elicit the pieces of information from the user required to fulfill the user intents from multiple intents. We can use follow-up intent to make the data flow to different intents, and any intent can utilize the data from the previous intents. Here we would be utilizing two concepts to be precise follow-up intent and contexts.

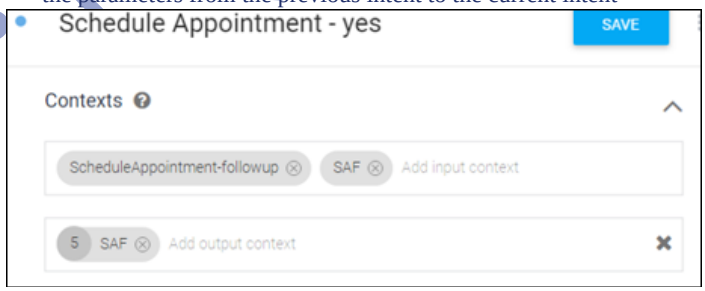
## Contexts

Contexts signify the current state of a user's request and allow the agent to carry information from one intent to another. We can use combinations of input and output contexts to control the conversational path the user takes through our dialog.

a. Output Contexts: When applied to an intent, an output context tells Dialogflow to activate a context if it's not already active or to carry the information(parameters) from the current intent to the follow-up intent.



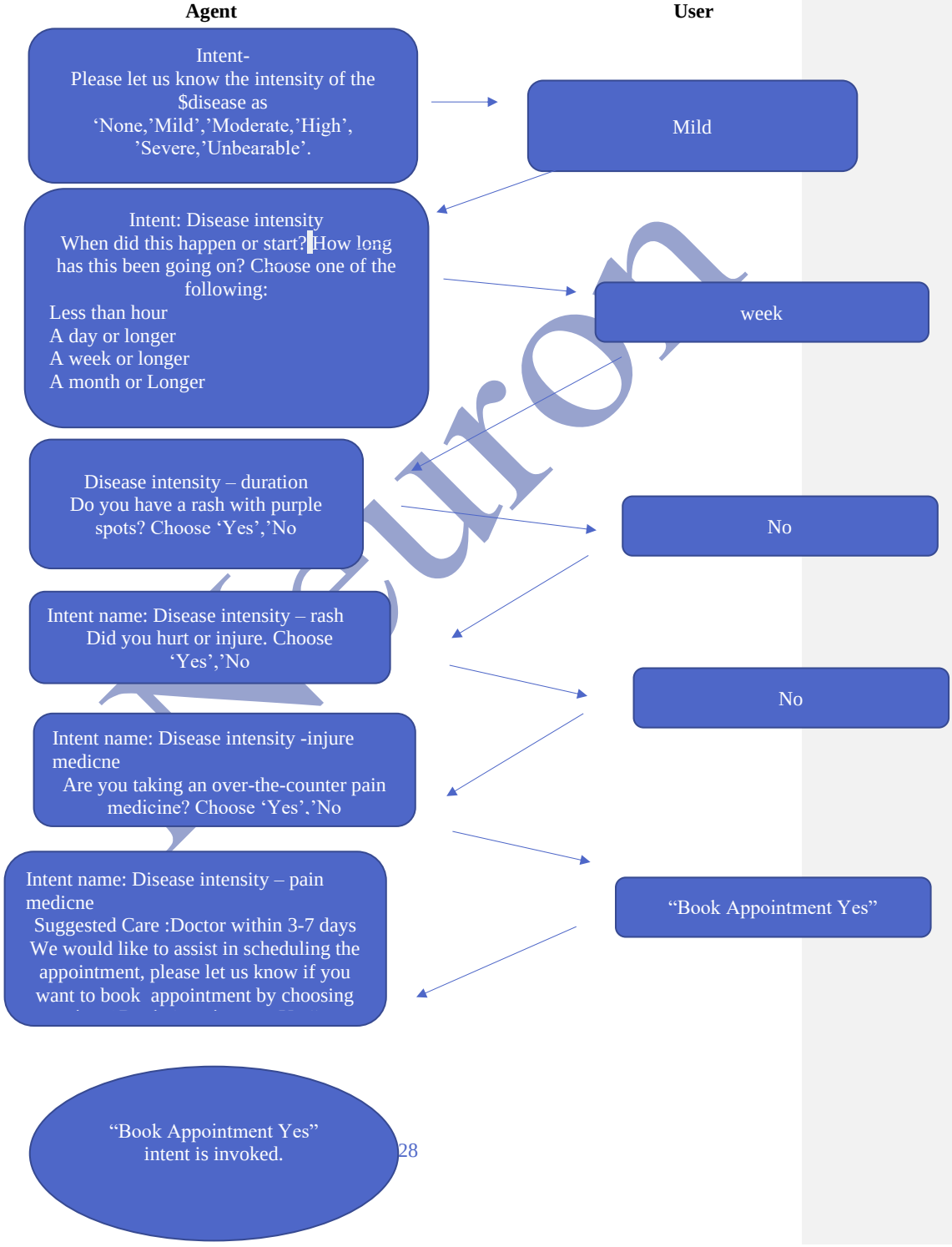

b. Input contexts: When applied to an intent, an input context tells Dialogflow to match the intent only if the user utterance is a close match and it helps to access the parameters from the previous intent to the current intent




## Follow-up Intent

You can use *follow-up intents* to set contexts for pairs of intents automatically. When we create a follow-up intent, an output context is automatically added to the parent intent, and an input context of the same name is added to the follow-up intent.

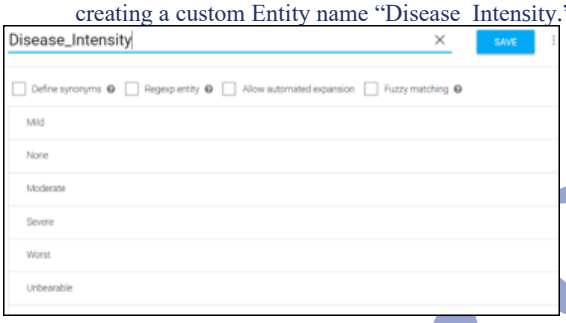
Let us consider the following flow where the Agent is requesting for user details



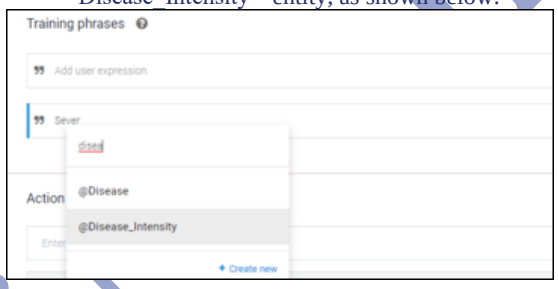
Steps to create linear Dialogflow with multiple intents using Context and Follow-up intents:

a. Go to the HealthCare bot and create a new intent named **Disease intensity**.

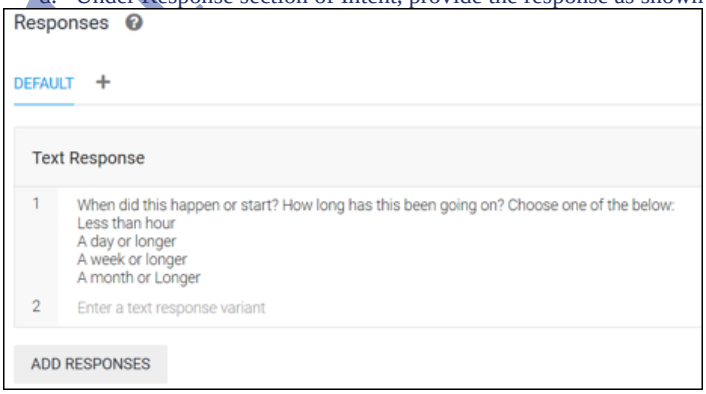

b. Go to Entities section to capture the Intensity of the Disease in a parameter by creating a custom Entity name "Disease_Intensity."




c. Go to **Training phases** and provide a phrase and assign data to Disease_Intensity" entity, as shown below:



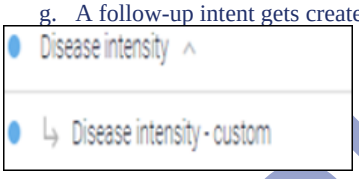

d. Under Response section of Intent, provide the response as shown below:



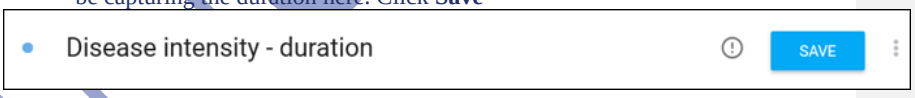
e. Click **SAVE** to save the current intent. Click on Intents to view the list of all available intents and select the **Disease Intensity** intent and click **Add Follow-up intent**.


f. A drop-down is displayed requesting the type of follow-up intent. Please select **Custom**


g. A follow-up intent gets created and gets displayed below the parent intent.



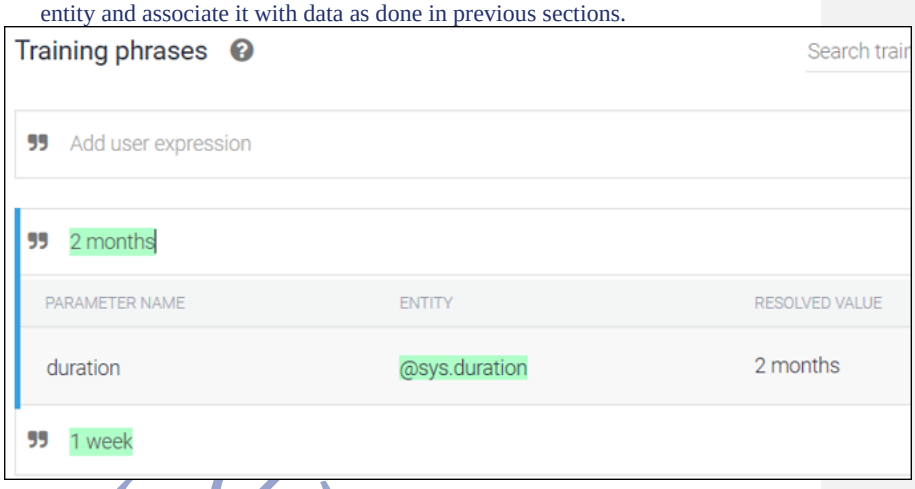

h. Select the newly created follow-up intent **Disease intensity-custom** to configure it. Edit the intent name **to Disease intensity – duration** as we would be capturing the duration here. Click **Save**




i. Go to Context, and you will observe that it is already populated with "Diseaseintensity-follow-up." It allows utilizing entities captured in previous intent in the current intent.


j. Create output Context to forward the current entities in the next intent. Type **Diseaseintensity-follow-up,** and it gets added with a default life span of 5. We can increase or decrease the life span based on our requirements.

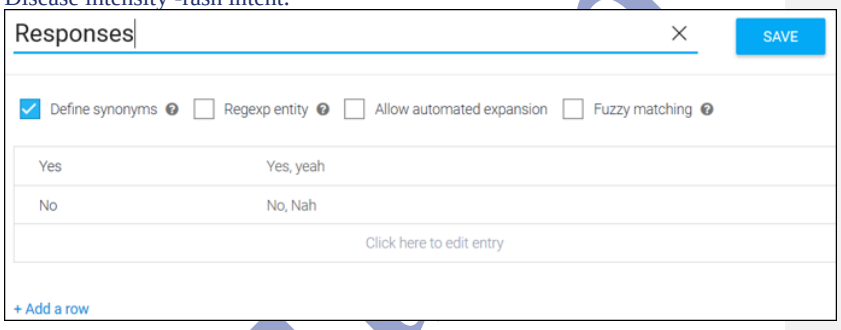

k. Go to the training phrase and provide the phrase for the duration. System entity @sys.duration automatically gets selected. We can create our custom entity and associate it with data as done in previous sections.




l. Provide the response as below and save the intent.


m. Go to the Disease intensity – duration and create a follow-up intent as Disease intensity -rash. Add the output contexts, as shown below. Click Save.

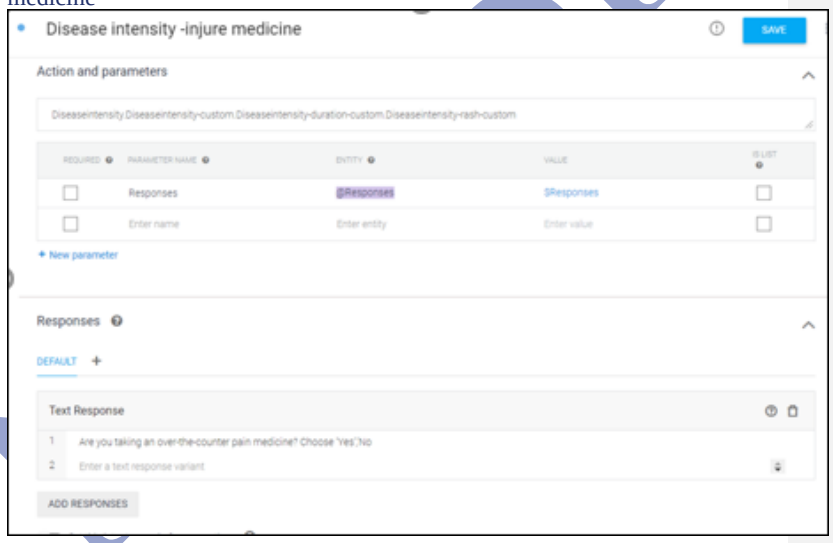

n. Create an entity Response to capture yes or no data. Click Save and got to Disease intensity -rash intent.



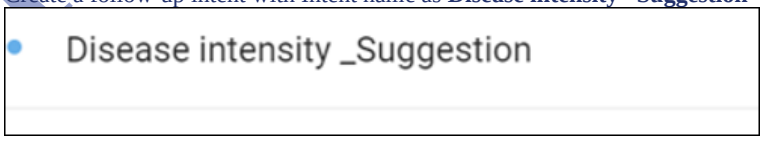

o. Add training phrases and associate it with entity response


p. Go to the response section of intent and add the following data.


q. Follow the steps m,o,p to create a follow-up intent Disease intensity -injure medicine

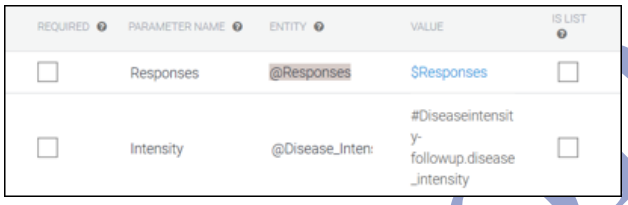



r. Create a follow-up intent with Intent name as **Disease intensity –Suggestion**



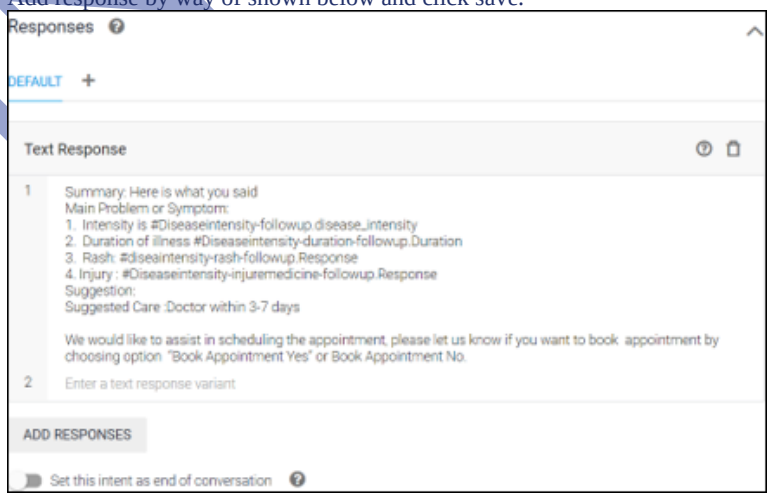

s. Add the training phrase


t. Add the parameter ‘intensity’ with the following values.



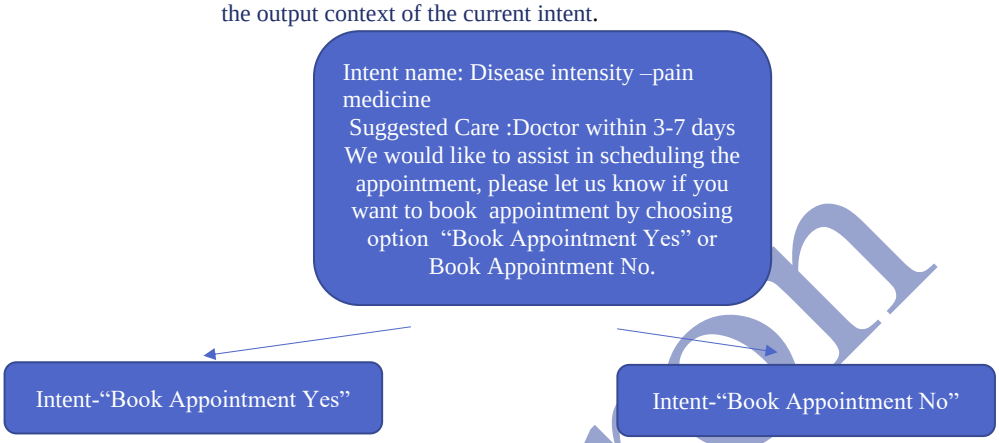

u. Similarly, add the parameters- duration, rash, injure.


v. Add response by way of shown below and click save.



## 3.2 Non- Linear Dialogflow:

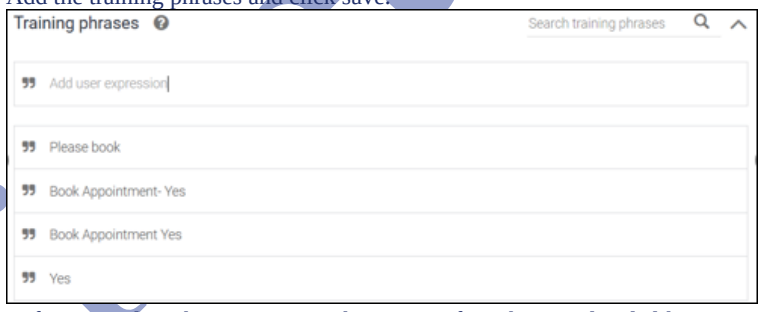
It requires more than one intent. Non-linear dialogs branch to the next intent based on responses from the previous intent given by the user. The branch is based on the input context, and the input context of next should be the same as the output context of the current intent.




Steps to create non-linear Dialogflow:

1. Create a new intent **Book Appointment Yes**

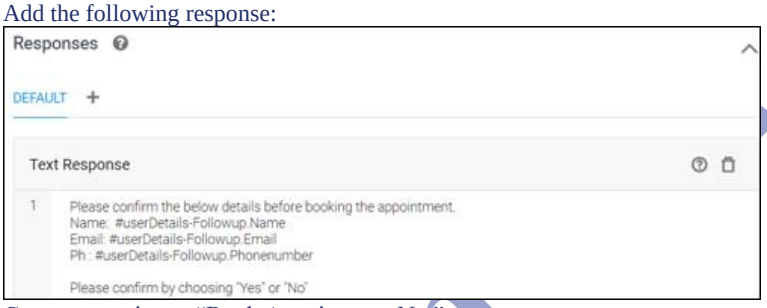

2. Add the training phrases and click save.



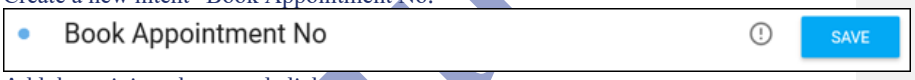

3. Before providing the response, we have to confirm the user details like name, email, and phone captured in "User Details" intent. For this, an output context "**userDetails-Follow-up**" is added in this intent.

4. Go to “Book Appointment Yes” and the “userDetails-Follow-up” context as input.

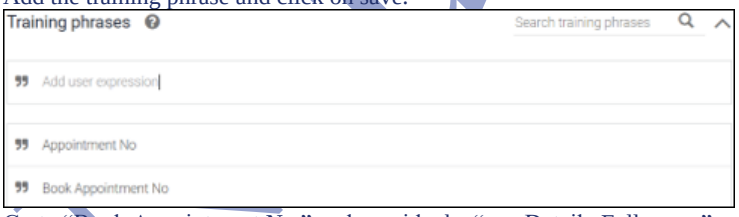

5. Add the following response:




6. Create a new intent “Book Appointment No.”




7. Add the training phrase and click on save.




8. Go to “Book Appointment No.” and provide the “userDetails-Follow-up” context as input.

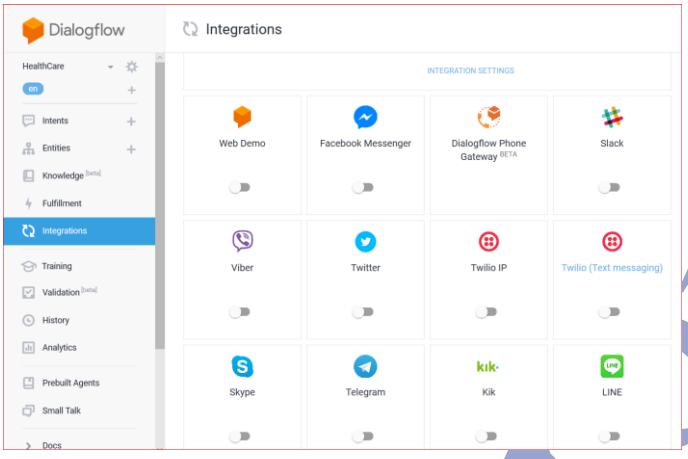

9. Add the following response:

iNeuron

# 4. Integrations

## Integrations:

Go to integrations tab on the left-hand side an enable the integrations

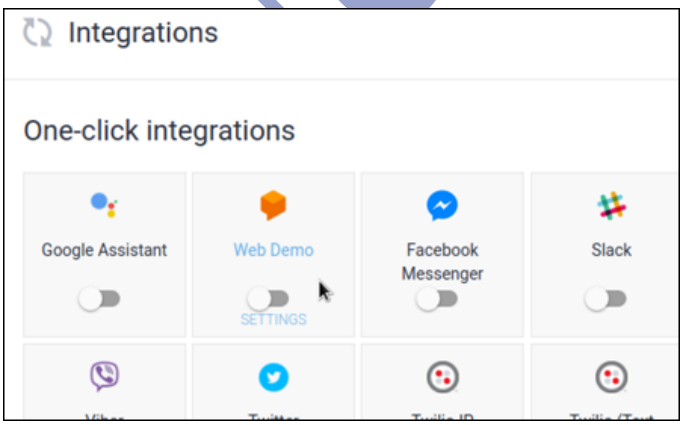



## 4.1 Dialogflow Web Demo:

The Web demo allows us to publicly share our agent through a generated page or by embedding it in our current portal/website.

### Setting Up a Web Demo

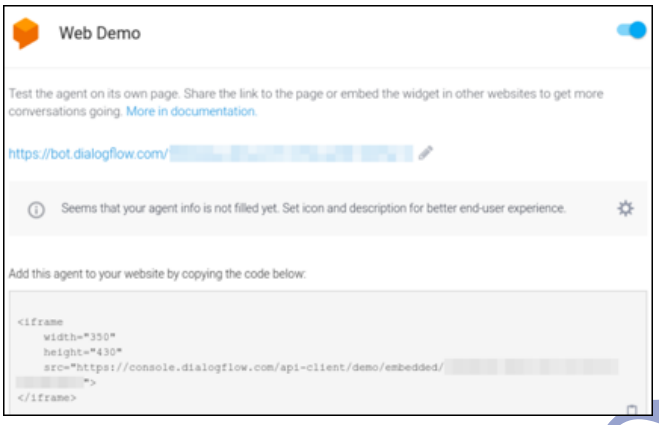
To create a web demo for our current agent, click on the **Integrations** option in the left menu of Dialogflow and then click the switch on the **Web Demo** tile.




After Web Demo is enabled, a window will be displayed with the following info:

- A URL to the generated webpage where our agent is hosted
- A link to icon and description settings, which affects our agent on the generated webpage

- Code to embed our agent in our website, via HTML

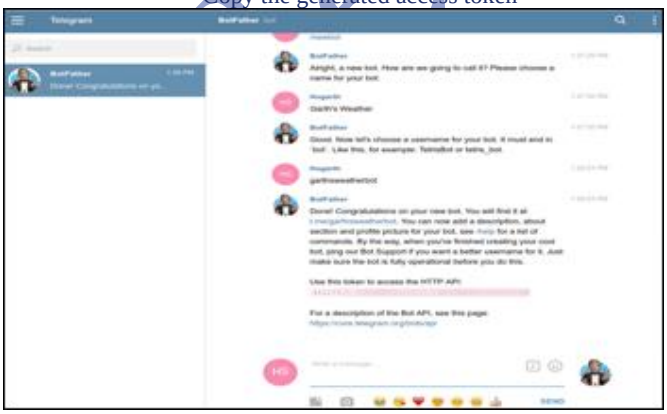



## 4.2 Telegram:

To set up the Telegram integration for our agent, you'll need a Telegram account.

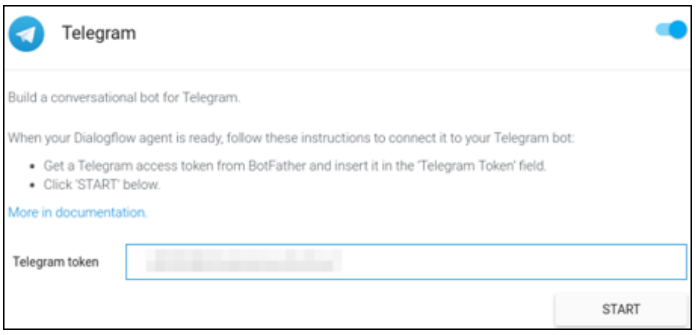
### Creating a Bot in Telegram

1. Log-in to Telegram using the link https://telegram.me/botfather
2. Click the **Start** button in the web interface or type /start in Telegram
3. Type **/newbot** and provide a name
4. Provide a username for the bot, ending in "bot" (e.g., garthsweatherbot)
   Copy the generated access token



### Setting Up Dialogflow

5. In Dialogflow, go to **Integrations** available in the left-hand menu
6. Click on the **Telegram** tile to enable the Telegram integrations.
7. Paste the Telegram **Access Token** into the related field. Click the **Start** button




## 5.1 Fulfilment and External API integrations:

**Fulfillment:**

If an intent requires some actions by the system or a dynamic response, we should enable fulfillment. When a fulfillment enabled intent is matched, Dialogflow sends a request to our *webhook* service with information about the matched intent. Our system can perform any required actions and respond to Dialogflow with information on how to proceed. The below diagram shows the processing flow for fulfillment.

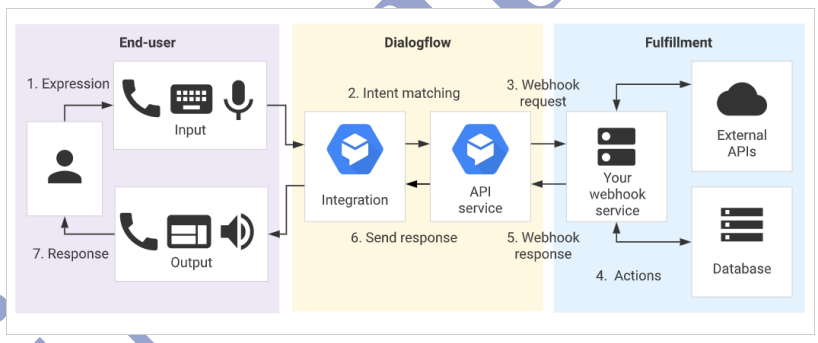



Image Courtesy: https://cloud.google.com/dialogflow/docs/basics

### 5.1 External API call - Clinics list based on specialty

In this section, we are extracting the Clinic list based on the specialty from an external API. We would display the contact number and name of the clinic based on the specialty.

#### 5.1.1 Generating the API key

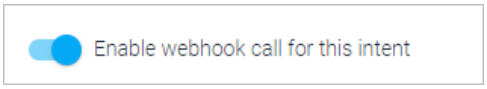
1.Register to generate the API key using the link https://developer.betterdoctor.com/

#### 5.1.2 Create Agent and Intent

1. Create an Agent and Intent with the following details:
    a. Create an Agent DoctorsClinic_Speciality
    b. Create an intent Doctors_Speciality
    c. Add Training phrases

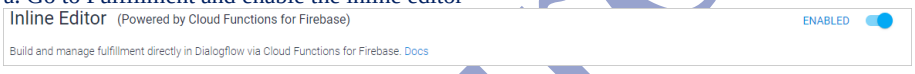
please suggest a clinic of Cardiologists


d. Add an entity named- specialty and add all the specializations.
e. Enable fulfillment

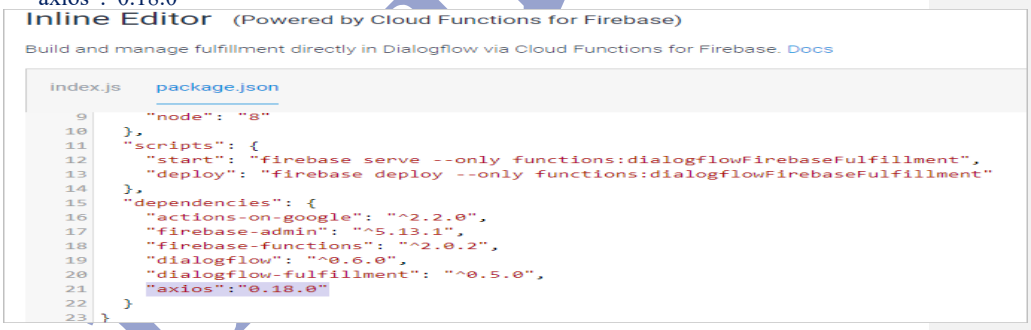



### 5.1.3 Fulfilment code:

a. Go to Fulfillment and enable the inline editor

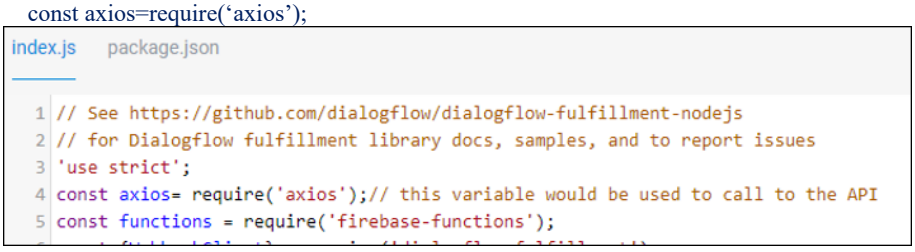



b. Add the 'axios' dependency under package.json.
"axios":"0.18.0"




Note: This package is used to call external API.

c. Goto Index.js, and add the constant axios
const axios=require('axios');



d. Add the following code for handler function and intent mapping

Handler function

```
function Dlist(agent) {
 const speciality= agent.parameters.speciality;
 agent.add('Nearby ${speciality} doctors hospitals and numbers are');
 return axios.get('https://api.betterdoctor.com/2016-03-
01/practices?name=pediatrician&skip=0&limit=4&user_key=371751d40c586e02eba9d
e992e678439')//add the key generated in first step
 //making a call to axios library and connects to the api and get the reponse
 //If we recived the result then execute this function
 .then((result) =>{
      //console.log(result.data);
   let responseData= result.data;


    let responseData1=responseData.data;

    responseData1.map(docobj =>{
      console.log(docobj.phones[0].number);
      console.log(docobj.name);

      agent.add(docobj.name);
      agent.add(docobj.phones[0].number);

    });

 });
  //agent.add(`Nearby ${speciality}`);

 }
```

Intent mapping:

```
intentMap.set('Doctors_Speciality', Dlist);
```

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

## 5.2 Email integration

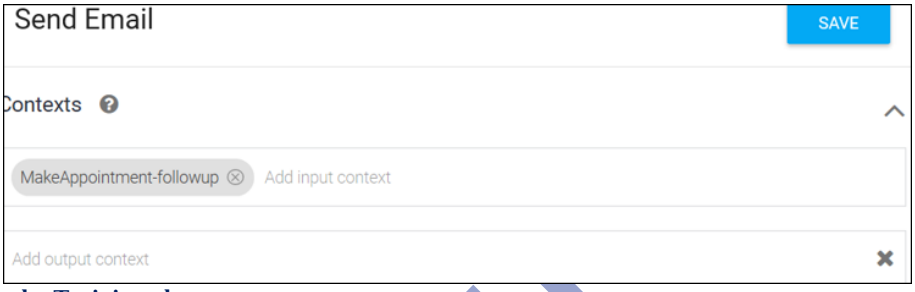
In this section, we would send an email with appointment details created in section 5.1 to the user.

### 5.2.1 Intent Creation

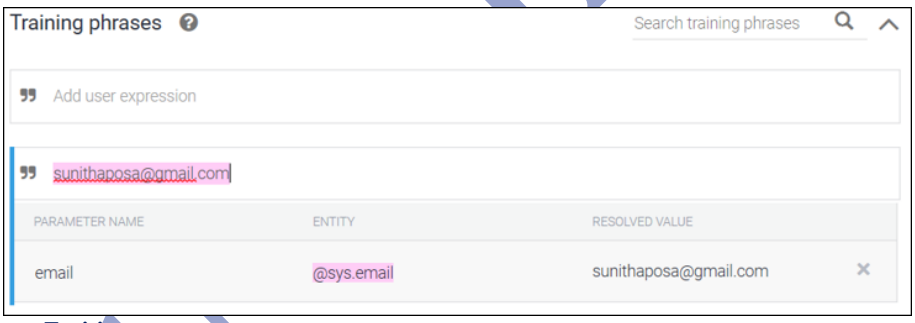
We would create a follow-up intent to **Make Appointment,** called **Send Email** with the following details:

**a. Intent Name**: Send Email

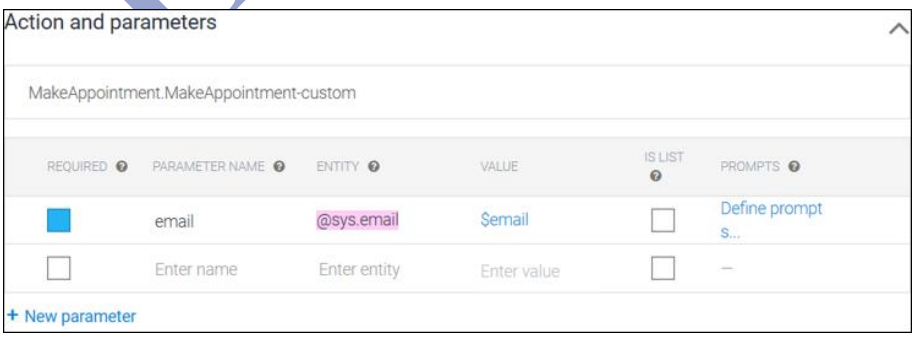
**Note**: Input Contexts automatically added to the newly created so that we can use the date and time captured in intent **Make Appointment.**




**b. Training phrases**:



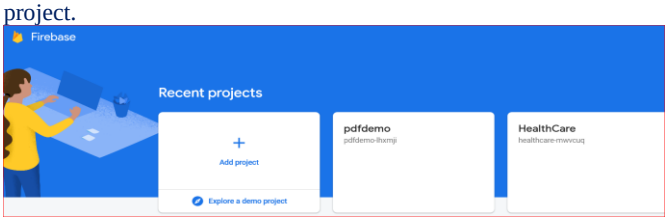

c. **Entities:**



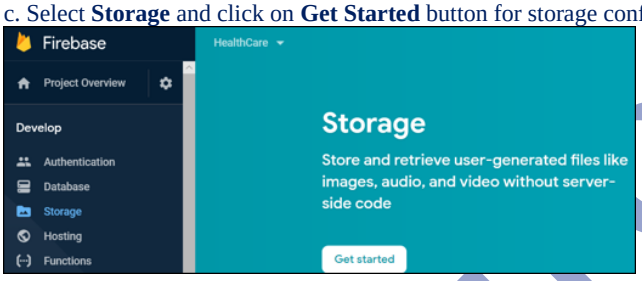

d. Enable fulfillment for this intent.

### 5.2.2 Firebase storage creation

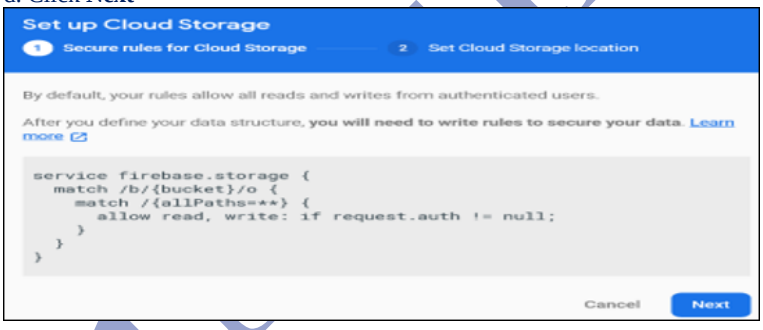
a. Go to Firebase console https://console.firebase.google.com using our google account.
b. All our google Dialogflow projects would be listed. Select our project to connect to the project.



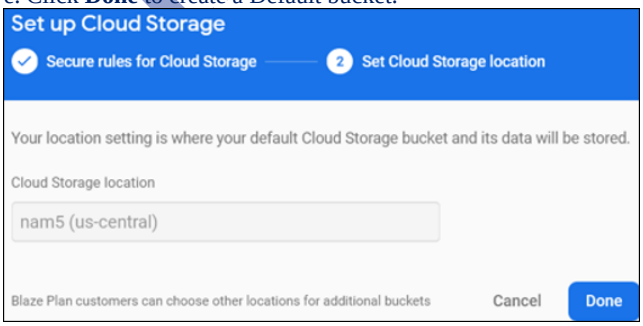

c. Select **Storage** and click on **Get Started** button for storage configuration.




d. Click **Next**




e. Click **Done** to create a Default bucket.




f. The bucket gets created.


### 5.2.3 Fulfilment code for Email Generation and sending pdf as an attachment

1. Add the following dependencies at the end of the **package.json**.

```
"nodemailer":"^4.4.2",
"dialogflow-fulfillment": "0.5.0",
"pdfkit":"0.10.0",
"firebase":"7.4.0"
```

2. Go to the index.js tab and add the following constants and variables.

```
const pdfkit = require("pdfkit");//Used to create pdf documents
const nodemailer=require("nodemailer");//used for email node configuration
var PROJECT_ID='healthcare-xxxx';//project name from Firebase
const {Storage} = require('@google-cloud/storage');//Used to get the reference the
project storage.
const storage = new Storage({
projectId: PROJECT_ID,
      });
const smtpTransport = nodemailer.createTransport({
        service: 'gmail',
          auth: {
            user: 'sunithaandraravi@gmail.com',//provide the email which would be
used to generate email
            pass: '*****'//provide the password
          }
        });


      // Initialize Firebase
var pdfproj=firebase.initializeApp(firebaseConfig);
```

3. In the index.js tab and insert the email and pdf **function.**

```
function emailandpdf(agent){
 agent.add("call to pdf function");
 const day1=agent.parameters.day;
 const time1=agent.parameters.time;
 var date1 = new Date(day1);
 var time2=new Date(time1);
 var day = date1.getFullYear()+'-'+(date1.getMonth()+1)+'-
'+date1.getDate();
 var time = time2.getHours() + ":" + time2.getMinutes() + ":" +
time2.getSeconds();
const email=agent.parameters.email;
 console.log(day);
 console.log(time);


  const bucket = storage.bucket(`${PROJECT_ID}.appspot.com`);
  const filename =  'output.pdf';
  const file = bucket.file(filename);
  const bucketFileStream = file.createWriteStream({resumable: false});

  // Pipe its output to the bucket
  doc.pipe(bucketFileStream);
  doc.fontSize(25).text('This is to confirm that the appointment is
confirmed on'+day+ 'at'+time+ 'time', 100, 100);
  doc.end();

    bucketFileStream.on("finish", function () {
    return sendOrderEmail(email, filename);
  });

  bucketFileStream.on("error", function(err) {
    console.error(err);
  });

  function sendOrderEmail(email1, filename) {
  const email = email1;
    console.log(email);
    console.log(filename);

    const mailOptions = {
    from: "admin@test.com",
    to: email,
    subject: "Appointment Confirmation",

     };
   console.log("mail options setting done");
  const bucket = storage.bucket(`${PROJECT_ID}.appspot.com`);
  const file = bucket.file(filename);
```

```
// mailOptions.html = mailTemplate;
 mailOptions.attachments =  [{
   filename: "output.pdf",
   contentType: "application/pdf",
   content: file.createReadStream()
 }];

smtpTransport.sendMail(mailOptions).then(() => {
   console.log("Appointment email sent to:", email);
 }).catch(error => {
   console.log(error);
 });
}
```

4. Add the step to map **Send Email** intent to the function handler **mainlandpdf** as shown below

```
let intentMap = new Map();
intentMap.set('Make Appointment', makeAppointment); // It maps the intent 'Make Appointment' to the function 'makeAppointment()'
intentMap.set('Send Email', emailandpdf);
agent.handleRequest(intentMap);
```

## 5.3 Webhook: WeatherCheck

In this section, we are creating a webhook to check the weather and then deploy it in Heroku.

### 5.3.1 Create Agent and Intent

Create an agent WeatherCheck and then create an intent CheckWeather with the following information:

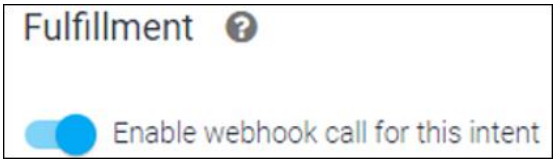
a. Intent name: CheckWeather

b. Training phrases:

- weather in Hyderabad  tomorrow
- weather forecast in Bangalore
- Weather forecast in Delhi on Thursday,

c. Add the parameters geo-city and date, as shown below:


c. Enable the Fulfillment



d. Enable Inline Editor and then add the following in the package.json tab under dependencies section:
"axios":"0.18.0"

```
dependencies": {
  "actions-on-google": "^2.2.0",
  "firebase-admin": "^5.13.1",
  "firebase-functions": "^2.0.2",
  "dialogflow": "^0.6.0",
  "dialogflow-fulfillment": "^0.5.0",
  "axios":"0.18.0"
```

## 5.32 Creating a Webhook and deployment in Heroku

a. Create a python file and then copy the below code and save it as webhook.py

```
import json #to convert list and dictionary to json
import os
import requests
from flask import Flask #it is microframework to develop a web app
from flask import request
from flask import make_response
#Falsk app for our web app

app=Flask(__name__)

# app route decorator. when webhook is called, the decorator would call the functions
which are e defined

@app.route('/webhook', methods=['POST'])

def webhook():
    # convert the data from json.
    req=request.get_json(silent=True, force=True)
    print(json.dumps(req, indent=4))
    #extract the relevant information and use api and get the response and send it
dialogflow.
    #helper function
    res=makeResponse(req)
    res=json.dumps(res, indent=4)
    r=make_response(res)
    r.headers['Content-Type']='application/json'
    return r

# extract parameter values, query weahter api, construct the resposne
def makeResponse(req):
    result=req.get("queryResult")
```

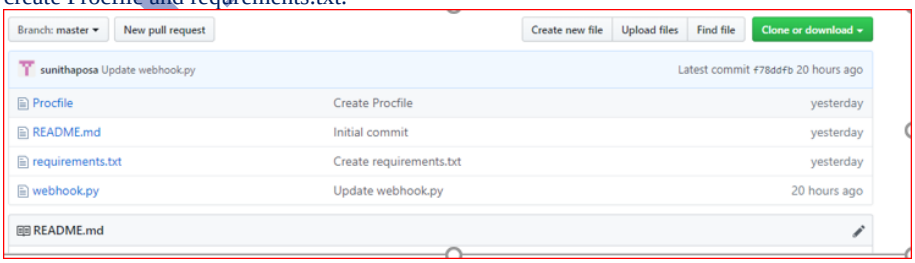
```
    parameters=result.get("parameters")
    city=parameters.get("geo-city")
    date=parameters.get("date")

r=requests.get('http://api.openweathermap.org/data/2.5/forecast?q=hyderabad,in&a
ppid=db91df44baf43361cbf73026ce5156cb')
    json_object=r.json()
    weather=json_object['list']


    #for i in range(0,40):
    #    if date in weather[i]['dt_txt']:
    #        condition=weather[i]['weather'][0]['description']
    condition=weather[0]['weather'][0]['description']
    speech="The forecast for "+city+ " for "+date+" is "+ condition
    return{
  "fulfillmentMessages": [
    {
      "text": {
        "text": speech
      }
    }]}
    #return {
    #    "speech": speech,
    #    "displayText":speech,
    #    "source":"apiai-weather-webhook"}

if __name__=='__main__':
    port=int(os.getenv('PORT',5000))
    print("starting on port %d" % port)
    app.run(debug=False, port=port, host='0.0.0.0')
```

b. Create a git repository and upload the webhook.py file. We would discuss the steps to create Procfile and requrements.txt.




Note: Git account setup and repository creation are required to deploy the code in Heroku.

c. Click on “create new file” with the name- requirements.txt and paste the following data and commit the changes.

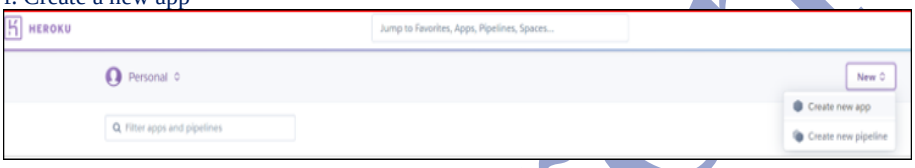
```
FLASK==0.12.2
Requests==2.18.4
```

d. Create the file  Procfile and paste the following data and then commit the changes:

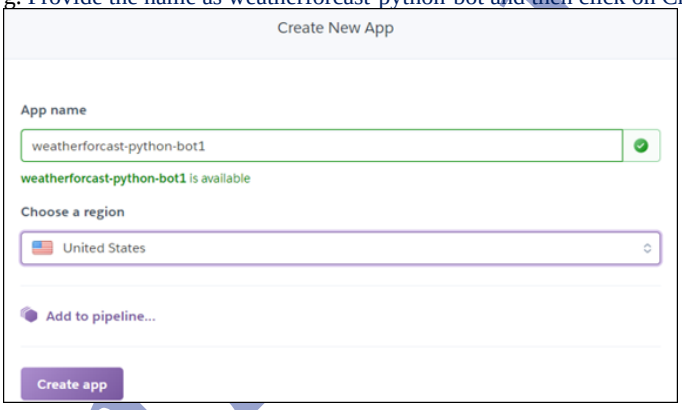
```
web: python webhook.py
```

e. Create an account in Heroku and login to it.
https://signup.heroku.com/

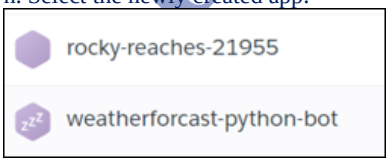
f. Create a new app



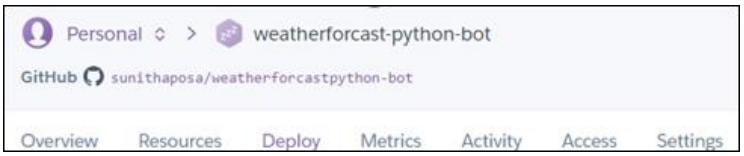

g. Provide the name as weatherforcast-python-bot and then click on Create app



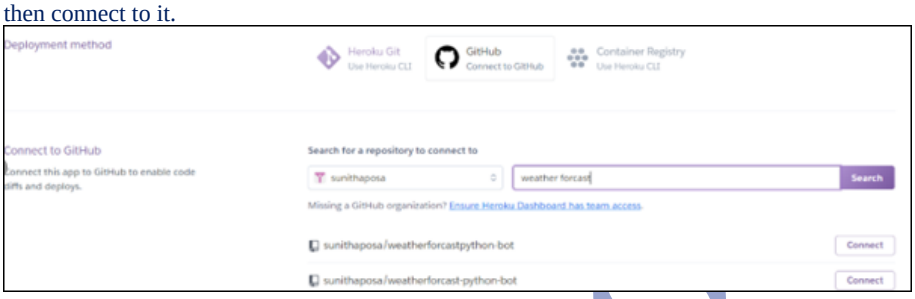

h. Select the newly created app.



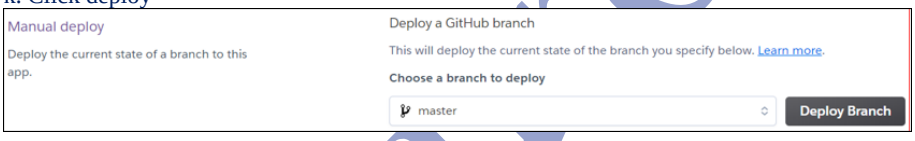

i. Select Deploy



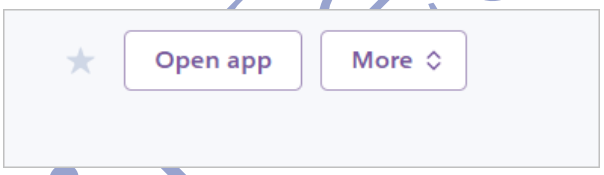

j. Under the Deployment method, select “Github” and then search for the GIT repository and then connect to it.




k. Click deploy




l. Click the open app button to get the webhook.




m. Copy the URL, go to the Dialogflow fulfillment and paste the URL under the webhook section of Fulfillment.