

Building Chatbot using Google Dialogflow

Contents

About This Guide	3
1.Chatbot Concept	4
1.1 What are chatbots	4
1.2 Use cases	4
1.3 Advantages of Chatbots	5
1.4 Models for a Chatbot	5
1.5 Architecture of AI chatbot	6
1.6 Chatbot Frameworks	7
2 Google Dialog Flow	8
2.1 Introduction	8
2.2 Why choose Google Dialogflow	8
2.3 How do Chatbots work?	8
2.4 Build your first chatbot	9
2.4.1 Signup for Dialogflow account	9
2.4.2 Creating an Agent	11
2.4.3 Creating an Intent	12
2.4.4 Test it Now	14
2.5 Understanding the building blocks of Dialogflow	15
2.5.1 Agent:	15
2.5.2 Intent	15
2.5.3 Entities	17
2.5.4 Fulfilment:	18
2.5.5 Integrations	19
2.5.6 Training:	20
2.5.6 History	20
2.5.7 Analytics	21
2.5.8 Pre-built Agents:	21
3. Dialogflow: Linear and Non-linear	22
3.1 Linear Dialogflow	22
3.1.1 Linear Dialogflow using Single intent:.....	22
3.1.2 Linear Dialogflow using multiple intents:	27
3.2 Non- Linear Dialogflow:	35
4. Integrations	37
4.1 Dialogflow Web Demo:	37
4.2 Telegram:	38
5.1 Fulfilment and External API integrations:.....	39

5.1 External API call - Clinics list based on specialty	39
5.1.1 Generating the API key	39
5.1.2 Create Agent and Intent	39
5.1.3 Fulfilment code:	40
5.2 Email integration	42
5.2.1 Intent Creation	42
5.2.2 Firebase storage creation	43
5.2.3 Fulfilment code for Email Generation and sending pdf as an attachment	44
5.3 Webhook: WeatherCheck	46
5.3.1 Create Agent and Intent	46
5.3.2 Creating a Webhook and deployment in Heroku	47

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.

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.

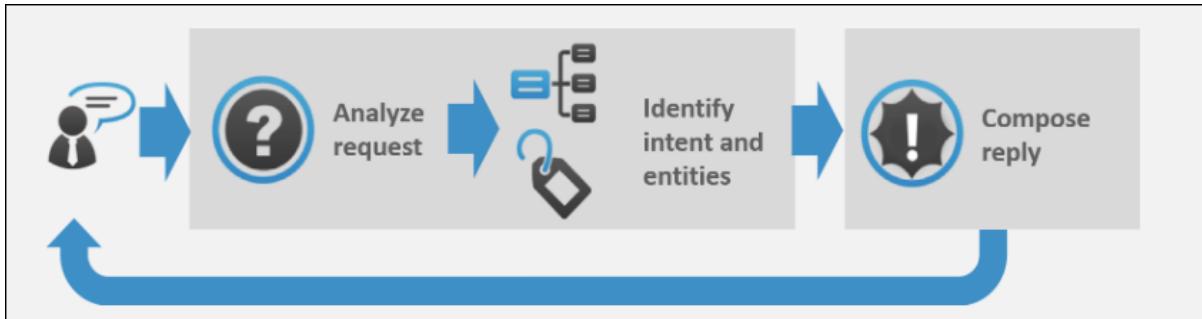


Image Courtesy: <https://expertsystem.com/chatbot/>

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 basic 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 provides reviews.

1.3 Advantages of Chatbots

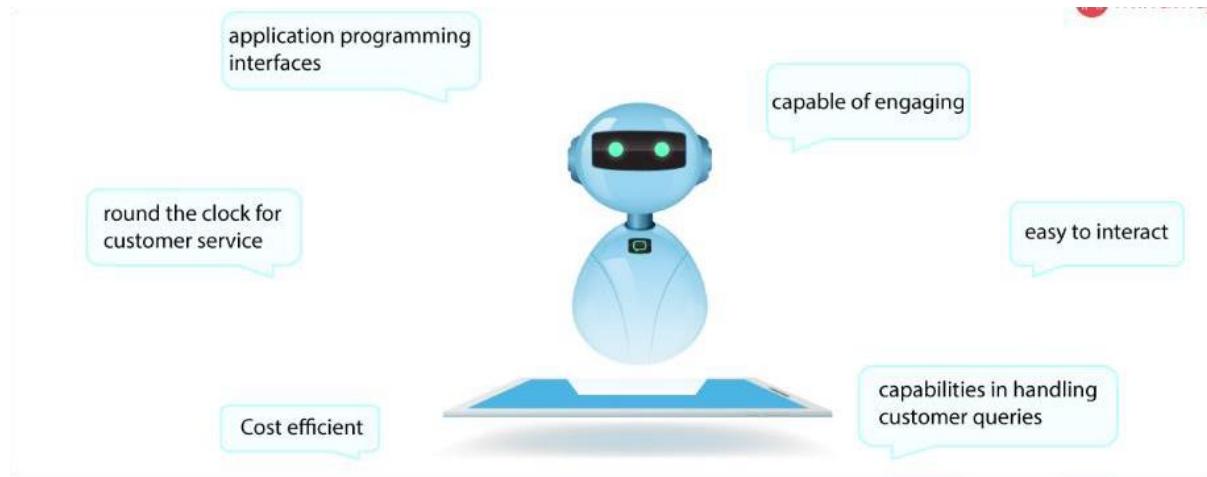


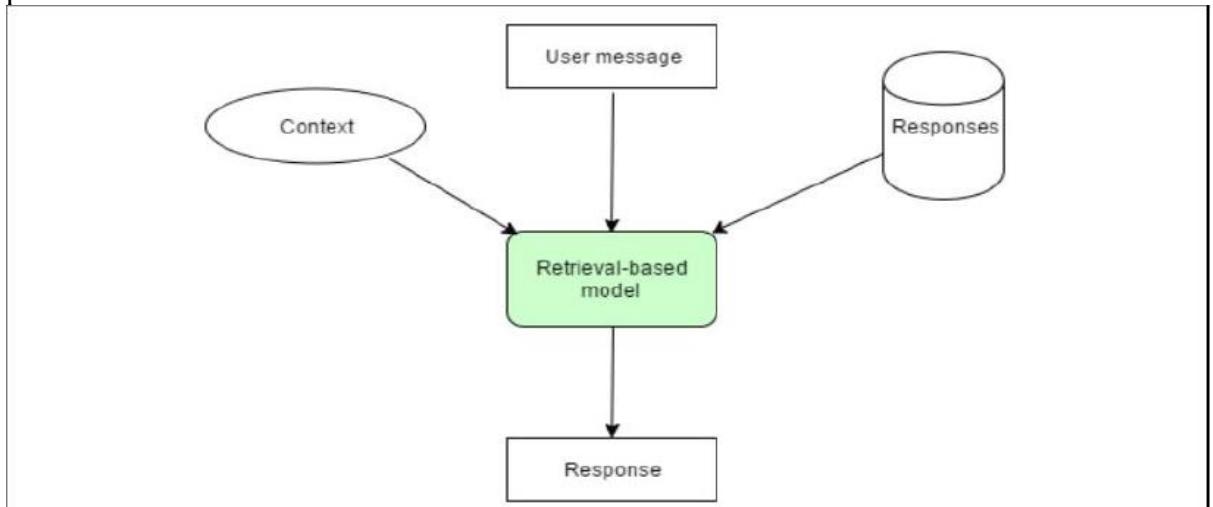
Image Courtesy: <https://mindmajix.com/chatbot>

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.



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."

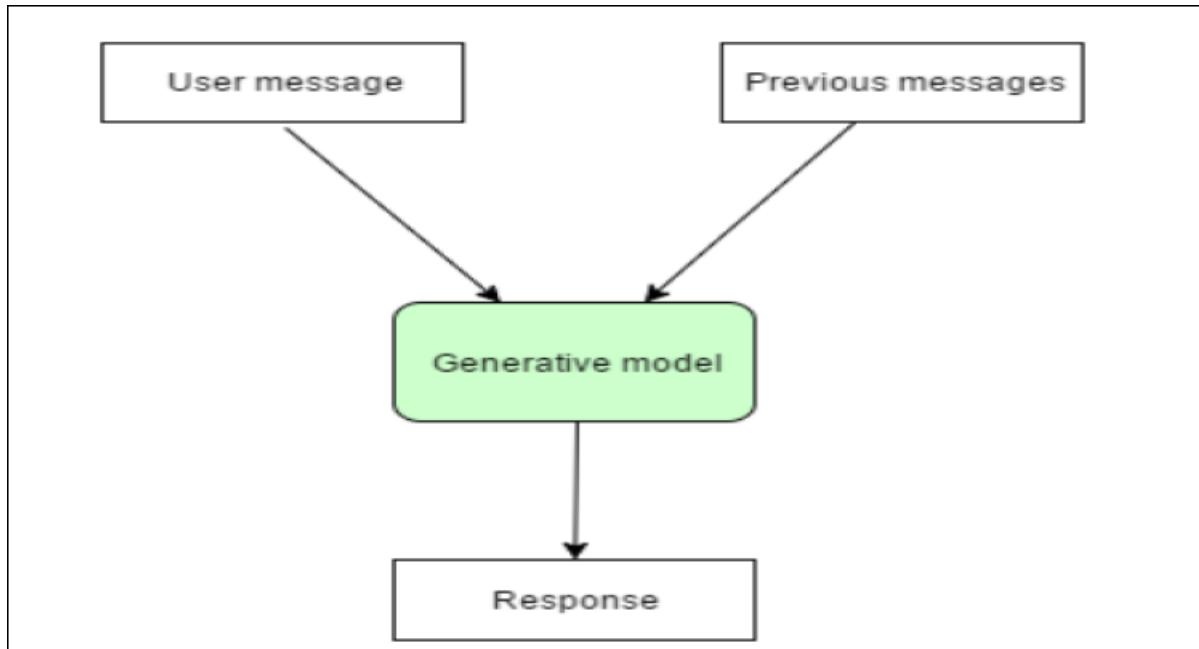


Image Courtesy: <https://mindmajix.com/chatbot>

1.5 Architecture of AI chatbot

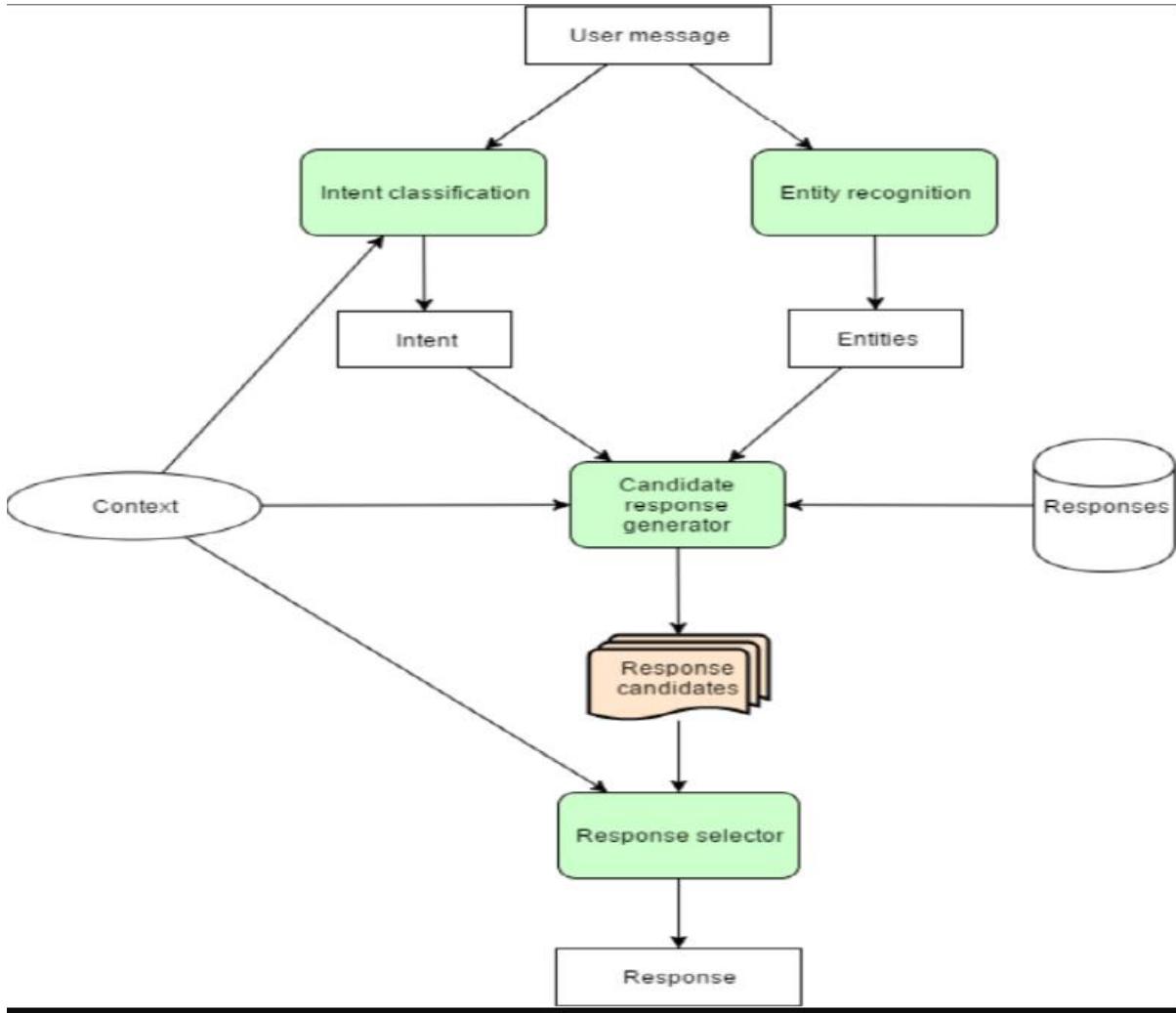


Image Courtesy: <https://mindmajix.com/chatbot>

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 intelligent enough to predict the hidden intention expressed in the input language. A Dialogflow chatbot maps 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-agnostic 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?

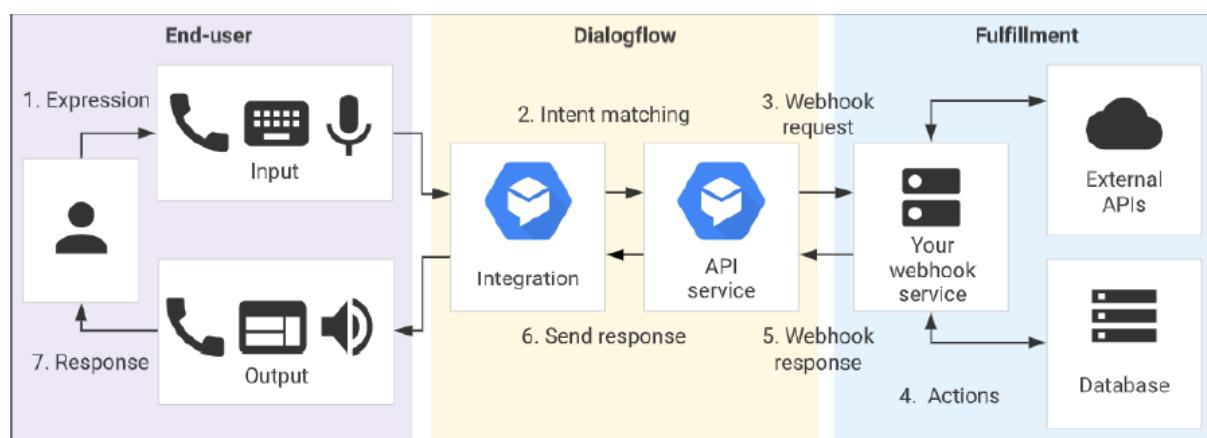


Image Courtesy: <https://cloud.google.com/dialogflow/docs/basics>

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

2.4 Build your first chatbot

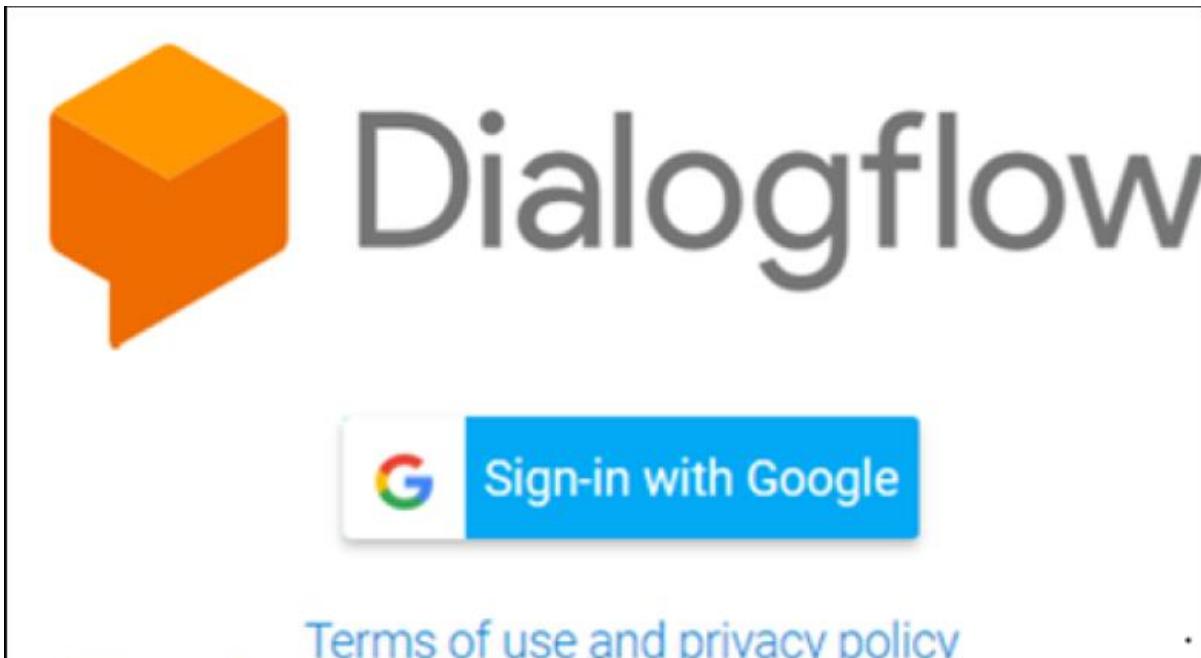
2.4.1 Signup for Dialogflow account

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

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

The screenshot shows the official Dialogflow website. At the top, there is a navigation bar with links for Overview, Case studies, Docs, Blog, Pricing, and Support. The main headline on the page reads "Build natural and rich conversational experiences". Below this, there is a descriptive paragraph about how Dialogflow allows users to interact with their products through various platforms like voice apps and chatbots. At the bottom of the page, there is an orange "Sign up for free" button.

- Click on Sign-in with Google



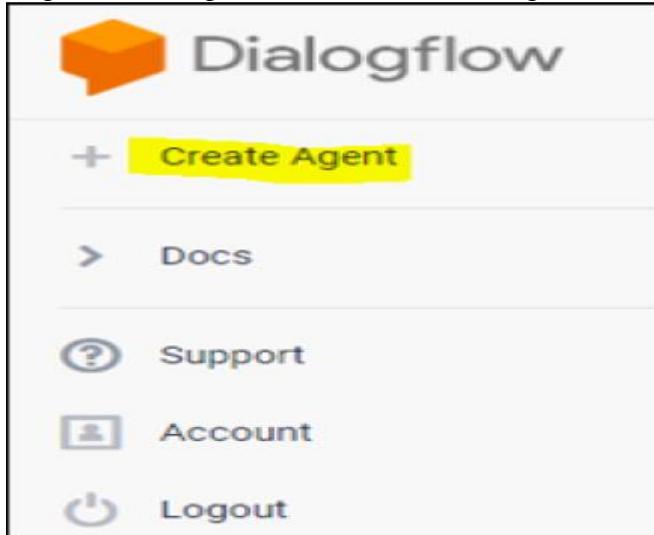
- Select your google account, and it will direct to the Google Dialogflow home page. Click on Go to console in the upper right corner to navigate to the home page of Google Dialogflow.



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.

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



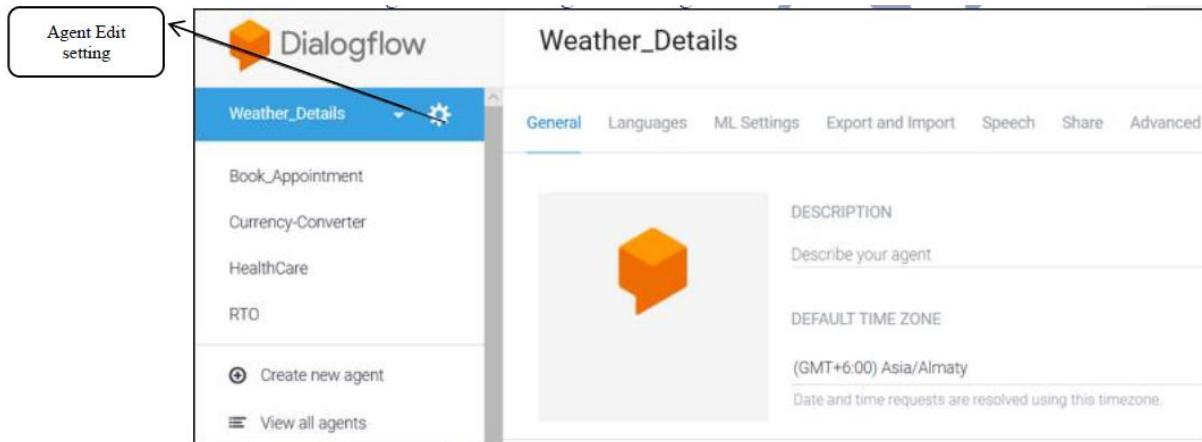
- Provide the name of the agent and click on the SAVE button to create the Agent.

The screenshot shows the 'Weather_Details' agent configuration page. At the top is the agent name 'Weather_Details' and a 'SAVE' button. Below are tabs for General, Languages, ML Settings, Export and Import, Speech, Share, and Advanced. The General tab is selected. On the left is a large orange speech bubble icon. The right side contains fields for 'DESCRIPTION' (with placeholder 'Describe your agent') and 'DEFAULT TIME ZONE' (set to '(GMT+6:00) Asia/Almaty'). A note below says 'Date and time requests are resolved using this timezone.'

- 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.

The screenshot shows the main Dialogflow interface. On the left is a sidebar with a list of agents: 'Book_Appointment', 'Currency_Converter', 'HealthCare', 'RTO', '+ Create new agent', and 'View all agents'. The 'Weather_Details' agent is selected. The main area shows its configuration details, including the orange speech bubble icon, 'DESCRIPTION' field ('Describe your agent'), and 'DEFAULT TIME ZONE' set to '(GMT+6:00) Asia/Almaty'. A note at the bottom says 'Date and time requests are resolved using this timezone.'

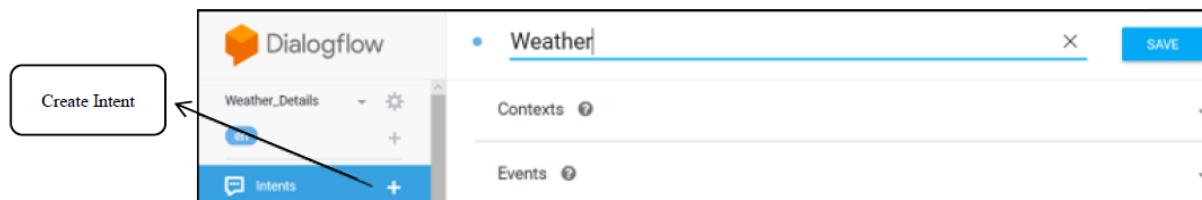
- Edit the Agent details using the setting icon.



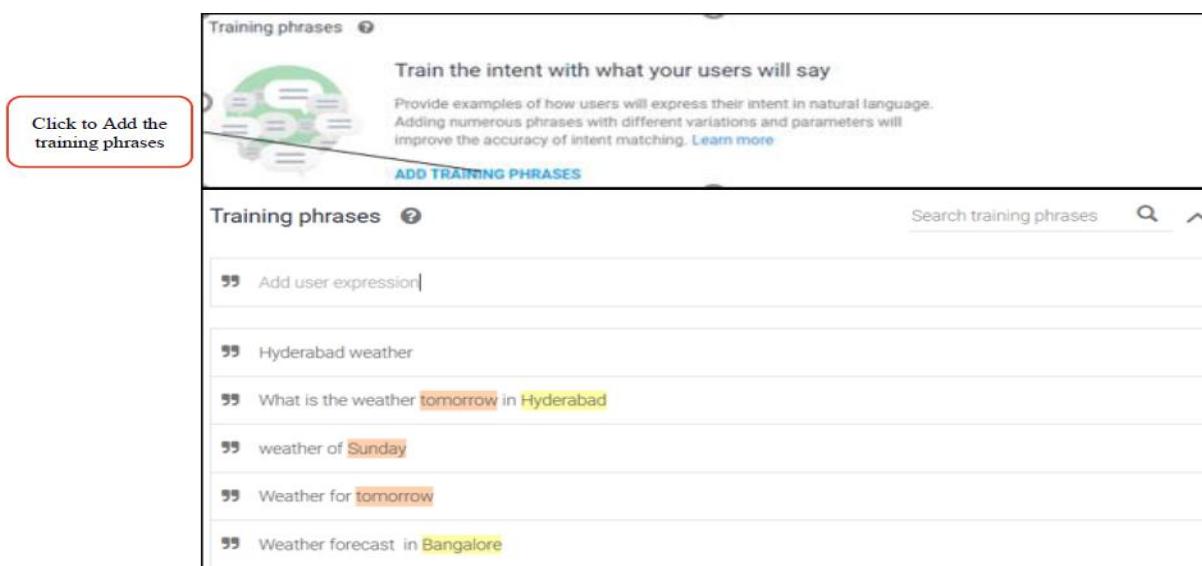
2.4.3 Creating an Intent

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

- 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.



- 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.



- 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.

55	What is the weather tomorrow in Hyderabad	
PARAMETER NAME	ENTITY	RESOLVED VALUE
date	@sys.date	tomorrow
geo-city	@sys.geo-city	Hyderabad
55	weather of Sunday	
55	Weather for tomorrow	

- Click on ADD RESPONSE to provide an appropriate response to user queries.

Responses 



Execute and respond to the user
Respond to your users with a simple message, or build custom rich messages for the integrations you support. [Learn more](#)

[ADD RESPONSE](#)

- Add the below Responses:

Responses 

DEFAULT 

Text Response		 
1	Sorry, I don't know the response.	
2	Weather is pleasant in \$geo-city	
3	Weather is hot \$da	
4	Enter a text response variant	

[ADD RESPONSES](#)

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

• Weather 

2.4.4 Test it Now

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

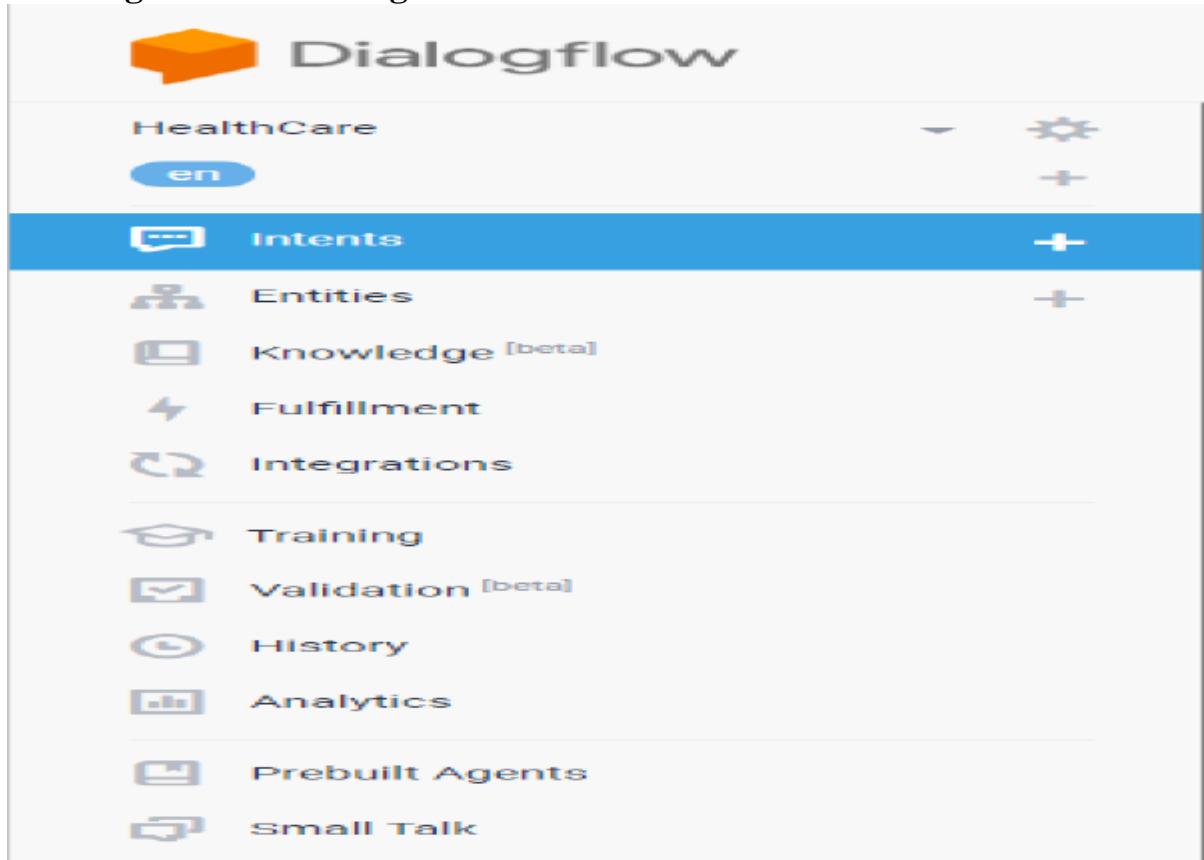
The screenshot shows the Dialogflow test console interface. At the top, there is a "Try it now" button and a microphone icon. Below that, a "See how it works in Google Assistant" link with a blue and yellow icon. The main area is titled "Agent". It displays the following information:

- USER SAYS:** what is the weather in hyderabad
- COPY CURL** button
- DEFAULT RESPONSE:** Weather is pleasant in Hyderabad
- INTENT:** Weather
- ACTION:** Not available
- PARAMETER** date
- VALUE**
- PARAMETER** geo-city
- VALUE** Hyderabad

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.

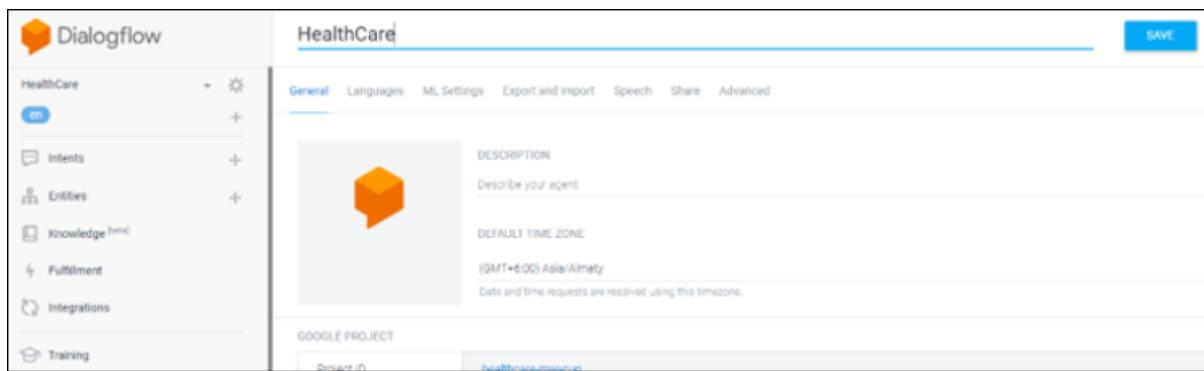
Building blocks of Dialogflow:



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.

The screenshot shows the Dialogflow interface for managing intents. At the top, there's a search bar labeled "Search intents" and a blue "CREATE INTENT" button. Below the search bar is a list of intents: "Default Fallback Intent", "Default Welcome Intent", and "Weather".

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.

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

The screenshot shows the Dialogflow interface for creating a new intent. The intent name is "Online Consultation". The sidebar on the left shows "HealthCare" selected under "Intents". The main area includes sections for "Contexts", "Events", and "Training phrases". A "SAVE" button is visible at the top right.

2) **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.

The screenshot shows the "Training phrases" section of the Dialogflow interface. It lists several user expressions: "Add user expression!", "Please provide suggestion", "Please advise", "we would like to have advise", "like to have a Consultation", and "Provide Online Consultation".

- 3) 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.

Action and parameters

Enter action name				
REQUIRED	PARAMETER NAME	ENTITY	VALUE	IS LIST
<input type="checkbox"/>	Consultation	@Consultation	\$Consultation	<input type="checkbox"/>
<input type="checkbox"/>	test	@sys.date	\$test	<input type="checkbox"/>
<input type="checkbox"/>	Enter name	Enter entity	Enter value	<input type="checkbox"/>

[+ New parameter](#)

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

Responses [?](#)

[DEFAULT](#) +

Text Response	
1	Thank you for choosing our services. Please provide your name, age and phone number as part of the process.
2	Please provide the details like name, age, phone number for our recording purpose.
3	Enter a text response variant

[ADD RESPONSES](#)

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.

The screenshot shows the Dialogflow Entities interface. On the left, there are navigation tabs for 'HealthCare' (selected), 'Intents', and 'Entities'. The main area displays a table with columns for 'NAME', 'TYPE', and 'DESCRIPTION'. One row is visible: 'HealthCare' (Custom Entity, System-defined). A search bar at the bottom shows the query 'Consultation'.

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.

The screenshot shows the 'Consultation' entity configuration page. At the top, there are several checkboxes: 'Define synonyms' (checked), 'Regexp entity', 'Allow automated expansion', and 'Fuzzy matching'. Below the checkboxes is a table listing synonyms. The table has two columns: 'Term' and 'Definition'. The terms listed are 'advise', 'appointment', 'consultation', 'support', 'suggestion', and 'help'. The definitions are 'advise, advises', 'appointment', 'consultation', 'support', 'suggestion, suggest', and 'help' respectively. A blue 'SAVE' button is located at the top right of the form.

Term	Definition
advise	advise, advises
appointment	appointment
consultation	consultation
support	support
suggestion	suggestion, suggest
help	help

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.

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

The screenshot shows the 'Fulfillment' section of an intent configuration. It includes a note: "To enable fulfillment for an intent, toggle **Enable webhook call for this intent** in the **Fulfillment** section of the intent and click **SAVE**." Below the note are two toggle switches: one for "Enable webhook call for this intent" (which is turned on) and another for "Enable webhook call for slot filling".

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

The screenshot shows the 'Fulfillment' section of the Dialogflow interface. It includes two main configuration sections: 'Webhook' and 'Inline Editor'. The 'Webhook' section is currently set to 'DISABLED' (indicated by a grey toggle switch). A descriptive text explains that your web service will receive a POST request from Dialogflow when a user query matches an intent with webhook enabled. The 'Inline Editor' section is set to 'ENABLED' (indicated by a blue toggle switch). Below it, there's a link to 'Build and manage fulfillment directly in Dialogflow via Cloud Functions for Firebase' with a 'Docs' link.

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

The screenshot shows the 'Code Editor' section of the Dialogflow interface. It displays a snippet of JavaScript code (labeled lines 32-41) which generates a card with an image URL, body text, a button text, and a button URL. Below the code editor is a large blue 'DEPLOY' button.

- 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.

The screenshot shows the 'Code Editor' section of the Dialogflow interface, specifically the 'index.js' file. The code contains a single line: 'use strict';. There is also a 'package.json' file listed in the sidebar. A cursor icon is visible over the code editor area.

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.

The screenshot shows the Dialogflow interface with the 'Integrations' tab selected. On the left sidebar, under 'HealthCare', there are sections for 'Intents', 'Entities', 'Knowledge', 'Fulfillment', and 'Integrations'. The 'Integrations' section is highlighted with a blue background. In the main area, there is a large graphic of four overlapping colored circles (blue, red, yellow, green). To the right, there is a section titled 'Google Assistant' with three circular icons representing different devices: a smartphone, a tablet, and a computer monitor. Below this, it says 'Build Actions for the Google Assistant to reach users through Google Home, Android phones, and more devices.' At the bottom of the main area, there are four small icons labeled 'Web Demo', 'Facebook Messenger', 'Dialogflow Phone Gateway', and 'Slack', each with a corresponding logo.

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.

The screenshot shows the Dialogflow interface with the 'Training' tab selected. On the left sidebar, under 'HealthCare', there are sections for 'Intents', 'Entities', 'Knowledge', 'Fulfillment', and 'Integrations'. The 'Training' section is highlighted with a blue background. In the main area, there is a table titled 'Training' with columns for 'Conversation', 'Requests', 'No match', and 'Date'. The table contains five rows of data: 'cardiologist' (1 request, 0 no match, Today), 'Book Appointment Yes' (1 request, 0 no match, Today), 'book appointment yes' (28 requests, 0 no match, Today), and 'no' (28 requests, 0 no match, Today). At the top right of the table is a blue 'UPLOAD' button. At the bottom right are 'PREVIOUS' and 'NEXT' buttons.

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.

The screenshot shows the 'History' section of the Dialogflow interface. On the left, a sidebar lists various project components: HealthCare, Intents, Entities, Knowledge, Fulfilment, Integrations, Training, Validation, and History. The 'History' option is selected and highlighted in blue. The main area is titled 'History' and displays a table of conversations. The columns include 'Conversation' (with examples like 'cardiologist'), 'Date' (with dates like 'Nov 6' and 'Today'), and a small orange circular icon with a number (e.g., '1'). A date range selector at the top right shows 'Oct 31, 2019' to 'Nov 12, 2019'.

2.5.7 Analytics

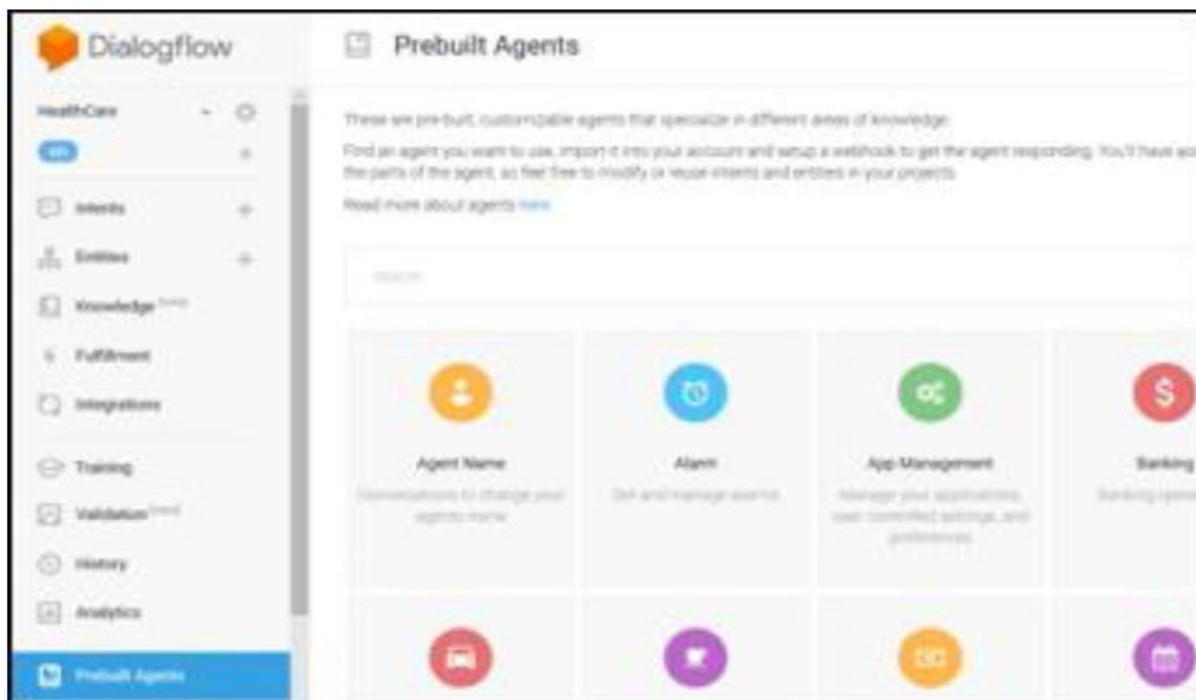
The Analytics screen of Dialogflow gives understanding into how well the agent is performing so that we can work on to improve further the user experience you're providing.

The screenshot shows the 'Analytics' section of the Dialogflow interface. The sidebar on the left is identical to the History screen, with 'Analytics' now selected. The main area is titled 'Analytics' and contains two primary sections: 'Sessions' and 'Intents'.
Sessions: It shows a chart titled 'Sessions last 7 days' with a value of '2'. Below the chart, it says 'Question per session last 7 days: 25.50'.
Intents: It shows a table of intents with columns: Intent, Questions, Count, and Agent %.

Intent	Questions	Count	Agent %
Default: Welcome Intent	1	23	21.34%
DISABILITY	1	16	12.80%

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.



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.

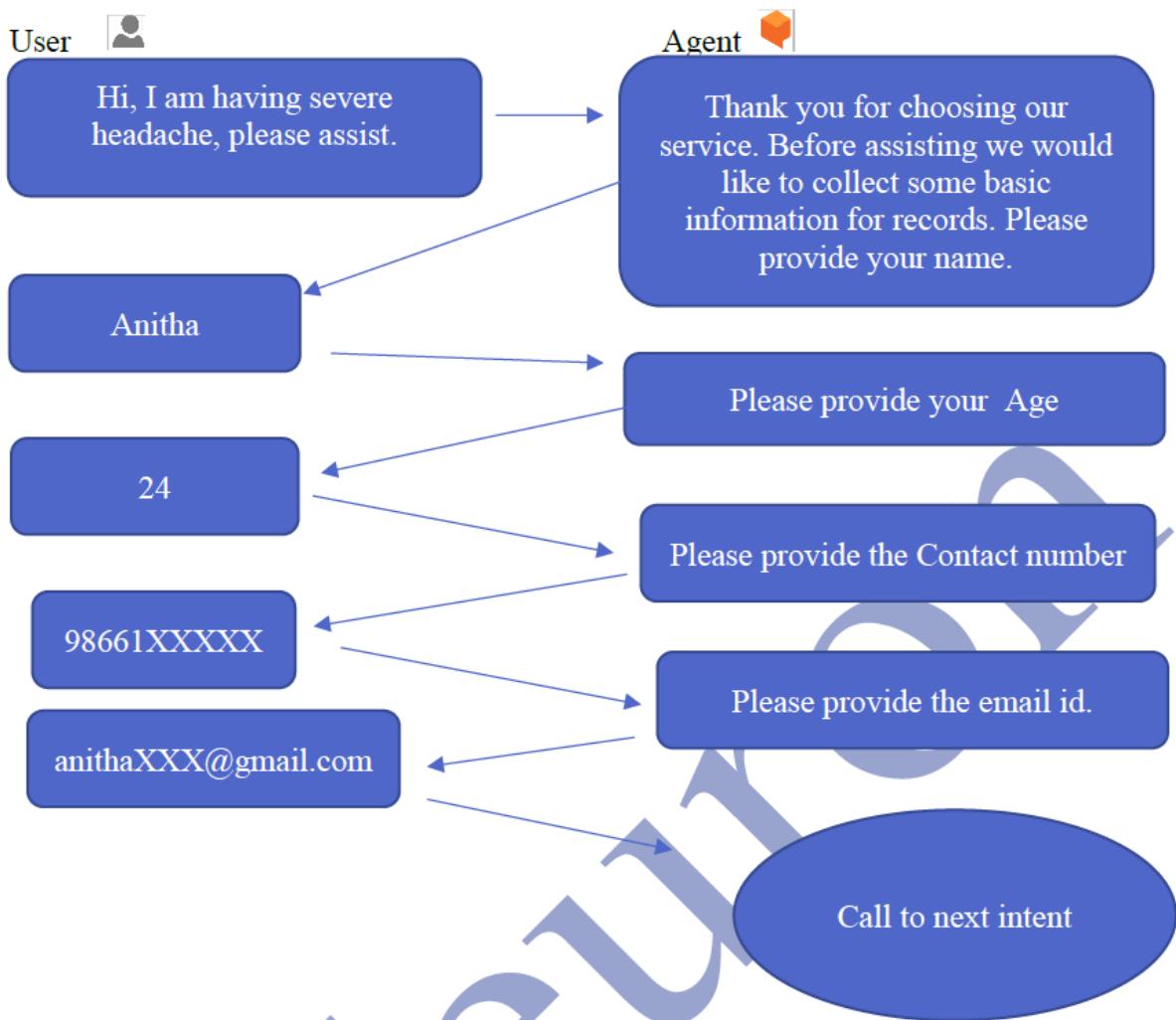
3.1 Linear Dialogflow

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.

Training phrases ?

99 Add user expression

99 Assist me with Cold and Cough

99 A little feverish

99 Please assist me with the throat infection.

99 Hi, I am having Headache

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.

Disease X SAVE ...

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

Cold and Cough	Cold and Cough
Fever	Fever, Feverish
Headache	Headache, pain in head
Stomachache	Stomachache, pain in stomach
Throat infection	Throat infection, pain in throat, pain in swallowing

Click here to edit entry

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.”

The screenshot shows two panels. The left panel is titled 'Action and parameters' and lists several entries under 'Training phrases'. One entry is 'Assist me with Cold' with a dropdown menu showing 'Cold' selected. The right panel is titled 'Training phrases' and shows a list of user expressions: 'A little feverish', 'Please assist me with @sys.geo-country-code', 'Hi, I am having Head', 'disease', 'A little feverish', 'Please assist me with @Disease', and 'Hi, I am having Head'. The word 'disease' is highlighted in the list.

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.

This screenshot shows a detailed view of a training phrase. The phrase 'A little feverish' is selected. Below it, a table shows the mapping between parameters and entities. The table has columns: 'PARAMETER NAME', 'ENTITY', and 'RESOLVED VALUE'. One row shows 'Disease' mapped to '@Disease' with the resolved value 'feverish'. There is also a row for the phrase 'Please assist me with the throat infection'.

PARAMETER NAME	ENTITY	RESOLVED VALUE
Disease	@Disease	feverish
Please assist me with the throat infection		

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

+ New parameter

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

Action and parameters

Action and parameters				
Enter action name				
REQUIRED	PARAMETER NAME	ENTITY	VALUE	IS LIST
<input type="checkbox"/>	Disease	@Disease	\$Disease	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	Name	@sys.given-name	\$Name	<input type="checkbox"/>

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

Action and parameters

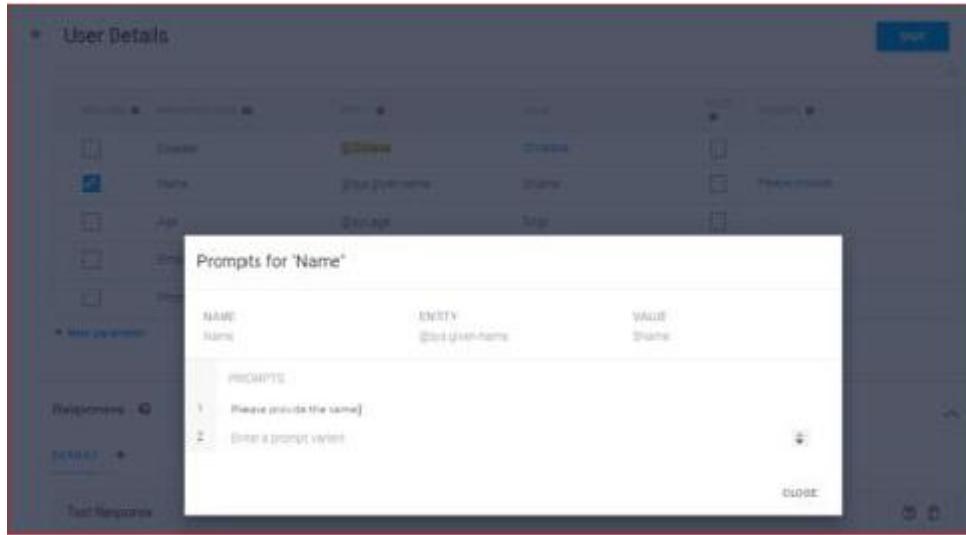
Action and parameters				
Enter action name				
REQUIRED	PARAMETER NAME	ENTITY	VALUE	
<input type="checkbox"/>	Disease	@Disease	\$Disease	
<input checked="" type="checkbox"/>	Name	@sys.given-name	\$Name	
<input checked="" type="checkbox"/>	Age	@sys.age	\$Age	
<input checked="" type="checkbox"/>	Email	@sys.email	\$email	
<input checked="" type="checkbox"/>	PhoneNumber	@sys.phone-nur	\$PhoneNumber	

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

Action and parameters

Action and parameters				
Enter action name				
REQUIRED	PARAMETER NAME	ENTITY	VALUE	IS LIST
<input type="checkbox"/>	Disease	@Disease	\$Disease	<input type="checkbox"/>
<input checked="" type="checkbox"/>	Name	@sys.given-name	\$Name	<input type="checkbox"/> Define prompts...

- 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.

REQUIRED	PARAMETER NAME	ENTITY	VALUE	IS LIST	PROMPTS
<input type="checkbox"/>	Disease	@Disease	\$Disease	<input checked="" type="checkbox"/>	—
<input checked="" type="checkbox"/>	Name	@sys.given-name	\$Name	<input type="checkbox"/>	Please provide ...

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

REQUIRED	PARAMETER NAME	ENTITY	VALUE	IS LIST	PROMPTS
<input type="checkbox"/>	Disease	@Disease	\$Disease	<input checked="" type="checkbox"/>	—
<input checked="" type="checkbox"/>	Name	@sys.given-name	\$Name	<input type="checkbox"/>	Please provide ...
<input checked="" type="checkbox"/>	Age	@sys.age	\$Age	<input type="checkbox"/>	Please provide ...
<input checked="" type="checkbox"/>	Email	@sys.email	\$Email	<input type="checkbox"/>	Please provide ...
<input checked="" type="checkbox"/>	PhoneNumber	@sys.phone-number	\$PhoneNumber	<input type="checkbox"/>	Please provide ...
<input type="checkbox"/>	Enter name	Enter entity	Enter value	<input type="checkbox"/>	—

- 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.

The screenshot shows the 'Responses' section for the 'DEFAULT' intent. It contains a 'Text Response' card with two entries:

- 1 Thank you for providing the details. Please let us know since when you are observing the SDisease
- 2 Enter a text response variant

Below the card is a 'ADD RESPONSES' button.

- i. Test the flow using the “Try it now” option discussed in the earlier section.

The screenshot shows the 'Contexts' section for the 'Schedule Appointment' intent. It includes an 'Add input context' field and a list of active contexts:

- 50 ScheduleAppointment-followup
- 5 SAF

Below the list is an 'Add output context' button.

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.

Schedule Appointment - yes

SAVE

Contexts ⓘ

ScheduleAppointment-followup ⓧ SAF ⓧ Add input context

5 SAF ⓧ Add output context ⓧ

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

Disease intensity

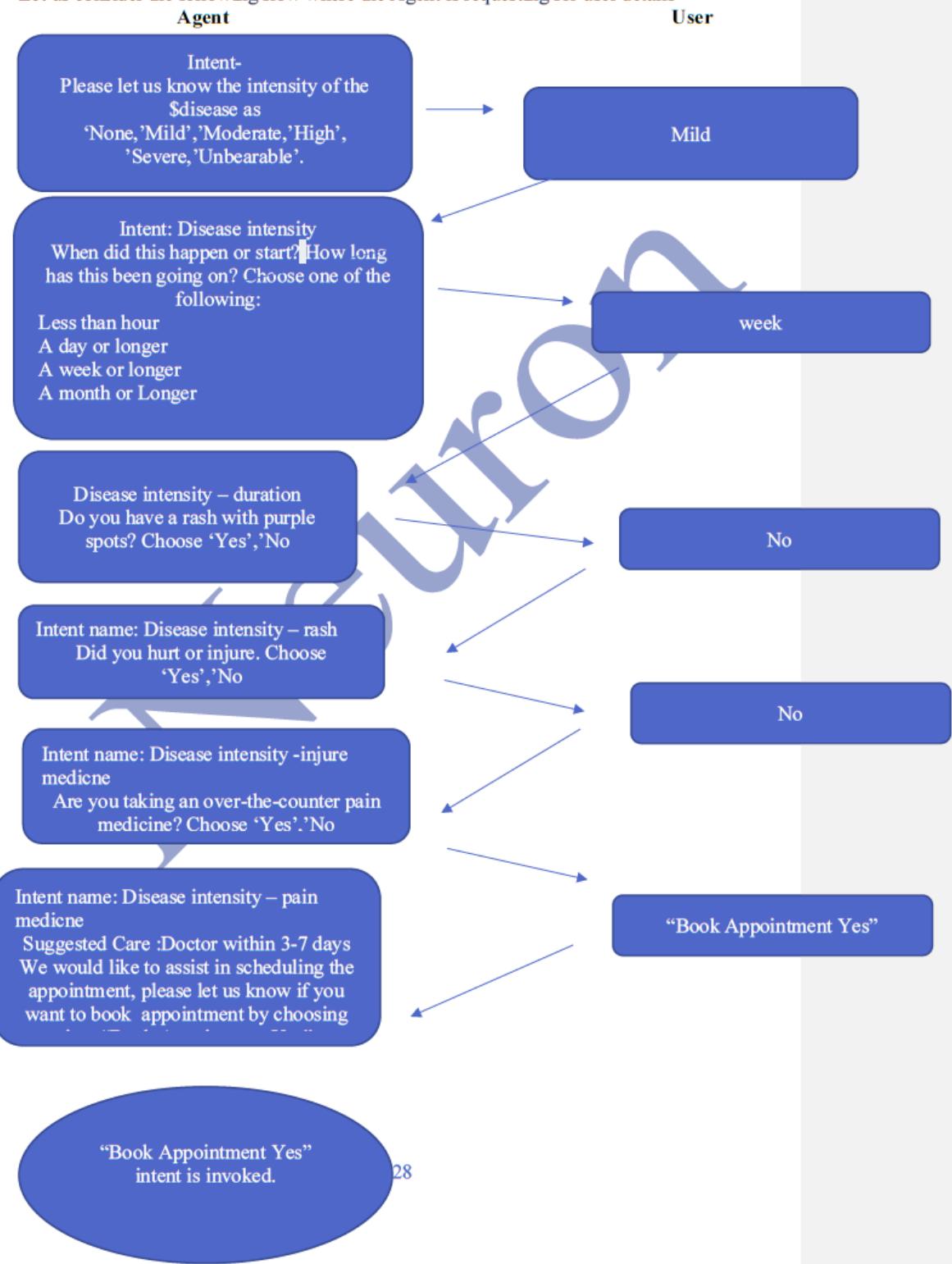
SAVE

Contexts ⓘ

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:

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

A screenshot of a user interface showing a list item titled "Disease intensity". To the right of the item is a blue "SAVE" button. Below the list item is a section labeled "Contexts" with a help icon (a question mark inside a circle).

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

A screenshot of the "Disease_Intensity" entity configuration screen. At the top, there are several checkboxes: "Define synonyms", "Regexp entity", "Allow automated expansion", and "Fuzzy matching". Below these are six entity values listed vertically: "Mild", "None", "Moderate", "Severe", "Worst", and "Unbearable". A blue "SAVE" button is located at the top right.

- c. Go to Training phrases and provide a phrase and assign data to Disease_Intensity” entity, as shown below:

A screenshot of the "Training phrases" section. A new entry is being created, starting with the prefix "Sever". The "Action" field contains "@Disease" and "@Disease_Intensity". A dropdown menu is open over the "@Disease_Intensity" field, showing the options "Enter" and "Create new". A "Create new" button is visible at the bottom of the dropdown menu.

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

Responses ?

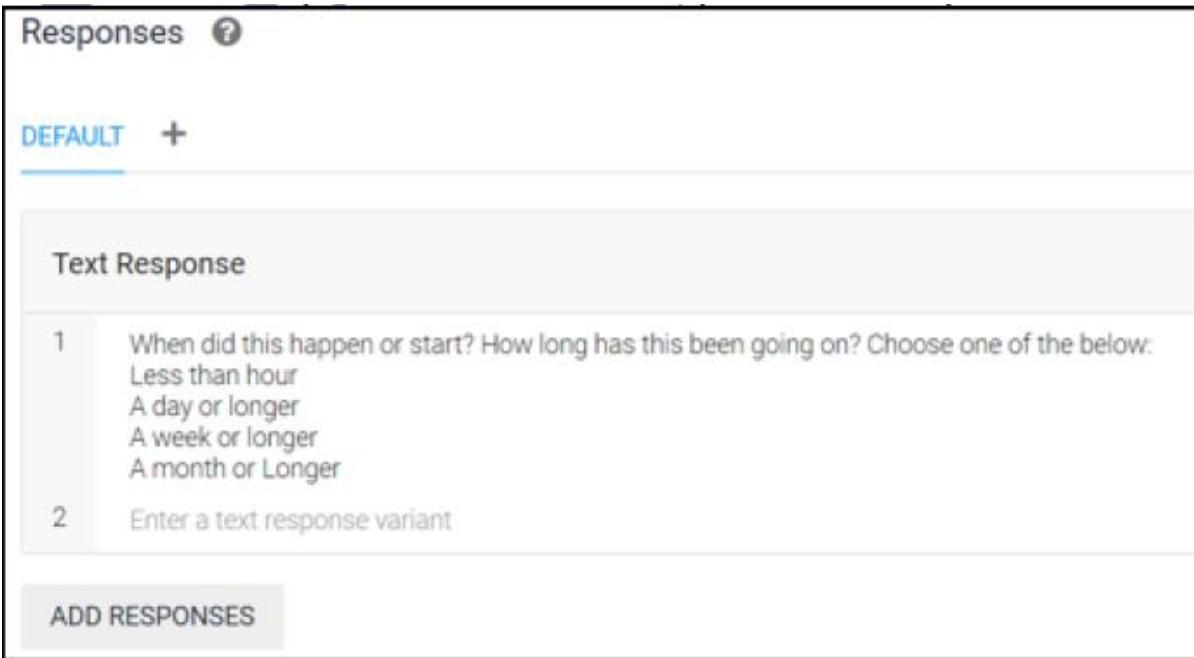
DEFAULT +

Text Response

1 When did this happen or start? How long has this been going on? Choose one of the below:
Less than hour
A day or longer
A week or longer
A month or Longer

2 Enter a text response variant

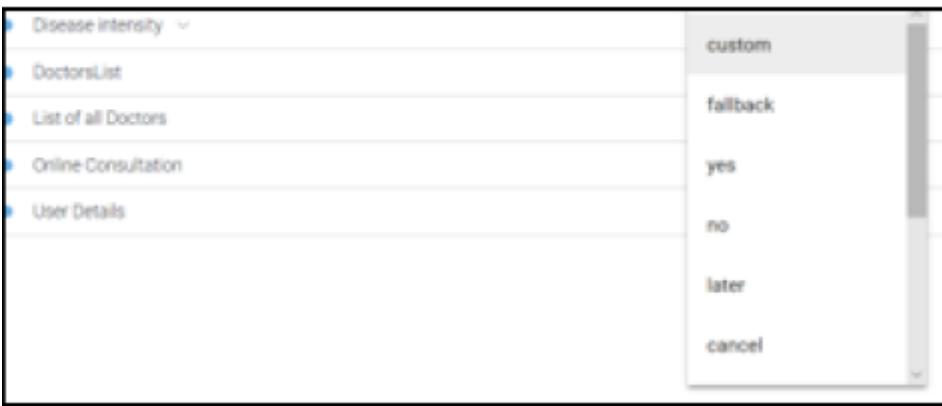
ADD RESPONSES



- 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.

The screenshot shows a list of intents under the heading 'Disease intensity'. The first item is 'Disease intensity' with a blue circular icon. Below it is 'Disease intensity - custom' with a blue circular icon and a right-pointing arrow. The 'Disease intensity - custom' item is highlighted with a dark blue border.

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**

The screenshot shows the configuration page for the 'Disease intensity - duration' intent. At the top, there is a list item with a blue dot next to 'Disease intensity - duration'. To the right of this list item are three buttons: a help icon, a blue 'SAVE' button, and a more options icon. The main area below the list is currently empty.

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.

The screenshot shows the 'Contexts' section for the 'Disease intensity - duration' intent. At the top, there is a list item with a blue dot next to 'Disease intensity - duration'. To the right of this list item are three buttons: a help icon, a blue 'SAVE' button, and a more options icon. Below the list, the word 'Contexts' is followed by a question mark icon. Underneath this, there is a list box containing the entity 'Diseaseintensity-followup' with a delete icon. At the bottom of the list box, there is a button labeled 'Add input context'.

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.

The screenshot shows the 'Contexts' section for the 'Disease intensity - duration' intent. At the top, there is a list item with a blue dot next to 'Disease intensity - duration'. To the right of this list item are three buttons: a help icon, a blue 'SAVE' button, and a more options icon. Below the list, the word 'Contexts' is followed by a question mark icon. Underneath this, there are two list boxes. The top list box contains the entity 'Diseaseintensity-followup' with a delete icon. The bottom list box contains two entries: '5 Diseaseintensity-followup' and '2 Diseaseintensity-duration-followup', each with a delete icon. At the bottom of the bottom list box, there is a button labeled 'Add output context'.

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.

Training phrases ?

” Add user expression

” 2 months

PARAMETER NAME	ENTITY	RESOLVED VALUE
duration	@sys.duration	2 months

” 1 week

- Provide the response as below and save the intent.

Responses ?

DEFAULT +

Text Response

- 1 Do you have rash? Choose 'Yes','No'
- 2 Enter a text response variant

ADD RESPONSES

- 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.

• **Disease intensity - rash** ! SAVE

Contexts ?

Diseaseintensity-followup (X) Diseaseintensity-duration-followup (X) Add input context

5 Diseaseintensity-followup (X) 5 diseaseintensity-duration (X) 2 Diseaseintensity-rash-followup (X) X

Add output context

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

Responses X SAVE

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

Yes	Yes, yeah
No	No, Nah
Click here to edit entry	

[+ Add a row](#)

o. Add training phrases and associate it with entity response

Training phrases ?		Search training phrases 🔍	
" " Add user expression			
" " yes			
PARAMETER NAME	ENTITY	RESOLVED VALUE	
Responses	@Responses	yes	×

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

Responses ?

DEFAULT +

Text Response	
1	Did you hurt or injure. Choose 'Yes';'No'
2	Enter a text response variant
ADD RESPONSES	

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

• Disease intensity -injure medicine

Action and parameters

Diseaseintensity Diseaseintensity-custom Diseaseintensity-duration-custom Diseaseintensity-rash-custom

REQUIRED	PARAMETER NAME	ENTITY	VALUE	IS LIST
<input type="checkbox"/>	Responses	@Responses	\$Responses	<input type="checkbox"/>
<input type="checkbox"/>	Enter name	Enter entity	Enter value	<input type="checkbox"/>

+ New parameter

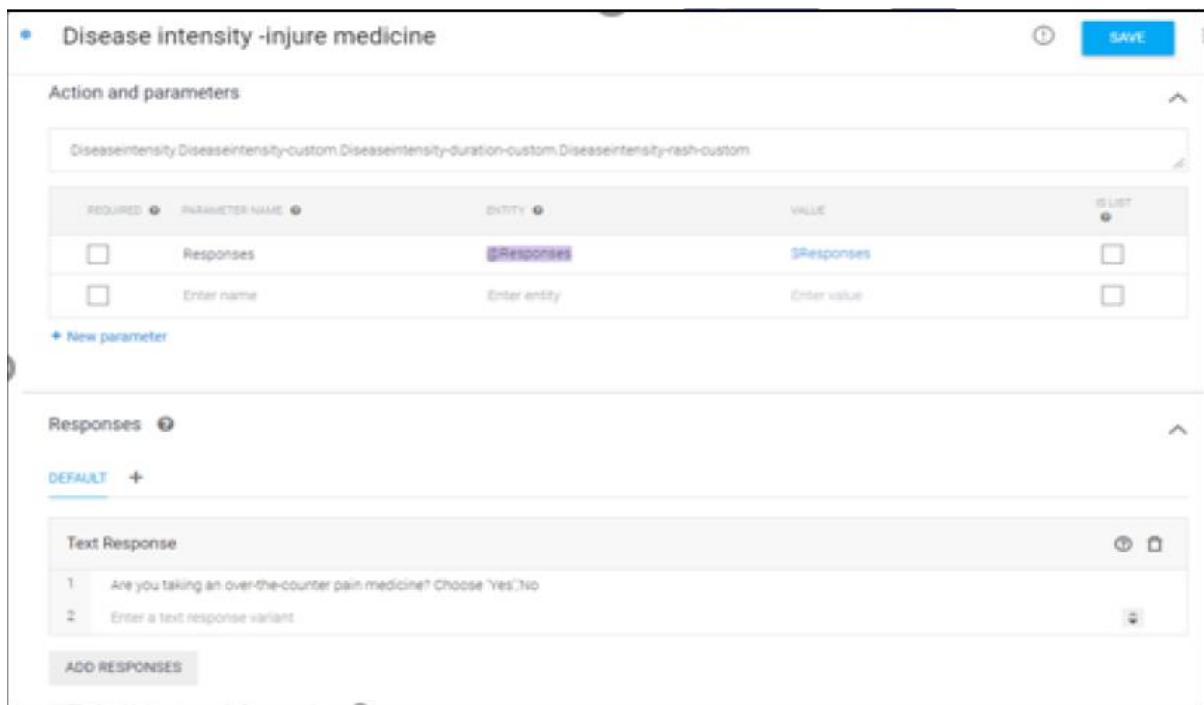
Responses

DEFAULT +

Text Response

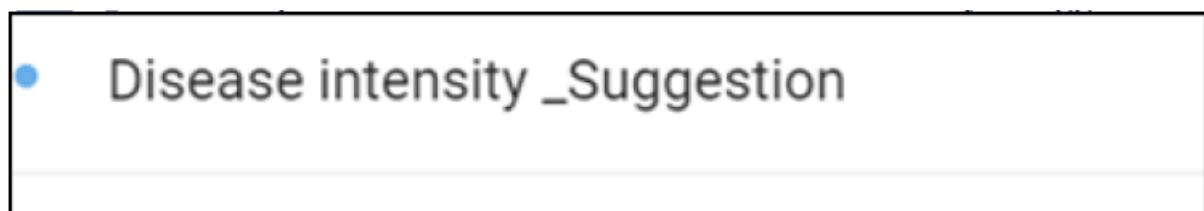
1 Are you taking an over-the-counter pain medicine? Choose 'Yes'/'No'
2 Enter a text response variant

ADD RESPONSES



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

• Disease intensity _Suggestion



s. Add the training phrase

Training phrases ⑦

Search training

99 Add user expression

99 yeah

PARAMETER NAME	ENTITY	RESOLVED VALUE
Responses	@Responses	yeah



t. Add the parameter 'intensity' with the following values.

REQUIRED	PARAMETER NAME	ENTITY	VALUE	IS LIST
<input type="checkbox"/>	Responses	@Responses	\$Responses	<input type="checkbox"/>
<input type="checkbox"/>	Intensity	@Disease_Inten:	#Diseaseintensit y- followup.disease _intensity	<input type="checkbox"/>

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

REQUIRED	PARAMETER NAME	ENTITY	VALUE	IS LIST
<input type="checkbox"/>	Responses	@Responses	\$Responses	<input type="checkbox"/>
<input type="checkbox"/>	Intensity	@Disease_Inten:	#Diseaseintensit y- followup.disease _intensity	<input type="checkbox"/>
<input type="checkbox"/>	Duration	@sys.duration	#Diseaseintensit y-duration- followup.Duration	<input type="checkbox"/>
<input type="checkbox"/>	Rash	@Responses	#diseaintensity- rash- followup.Response	<input type="checkbox"/>
<input type="checkbox"/>	Injure	@Responses	#Diseaseintensit y- injureremedicine- followup.Response	<input type="checkbox"/>
<input type="checkbox"/>	Enter name	Enter entity	Enter value	<input type="checkbox"/>

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

Responses 

DEFAULT 

Text Response

1 Summary: Here is what you said
 Main Problem or Symptom:
 1. Intensity is #Diseaseintensity-followup.disease_intensity
 2. Duration of illness #Diseaseintensity-duration-followup.Duration
 3. Rash: #diseaintensity-rash-followup.Response
 4. Injury : #Diseaseintensity-injureremedicine-followup.Response
 Suggestion:
 Suggested Care :Doctor within 3-7 days

We would like to assist in scheduling the appointment, please let us know if you want to book appointment by choosing option "Book Appointment Yes" or Book Appointment No.

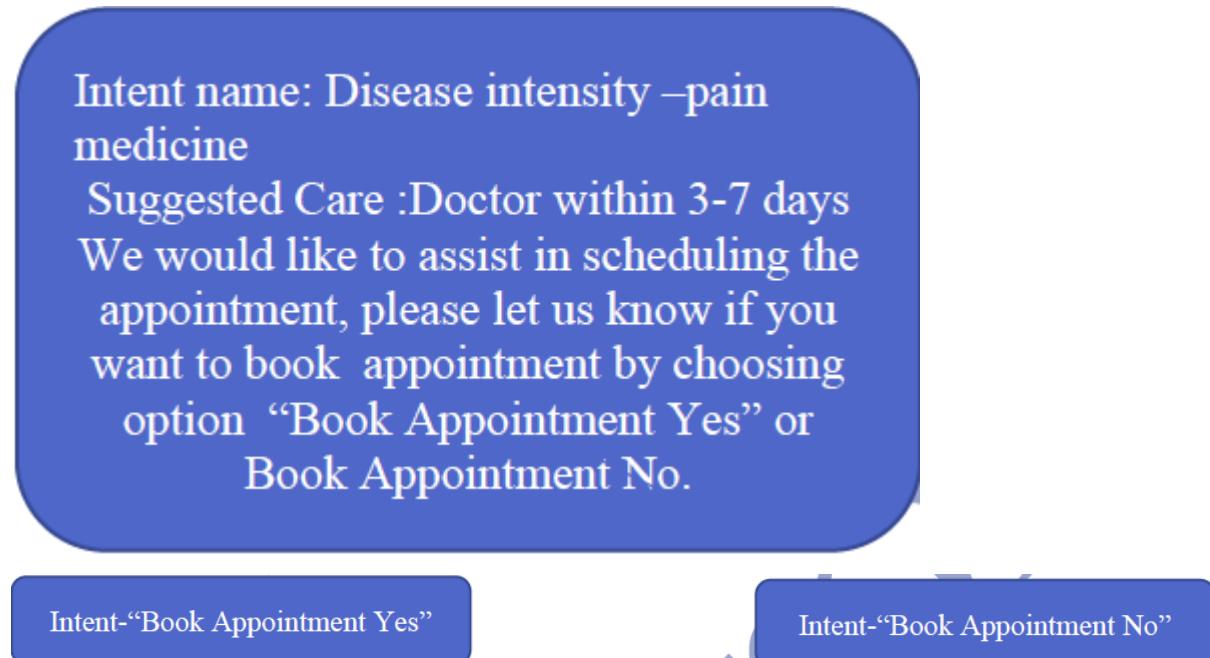
2 Enter a text response variant

ADD RESPONSES

Set this intent as end of conversation 

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.

A screenshot of the Dialogflow interface showing the "Training phrases" list for the "Book Appointment Yes" intent. The list contains five entries: "Add user expression", "Please book", "Book Appointment- Yes", "Book Appointment Yes", and "Yes". Each entry has a small "Delete" icon to its left. The top right corner of the interface shows a search bar and navigation icons.

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.

The screenshot shows the 'User Details' intent configuration. At the top, there is a blue circular icon with a white dot and a 'SAVE' button. Below the title, the word 'Contexts' is followed by a question mark icon. A button labeled 'Add input context' is present. In the main area, there is a list item '10 userdetails-followup' with a delete icon (an 'X'). Below it, a button labeled 'Add output context' is visible. The entire interface is enclosed in a light gray border.

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

The screenshot shows the 'Book Appointment Yes' intent configuration. At the top, there is a blue circular icon with a white exclamation mark and a 'SAVE' button. Below the title, the word 'Contexts' is followed by a question mark icon. A list item 'userDetails-Followup' is shown with a delete icon. Below it, another list item '2 BookAppointmentYes-followup' is shown with a delete icon. A button labeled 'Add input context' is located above the second list item. The entire interface is enclosed in a light gray border.

5. Add the following response:

The screenshot shows the 'Responses' section for the 'Book Appointment Yes' intent. At the top, there is a blue circular icon with a question mark and a 'Responses' label. Below it, a 'DEFAULT' label with a plus sign is followed by a light gray '+' button. Under the 'Text Response' heading, there is a numbered list: '1 Please confirm the below details before booking the appointment. Name: #userDetails-Followup.Name Email: #userDetails-Followup.Email Ph : #userDetails-Followup.Phonenumber'. Below the list, a note says 'Please confirm by choosing "Yes" or "No"' with a question mark and a trash can icon. The entire interface is enclosed in a light gray border.

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

The screenshot shows the 'Book Appointment No.' intent configuration. At the top, there is a blue circular icon with a white exclamation mark and a 'SAVE' button. Below the title, the word 'Contexts' is followed by a question mark icon. The entire interface is enclosed in a light gray border.

7. Add the training phrase and click on save.

The screenshot shows a list of training phrases. At the top, there is a search bar labeled "Search training phrases" with a magnifying glass icon. Below the search bar, the list contains three items, each preceded by a small blue circular icon with a white question mark:

- Add user expression
- Appointment No.
- Book Appointment No.

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

The screenshot shows the "Contexts" section. It displays two contexts listed vertically:

- userDetails-Followup (input context)
- userDetails-Followup (output context)

9. Add the following response:

The screenshot shows the "Responses" section. It has a "DEFAULT" tab selected. Under the "Text Response" heading, there are two variants:

- Thank you for choosing our service. Please reach us for any suggestion or to book an appointment.
- Enter a text response variant

At the bottom left, there is a button labeled "ADD RESPONSES".

4. Integrations

Integrations:

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

The screenshot shows the Dialogflow interface with the 'Integrations' tab selected in the sidebar. The main area displays a grid of integration settings for various platforms. Each platform has a switch icon next to its name, indicating whether it is enabled or disabled. The platforms listed are: Web Demo, Facebook Messenger, Dialogflow Phone Gateway (BETA), Slack, Viber, Twitter, Twilio IP, Twilio (Text messaging), Skype, Telegram, Kik, and LINE.

INTEGRATION SETTINGS			
Web Demo	Facebook Messenger	Dialogflow Phone Gateway <small>BETA</small>	Slack
Viber	Twitter	Twilio IP	Twilio (Text messaging)
Skype	Telegram	Kik	LINE

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

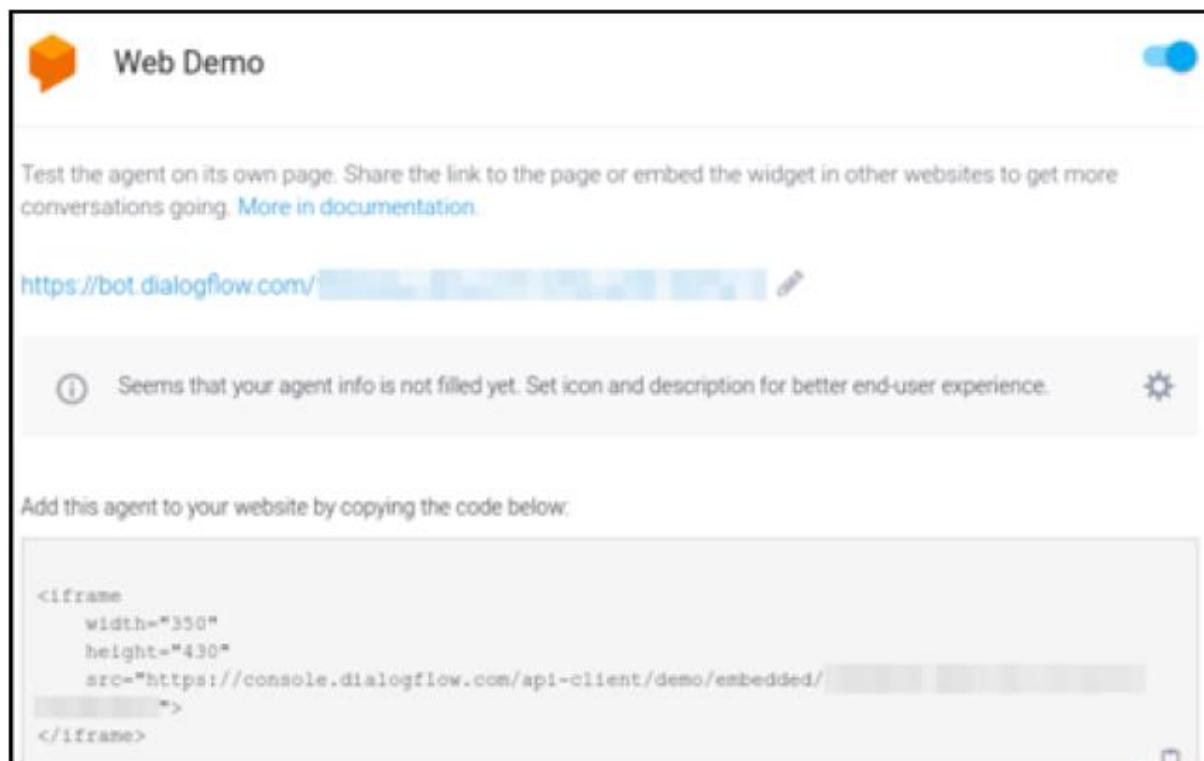
To create a web demo for our current agent, click on the **Integrations** option in the left menu of Dialogflow and then click the switch on the **Web Demo** tile.

This screenshot shows the 'One-click integrations' section within the Dialogflow Integrations interface. It features a 2x4 grid of tiles, each representing a different integration. The tiles are: Google Assistant, Web Demo, Facebook Messenger, Slack; and Viber, Twitter, Twilio IP, Twilio (Text messaging). The 'Web Demo' tile is highlighted with a cursor icon pointing to its switch, which is currently off. The other tiles are also shown with their respective icons and names.

Google Assistant	Web Demo	Facebook Messenger	Slack
Viber	Twitter	Twilio IP	Twilio (Text messaging)

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



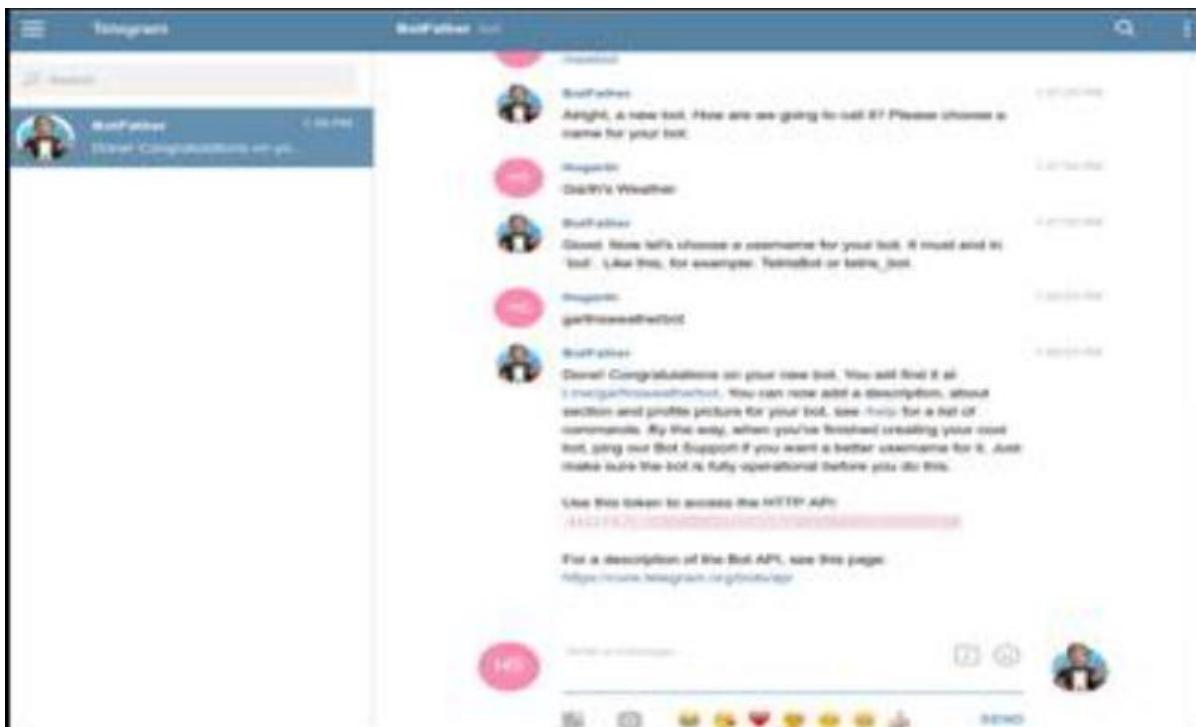
4.2 Telegram:

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

Creating a Bot in Telegram

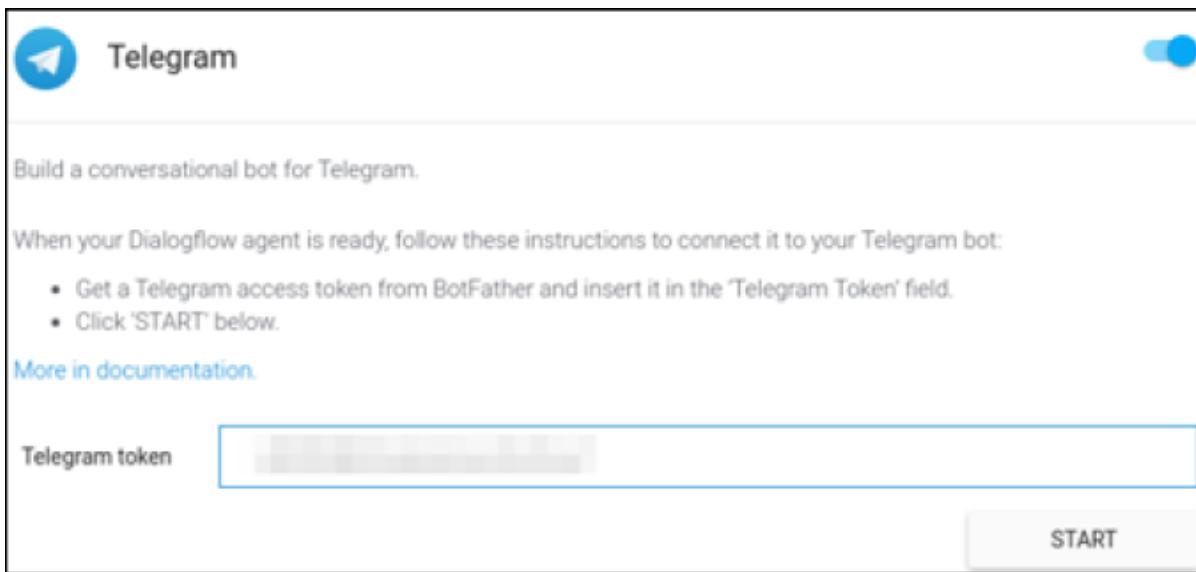
1. Log-in to Telegram using the link <https://telegram.me/botfather>
2. Click the **Start** button in the web interface or type /start in Telegram
3. Type **/newbot** and provide a name
4. Provide a username for the bot, ending in "bot" (e.g., garthsweatherbot)

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.

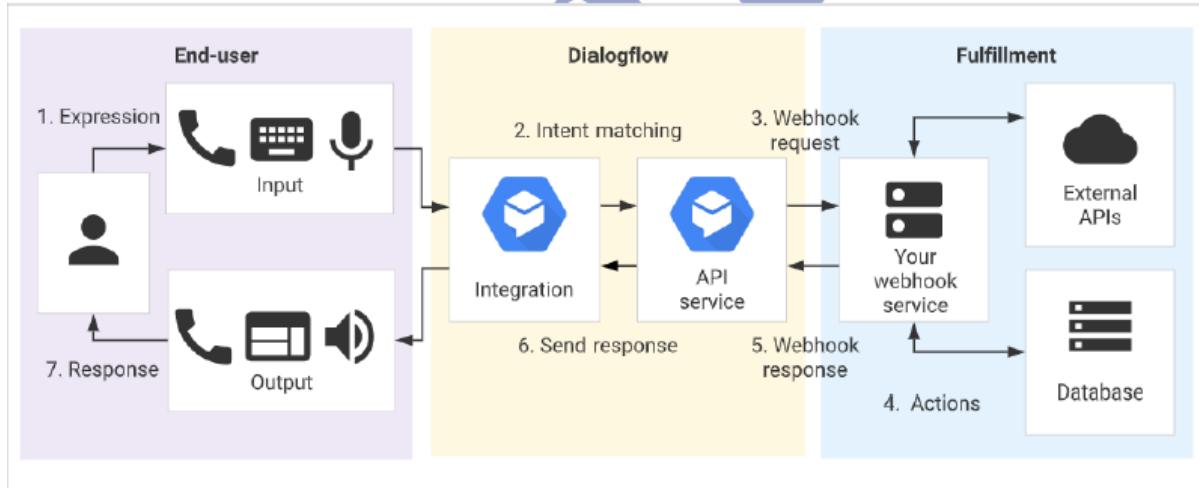


Image Courtesy: <https://cloud.google.com/dialogflow/docs/basics>

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

Training phrases			Search training phrases
Add user expression			X
please suggest a Cardiologist			X
PARAMETER NAME	ENTITY	RESOLVED VALUE	X
specialty	@specialty	Cardiologists	X

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

e. Enable fulfillment



Enable webhook call for this intent

5.1.3 Fulfilment code:

- Go to Fulfillment and enable the inline editor

Inline Editor (Powered by Cloud Functions for Firebase)

Build and manage fulfillment directly in Dialogflow via Cloud Functions for Firebase. [Docs](#)

ENABLED

- Add the ‘axios’ dependency under package.json.

“axios”:”0.18.0”

Inline Editor (Powered by Cloud Functions for Firebase)

Build and manage fulfillment directly in Dialogflow via Cloud Functions for Firebase. [Docs](#)

index.js	package.json
----------	--------------

```
index.js  package.json
  9  "node": "8"
10 },
11 "scripts": {
12   "start": "firebase serve --only functions:dialogflowFirebaseFulfillment",
13   "deploy": "firebase deploy --only functions:dialogflowFirebaseFulfillment"
14 },
15 "dependencies": {
16   "actions-on-google": "^2.2.0",
17   "firebase-admin": "^5.13.1",
18   "firebase-functions": "^2.0.2",
19   "dialogflow": "^0.6.0",
20   "dialogflow-fulfillment": "^0.5.0",
21   "axios": "0.18.0"
22 }
23 }
```

Note: This package is used to call external API.

- Goto Index.js, and add the constant axios

```
const axios=require('axios');
```

index.js	package.json
----------	--------------

```
index.js  package.json
  1 // See https://github.com/dialogflow/dialogflow-fulfillment-nodejs
 2 // for Dialogflow fulfillment library docs, samples, and to report issues
 3 'use strict';
 4 const axios= require('axios');// this variable would be used to call to the API
 5 const functions = require('firebase-functions');
 6
 7 module.exports = functions.https.onRequest((request, response) => {
 8   const query = request.query;
 9   const params = query.params;
10   const intent = query.intent;
11   const parameters = query.parameters;
12   const fulfillmentText = query.fulfillmentText;
13   const fulfillmentMessages = query.fulfillmentMessages;
14   const payload = query.payload;
15   const source = query.source;
16   const user = query.user;
17   const session = query.session;
18   const sessionId = query.sessionId;
19   const event = query.event;
20   const eventTime = query.eventTime;
21   const eventSource = query.eventSource;
22   const eventVersion = query.eventVersion;
23   const eventTimestamp = query.eventTimestamp;
24   const eventTimezone = query.eventTimezone;
25   const eventTimezoneOffset = query.eventTimezoneOffset;
26   const eventTimezoneOffsetInMinutes = query.eventTimezoneOffsetInMinutes;
27   const eventTimezoneOffsetInHours = query.eventTimezoneOffsetInHours;
28   const eventTimezoneOffsetInDays = query.eventTimezoneOffsetInDays;
29   const eventTimezoneOffsetInWeeks = query.eventTimezoneOffsetInWeeks;
2
30   // Your fulfillment logic here
31   // ...
32
33   return response.status(200).send(`Hello, ${intent.name}!`);
34 })
```

- 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=371751d40c586e02eba9de992e678439')//add the key generated in first step
    //making a call to axios library and connects to the api and get the response
    //If we received the result then execute this function
    .then((result) =>{
      .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:

Training phrases			Search training phrases	▲
Add user expression				×
»	sunithaposa@gmail.com			
PARAMETER NAME	ENTITY	RESOLVED VALUE		
email	@sys.email	sunithaposa@gmail.com	x	

c. Entities:

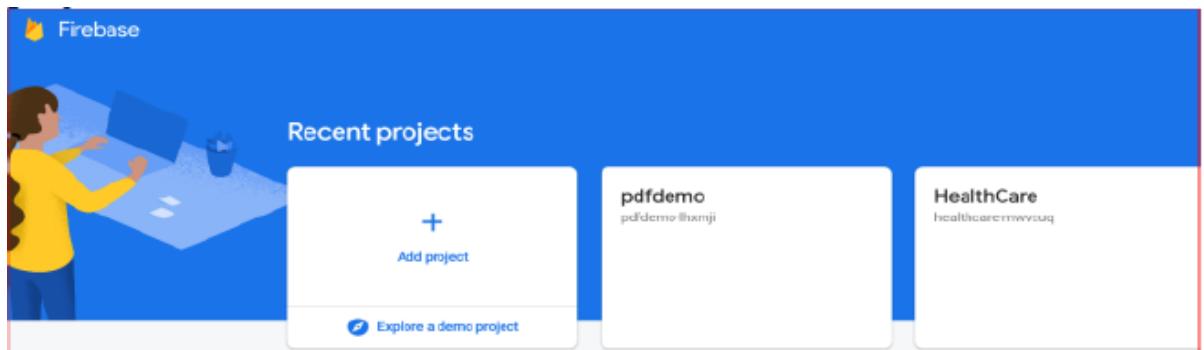
Action and parameters					
MakeAppointment.MakeAppointment-custom					
REQUIRED	PARAMETER NAME	ENTITY	VALUE	IS LIST	PROMPTS
<input checked="" type="checkbox"/>	email	@sys.email	\$email	<input type="checkbox"/>	Define prompts s...
<input type="checkbox"/>	Enter name	Enter entity	Enter value	<input type="checkbox"/>	-

+ New parameter

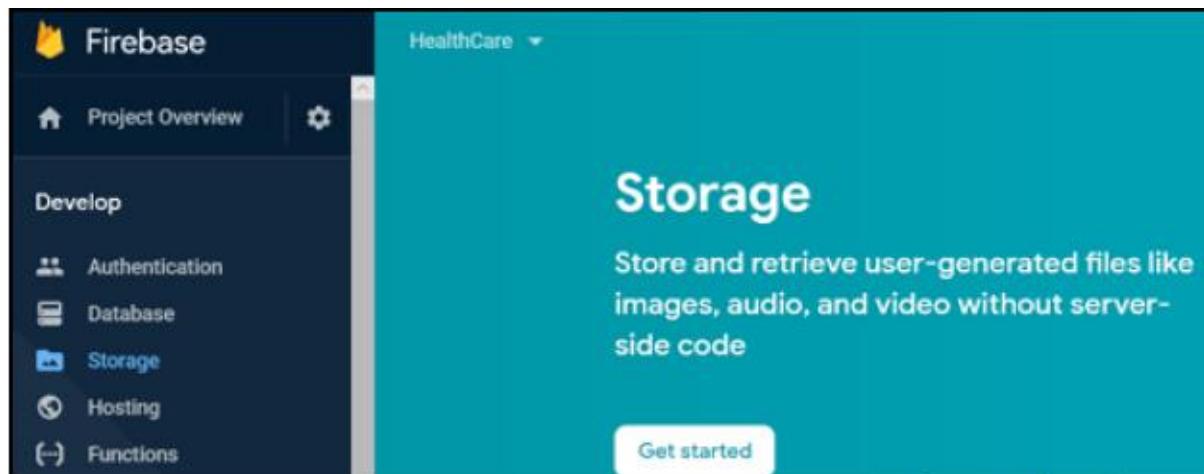
d. Enable fulfillment for this intent.

5.2.2 Firebase storage creation

- a. Go to Firebase console <https://console.firebaseio.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**

Set up Cloud Storage

1 Secure rules for Cloud Storage 2 Set Cloud Storage location

By default, your rules allow all reads and writes from authenticated users.

After you define your data structure, you will need to write rules to secure your data. [Learn more](#)

```
service firebase.storage {  
  match /b/{bucket}/o {  
    match /{allPaths=**} {  
      allow read, write: if request.auth != null;  
    }  
  }  
}
```

Cancel Next

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

Set up Cloud Storage

1 Secure rules for Cloud Storage 2 Set Cloud Storage location

Your location setting is where your default Cloud Storage bucket and its data will be stored.

Cloud Storage location

nam5 (us-central)

Blaze Plan customers can choose other locations for additional buckets

Cancel Done

f. The bucket gets created.

Firebase

Project Overview Settings

HealthCare

Storage

Develop

Authentication Database Storage Functions

Files Rules Usage

gs://healthcare-mwvcuq.appspot.com

Name

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: 'sunithaandraruvi@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(emai1l, filename) {
    const email = emai1l;
    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

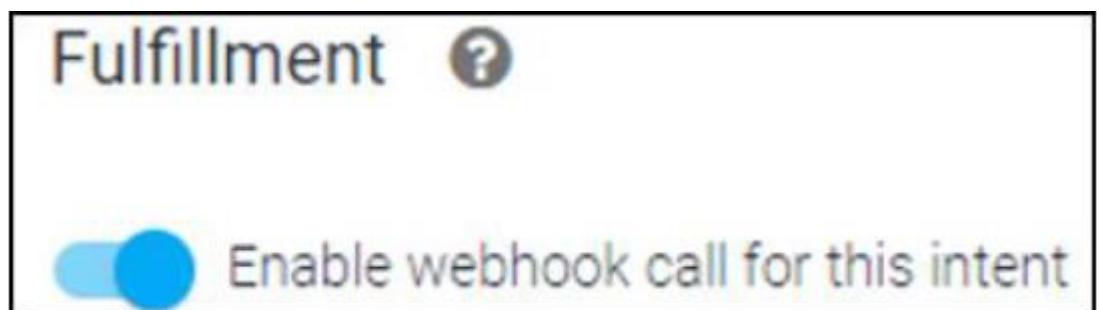
weather forecast in Bangalore

weather forecast in Delhi on Thursday,

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

weather forecast in Delhi Thursday		
PARAMETER NAME	ENTITY	RESOLVED VALUE
geo-city	@sys.geo-city	Delhi
date	@sys.date	Thursday

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
#Flask app for our web app

app=Flask(__name__)

# app route decorator. when webhook is called, the decorator would call the functions
# which are 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 weather api, construct the response
def makeResponse(req):
    result=req.get("queryResult")
```

```

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

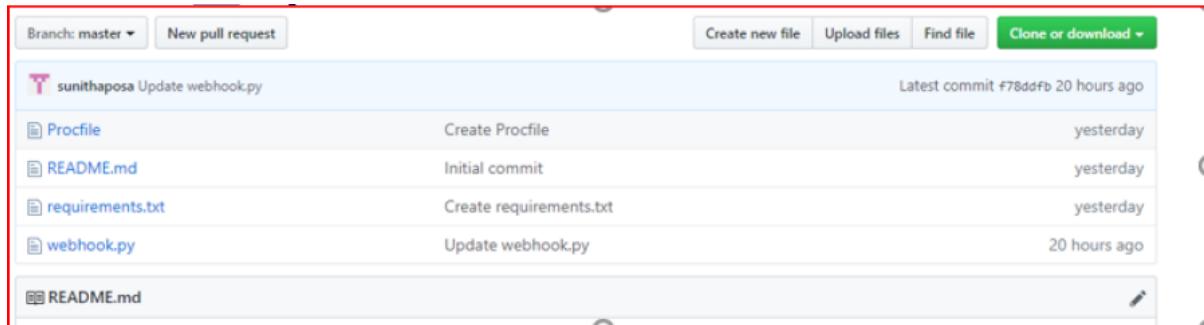
r=requests.get('http://api.openweathermap.org/data/2.5/forecast?q=hyderabad,in&appid=db91df44baf43361cbf73026ce5156cb')
json_object=r.json()
weather=json_object['list']

#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 requirements.txt.



The screenshot shows a GitHub repository interface. At the top, there are buttons for 'Branch: master', 'New pull request', 'Create new file', 'Upload files', 'Find file', and 'Clone or download'. Below this is a list of files and their details:

- sunithaposa** Update webhook.py - Latest commit f78ddfb 20 hours ago
- Procfile - Create Procfile - yesterday
- README.md - Initial commit - yesterday
- requirements.txt - Create requirements.txt - yesterday
- webhook.py - Update webhook.py - 20 hours ago
- README.md

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
```

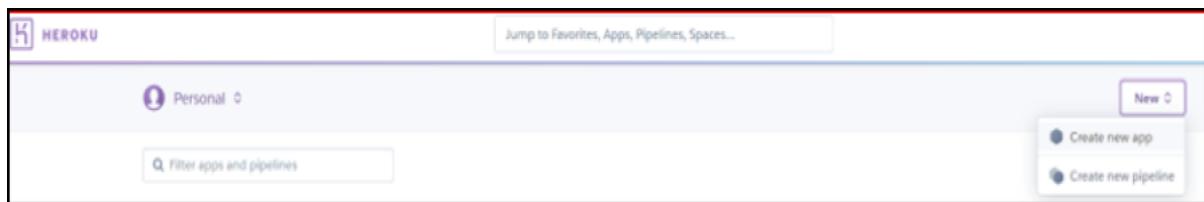
- d. Create the file Procfile and paste the following data and then commit the changes:

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

- e. Create an account in Heroku and login to it.

<https://signup.heroku.com/>

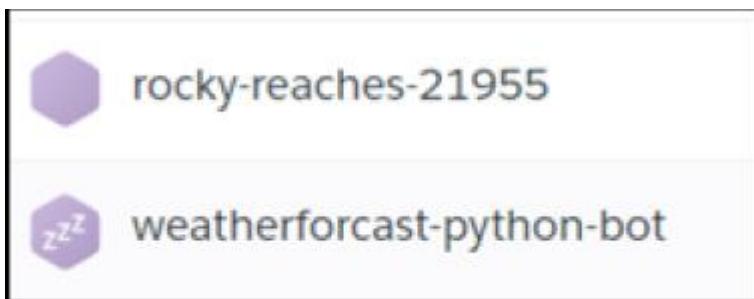
- f. Create a new app



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

A screenshot of the "Create New App" form. The title is "Create New App". It has fields for "App name" containing "weatherforcast-python-bot1" (with a green checkmark icon), "Choose a region" with "United States" selected (indicated by a purple border), and a "Add to pipeline..." button. At the bottom is a large purple "Create app" button.

h. Select the newly created app.



i. Select Deploy

A screenshot of the Heroku app settings page for 'weatherforecast-python-bot'. At the top, there's a navigation bar with 'Personal' and the app name. Below it, the GitHub URL 'GitHub sunithaposa/weatherforecastpython-bot' is shown. The bottom navigation bar has tabs: Overview, Resources, Deploy (which is selected and highlighted in blue), Metrics, Activity, Access, and Settings.

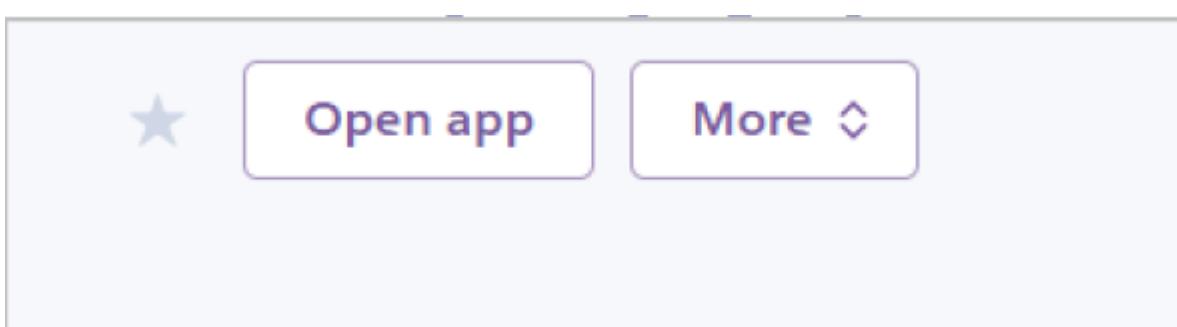
j. Under the Deployment method, select “Github” and then search for the GIT repository and then connect to it.

A screenshot of the 'Deployment method' section in the Heroku app settings. It shows three options: 'Heroku Git' (Use Heroku CLI), 'GitHub' (Connect to GitHub, which is selected and highlighted in blue), and 'Container Registry' (Use Heroku CLI). Below this, there's a 'Connect to GitHub' section with a note about enabling code shifts and deploys. A search bar shows 'sunithaposa/weather forecast' with a 'Search' button. Below the search bar, it says 'Missing a GitHub organization? Ensure Heroku Dashboard has team access.' and lists two repository results: 'sunithaposa/weatherforecastpython-bot' and 'sunithaposa/weatherforecast-python-bot', each with a 'Connect' button.

k. Click deploy

A screenshot of the 'Manual deploy' section in the Heroku app settings. It shows a 'Deploy a GitHub branch' section with a note: 'This will deploy the current state of the branch you specify below.' and a 'Learn more' link. Below it is a 'Choose a branch to deploy' dropdown set to 'master' and a 'Deploy Branch' button.

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.

Webhook	ENABLED <input checked="" type="checkbox"/>
Your web service will receive a POST request from Dialogflow in the form of the response to a user query matched by intents with webhook enabled. Be sure that your web service meets all the webhook requirements specific to the API version enabled in this agent.	
URL*	<input type="text" value="https://weatherforcast-python-bot.herokuapp.com/webhook"/>

Thankyou!