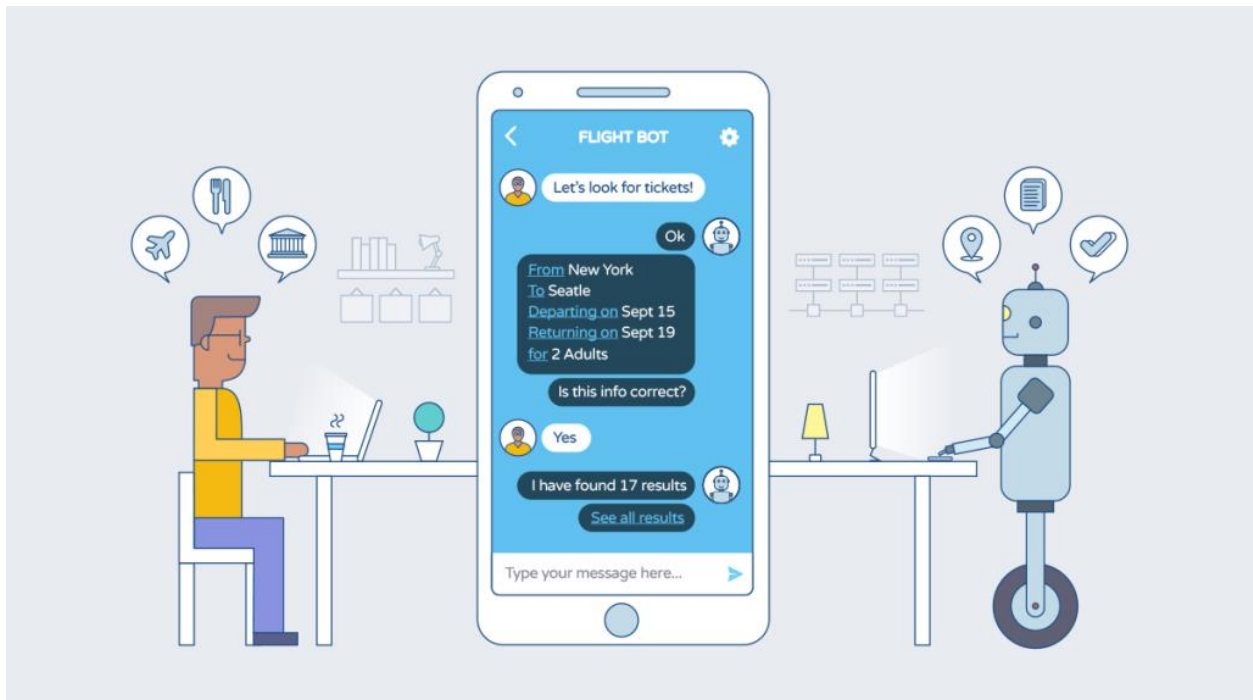


Chatbot Development

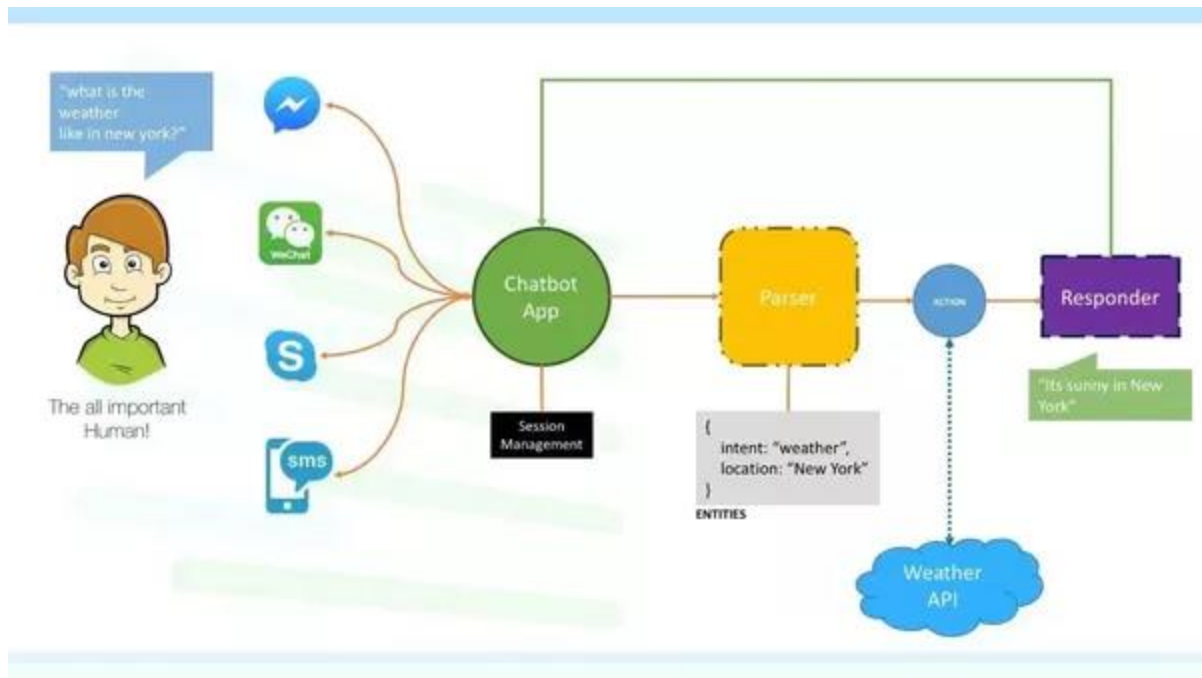


According to Google Chatbot means “a computer program designed to simulate conversation with human users, especially over the Internet.” But I told you guys to read about “Turing Test” in few posts ago. If you want to know the exact meaning of Chatbot and how it works then “Turing test” is good to start with!! A chatbot also known as interactive agent is a computer program which conducts a conversation or answers your question via textual or auditory methods. Such programs are often designed to convincingly simulate how a human would behave as a conversational partner, thereby passing the **Turing test**. Some chatbots use sophisticated natural language processing systems, but many simpler systems scan for keywords within the input, then pull a reply with the most matching keywords, or

the most similar wording pattern, from a database. I will try to show you both here in this few pages of PDF.



The term "ChatterBot" was originally coined by Michael Mauldin (creator of the first Verbot, Julia) in 1994. Then once it got limelight so this global world is so fast that GA and Alexa originated. You are fast too, aren't you? You will go and develop a powerful chat bot after this PDF. How chatbot works? A simpler representation from the internet image is as follows:

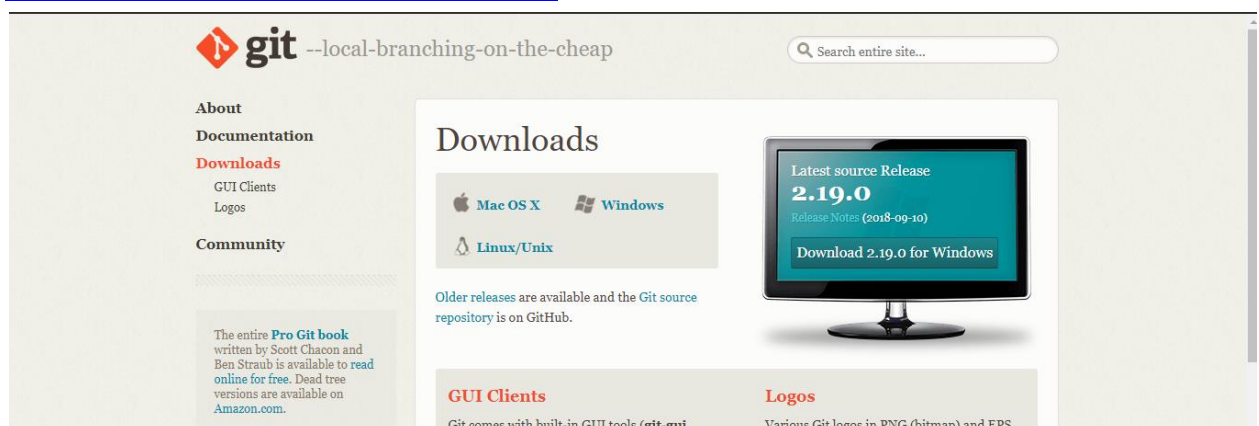


I don't like to explain theory too much. I will be explaining while doing the real deal. Before starting, I would recommend you to get ready with the following things (These are must). Just follow the instructions:-

Tools and Techniques

1. Download "Git Bash" from this site

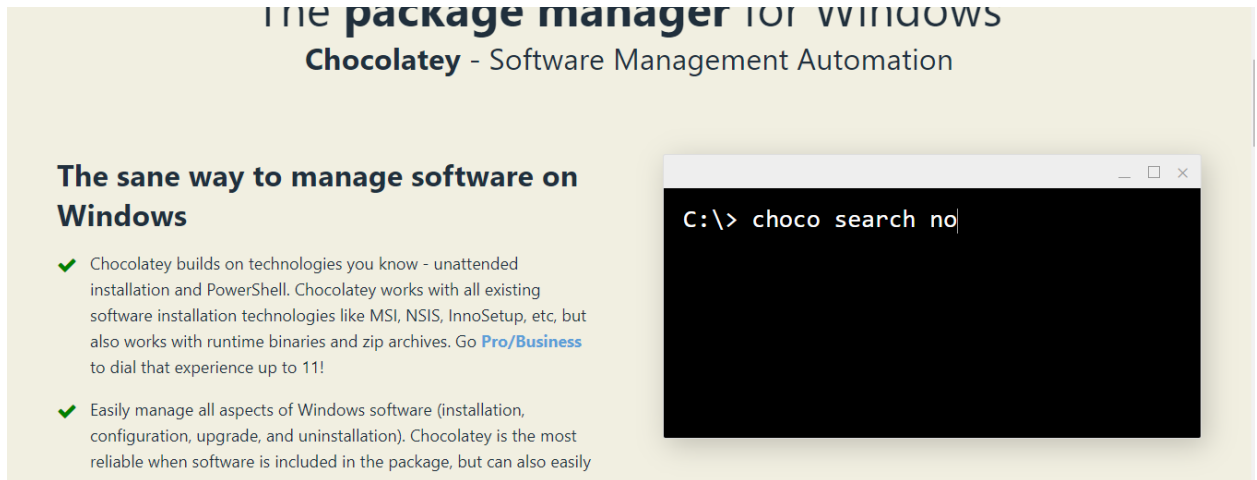
<https://git-scm.com/downloads>



Download the .exe file according to your OS. You can read about Git on the same website.

2. Install Chocolatey package manager for windows. I have seen people suffered with “ngrok”. So you must download this package manager.

Install link- <https://chocolatey.org/>



3. Download “ngrok”. Ngrok is a reverse proxy that creates a secure tunnel between from a public endpoint to a locally running web service. Ngrok captures and analyzes all traffic over the tunnel for later inspection and replay.

Download process:-

- After installing Chocolatey successfully open a new command prompt. Write the following command to download, install and automatic path settling for ngrok-

C:\> choco install ngrok.portable

That’s it. It will globally install your ngrok.

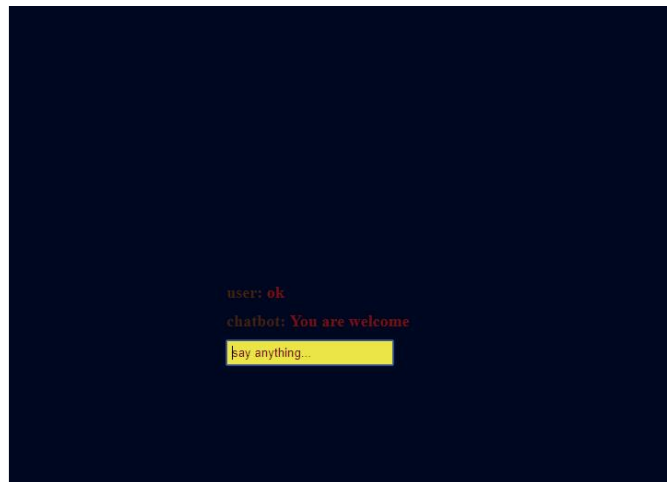
4. A Gmail account
5. Go to Dialogflow (formerly API.AI). Just open it in your tab for now.
<https://dialogflow.com/>

6. I assume you have Anaconda installed (if not then download it from the following link)
<https://www.anaconda.com/>
7. Open Pusher website (You can create real time powerful apps with Pusher)
<https://www.pusher.com/>

I think it's too much for now. I have mentioned in my post that I will be sharing both a small manual chat bot (Without AI) using JavaScript and Speech recognition in browser and an AI based chat bot using Dialogflow, Flask and ngrok. I will start with the earlier one first. Let's start:-

Part 1

I will be sharing here a very simple JS Based application which is an auditory featured question-answer app using your manual data.



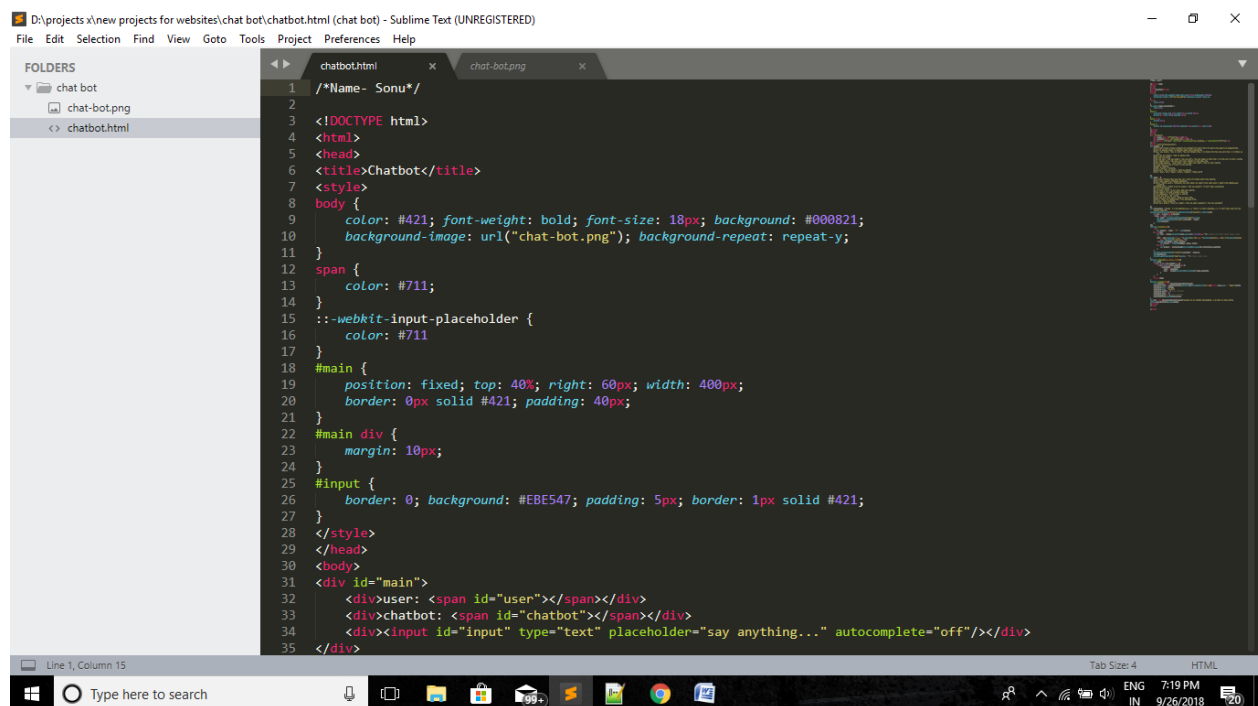
I am not at all good at designing. You can design the frontend as you like.

Steps:-

1. Create a folder "chat bot"
2. Create a HTML file "chatbot.html"
3. You can choose any pic from the internet and save it in the same folder.

Open "chatbot.html" and write the following code in it.

First we will write code for the frontend design (that's pretty weird) as follows:-



The screenshot shows a Sublime Text editor window titled "chatbot.html (chat bot) - Sublime Text (UNREGISTERED)". The editor displays the following HTML code:

```
1 /*Name- Sonu*/
2
3 <!DOCTYPE html>
4 <html>
5 <head>
6 <title>Chatbot</title>
7 <style>
8 body {
9     color: #421; font-weight: bold; font-size: 18px; background: #000821;
10    background-image: url("chat-bot.png"); background-repeat: repeat-y;
11 }
12 span {
13     color: #711;
14 }
15 ::-webkit-input-placeholder {
16     color: #711
17 }
18 #main {
19     position: fixed; top: 40%; right: 60px; width: 400px;
20     border: 0px solid #421; padding: 40px;
21 }
22 #main div {
23     margin: 10px;
24 }
25 #input {
26     border: 0; background: #EBE547; padding: 5px; border: 1px solid #421;
27 }
28 </style>
29 </head>
30 <body>
31 <div id="main">
32     <div>user: <span id="user"></span></div>
33     <div>chatbot: <span id="chatbot"></span></div>
34     <div><input id="input" type="text" placeholder="say anything..." autocomplete="off"/></div>
35 </div>
```

If you can't see the codes then don't worry, I will upload it to my drive and will share with you. Just beneath this we will take a trigger where we will keep all our questions and it's variant that what a user can ask for.

```
<script type="text/javascript">
var trigger = [
  ["hi","hey","hello","hello brother","hi brother","hi bro","hii","hi girl","hey girl","hi beautiful"],
  ["what is special about u","special about you","special"],
  ["how r you doing", "how is life", "how are things","how r u","kaisi ho","how are you","how r u","Whats up"],
  ["what are you doing", "what is going on"],
  ["how old are you"],
  ["who are you", "are you human", "are you bot", "are you human or bot","who r u","who are u","who r you"],
  ["who created you", "who made you","who created u","who made u"],
  ["your name please", "your name", "may i know your name", "what is your name"],
  ["i love you","i love u","love u","my love"],
  ["happy", "good"],
  ["bad", "bored", "tired"],
  ["help me", "tell me story", "tell me joke"],
  ["ah", "yes", "ok", "okay", "nice", "thanks", "thank you"]
];
```

We created a variable “trigger” where we will keep our questions that a user can ask. You can manually increase the questions and its variant at any size. Now after keeping all the questions, we have to prepare answer set too in the similar fashion as follows:

```
var reply = [
  ["Hi","Hey","Hello","hey dear How can i help u?","hello dear","hey dear"],
  ["24*7 user support","friendly nature"],
  ["Fine", "Pretty well", "Fantastic and what about you dear","Fine what about u mate","sab badhiya,aap btaye!"],
  ["Nothing much", "About to go to sleep", "Can you guest?", "I don't know actually"],
  ["I am 1 day old"],
  ["I am just a bot", "I am a bot. What are you?"],
  ["Sonu Kumar", "My God","webhub studio"],
  ["I am nameless", "I don't have a name"],
  ["I love you too", "Me too"],
  ["Have you ever felt bad?", "Glad to hear it"],
  ["Why?", "Why? You shouldn't!", "Try watching TV"],
  ["I will", "What about?"],
  ["Tell me a story", "Tell me a joke", "Tell me about yourself", "You are welcome"]
];
```

We took a variable reply and prepare the replies according to same as question above. Keep in the pattern. After it, we have to choose an alternative if the question doesn’t match with the prepared trigger.

```
var alternative = ["oops i m not getting you...", "sorry i m still learning...", "i don't know what are you talking about"];
```

```
var alternative = ["oops I m not getting you...", "sorry I m still learning...", "I don't know what are you talking about"];
```

Here alternative variable has 3 different answers. If the question doesn't match then anyone of the above will fired.

Now we will write our event listener which will keep track of our input every time as follows:-

```
document.querySelector("#input").addEventListener("keypress", function(e){
    var key = e.which || e.keyCode;
    if(key === 13){ //Enter button
        var input = document.getElementById("input").value;
        document.getElementById("user").innerHTML = input;
        output(input);
    }
});
```

Now we will write a function which will ignore all the special characters and it will also neglect case sensitivity.

```
function output(input){
    try{
        var product = input + "=" + eval(input);
    } catch(e){
        var text = (input.toLowerCase()).replace(/^[^w\s\d]/gi, ""); //remove all chars except words, space and
        text = text.replace(/ a /g, " ").replace(/i feel /g, "").replace(/whats/g, "what is").replace(/please /g, "").replace(/ please/g, "");
        if(compare(trigger, reply, text)){
            var product = compare(trigger, reply, text);
        } else {
            var product = alternative[Math.floor(Math.random()*alternative.length)];
        }
    }
    document.getElementById("chatbot").innerHTML = product;
    speak(product);
    document.getElementById("input").value = ""; //clear input value
}
```

The function “output” will work for the input. We have a text variable that contains an if-else conditional which match the trigger and reply. If it matches then answer will be generated else the alternative will be fired. After the generation of text, input field will be cleared.

I am using JS speech recognition script that generates speech of your text that have you prepared.


```

function speak(string){
    var utterance = new SpeechSynthesisUtterance();

    utterance.voice = speechSynthesis.getVoices().filter(function(voice){return voice.name
== "Agnes";}))[0];

    utterance.text = string;

    utterance.lang = "en-US";

    utterance.volume = 1; //0-1 interval

    utterance.rate = 1;

    utterance.pitch = 2; //0-2 interval

    speechSynthesis.speak(utterance);
}

```

We have a function speak. This function has a variable utterance which calls the *SpeechSynthesisUtterance()*

Constructor. You can change the voice (currently it is “Agnes”) and language too (currently it is “US”). Similarly pitch, volume etc are changeable as your choice.

```

function speak(string){
    var utterance = new SpeechSynthesisUtterance();
    utterance.voice = speechSynthesis.getVoices().filter(function(voice){return voice.name == "Agnes";}))[0];
    utterance.text = string;
    utterance.lang = "en-US";
    utterance.volume = 1; //0-1 interval
    utterance.rate = 1;
    utterance.pitch = 2; //0-2 interval
    speechSynthesis.speak(utterance);
}

```

For generating a welcome message just after opening the web page every time I am using window speech synthesis of JavaScript as follows:

```

var msg = new SpeechSynthesisUtterance('welcome to my chatbot development. I am here to help you');

window.speechSynthesis.speak(msg);

```

```
var msg = new SpeechSynthesisUtterance('welcome to my chatbot development. I am here to help you');  
window.speechSynthesis.speak(msg);
```

That's it. Save the file and open "chatbot.html" in your browser. Make sure you have your microphone on in the system. Download the file of above part from the following link as follows:

https://drive.google.com/open?id=1P9GmqXlB_Sa842-wZdeKssFxAsylX4xL

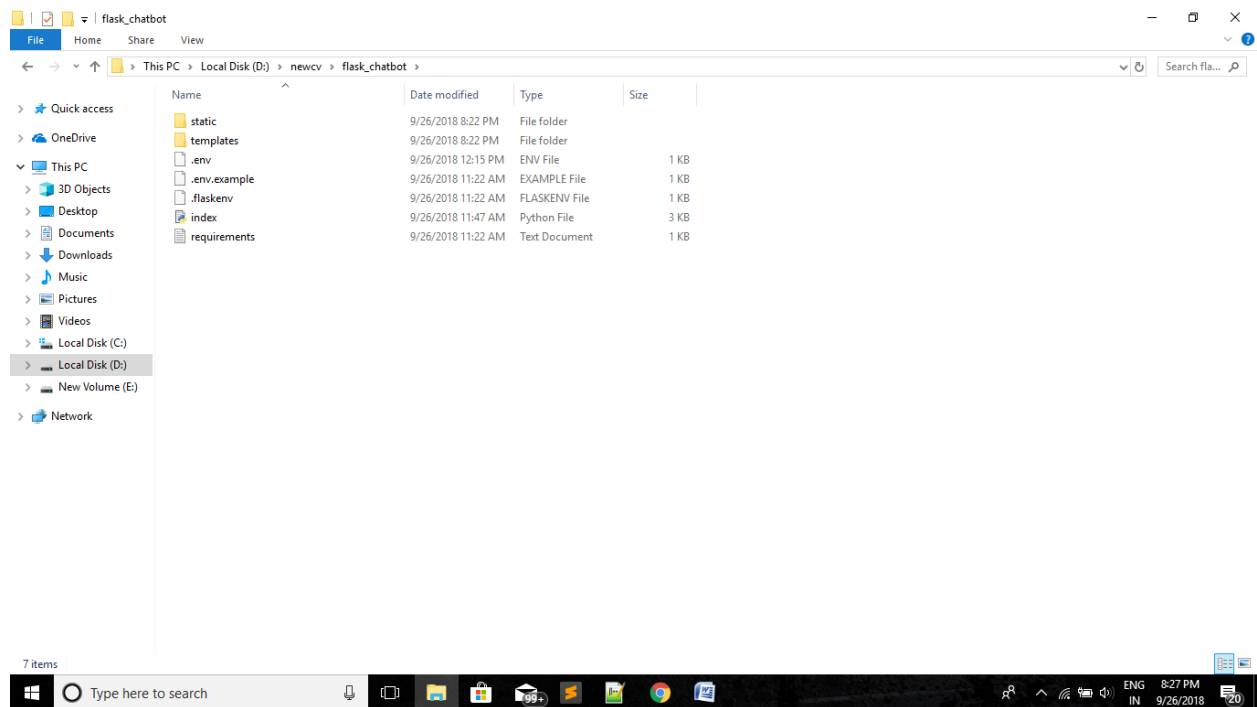
Part 2

We'll build a chatbot that will help us get details of a movie. You will need OMDb API. The OMDb API is a Restful web service to obtain movie information. When a user asks our bot for details about a movie, we'll make a request to OMDb to get the detail about the movie. OMDb requires that we have an API key before you can use their service. To get a free API access key, head to OMDb. [Sign up for a free account](#). An email will be sent to you that contain your API key. You also need to activate your account from the email sent to you.

First of all download the source code folder from this link:-

<https://drive.google.com/open?id=1dq1Q956R6eKaRdic1XQmm2siYWj1Umf4>

Download the file and save it anywhere in your system. It has following files:-



CREATING A VIRTUAL ENVIRONMENT

Virtualenv is a tool to create isolated Python environments. It creates a folder which contains all the necessary executables to use the packages that a Python project would need. Open Git bash in your folder and type the following command to create a virtual environment:-

```
$ python -m venv env
```

If you are using Windows, activate the virtualenv with the below command:

```
$ env/Scripts/activate
```

Now You will see an env folder inside “flask_chatbot”.

Install the libraries in requirements.txt:

```
pip install -r requirements.txt
```

It will install Flask==1.0.2

requests==2.18.4

dialogflow==0.4.0

python-dotenv==0.8.2

pusher==2.0.1

Now good with installation.

STARTING UP FLASK

In the index.py file, add the following code to it: Actually You don't have to write any code because I have already given the folder with codes. You just have to change the keys few times.

```
# /index.py

from flask import Flask, request, jsonify, render_template
import os
import dialogflow
import requests
import json
import pusher

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

# run Flask app
if __name__ == "__main__":
    app.run()
```

Here, we created a route - / -, which will render the index.html file in the templates folder.

Open git bash in the “flask_chatbot” by right click on the folder. It will say open with “Git bash”. Write the following command to run the flask app:

```
flask run
```

Your app should run by the default on localhost:5000

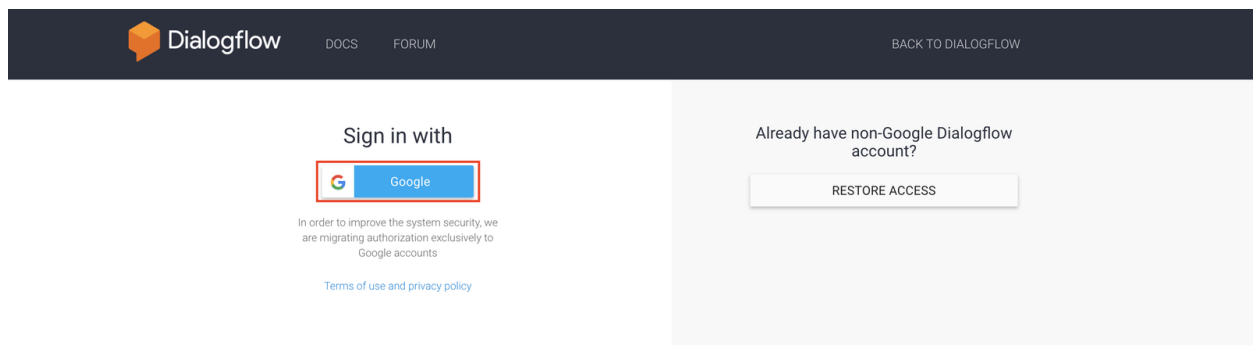
```
(env) → movie_bot flask run
* Serving Flask app "index.py" (lazy loading)
* Environment: development
* Debug mode: on
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 146-855-672
```

Now, visit <http://127.0.0.1:5000>. You'll get a blank page because there is no content in /templates/index.html yet.

SETTING UP YOUR DIALOGFLOW ACCOUNT

Dialogflow is a Google-owned developer of human–computer interaction technologies based on natural language conversations. It will make our chatbot intelligent by using machine learning to understand what our users are saying. All we need to do is train it.

Now head to Dialogflow’s website and create a free [account](#) if you don’t have one already. You’ll need a google account to be able to create an account. Once you are on the page, click on the sign in with Google button:



If you already have an account, go ahead and click on the button to log in.

DIALOGFLOW SETUP

When a user submits a message, we'll send the message to Dialogflow. Then Dialogflow will detect the intent of the message and send back a reply (fulfillment text) to us. This will be the basic flow of our bot.

For example:

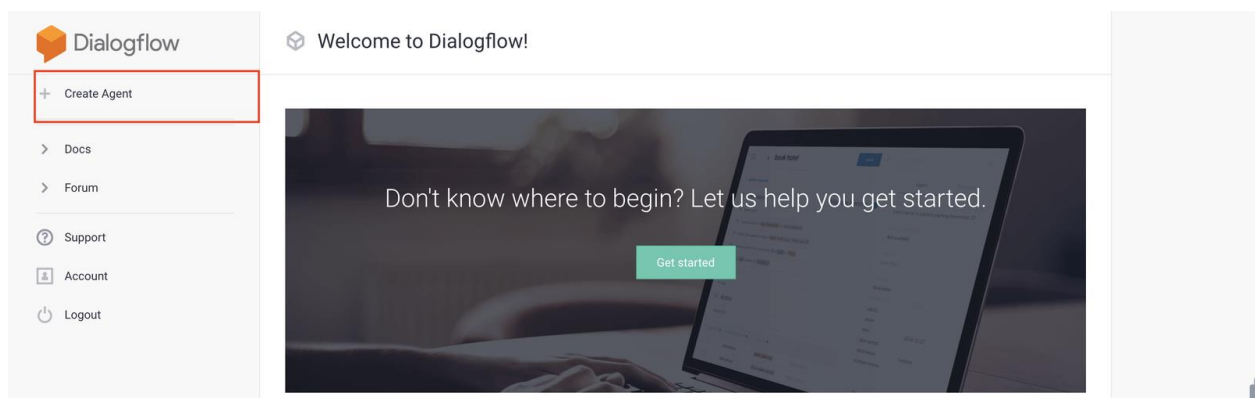
User: Hi **Bot:** hello! **User:** Who are you? **Bot:** I am just a messenger **User:** Please, show me more details about The mummy **Bot:** ...

Now, for our bot to know the intent of our user, be it greeting, asking about the bot or asking for details of a movie, we need to train our bot to understand that.

CREATING OUR AGENT

The agent is our bot itself.

- Go to your Dialogflow [dashboard](#):



- Then, click on **Create Agent** to create a new agent.

- Next, type in the bot name - Movie_Bot - and then click on the **CREATE** button to create the bot:

Movie_Bot

CREATE

DEFAULT LANGUAGE ?

DEFAULT TIME ZONE

Select language

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

Date and time requests are resolved using this timezone.

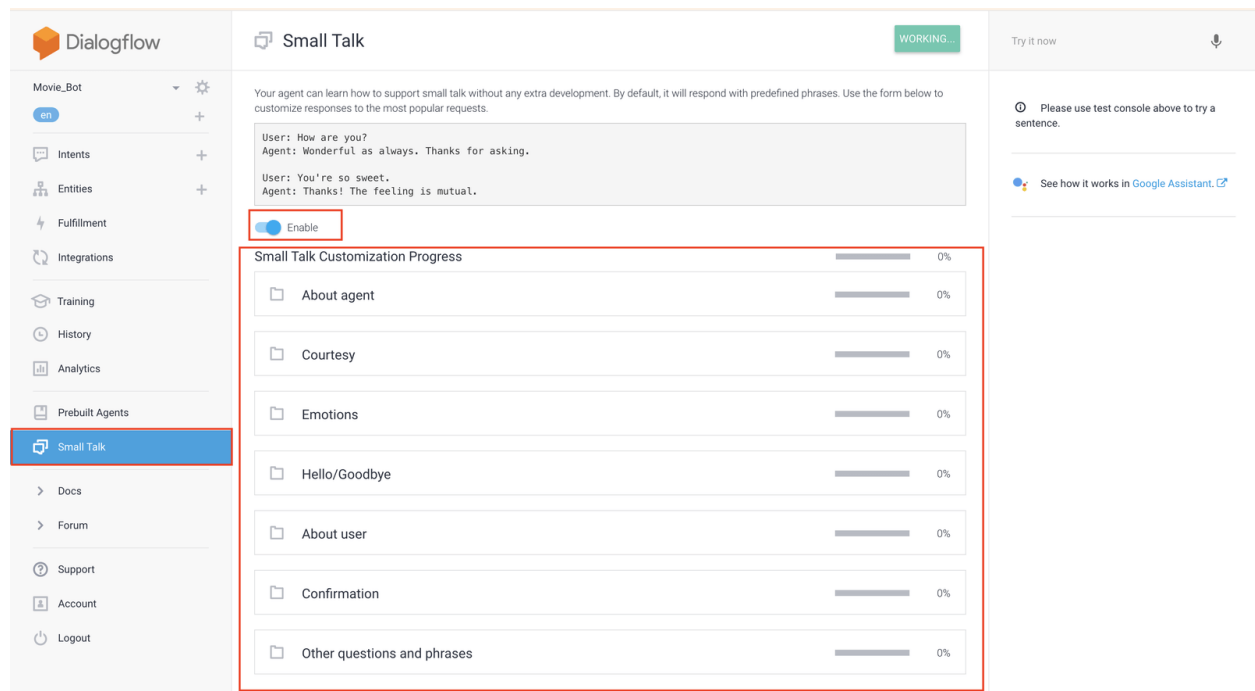
GOOGLE PROJECT

New GCP project will be automatically linked to the agent after saving

SMALL TALK

For sure, we know, when some users want to interact with our bot. They might want to exchange greetings. By default, Dialogflow has this intent set already. All we need to do is enable it. When enabled, The bot will know when the user is greeting and reply with the appropriate response.

On the sidebar of your console dashboard, click on **Small Talk**:



I will recommend you to make small talk 100% by answering all the questions in segment like about agent, courtesy, emotions, hello/goodbye etc. It will look your conversation very natural.

Now, enable Small Talk using the toggle button as indicated on the image above. Feel free to customize the responses as you like. Once you are done, click on the **SAVE** button.

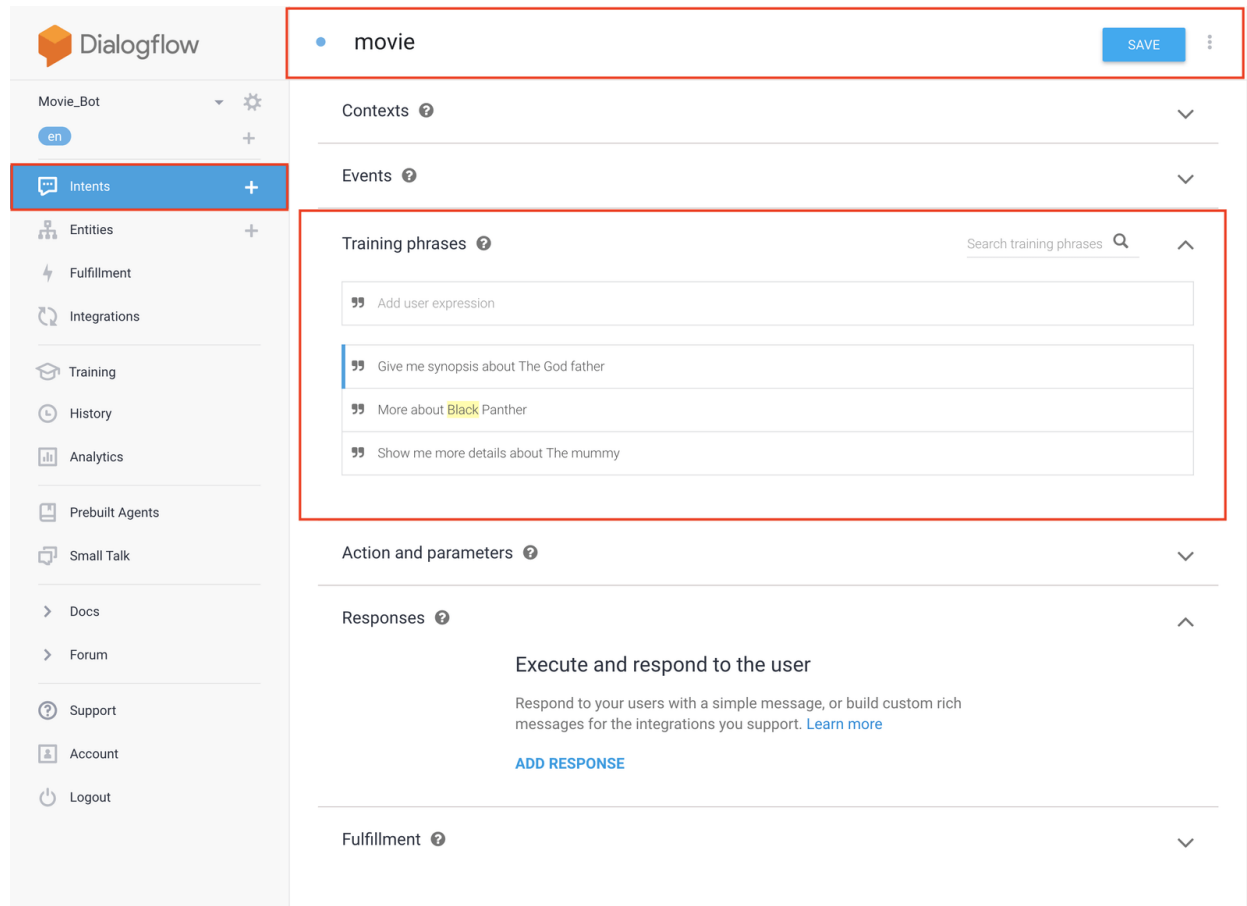
MOVIE INTENT

When users ask about a movie, at the moment our bot won't know what to respond with because it does not know about movies. So, let's train it to understand when the user is asking about a movie. This will be a **movieintent**.

Now,

- Click on **Intents** on the sidebar of your dashboard
- Then click on the **CREATE INTENT** button

- Next, type in **movie** as the intent name
- Finally, click on the **SAVE** button



Next, click on **ADD TRAINING PHRASES** under the **Training Phrases** section as indicated in the image above.

Now, type in texts that a user is likely going to use to ask about a movie as seen in the image above.

For example:

- Show details about **The Mummy**
- More about **Black Panther**
- Give me synopsis about **The Godfather**

- I want to know more about **Upgrade**

The more your input, the more intelligent your bot will be.

When you are done typing it in, click on the **SAVE** button.

CREATING OUR ENTITY

The entity is used for extracting parameter values from the user input. Let's train our bot to know which among the texts is the actual movie name. All we need is the name of the movie so we can search for the movie.

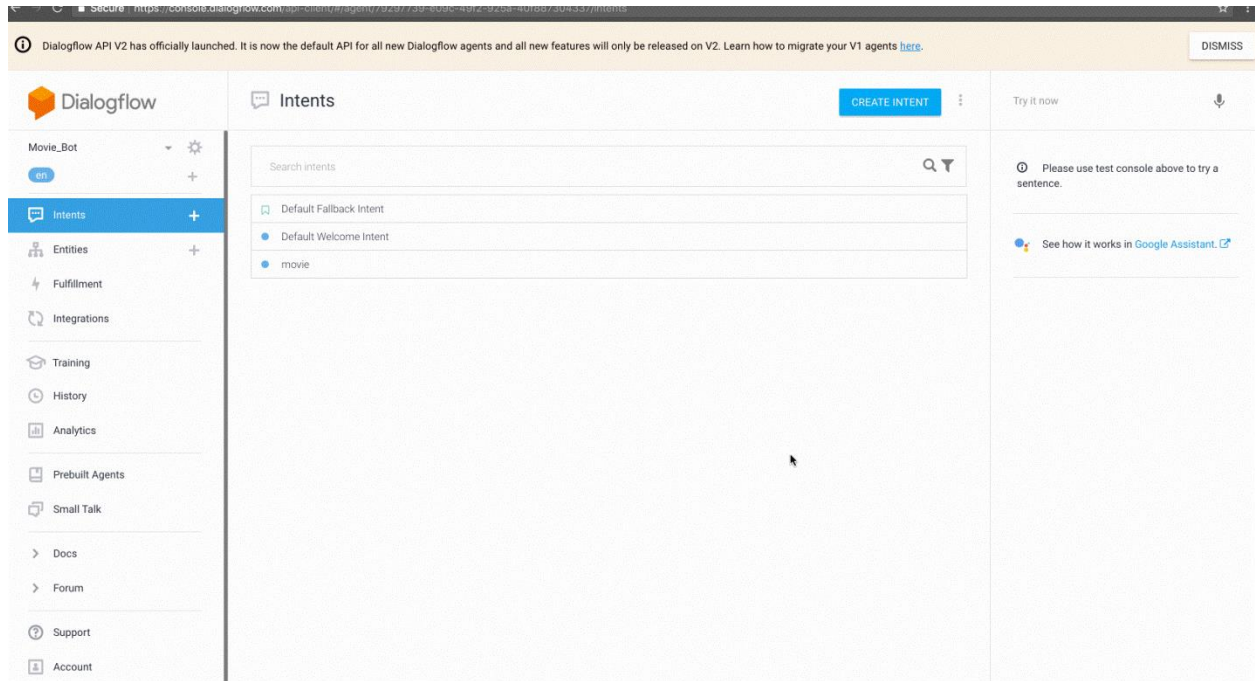
Now,

- Click on **Entities** on the sidebar.
- Next, click on the **CREATE ENTITY** button.
- Type in **movie** as the **ENTITY NAME** as seen in the image below.
- Then add one or more movie name of your choice as seen in the image below.
- Finally, click on the **SAVE** button.

The screenshot shows the Dialogflow console interface. On the left sidebar, the 'Entities' tab is selected for the 'Movie_Bot' project. The main area displays the 'movie' entity configuration. The entity name 'movie' is entered at the top, with a 'SAVE' button to its right. Below the name, there are two checkboxes: 'Define synonyms' (which is checked and highlighted with a red box) and 'Allow automated expansion'. A table below these options lists example values for the entity: 'The mummy', 'The dark Knight', 'Black Panther', and two rows with the placeholder text 'Enter value'. At the bottom left of the table, there is a '+ Add a row' link.

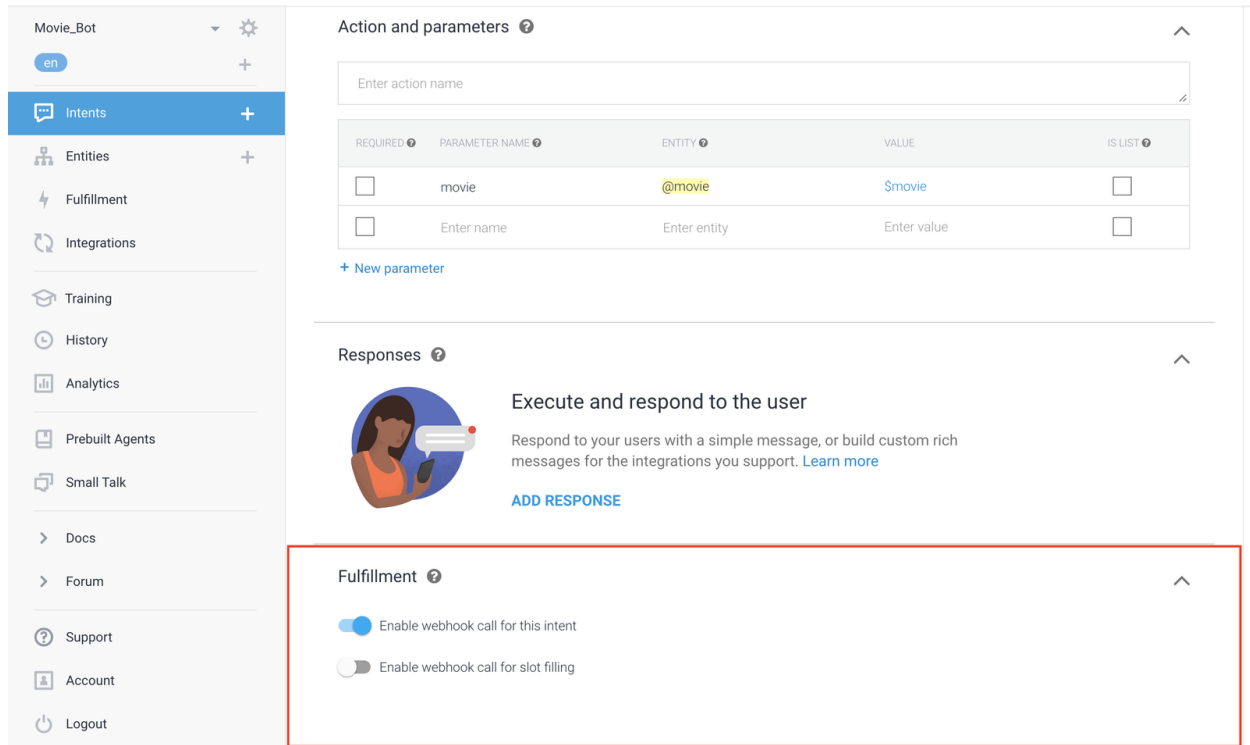
Next, click on **Intents** on the sidebar. Under the intents lists, click on **movie**.

Then for each of the intent you have typed earlier, highlight the movie name. You'll get an entity option, then select **@movie**.



When you are done, click on the **SAVE** button.

Finally, on the same page, scroll down to **Fulfillment** and enable it to use webhook then **SAVE**.



Movie_Bot

en

Intents

Entities

Fulfillment

Integrations

Training

History

Analytics

Prebuilt Agents

Small Talk

> Docs

> Forum

? Support

Account

Logout

Action and parameters ?

Enter action name

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

+ New parameter

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

Fulfillment ?

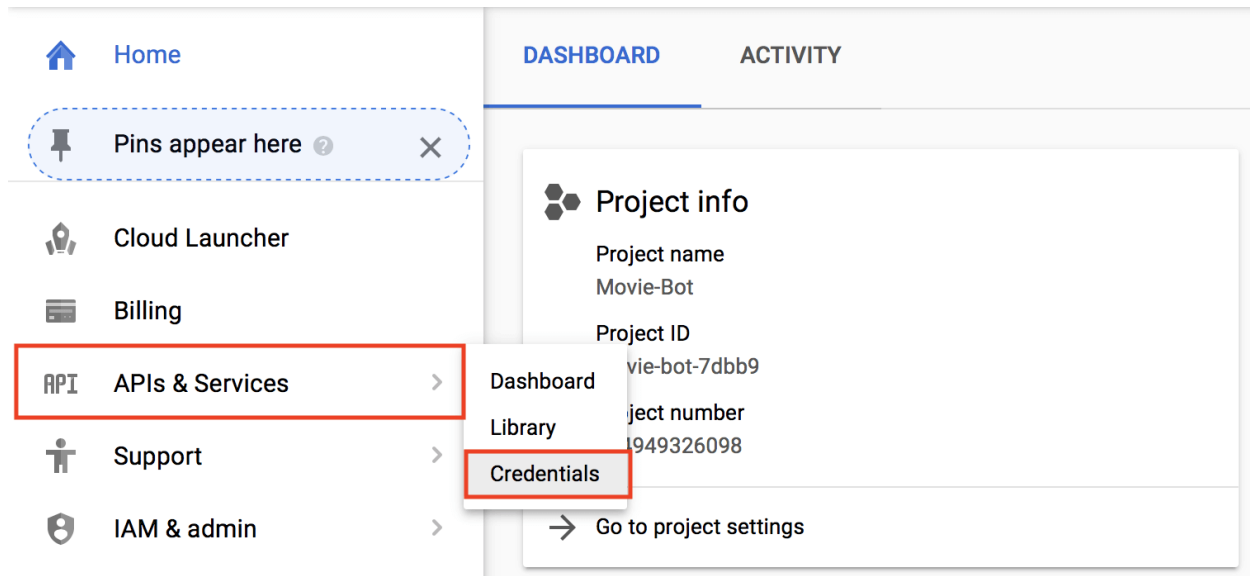
☒ Enable webhook call for this intent

☐ Enable webhook call for slot filling

GETTING OUR API KEYS

To make use of Dialogflow Python library, it requires that we have an API key. So we need to get it.

- Go to [Google Cloud Platform](#).
- Select our bot project name - **Movie-Bot** (This is the same as the name we gave our Agent)

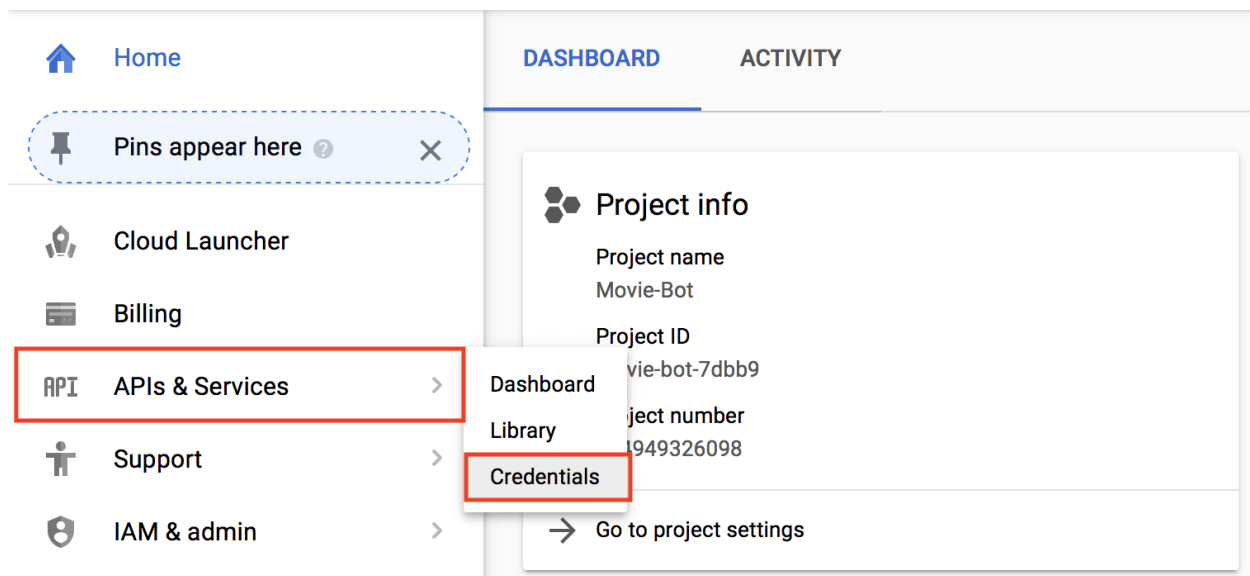


- Copy out the project_id. Then add the project_id to the .env file:

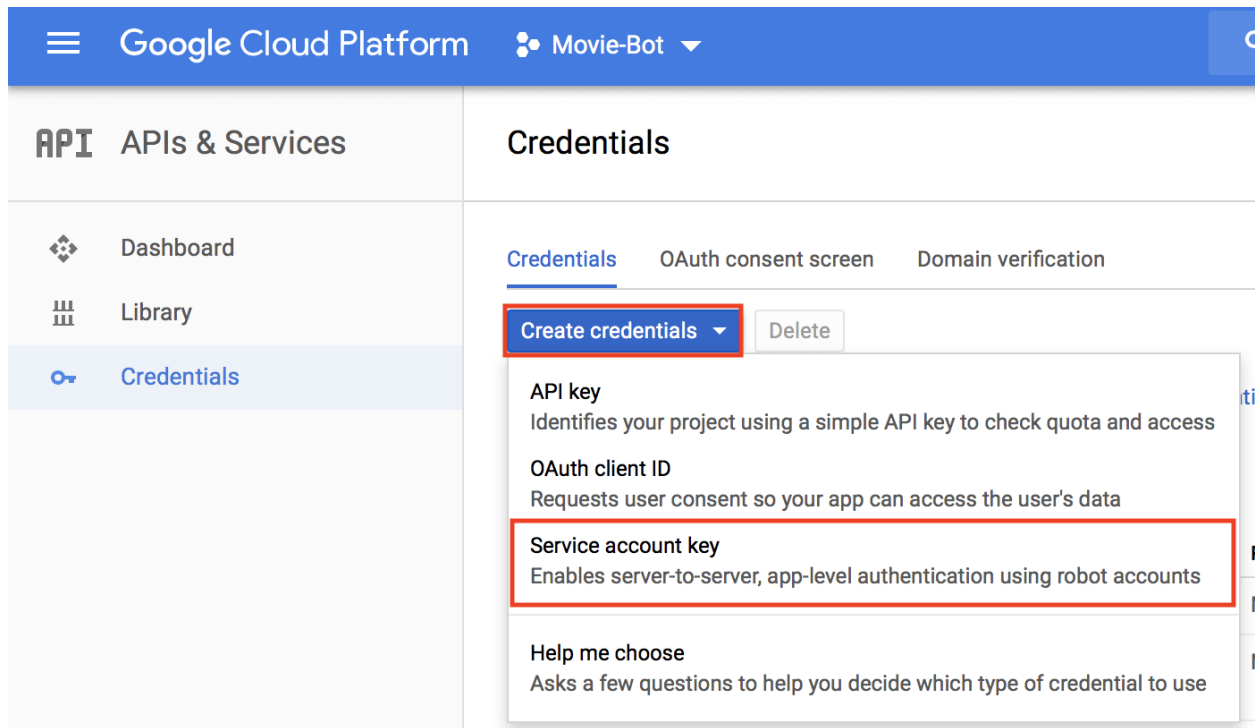
DIALOGFLOW_PROJECT_ID=project_id

Replace project_id with the project ID you just copied.

- Next, go to **APIs & Services** then **Credentials**

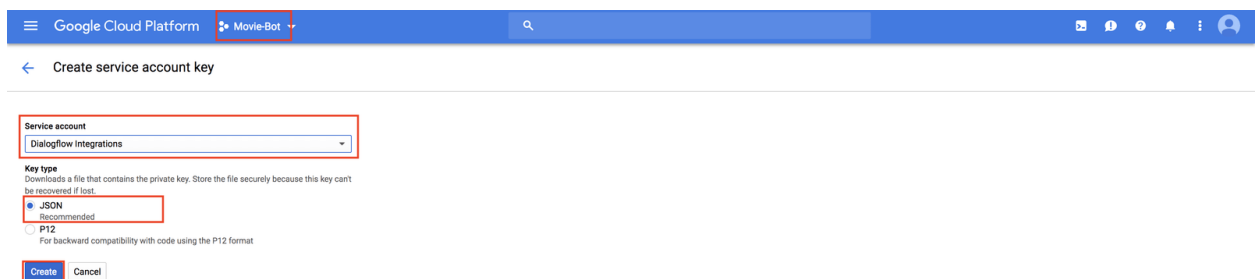


- Under **Create credentials**, click on **Service account key**



Once the page loads,

- Select **Dialogflow integrations** under **Service account**.
- Then select **JSON** under **key type**.



- Finally, click on the **Create** button to download your API key
- Your API key will be downloaded automatically.

Now, copy the downloaded JSON file (Movie-Bot-*.json) to the root folder of the project - movie_bot.

Note: This is your API key, you should not commit it to a public repository.

Next, add the the path to the file to the .env file:

```
GOOGLE_APPLICATION_CREDENTIALS=Movie-Bot-*.json
```

Make sure to use the correct file name instead of Movie-Bot-*.json.

SETTING UP A WEBHOOK

Now when a user asks our bot about a movie, it will detect the intent of the user but can't reply with the movie detail. That is where webhooks come in.

When our bot detects the movie keyword, it will make a request to our webhook using the movie keyword. Now using the movie keyword, we'll query OMDb for details about the movie. Then send the result back to Dialogflow.

Lets a create a new route which we'll use as our webhook. Add the following code to index.py:

```
# /index.py
...
@app.route('/get_movie_detail', methods=['POST'])
def get_movie_detail():
    data = request.get_json(silent=True)
    movie = data['queryResult']['parameters']['movie']
    api_key = os.getenv('OMDB_API_KEY')

    movie_detail =
requests.get('http://www.omdbapi.com/?t={0}&apikey={1}'.format(movie,
api_key)).content
    movie_detail = json.loads(movie_detail)
```

```

        response = """
            Title : {0}
            Released: {1}
            Actors: {2}
            Plot: {3}
        """.format(movie_detail['Title'], movie_detail['Released'],
movie_detail['Actors'], movie_detail['Plot'])

        reply = {
            "fulfillmentText": response,
        }

        return jsonify(reply)
    ...

```

In the preceding code:

- We created a route named `get_movie_detail`, which is a POST method.
- Then, we got the json request that will be sent by Diagflow to the route using `request.get_json()`.
- Next, we extracted the movie keyword using `data['queryResult']['parameters']['movie']` ['movie'] from the request data.
- Finally, we returned a JSON response of the movie detail

IMPORTANT NOTE

Make sure you have changed all the keys in env files by your own keys. For pusher, Go to pusher website and create an app. Give it a name and under keys write down all the 4 important keys. YOU Have to change keys in custom.js too.

After saving the keys. Save all the files.

EXPOSING OUR LOCALHOST TO THE WORLD

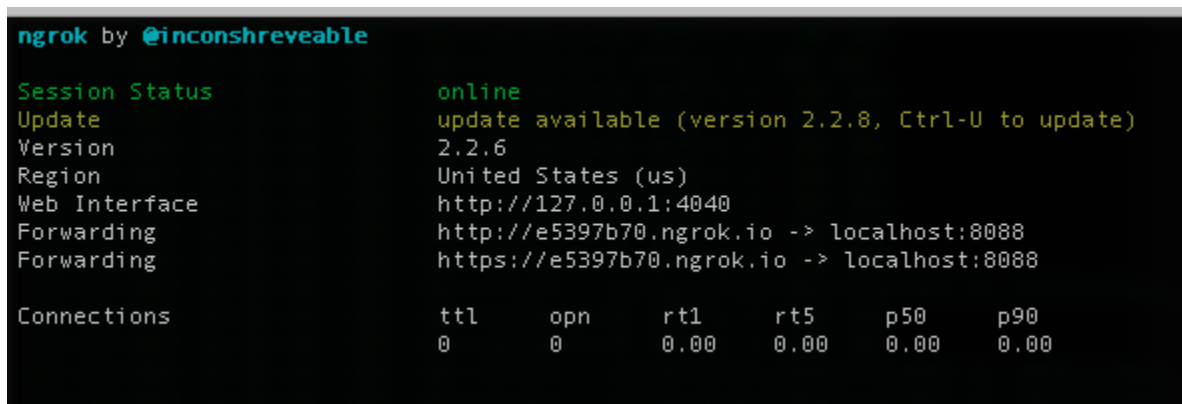
Now that we have created our webhook for getting details about a movie, let's add the webhook to Dialogflow. Oh wait, this is localhost, Dialogflow can't access it!

Hold on, Ngrok can help us out here.

Open up a new window of your command line, and type the following command:

```
ngrok http 5000
```

Now, you should get a screen like below



```
ngrok by @inconshreveable

Session Status      online
Update              update available (version 2.2.8, Ctrl-U to update)
Version             2.2.6
Region              United States (us)
Web Interface        http://127.0.0.1:4040
Forwarding           http://e5397b70.ngrok.io -> localhost:8088
Forwarding           https://e5397b70.ngrok.io -> localhost:8088

Connections         ttl    opn    rt1    rt5    p50    p90
                   0      0      0.00   0.00   0.00   0.00
```

If you are getting something like this then you are good to go.

Copy any of the forwarding URLs, the URL similar to this <https://0a36e90a.ngrok.io>.

Next,

- Click on Fulfillment on the sidebar of your Dialogflow console.
- Next, enable the webhook using the toggle button as indicated on the image below.

- Then, add the URL you just copied from Ngrok -
https://0a36e90a.ngrok.io/get_movie_detail(Replace https://0a36e90a.ngrok.io with yours).
- Finally, click the **SAVE** button to save

The screenshot shows the Dialogflow Fulfillment page. On the left is a sidebar with navigation options: Movie_Bot, Intents, Entities, Fulfillment (selected), Integrations, Training, History, Analytics, Prebuilt Agents, Small Talk, Docs, Forum, Support, Account, and Logout. The main area is titled 'Fulfillment' and contains a 'Webhook' section. The 'Webhook' section has a toggle switch set to 'ENABLED'. Below it, there's a 'Webhook example' section with a 'URL*' field containing 'https://e610a704.ngrok.io/get_movie_detail'. There are also fields for 'BASIC AUTH' (username and password) and 'HEADERS' (key and value). Below these is an 'Inline Editor' section, which is currently 'DISABLED'. The editor shows a JavaScript code snippet for handling intents. At the bottom right of the editor is a 'SAVE' button.

After saving it, just go to the URL that you have in ngrok prompt.

ADDING REALTIME FUNCTIONALITY

Let's now make our chatbot more interesting. Any user on the chatbot page should see in realtime conversation currently going on.

Initialize the Pusher Python library by adding the below code to index.py after the **imports**:

```
# /index.py
```

...

initialize Pusher

```
pusher_client = pusher.Pusher(  
    app_id=os.getenv('PUSHER_APP_ID'),  
    key=os.getenv('PUSHER_KEY'),  
    secret=os.getenv('PUSHER_SECRET'),  
    cluster=os.getenv('PUSHER_CLUSTER'),  
    ssl=True)
```

...

TRIGGERING AN EVENT

Next, trigger an event to Pusher when a user sends a message.

Add the following to the `send_message` function exactly before `*return* jsonify(response_text)` in `index.py` file:

```
socketId = request.form['socketId']  
  
pusher_client.trigger('movie_bot', 'new_message',  
    {'human_message': message, 'bot_message':  
fulfillment_text},  
    socketId)
```

In the preceding code:

- We'll trigger an event to Pusher on a channel - `movie_bot` - with an event named `-new_message`
- Then, we'll also pass along the message the user typed - `message` - and the reply of our bot - `fulfillment_text` - along with the connected user `socketId`

We are passing the `socketId` so that the user triggering the event will not get the message back.

LISTENING FOR AN EVENT

Now let's listen for the `new_message` event on the `movie_bot` channel. When the event occurs, we'll attach the message to the HTML DOM.

Add the following code to the topmost part of `static/custom.js`:

```
// /static/custom.js
```

```
// Initialize Pusher
```

```
const pusher = new Pusher('<PUSHER_KEY>', {  
  cluster: '<CLUSTER>',  
  encrypted: true  
});
```

```
// Subscribe to movie_bot channel
```

```
const channel = pusher.subscribe('movie_bot');
```

```
// bind new_message event to movie_bot channel
```

```
channel.bind('new_message', function(data) {
```

```
// Append human message
```

```
$('.chat-container').append(`
```

```
  <div class="chat-message col-md-5 human-message">
```

```
    ${data.human_message}
```

```
  </div>
```

```
`)
```

```
// Append bot message
```

```
$('.chat-container').append(`
```

```
  <div class="chat-message col-md-5 offset-md-7 bot-  
message">
```

```
    ${data.bot_message}
```

```
  </div>
```

```
`)
```

```
});
```

Don't forget to replace <PUSHER_KEY> and <CLUSTER> with your correct Pusher details.

Finally, update the POST request in the submit_message function in the static/custom.js file so that it passes along the Pusher socket_id of the user sending the message:

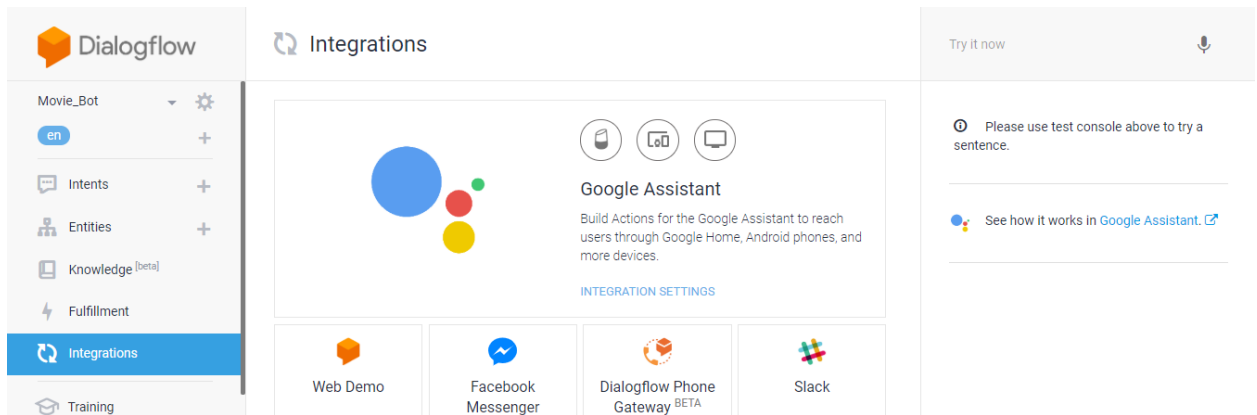
```
...  
$.post( "/send_message", {  
    message: message,  
    socketId: pusher.connection.socket_id  
}, handle_response);  
...
```

To test the app, restart the server. Ensure ngrok is running.

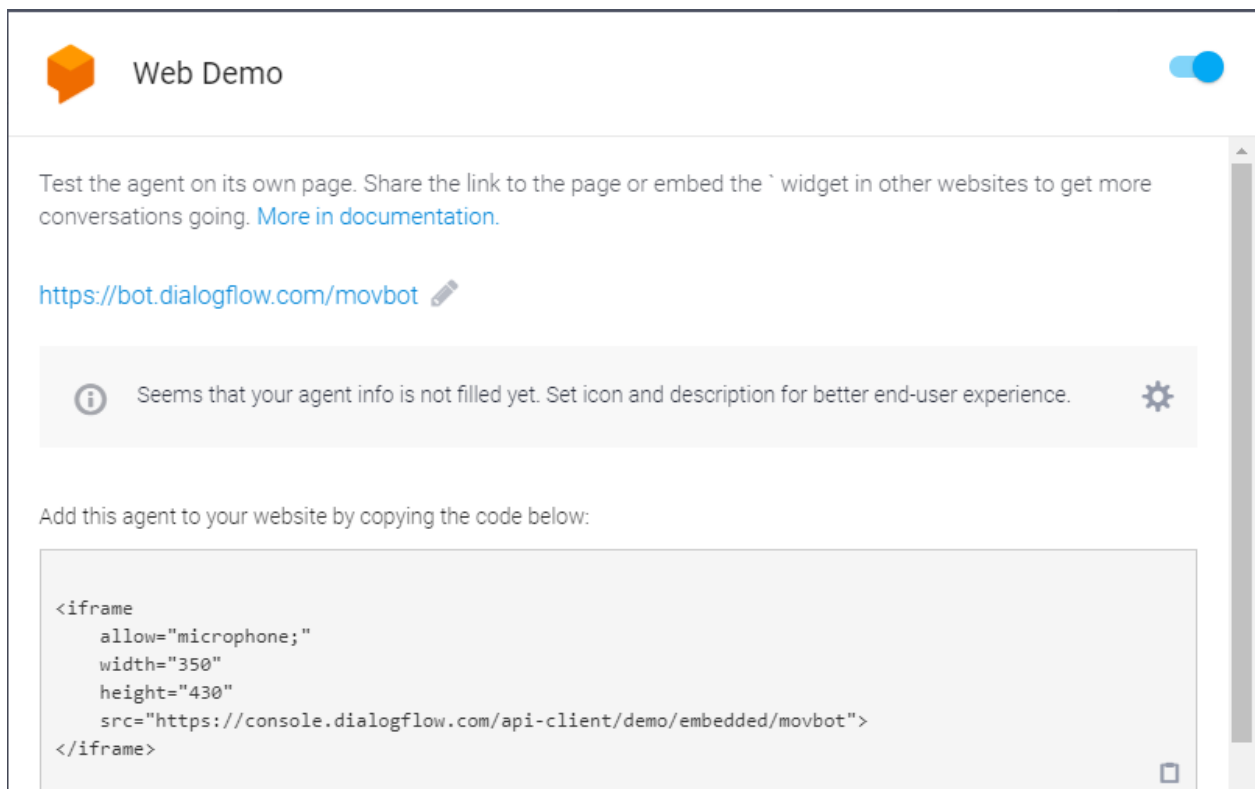
NOTE:- Every time you run the app from command then you have to change the web hook in the Dialogflow fulfillment .

WEB DEMO INTEGRATION

Go to Dialogflow integration to the left side of the dashboard and click web-demo enabled by toggling it.



When you will click then you will see something like that:



You will get a URL. You can change the number with any name in your URL. You can also integrate the bot to your website by attaching the script.

Best reference for powerful chatbot (AI Based)-

https://www.youtube.com/watch?v=dvOnYLDg8_Y