

BOT building for Covid19 – using Google Dialogflow

May 2020

Authored by: Sangeetha Yemisetty

Contents

Setting the Context.....	5
What is a Chatbot?.....	6
Business Problem.....	7
User Journey	7
Architecture	9
Intro to Dialogflow.....	10
Building the BOT.....	10
Account Creation in Dialogflow.....	10
Dialogflow Console.....	11
Agent Creation	11
Intents.....	11
Entities	16
Knowledge Base	16
Fulfillment	18
Building the Flask App	19
Folder Structure.....	20
Sequence Diagram	20
3 rd Party API.....	22
Python Packages	22
Testing the Flask app	23
Webhook Request and Response Structure	23
Deployment in Heroku	26
Heroku CLI Installation.....	26
Deployment Specific Files.....	27
Executing GIT.....	28
Channel Integration	28

Integration with Telegram.....	29
Integration with Facebook Messenger.....	30
Results.....	33
Facebook Screen Shots	33
Telegram Screen Shots.....	35
Email Screen Shots	38
Wrapping it up	39
Appendix	40
References.....	41

Figure 1 User Stories	7
Figure 2 User journey path 1	8
Figure 3 User journey path 2	8
Figure 4 Functional Architecture.....	9
Figure 5 Dialogflow account creation	10
Figure 6 Agent Creation in Dialogflow	11
Figure 7 Creation of Intent.....	11
Figure 8 Intent Creation - 1	12
Figure 9 Context setting in localcovidinfo	13
Figure 10 Context and Parameter setting for globalcovidinfo	13
Figure 11 Intent Creation - 2.....	14
Figure 12 Intent Creation - 3.....	15
Figure 13 Intent Creation - 4.....	15
Figure 14 Entity Creation	16
Figure 15 Enable Beta Feature for agent.....	16
Figure 16 Create Knowledge Base.....	17
Figure 17 Enable Knowledge Base.....	17
Figure 18 Knowledge Base Response Add.....	17
Figure 19 Fulfillment Webhook Service	18
Figure 20 Key Steps in Dialogflow.....	19
Figure 21 Folder Structure of Python Flask App.....	20
Figure 22 Sequence Diagram Path 1.....	21
Figure 23 Sequence Diagram Path 2.....	21
Figure 24 3rd Party API Details	22
Figure 25 Python Packages Used.....	22
Figure 26 POSTMAN Request in local machine	23
Figure 27 Telegram Setup	29
Figure 28 Telegram setting in Dialogflow.....	30
Figure 29 Facebook Create Page	30
Figure 30 Add a Button.....	30
Figure 31 Create New Facebook APP.....	31
Figure 32 Add product and Page with Facebook app.....	31
Figure 33 Generate Token in Facebook App.....	31

Figure 34 Setting Token in Dialogflow	32
Figure 35 Webhook setting in Facebook Developer	32
Figure 36 Add Subscriptions for Facebook app	33
Figure 37 Facebook Screen 1	33
Figure 38 Facebook Screen 2	34
Figure 39 Figure 36 Facebook Screen 3	35
Figure 40 Telegram Screen 1	35
Figure 41 Telegram Screen 2	36
Figure 42 Telegram Screen 3	36
Figure 43 Telegram Screen 4	37
Figure 44 Telegram Screen 5	37
Figure 45 Email Screen Shot 1.....	38
Figure 46 Global View from email attachment (plotly interactive graph)	38
Figure 47 Key Milestones of the Project	39

Setting the Context

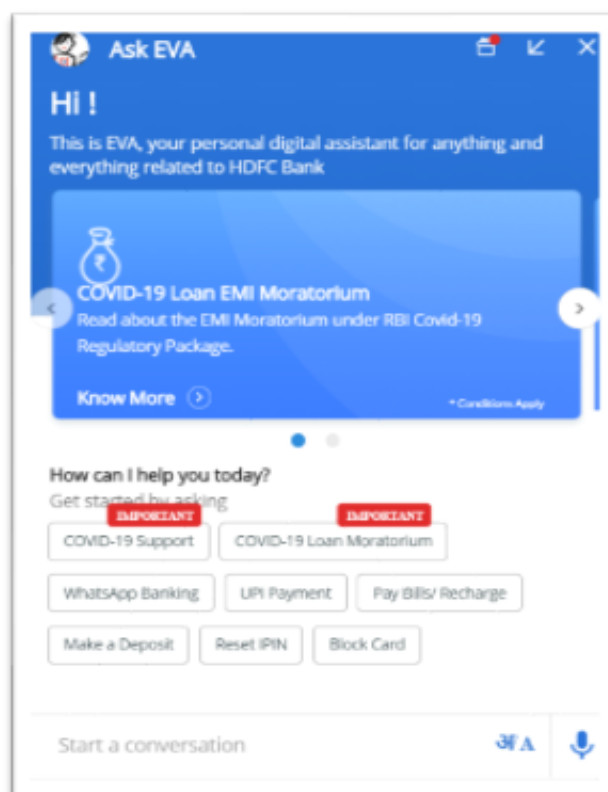
This document provides a step by step guide of building a BOT using Google Dialogflow. The document starts with the business requirement and ends at deploying the BOT. It touches upon all the key components of BOT building and assumes that someone has basic python knowledge. Attempt is made to elaborate the process in a business-oriented language rather than going too technical in nature. Hope you enjoy reading and implementing the same. ☺.

What is a Chatbot?

When I searched the internet, figured out almost thousand of definitions. So, thought of simplifying the same in my write-up. As the name suggests, there are 2 parts.

- Chat – which means capable of doing a conversation
- Bot – which means it is non-human being (often times a software application)

To put in context, a chatbot is something which will understand human language and reply back accordingly and carry on with the conversation till you are fulfilled. With the rapid advancement of AI & Machine Learning, conversational AI has improved a lot. Chatbot is nothing but conversational AI in action. While traditional chatbots used to be rule based, which is very cumbersome to be maintained, majority of the current chatbot frameworks are AI based.



Caution- Please don't expect that chatbots are emotional beings. These are applications which are trained to do some pre-defined conversation. So, it will not be able to fulfil all types of conversation as a normal human being can.

Some of the key benefits of chatbots are –

- **Customer Service** – Almost the entire service sector (banks, insurance, college etc) have deployed chatbots in their communication channels as a 1st touchpoint to their customers.
- **Cost Effective** – Once build & deployed, running cost is almost negligible.
- **Automation** – zero human touch
- **Time to launch** – time to build & deploy is in days. Also these applications can be easily integrated with whole host of channels. (company website, facebook messenger, twitter etc)

Business Problem

As we all know that at present world is going through a pandemic called 'Covid19', thus there is a need to share correct and relevant information to the people, so that people stay informed and take precautions. This chatbot is constructed under the same underlying guiding principles. While there are many use cases to solve, the following diagram depicts the high-level use cases that are considered for our BOT. This is to limit the scope and build something end to end within a short timeframe.

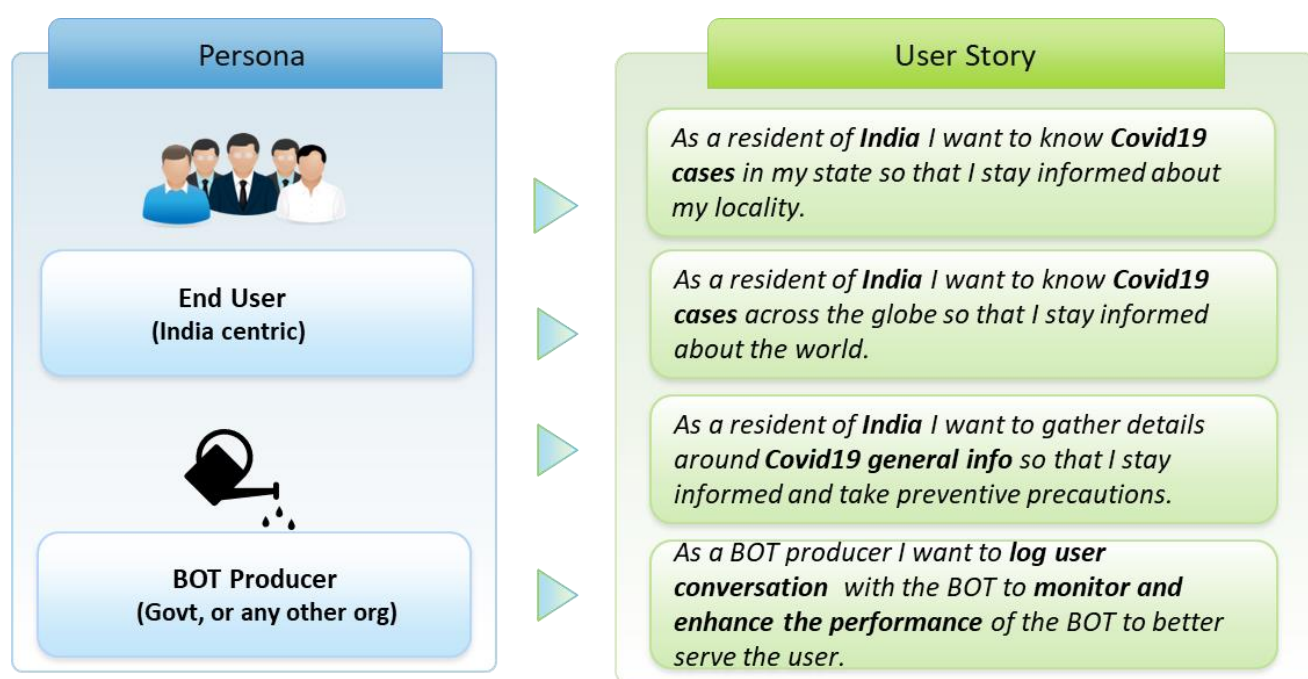


Figure 1 User Stories

As depicted above we are targeting two main set of personas. **End user** is obviously our prime persona. It is the intent to share the information to the end user in the easiest & quick possible way. While we discuss further on the business use cases, we have taken some assumptions to limit the scope.

User Journey

We have considered the following 2 user journey for our scope. Please note in a chatbot there can be 'n' no of paths a user can navigate. It is the design of the BOT that should manage to handle those unseen paths.

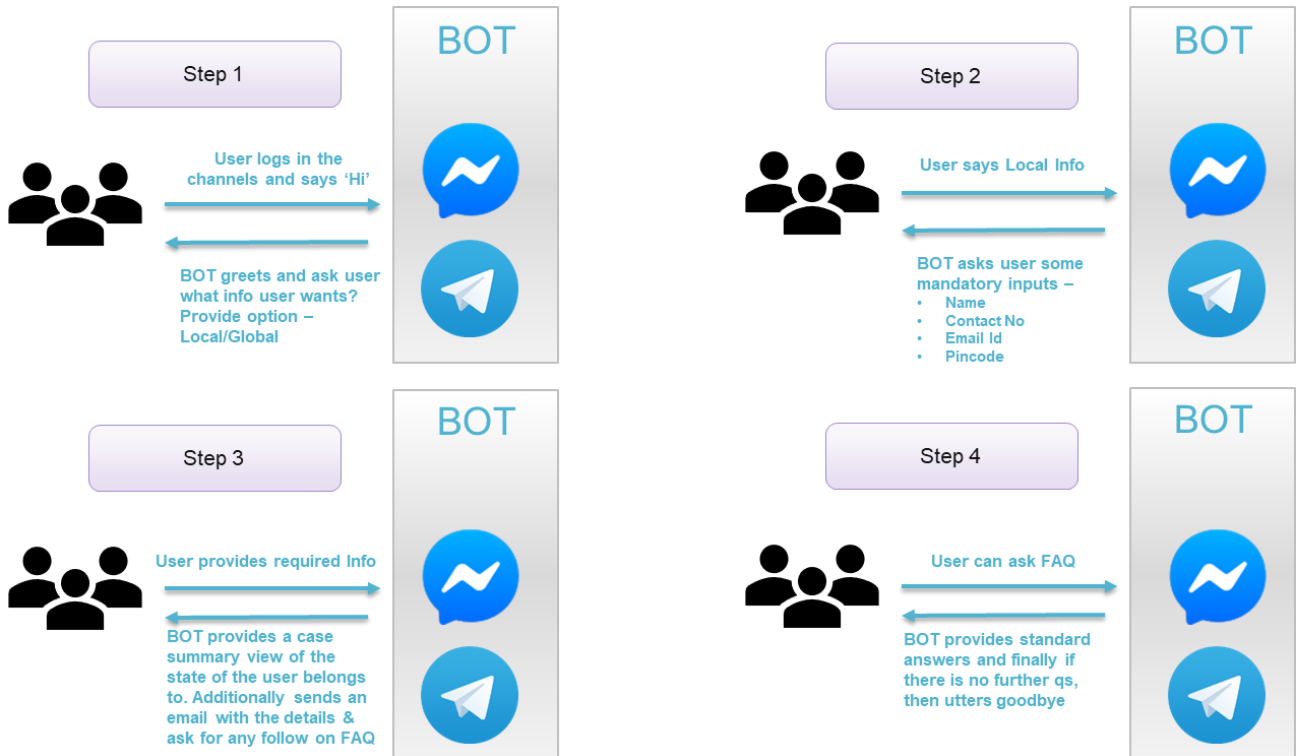


Figure 2 User journey path 1

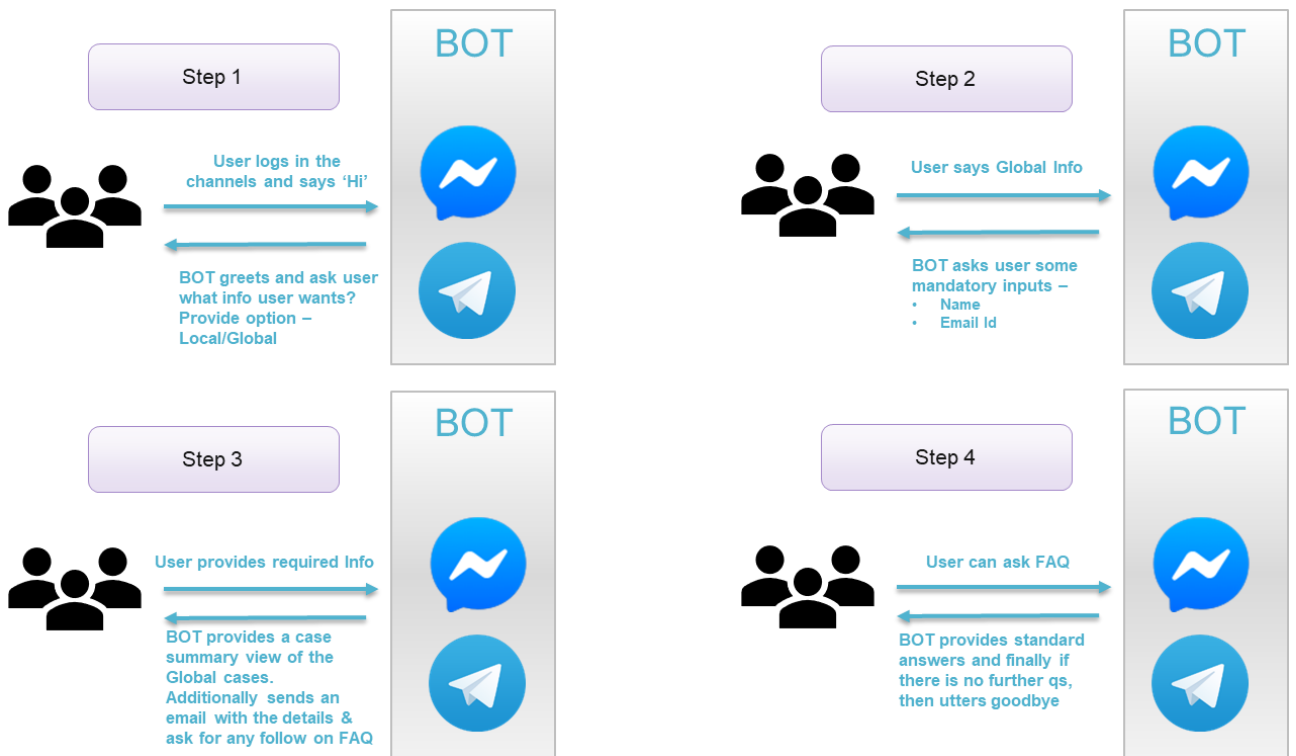


Figure 3 User journey path 2

Architecture

The following diagram depicts the functional architecture for the BOT. We will discuss each element of the architecture as we proceed. The below diagram depicts Google Dialogflow as the chatbot framework. Please refer appendix A, for a brief of some of the chatbot frameworks available in the market.

The diagram is a simplified view of the architecture. As depicted, there are primarily 4 layers that exist.

They are as follows –

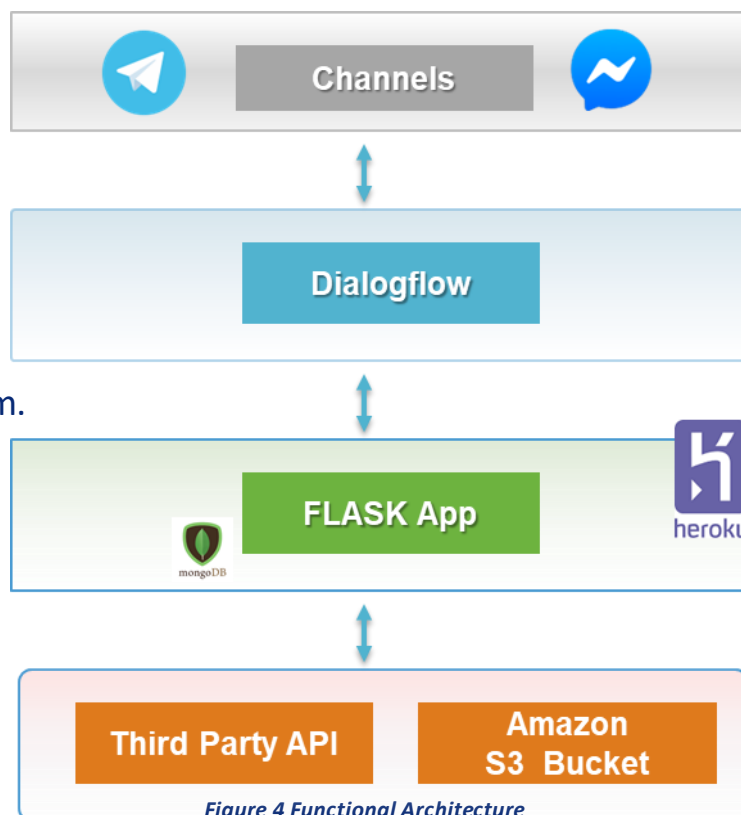
1) **Channels** – Two channels have been used for this BOT. Facebook Messenger & Telegram.

2) **Dialogflow** – Google Dialogflow has been used as the chatbot engine

3) **Flask App** – this is python flask-based application which gets the input from the user and provides the case data accordingly. Also, it sends

email to the user with the Covid case details. The app has been deployed in Heroku cloud.

4) **Third party** – This layer consists of 3rd party API's which have been used to get the Covid case data in real time. Amazon S3 bucket has also been used to store the graph images created by the Flask App, so that it can be accessed by the channels.



Note – This application has been built on a personal capacity. So, all the resources used are free of cost. MongoDB has been used in the local machine for storing the logs. As it is not freely available in the cloud, have used file-based logging during deployment to cloud. 😊

Intro to Dialogflow

Google Dialogflow is a conversation management tool, based on natural language understanding and machine learning tools.

The key features of the framework are –

- **Multichannel integration** – It provides easy integration with multiple social media channels through configuration only. In our case we have integrated with Facebook messenger & Telegram.
- **Virtual Agents** – These are the assistants or BOTS which manages the conversational flow. This is the heart of any chatbot framework. Down the line we will see how we have created our agent for Covid19 and trained it.
- **Fulfillment** – In order to provide data inputs to the users, it is often necessary to interface with external service providers to fetch data. Hence this layer is critical for the conversation to be successful. In our case Flask App is the service provider which provides data to the agent.

Building the BOT

Account Creation in Dialogflow

As a very 1st step, we need to get an account created in Dialogflow. Please follow the below steps to get the account created.

- 1) Go to <https://dialogflow.com/> and click on 'Sign Up for Free' button.
- 2) Click 'sign-in with Google'.
- 3) Select your google account and once you are redirected, click on 'Go To Console' on the upper right corner of the screen.

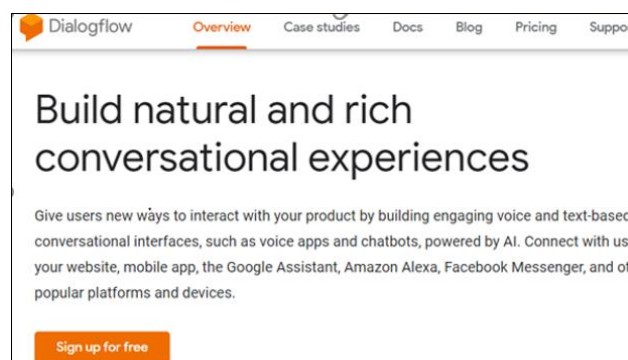


Figure 5 Dialogflow account creation

Dialogflow Console

Agent Creation

Click on the create agent link to create your first agent. As I have created few agents already it is showing up in the tab. On the right-hand side provide the name of the agent and then click CREATE. Well done your first agent has been created. Have named it as covid19.

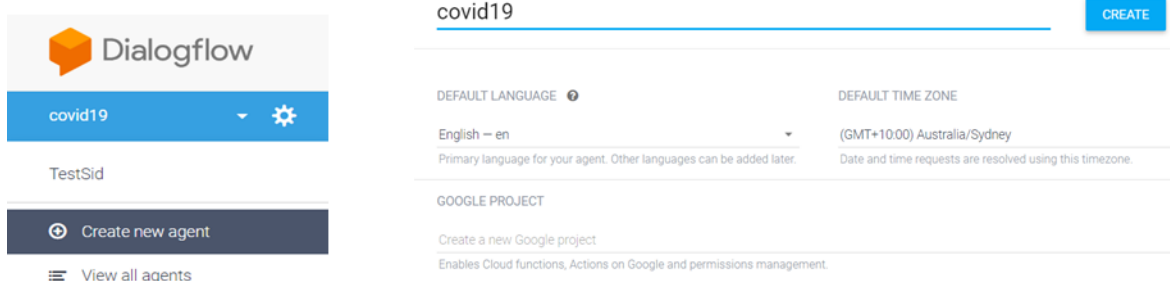


Figure 6 Agent Creation in Dialogflow

Intents

Once you have created the agent & saved it, Dialogflow engine will automatically train it. The next phase is to create Intent & Entities. The screen will look as depicted.

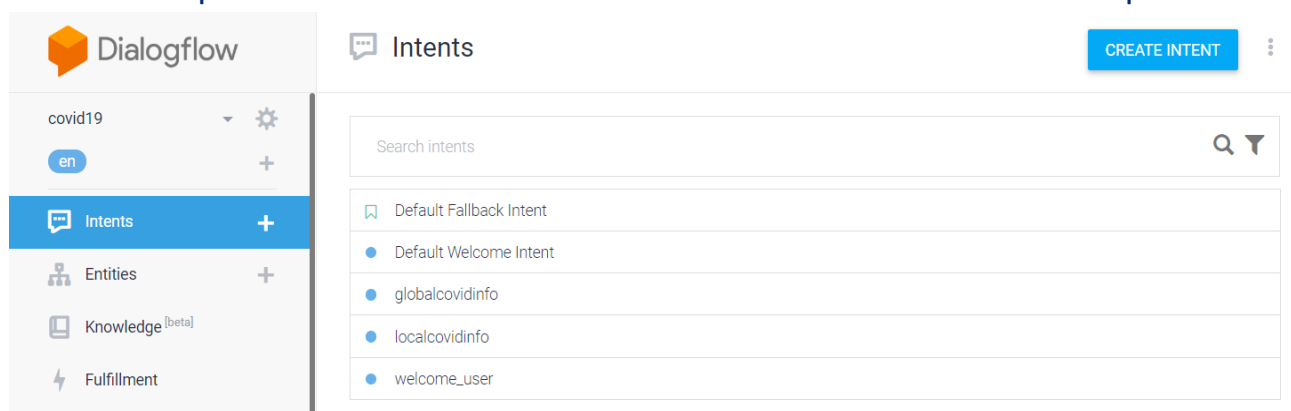


Figure 7 Creation of Intent

It's important to understand what are intents. In simple language an intent is an objective that user of the BOT wants to satisfy. For any project that we need to do, we need to create as many intents possible so that the BOT gets trained. Each intent should be associated with training phrases and a corresponding response. The training phrases are those ones which user will ask for. Based on the training phrase, agent will identify the corresponding intent. Once it identifies the intent, it will respond back according to the

response definition. As seen above for our Covid19 project we have created some intents. There are 2 intents which are already defined by Dialogflow. So, whenever we create a new agent those 2 are automatically created. They are –

- **Default Fallback Intent** – If for any user phrase, BOT is unable to identify an intent, then it will fall back to this default intent. This is equivalent to exception handling scenario in the programming world. The objective should be the agent should be trained with variety of phrases so that, BOT need not fallback to this intent.
- **Default Welcome Intent** – This is the intent that gets identified by the BOT when user starts with the 1st message after entering the BOT. Typical phrases will ‘Hi’. ‘Hello’ etc.

Note – Although these intents are system defined, we can override the same and create our own fallback & welcome intent. For Covid19 am using the system defined ones. We can add more phrases to train the model.

Now let’s start focus on the process of creating an intent and the associated steps. Have numbered the items for easy reference with the figure.

The screenshot displays the Dialogflow console interface for creating a new intent. The intent name 'localcovidinfo' is entered at the top, marked with a red circle '1'. Below it, the 'Contexts' section is marked with a red circle '2', and the 'Events' section is marked with a red circle '3'. The 'Training phrases' section is marked with a red circle '4' and contains a search bar and a list of example phrases: 'Add user expression', 'I want to know about covid details in my state', 'i want to have details of my locality', 'I need info of my place', and 'I need covid info of locality'. A 'SAVE' button is visible next to the intent name.

1) **localcovidinfo** - This is the name of the intent. Free text. For our covid BOT, there is an user path where customer wants details of local area. This is the intent that the BOT should identify based on user phrases. For globalinfo user path 2, have created a **globalcovidinfo** intent.

2) **Contexts** – If there are any parameters that needs to be shared across multiple intents then this can be set. For eg- customer name & email in our project.

Figure 8 Intent Creation - 1

- localcovidinfo

Contexts ?

Add input context

5 name (x) 5 emailid (x) Add output context

For localcovidinfo we are getting the user name & email id. Those are configured as output context for this intent. The same name & email id will be used for globalcovidinfo intent. Hence, we are using the same as input context for this intent. This avoids asking the customer email id again. PFB the screens on how it is configured. Also, the value of the parameters is given as #name.cust_name & #emailid.cust_email. This is how we can access the parameters in this intent although captured in another intent.

Figure 9 Context setting in localcovidinfo

- globalcovidinfo

Contexts ?

name (x) emailid (x) Add input context

Add output context

Action and parameters

Enter action name

REQUIRED ?	PARAMETER NAME ?	ENTITY ?	VALUE	IS LIST ?	PROMPTS ?
<input checked="" type="checkbox"/>	cust_name	@sys.any	#name.cust_name	<input type="checkbox"/>	Define prompt S...
<input checked="" type="checkbox"/>	cust_email	@sys.email	#emailid.cust_email	<input type="checkbox"/>	Define prompt S...
<input type="checkbox"/>	Enter name	Enter entity	Enter value	<input type="checkbox"/>	—

+ New parameter

Figure 10 Context and Parameter setting for globalcovidinfo

3) **Events** – Google provides some inbuilt phrases for particular scenarios. If our intent use case matches with the inbuilt events, we can reuse the same instead of defining new phrases.

4) **Training Phrases** – These will be the user utterances based on which agent will identify the correct intent. This part is very critical as you want machine to learn as much as possible with variety of utterances.

5) **Actions and parameters** –

In this part we are training the agent of what it needs to perform once it identifies the intent based on the user utterances. Based on user journey path 1, we need to collect some mandatory information from customer. So, we have configured the parameters accordingly. The following are the details of each field –

Action and parameters 5

Gather_cust_details

REQUIRED ?	PARAMETER NAME ?	ENTITY ?	VALUE	IS LIST ?	PROMPTS ?
<input checked="" type="checkbox"/>	cust_name	@sys.any	\$cust_name	<input type="checkbox"/>	Provide your n a...
<input checked="" type="checkbox"/>	cust_email	@sys.email	\$cust_email	<input type="checkbox"/>	Provide your e m...
<input checked="" type="checkbox"/>	cust_contact	@sys.phone-num	\$cust_contact	<input type="checkbox"/>	Provide your m o...
<input checked="" type="checkbox"/>	cust_pincode	@sys.any	\$cust_pincode	<input type="checkbox"/>	Please provide ...
<input type="checkbox"/>	Enter name	Enter entity	Enter value	<input type="checkbox"/>	–

+ New parameter

Figure 11 Intent Creation - 2

- a. **Required** – whether parameter is mandatory/optional
- b. **Parameter Name** – the name of the parameter
- c. **Entity** – google already provides system defined entities. It is nothing but variable type with validation rules inbuilt. For eg. For email we are using @sys.email. If we provide the system defined entity, then agent has the ability to extract the value form the user input. We can create our own entity which will be discussed later. @sys.any is catch-all kind of datatype.
- d. **Value** – as this info will be provided by the user we add a \$ and the paramter name. These will be run time values.
- e. **Is List** – If the parameter has a list of values, then we can tick that.
- f. **Prompts** – This is the phrase that BOT will ask to the customer. For eg. Pls provide your name? for cust_name parameter

6) **Responses** – This is the final output of the intent. This is what BOT should provide as a final response to the user for this intent. As seen, there is a default response. We can add channel specific response as well. This is more of a static response. We can also add a custom payload, which can call a service to generate a response instead of a direct static response. There is an option to select, whether this response is the last response of the conversation. Typically, the goodbye intent will have this checked.

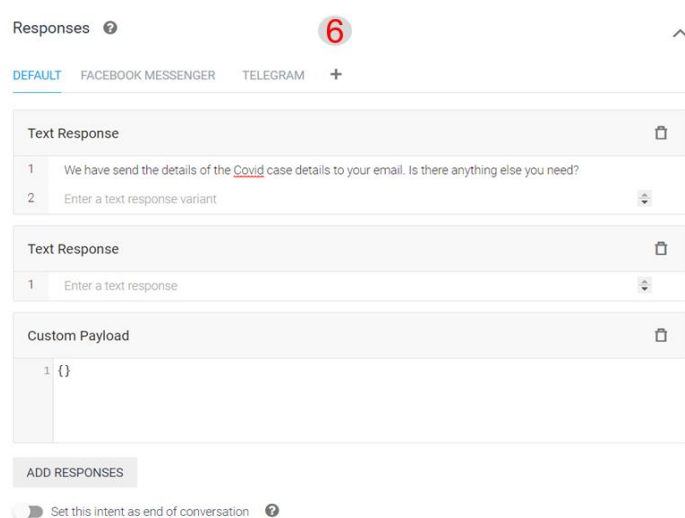


Figure 12 Intent Creation - 3

7) **Fulfillment** – if for a particular intent, we need to fetch data from external provider, then we need to enable fulfillment. Also, this enables a more dynamic response. There are 2 varieties. –

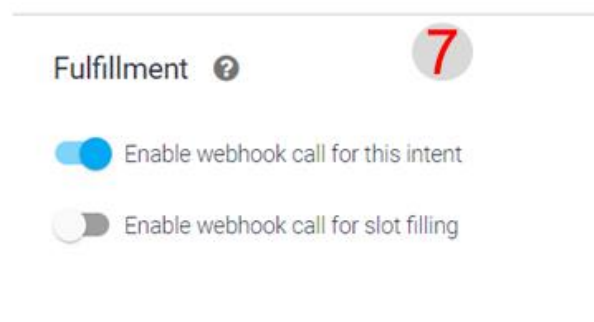


Figure 13 Intent Creation - 4

- a. **Enable webhook call for this intent** – if enabled, then agent will call a webhook service (external API). We are using this for our covid19 project, as we will call the Flask App by this webhook functionality.
- b. **Enable webhook service call for slot filling** – slot filling is parameter filling. If user misses to fill in some parameters which were not asked explicitly, then using this webhook call we can dynamically ask user to fill in those parameters. For this also we need to write custom code.

Entities

Entities are user provided parameters which are not pre-defined datatypes in the Dialogflow engine. In that case we need to create our custom entity type and train the agent with appropriate training words. We haven't used any entity for our project. This entity is just for demonstration purpose. We have created an entity named `course_details`. We have given some synonymous words. So, system will learn according to the training words & extract the `course_details` attribute from the user utterances.

The screenshot shows the configuration for an entity named `course_details`. At the top right is a blue **SAVE** button. Below the entity name, there are four checkboxes: ☒ Define synonyms, ☐ Regexp entity, ☐ Allow automated expansion, and ☐ Fuzzy matching. A grey tooltip box states: "Separate synonyms by pressing the enter, tab or ; key." Below this is a table with columns for adding synonyms. The first row contains the text "subject" in a box, followed by "subject" and "chapter" in separate boxes, and then "Enter synonym". The subsequent three rows each contain a "Click here to edit entry" link. At the bottom left, there is a "+ Add a row" link.

Figure 14 Entity Creation

Knowledge Base

This is a beta feature in Dialogflow. So, before we can use it, we need to enable beta features for this agent. Click on the agent and enable the beta features as shown. This is a very useful feature wherein we can train the agent with a standard set of frequently asked questions about the topic. In this project we have loaded FAQ about Covid19. Once we click create document, the following screen will appear. For Covid19 we have taken the FAQ from WHO site. Once we click the create button, system will parse the html page of the site and pull out all the questions & answers. Next time if user ask a similar question, then agent will provide the answer from this knowledge. Thus, it becomes efficient if we upload FAQ. In that case there is no need of manual intent creation with responses.

The screenshot shows the "API VERSION" and "BETA FEATURES" settings for an agent. Under "API VERSION", the "V2 API" is selected with a radio button, and the text says "Use Cloud API as default for the agent. Your webhook will receive and return V2 format messages." Under "BETA FEATURES", the "Enable beta features and APIs" toggle is turned on (blue), and the text below says "Be the first to get access to the newest features and latest APIs. (Full V2-beta API reference)".

Figure 15 Enable Beta Feature for agent

Figure 16 Create Knowledge Base

Once we upload the questions, then we need to enable the Knowledge Base.

Figure 17
Enable Knowledge

Figure 18 Knowledge Base Response Add

Finally, we need to add the response in the following fashion. The `index[1]` states return a single response with the highest matching probability. This completes the configuration of Knowledge Base.

Fulfillment

This is another critical element of the agent functioning. As mentioned earlier for both intents (localcovidinfo & globalcovidinfo), we will fetch data from external provider. Hence, we have enabled webhook service for both the intents. In the fulfillment section we will provide the API URL which the webhook service will be calling to fetch the information. As mentioned below Dialogflow will call an endpoint (POST request) and the response of the endpoint has to be in the format as expected by Dialogflow. From our Covid project standpoint, this endpoint is being provided by our Flask App when deployed in cloud. We will cover the details in our Flask APP section.

Webhook

ENABLED 

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


URL*	<input type="text" value="https://covidchatbotot.herokuapp.com/webhook"/>	
BASIC AUTH	<input type="text" value="Enter username"/>	<input type="text" value="Enter password"/>
HEADERS	<input type="text" value="Enter key"/>	<input type="text" value="Enter value"/>
	<input type="text" value="Enter key"/>	<input type="text" value="Enter value"/>
	 Add header	
SMALL TALK	<input type="checkbox"/> Disable webhook for Smalltalk	

Figure 19 Fulfillment Webhook Service

Before we wrap up the Dialogflow part let us do a recap of the critical steps that is performed to setup the BOT. One part which is not discussed yet is the integration with 3rd party channels like Facebook Messenger & Telegram. We will discuss in a separate section. The next picture highlights the keys steps executed so far.

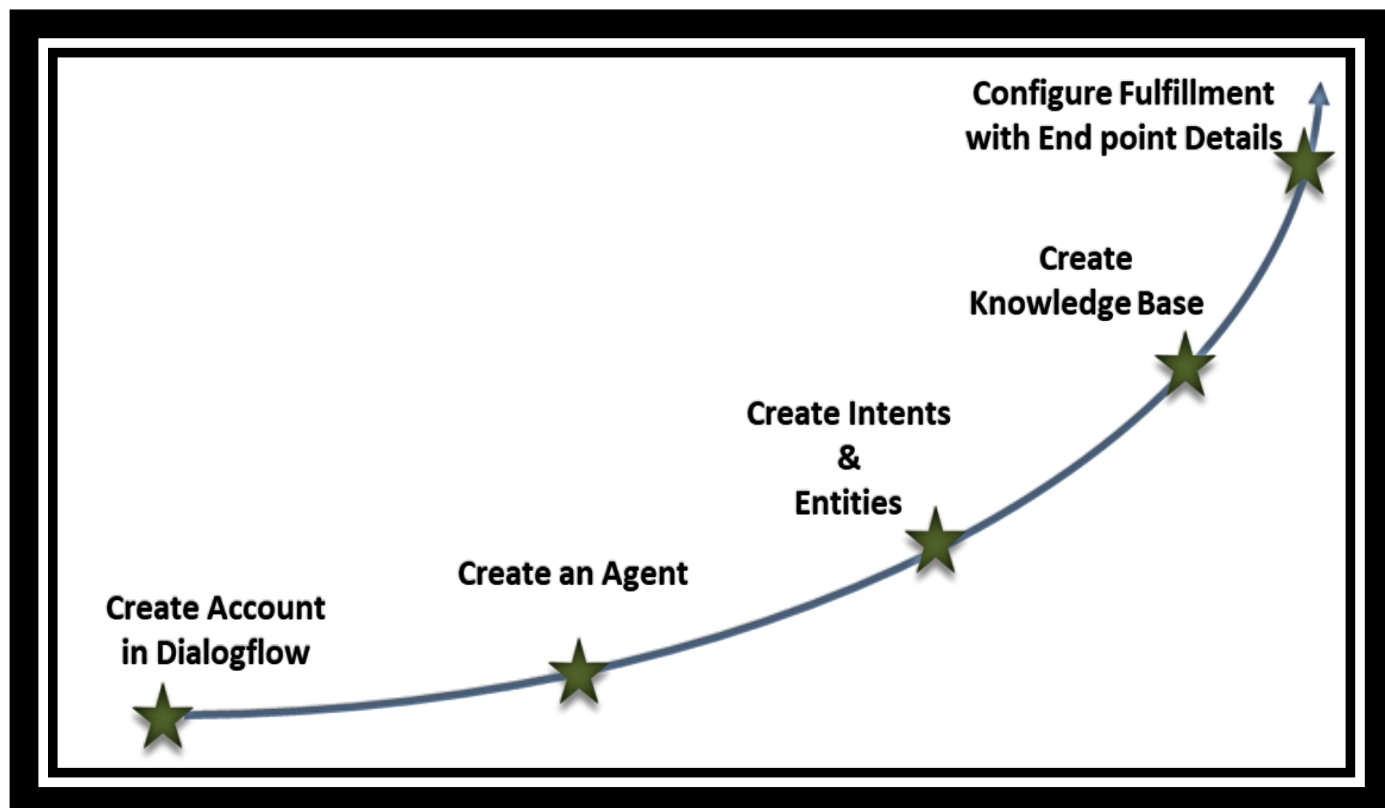


Figure 20 Key Steps in Dialogflow

Building the Flask App

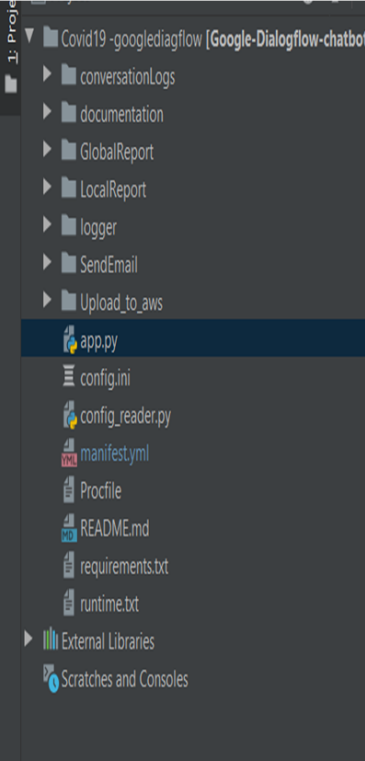
This is the heart of the overall BOT application. The full code is available in the [Github account](#). Have used PyCharm as the IDE for python. Please refer to the appendix A on how to install PyCharm. It is recommended to create a new conda env for this project. You can create a new environment using PyCharm easily. Please note this program requires quite a lot of libraries. All are listed in the requirements.txt file. All are simple pip installations.

Key Functions –

- 1) Fetch Covid case data (local/global) and pass it to Dialogflow
- 2) Send email to user

Folder Structure

Below diagram depicts the folder structure and the purpose of each folder.



File/Folder	Purpose
conversationLogs	This folder is to store the session logs
documentation	This folder maintains document related to this application
GlobalReport	This folder contains the image file for globalreport which is send as email.
LocalReport	This folder contains the image file for localreport which is send as email.
logger	This folder has logger.py. This is a support python program to write the logs
SendEmail	Folder having the python program to send an email
Upload_to_aws	This is a support program to upload image file to S3 bucket
app.py	This is the starting point of the program. We have instantiated a Flask POST Rest API.
config.ini	This holds the various end point details and keys that will be used to call 3 rd party services
config/config_reader.py	Python program to read a configuration file
logger/logger.py	This is a support program to write the logs
manifest.yml	File required for deployment to PCF
Procfile	File required for any cloud deployment
requirements.txt	This is the file contains the list of libraries that are required for deployment
runtime.txt	This file has the python version required for cloud deployment

Figure 21 Folder Structure of Python Flask App

Sequence Diagram

The below diagrams depict the sequence of action for both the parts.

- Local Covid Info
- Global Covid Info

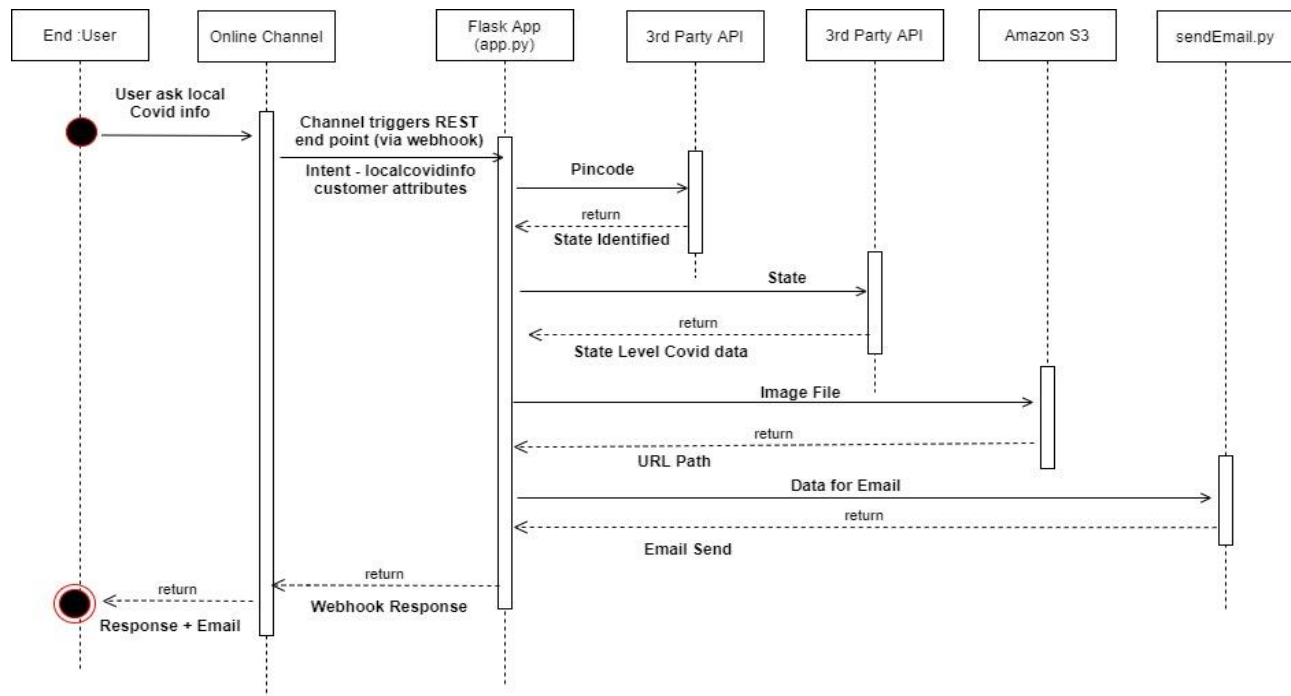


Figure 22 Sequence Diagram Path 1

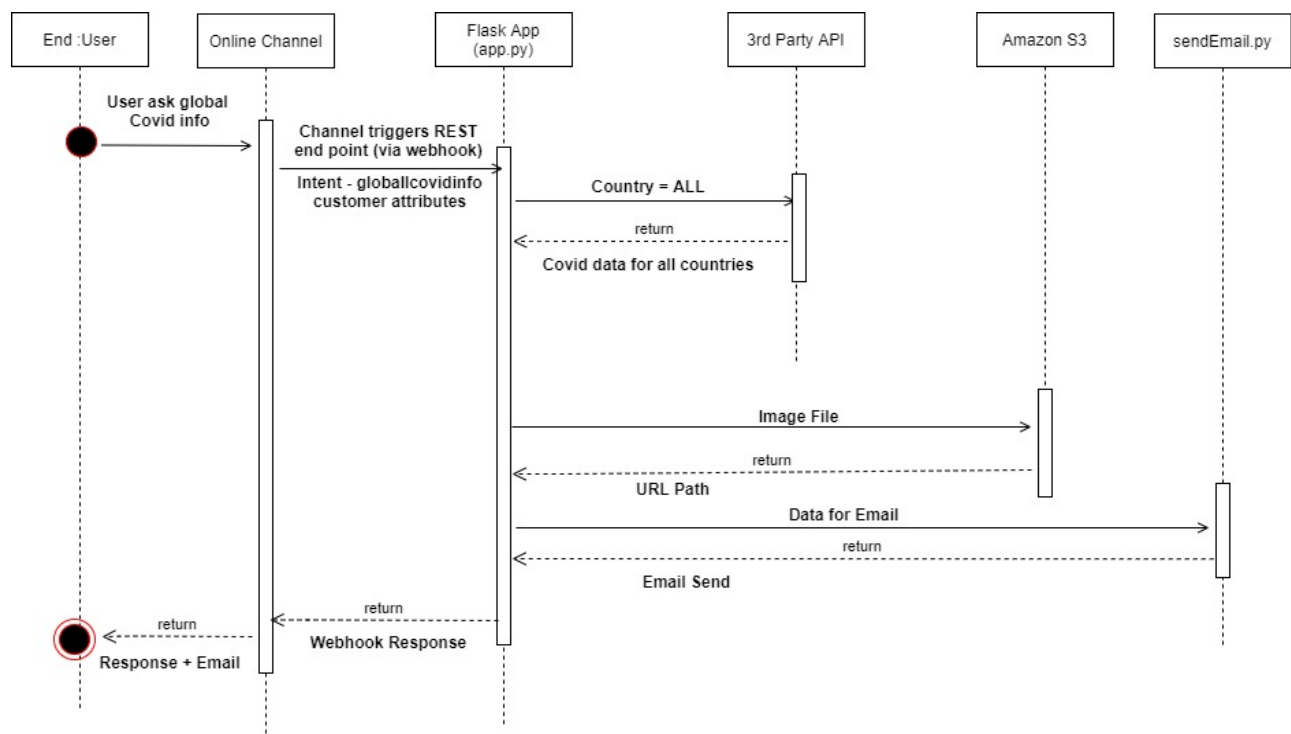


Figure 23 Sequence Diagram Path 2

Key Notes –

- 1) The path differs based on the intent identified. The webhook payload to flaskApp will have the intent. So, it is critical that agent identifies the correct intent based on user utterances.
- 2) We are storing the report images in Amazon, as webhook response expects a URL (accessible through internet) if sending an image. This is an area of further exploration if there any better methods of sending image (rich text) response.

3rd Party API

The following 3rd party API has been used to fetch Covid data.

API (end point URL)	Purpose	Input Parameter
"https://pincode.p.rapidapi.com/"	Returns the State of India based on pincode	Pincode
"https://covid19india.p.rapidapi.com/getStateData/"	Returns Covid stats for a state.	State in ISO format
"https://covid-193.p.rapidapi.com/statistics"	Returns Covid stats for all countries	-

Figure 24 3rd Party API Details

Python Packages

Let's discuss some of the python packages that have been used and its purpose.

Package	Purpose
Flask (sub-module - Flask, request, make_response)	Main package to create the REST (POST method) end point,;receiving the webhook request, deconstructing the message, sending the response back to the webhook service
json	This is used to convert the string response in JSON format before sending the response back to webhook call
flask_cors	Enable Cross Origin Resource Sharing for the flask end point (CORS is a W3C standard that allows you to get away from the same origin policy adopted by the browsers to restrict access from one domain to resources belonging to another domain)
uuid	This enables to generate unique ids. This unique id is being used to create the image filename before storing in S3 bucket
pandas	This is standard python library for data processing
plotly.express	This is a library to make interactive images (have used for the global covid view)
smtplib	Sending email
email.mime	Multipurpose Internet Mail Extensions (mime) – this is a standard to extend the format of the email messages to support text and other attachments of audio, video, images etc.
configparser	This is used to read values from a .ini file (key, value pair)
pymongo	This library helps in connecting with the mongo db
boto3	This is used to talk to Amazon Web Services (AWS) from Python. (EC2, S3 etc)

Figure 25 Python Packages Used

Testing the Flask app

Before we deploy the application, it is better to test the flask app locally. In order to do the test, we need to install POSTMAN. Follow the steps below.

1. Visit <https://www.postman.com/> and download the setup file based on your computer settings.
2. Once downloaded install the file
3. Open the POSTMAN app.
4. Execute the app.py file in PyCharm. This will give a local end point. (http://0.0.0.0:5000/). Note this end point
5. In POSTMAN fire the end point. Add /webhook with the above URL. The POST API will be running at that endpoint. PFB a screenshot. It is bit important to understand the payload of the webhook that we are sending to the endpoint.

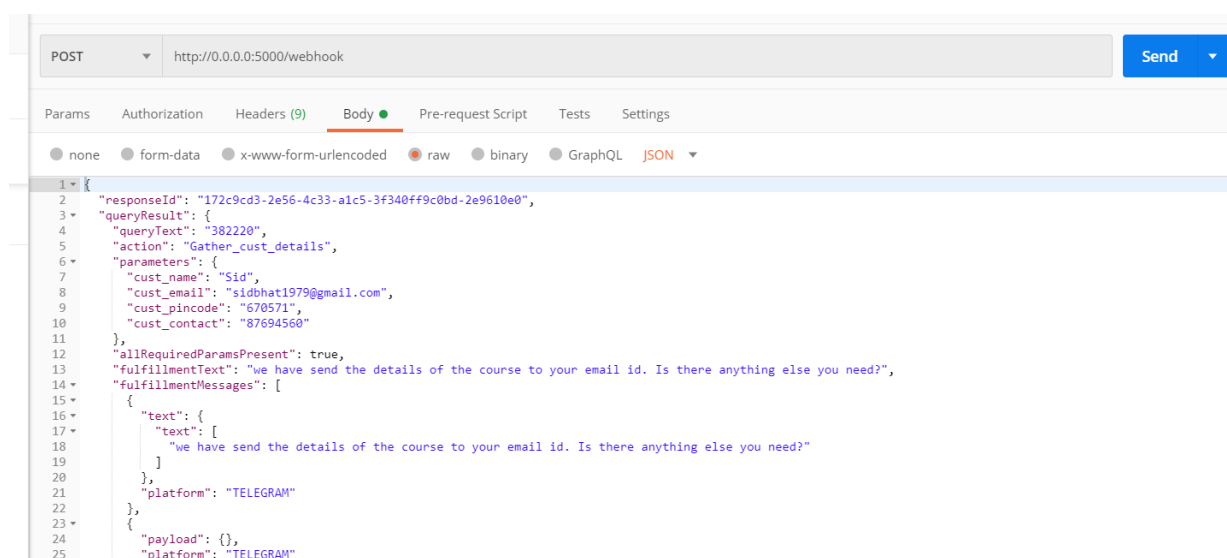


Figure 26 POSTMAN Request in local machine

Webhook Request and Response Structure

The below is the JSON structure. This structure you can get from the 'DIAGNOSTIC INFO' button once you execute a particular intent. Have highlighted the sections which are critical for request and for response. This is the structure as well to be used when we test in POSTMAN. The body of the request should have this structure. It is extremely critical that we adhere to the JSON structure that Dialogflow understands otherwise it will not work. Also note for each channel there is a section in the response.

```
{
  "responseId": "4731ee1f-c804-47c7-af6e-ef29e5ae3b27-2e9610e0", unique id of the session
  "queryResult": {
    "queryText": "500016",
    "action": "Gather_cust_details", - inputs provided by customer (you can check all the attributes are present)
    "parameters": {
      "cust_pincode": "500016",
      "cust_contact": "78956789",
      "cust_name": "corona",
      "cust_email": "sangeethayemisety369@gmail.com"
    },
    "allRequiredParamsPresent": true, - The entire portion in gray is the output. This is the output that our app has to generate. In Postman the value of the keys can be null while you send the request
    "fulfillmentText": "An email has been sent to your email id with the details. Do you have any further queries?",
    "fulfillmentMessages": [
      {
        "text": {
          "text": [
            "An email has been sent to your email id with the details. Do you have any further queries?"
          ]
        },
        "platform": "TELEGRAM"
      },
      {
        "payload": {},
        "platform": "TELEGRAM"
      }
    ]
  }
}
```



```
{  
  "image": {-- URL location in S3  
    "imageUri": "https://covid-19-news.s3-ap-northeast-  
1.amazonaws.com/report06bb10bee1324af1bdfc72a0120467cf.jpg"  
  },  
  "platform": "TELEGRAM"  
},  
  
{  
  "image": {- URL location in S3  
    "imageUri": " https://covid-19-news.s3-ap-northeast  
2.amazonaws.com/LocalReport/reportf04ba29c06db4a8389e40b5e2820e7fc.png"  
  },  
  "platform": "FACEBOOK"  
},  
  
{  
  "text": {  
    "text": [  
      "An email has been sent to your email id with the details. Do you have any further queries?"  
    ]  
  }  
},  
  
{  
  "text": {  
    "text": [  
      "Dummy Text"  
    ]  
  }  
},  
  
{  
  "payload": {}  
}
```

```
}
```

```
],
```

"intent": { - This is the portion where the intent is provided to the flask app. This intent need to be extracted.

"name": "projects/course-selection-cjabjl/agent/intents/e9812b1c-61eb-4cff-b4ba-1e64572636b5",

"displayName": "localcovidinfo"

```
},
```

"intentDetectionConfidence": 0.01,

"diagnosticInfo": {

"webhook_latency_ms": 3371

```
},
```

"languageCode": "en"

```
},
```

"webhookStatus": {

"message": "Webhook execution successful"

```
}
```

```
}
```

It is strongly recommended to follow Dialogflow documents for request & response structure. Refer to the following [link](#) for further details.

Deployment in Heroku

Heroku CLI Installation

Follow the following steps to create an account and installation of Heroku CLI.

- 1) Go to <https://id.heroku.com/login> and create an account
- 2) Download Heroku CLI from <https://devcenter.heroku.com/articles/heroku-cli>
- 3) Install Heroku CLI
- 4) Open the command prompt and fire **heroku login -i**

This will ask for credential details of the account. Enter the user id and passwd.
If login is successful it means Heroku cli is able to talk to Heroku server.

Deployment Specific Files

Earlier we mentioned some files which are required to be created for deployment. Let's discuss those files a bit more.

Procfile –

This file is to command Heroku what it needs to execute once the deployment is done. For our project web indicates that Heroku needs to kick start a web server process and fire the command `python app.py`.

```
web: python app.py
```

requirements.txt -

This file contains the list of python packages that are required to be installed as part of the deployment process. Open a python terminal in PyCharm. Move to the root folder of the project. Fire the command `pip freeze > requirements.txt`. This file is very critical as all the packages has to be installed in Heroku cloud for the flask app to run.

```
boto3==1.12.41
botocore==1.15.41
certifi==2020.4.5.1
chardet==3.0.4
click==7.1.1
cyclr==0.10.0
docutils==0.15.2
Flask==1.1.2
Flask-Cors==3.0.8
idna==2.9
itsdangerous==1.1.0
Jinja2==2.11.2
jmespath==0.9.5
kiwisolver==1.2.0
MarkupSafe==1.1.1
matplotlib==3.2.1
numpy==1.18.2
pandas==1.0.3
patsy==0.5.1
plotly==4.6.0
plotly-express==0.4.1
pyparsing==2.4.7
python-dateutil==2.8.1
pytz==2019.3
requests==2.23.0
retrying==1.3.3
s3transfer==0.3.3
scipy==1.4.1
six==1.14.0
statsmodels==0.11.1
urllib3==1.25.9
```

```
Werkzeug==1.0.1  
wincertstore==0.2
```

runtime.txt

This file just mentions the version of python to be installed.

```
python-3.7.5
```

Executing GIT

You need to install GIT in local machine before executing the below steps

1. Open a command prompt in your local machine
2. Change directory to the top folder of your project
3. Run **heroku login** and provide credentials to login
4. Run **heroku create <appname>** eg. Covid19botsid
5. Run **git init** to initialize an empty git repository
6. Use **git add .** to add all the files to the local git repository.
7. Use **git commit -m "my first commit"** to commit the code to the local git repo.
8. Run **heroku git:remote -a <appname>** (tying the local git repo with the server git repo)
9. Push the code to the remote repo using **git push heroku master**

The last step will finish the deployment. At the end there will be an URL. For my project the URL is - <https://covidchatbotot.herokuapp.com>. Configure the URL in the URL field in the fulfillment section of Dialogflow. Please ensure that you add **'/webhook'** at the end of the URL.

Channel Integration

The last part of the project is to setup the channel integration of the BOT with Facebook messenger and Telegram. Post that we will test our BOT and check the results in those channels.

Integration with Telegram

- 1) Login to Telegram web. Search for **BotFather**. This helps developers to register a new bot in Telegram application. Provide the command /newbot. Post that it will ask name & username of the BOT. Username must end with **bot**. When it is done it will provide a token. This will be utilized in step 2. A snapshot of Telegram is provided below.

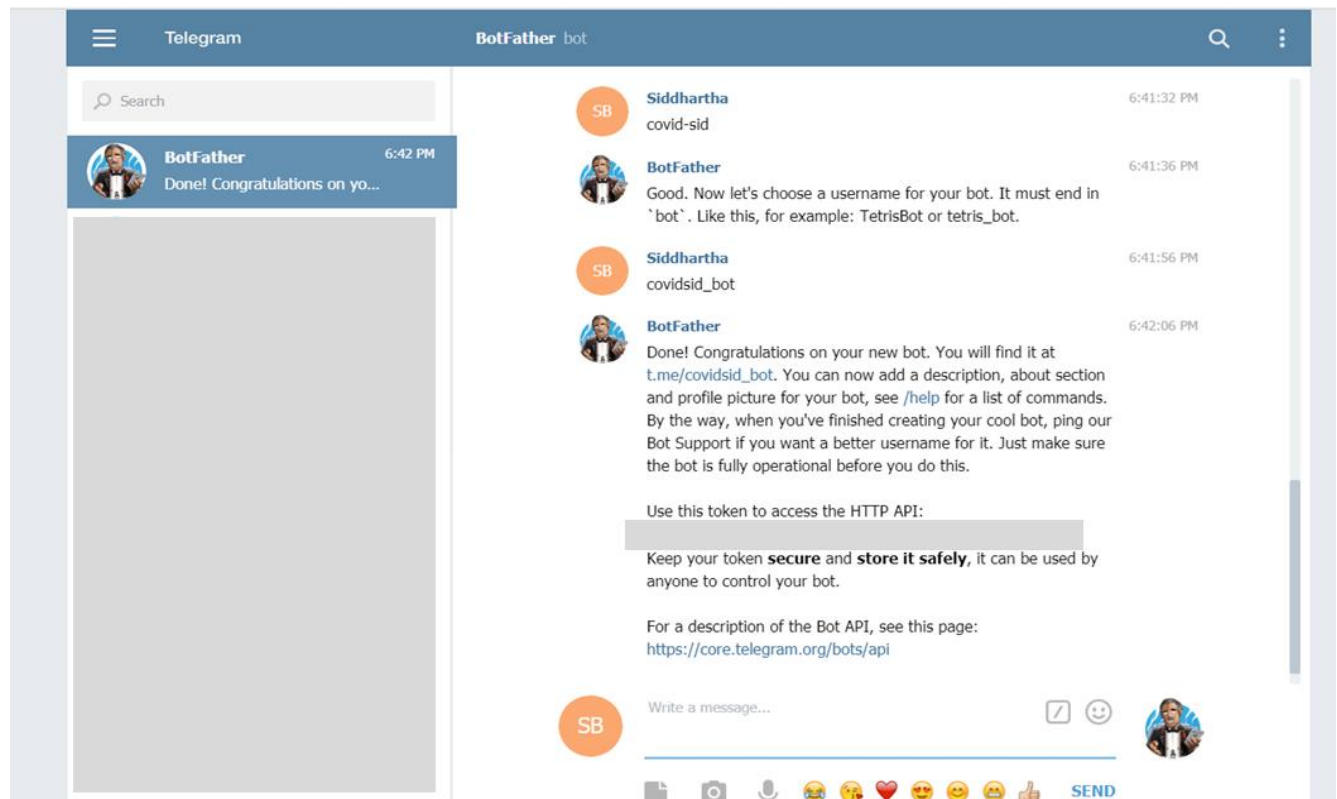


Figure 27 Telegram Setup

- 2) Note the token. Login to Dialogflow and click Integrations. Then go to the telegram icon. The below screen will pop-up. Input the token. Integration with Telegram is done.

Figure 28 Telegram setting in Dialogflow

Integration with Facebook Messenger

While the integration with Telegram is super easy, Facebook is bit complex.

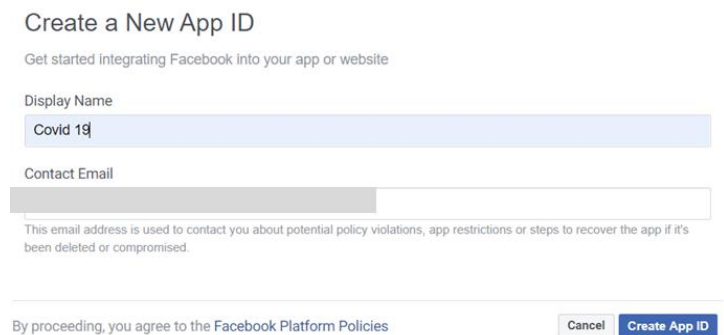
- 1) Create a Facebook page. It is better to create a new one even if you already have a Facebook page.
- 2) Click on Pages link in your facebook profile. Fill up the required details and continue to finish the creation.

Figure 30 Add a Button

Figure 29 Facebook Create Page

- 3) In this new page click on the 'Add a button'. The following page will open. Select send message option and finish the flow.

- 4) Next login to <https://developers.facebook.com/>.
- 5) Click on create a new app.
- 6) Once the app is created click '**Add a product**'. Choose Messenger.
- 7) Then it will ask for the page to link this app. Choose the page created in step 2.



Create a New App ID

Get started integrating Facebook into your app or website

Display Name
Covid 19

Contact Email

This email address is used to contact you about potential policy violations, app restrictions or steps to recover the app if it's been deleted or compromised.

By proceeding, you agree to the Facebook Platform Policies

Cancel Create App ID

Figure 31 Create New Facebook APP

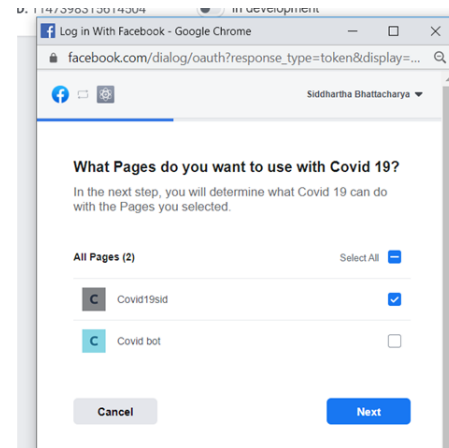
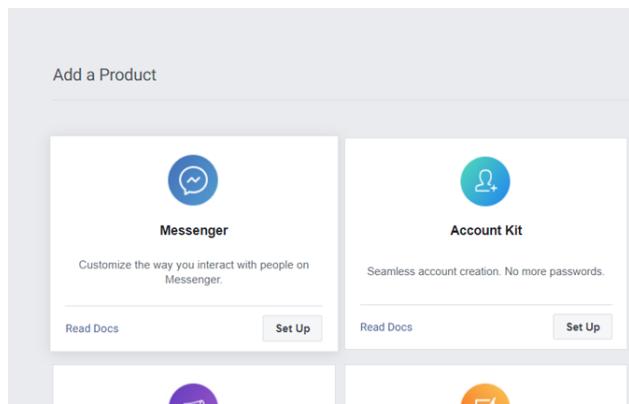


Figure 32 Add product and Page with Facebook app

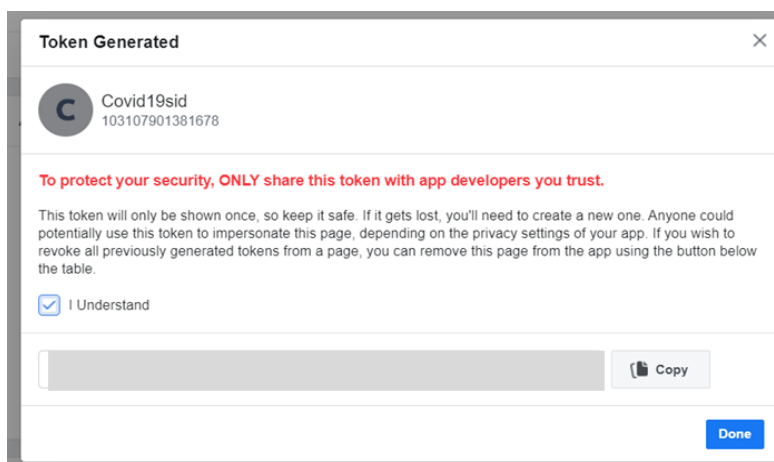


Figure 33 Generate Token in Facebook App

- 8) In the next page under the section "**Access Tokens**", click **generate token**. This token will be required to be fed in Dialogflow.

- 9) Next, we need to set up the key in google Dialogflow. Click on integration, go to messenger. Page Access Token = the token from step 8. Note down the callback URL and provide a string in verify token.

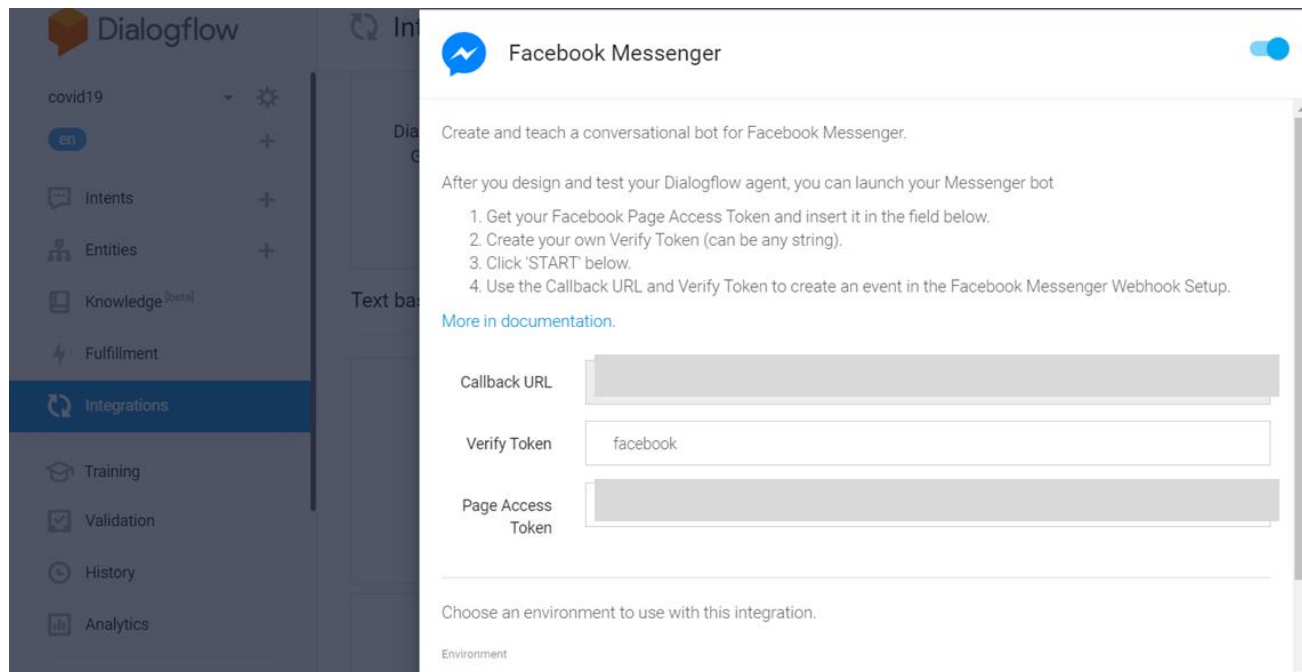


Figure 34 Setting Token in Dialogflow

- 10) Next go to facebook developer and click the “**Add Callback URL**” under webhook section. Add the callback URL and verify token from step 9. Once we click verify & save, this will connect the messenger with Dialogflow successfully.

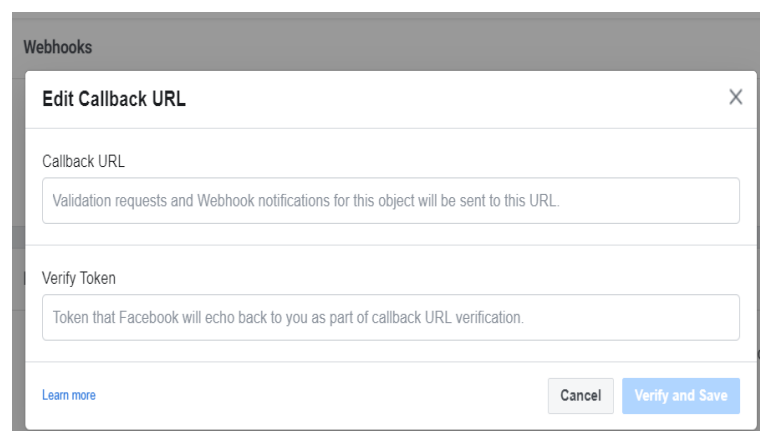


Figure 35 Webhook setting in Facebook Developer

- 11) Finally click ‘**Add subscriptions**’ against the app, and select ‘messages’ and ‘messaging_postbacks’.

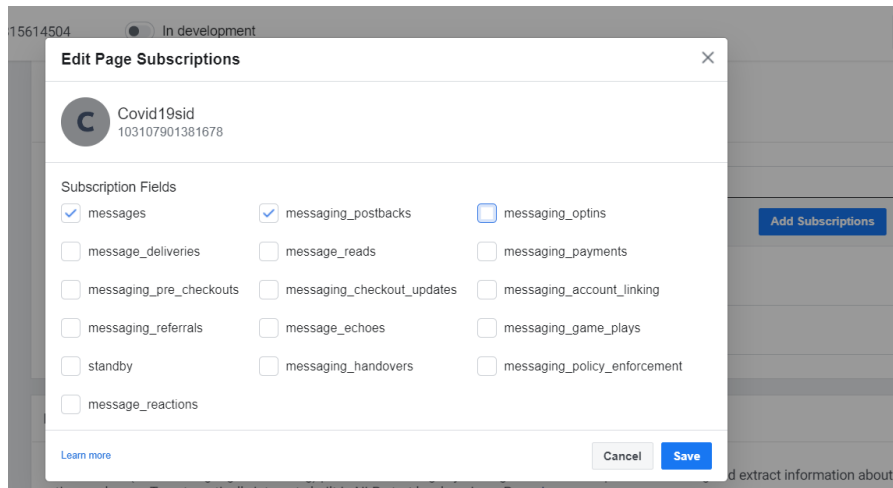


Figure 36 Add Subscriptions for Facebook app

Results

Now as we are done with the integration part let's check out how the BOT is behaving based on user inputs. PFB a series of screenshots for both the channels. I will discuss some of the challenges encountered during this testing and what tweaks have been done.

Facebook Screen Shots

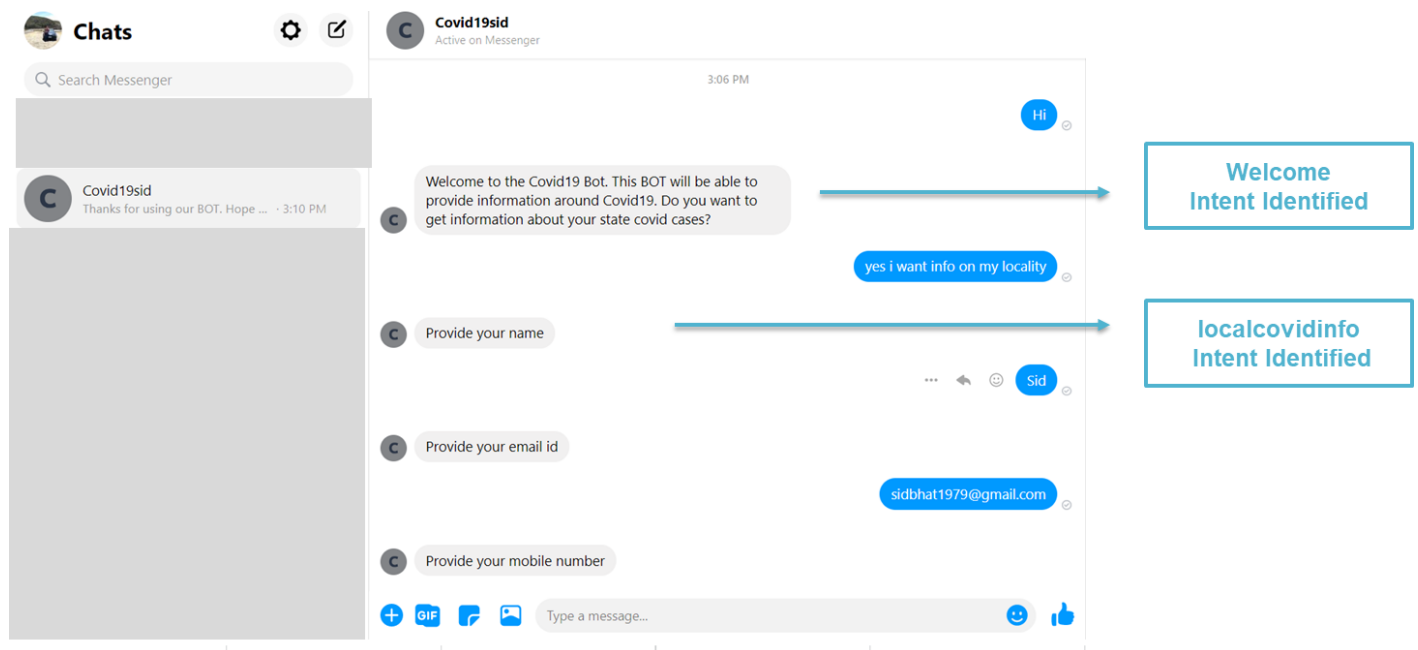


Figure 37 Facebook Screen 1

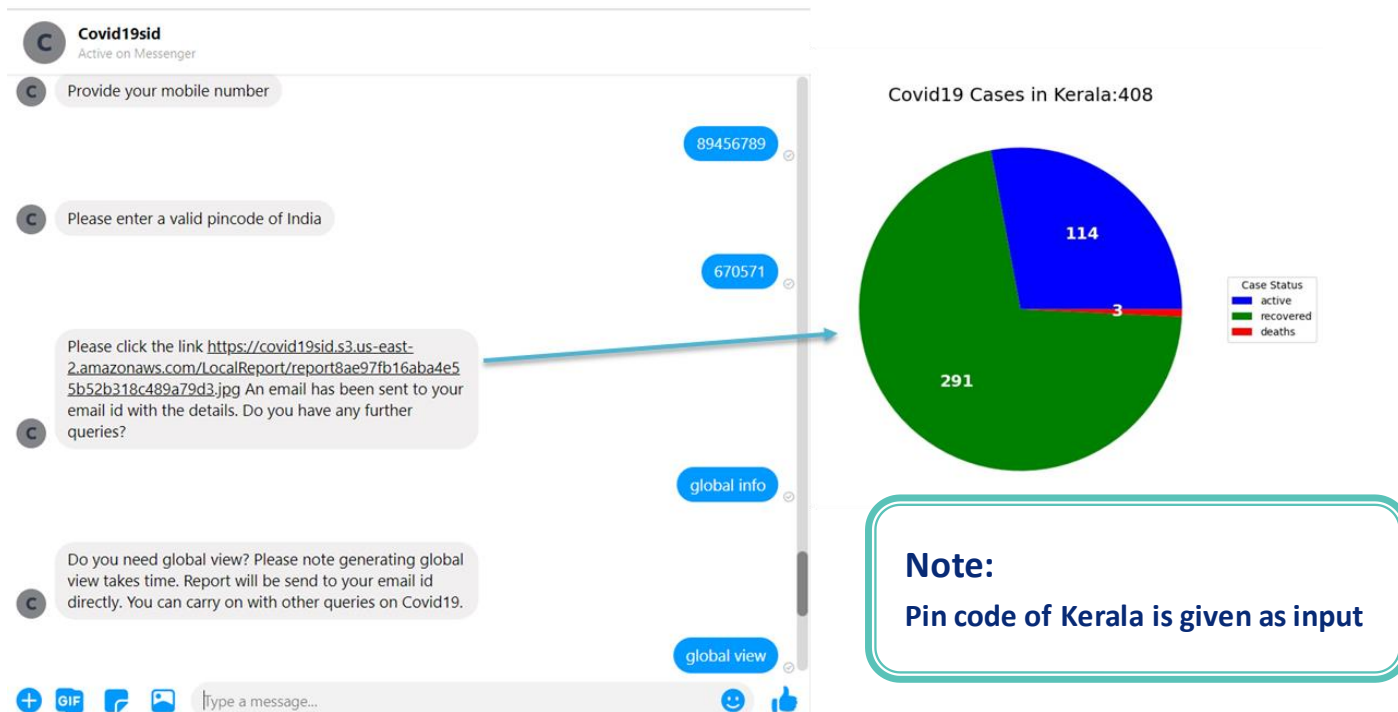


Figure 38 Facebook Screen 2

Challenges encountered & Potential scope of future work–

- 1) As per Dialogflow docs, if we pass image url (publicly available) in the webhook response, then messenger can display the image. In spite of sending such an URL, it didn't work. So, need to send the URL link as a message. One can click the link to view the image as shown. The URL is Amazon S3 bucket link.
- 2) Dialogflow webhook call gets disconnected if the response is not within 5 secs. For the global view, in my case flask app was taking more than 5 secs. Hence the webhook call was failing. So, in order to mitigate the same, providing the info, that it will be send to email only. There is no option to increase the response wait time in Dialogflow.

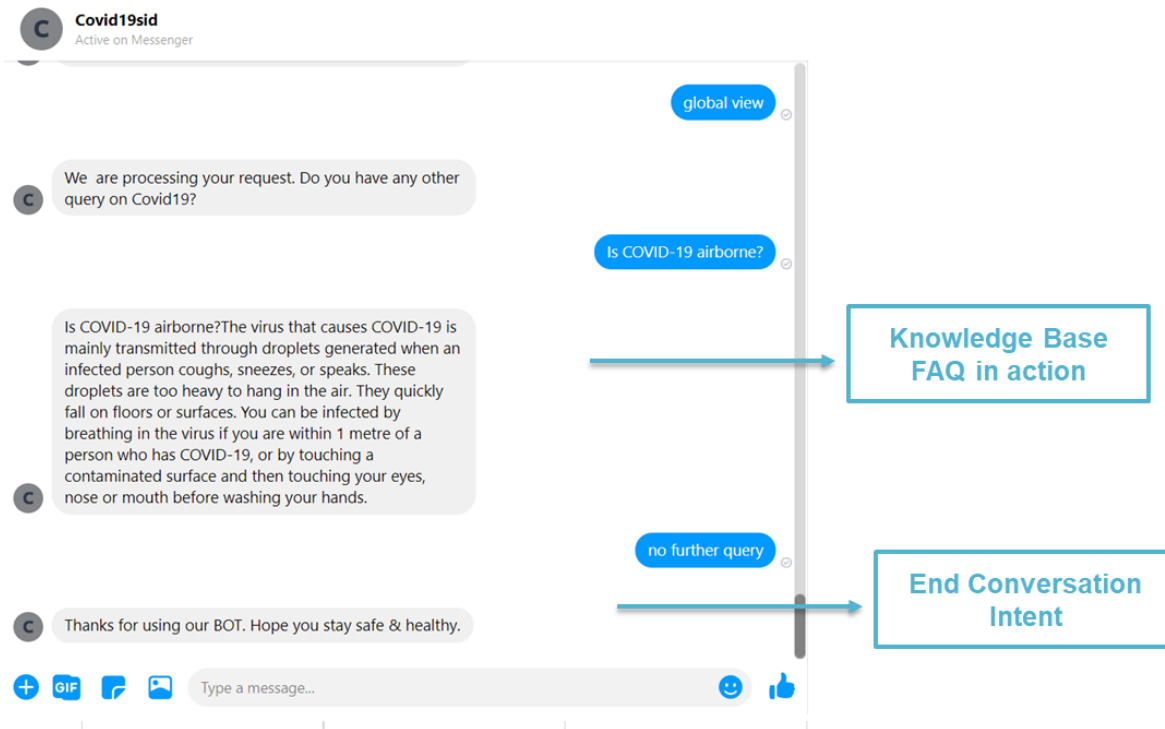


Figure 39 Figure 36 Facebook Screen 3

Telegram Screen Shots

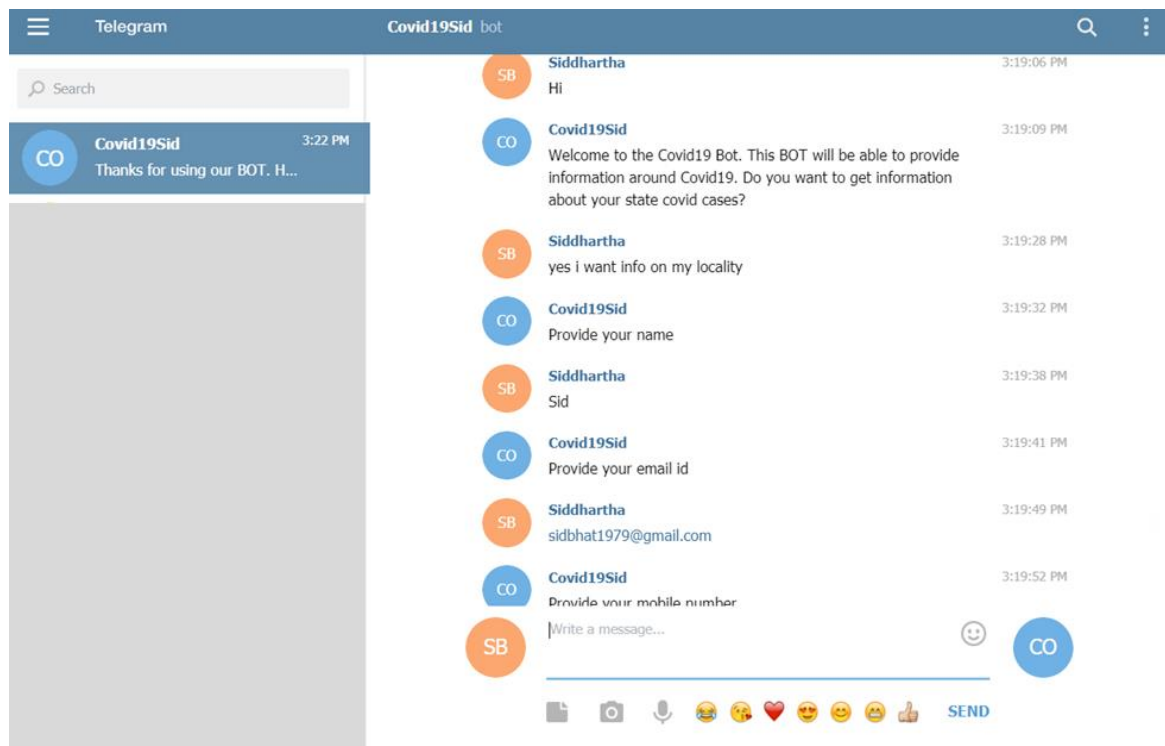
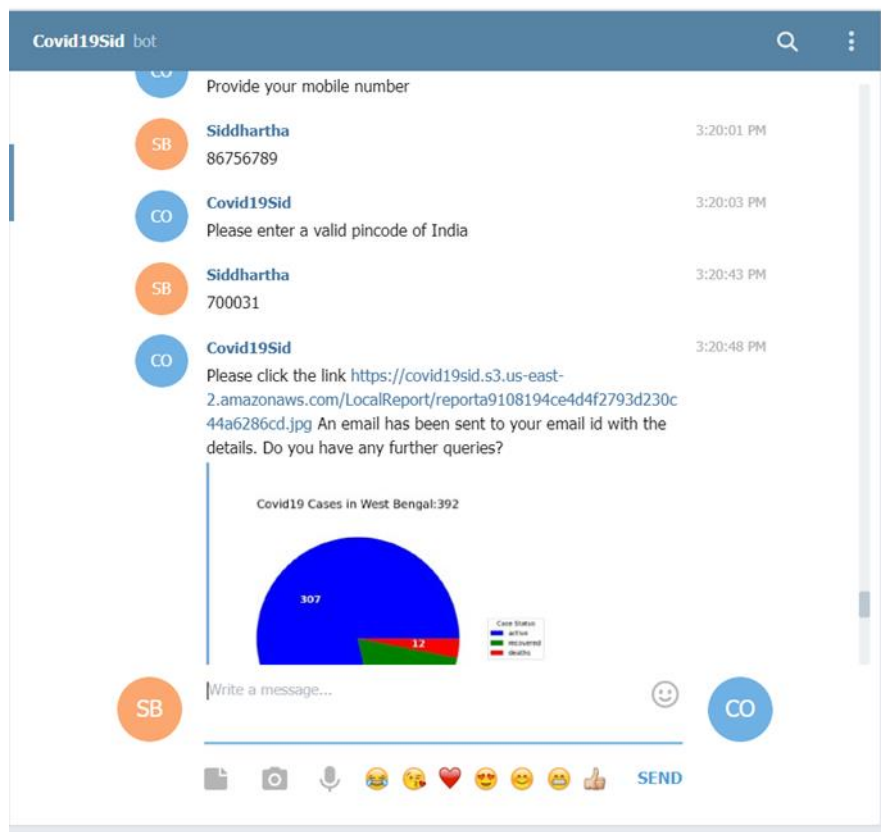


Figure 40 Telegram Screen 1



Note: Telegram showed the image in the screen. So, challenge 1 is not present.

Note: Pincode given of West Bengal

Figure 41 Telegram Screen 2

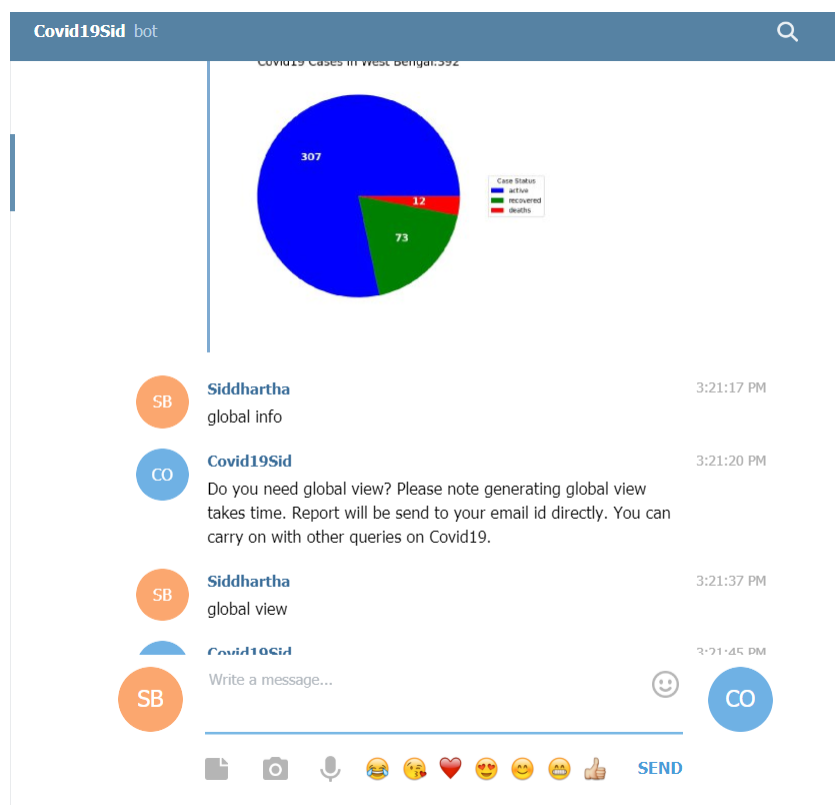


Figure 42 Telegram Screen 3

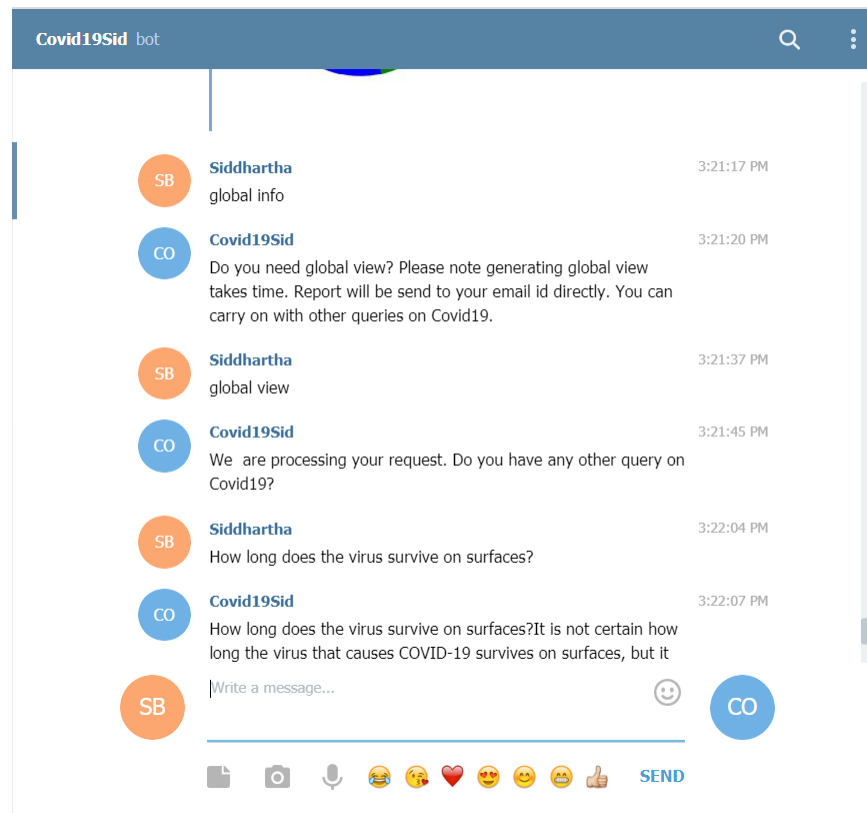


Figure 43 Telegram Screen 4

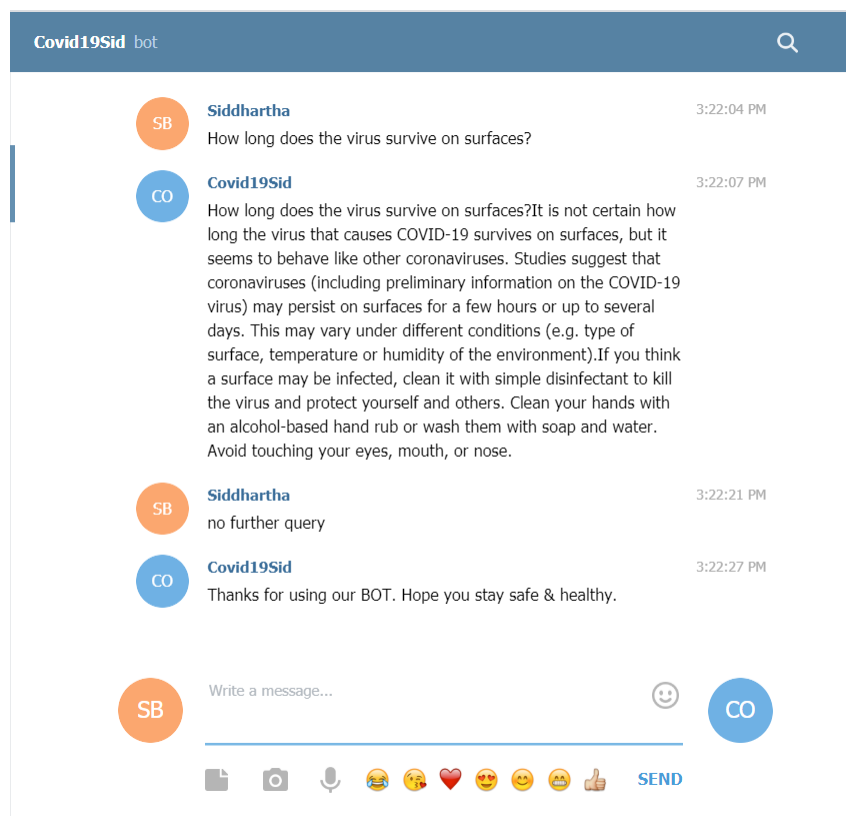


Figure 44 Telegram Screen 5

Email Screen Shots



Figure 45 Email Screen Shot 1

Note:

To send email from Gmail id through python app please do the following –

- Disable 2 factor authentication of your email settings
- Turn on Less secure app access

World View of Covid19 cases

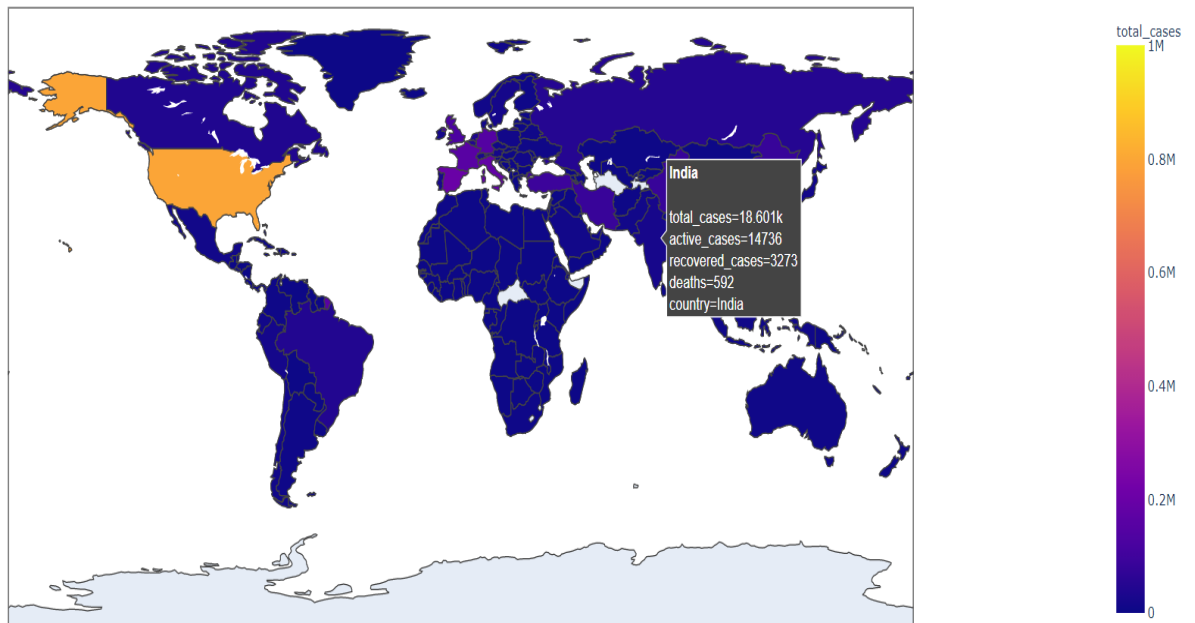


Figure 46 Global View from email attachment (plotly interactive graph)

Wrapping it up

Finally, to end let's have a look at the key steps and critical points to consider to build the BOT & deploy. Hope you have enjoyed reading and got some directions to build your own BOT. Signing off.

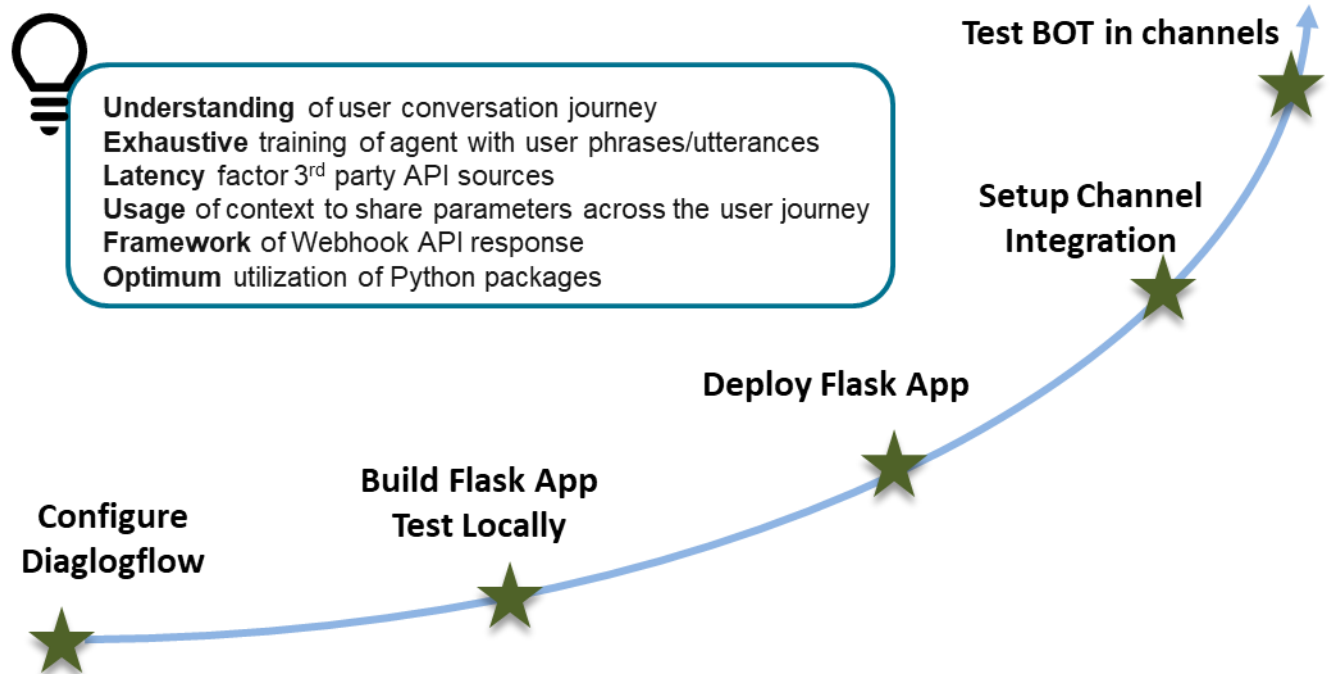
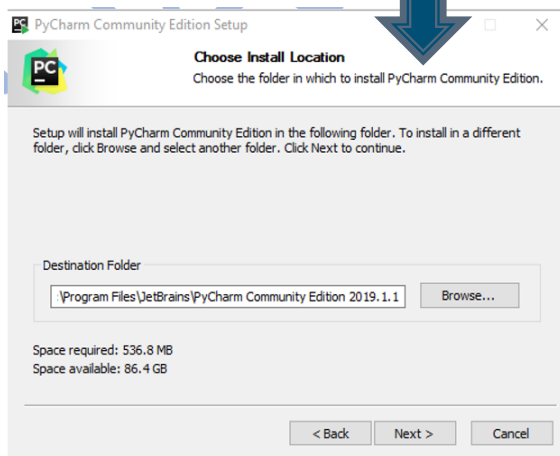


Figure 47 Key Milestones of the Project

Appendix

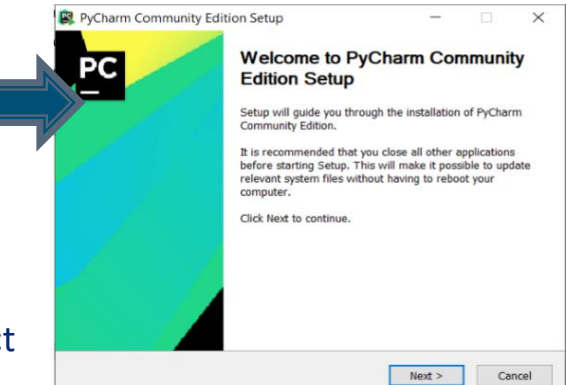
Quick steps to install PyCharm.

1. Go to the [link](#)
2. Double click on the installation file to start the installation process and click next to continue.
3. Select the directory to install PyCharm and then click next

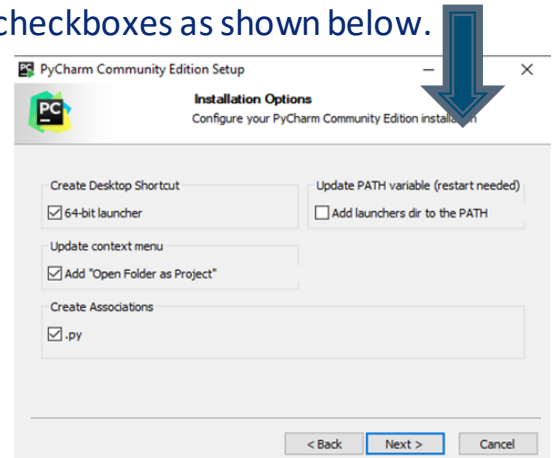


4. Select the appropriate checkboxes as shown below.

Don't miss to check the option **"Open Folder as Project"**



5. Choose the name of the start menu folder and then click on install to finish the installation.



References

<https://dialogflow.com/docs>

<https://cloud.google.com/dialogflow/docs>

<https://realpython.com/python-boto3-aws-s3/>

<https://www.youtube.com/watch?v=5XaU0DCMuL4>

<https://medium.com/swlh/how-to-build-a-chatbot-with-dialog-flow-chapter-1-introduction-ab880c3428b5>