

DOCUMENTATION ON GOOGLE DIALOGFLOW

CHATBOT of Covid-19

1. Chatbot Fundamentals

1.1 What is a chatbot?

A chatbot is a software application that mimics conversation with a human in natural languages through various platforms like messaging, websites, mobiles etc. The chatbot responds by identifying the intent of the conversation and then responding accordingly.

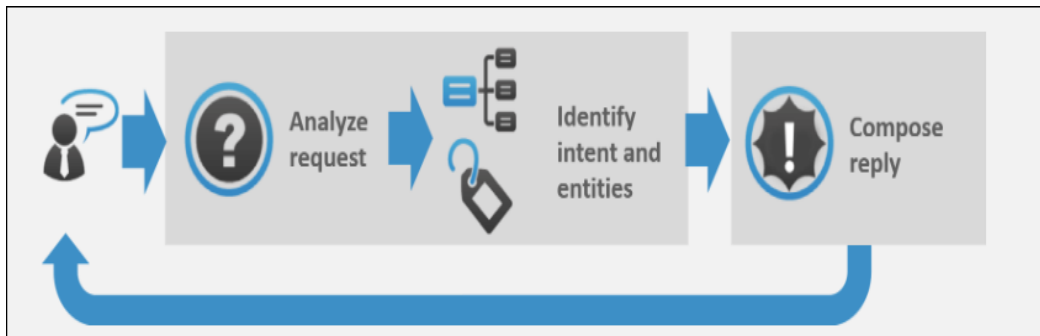


Image Courtesy: <https://expertsystem.com/chatbot/>

Broadly, chatbots can be classified into two categories:

1. Rule-Based Chatbots: This is an extremely fundamental type of chatbot which generally works on simple 'if-else' constructs. It can respond to only simple predefined queries. The performance of this application highly depends on the programming skills of the developer.
2. Chatbots with Natural Language Understanding: At the core, it has a language processing and understanding model with pre-trained instances using Deep Learning. It can communicate through both text and speech.

1.2 Uses of chatbots:

- a) Can be used to answer FAQs.
- b) Can be used for grievance handling.
- c) Internal organizational automation.
- d) To do the flight, hotel, appointment bookings etc.
- e) Can guide customers to buy the correct product by answering their questions
- f) Can be used for Customer Relationship Management.

1.3 Advantages of Chatbots

- a) 24*7 customer support.
- b) Uniform customer experience.
- c) Cost-efficient.
- d) Build once and deploy everywhere.

- e) Integration with various channels and platforms
- f) Better monitoring and insight generation.

1.4 Frameworks Present in the Market:

- a) **Google Dialogflow:** Dialogflow is a natural language understanding platform that makes it easy to design and integrate a conversational user interface into your mobile app, web application, device, bot, interactive voice response system, and so on.
- b) **Microsoft Bot Builder with LUIS:** Azure Bot Service enables you to build intelligent, enterprise-grade bots with ownership and control of your data.
- c) **Amazon Lex:** Amazon Lex is a service for building conversational interfaces into any application using voice and text. Amazon Lex provides the advanced deep learning functionalities of automatic speech recognition (ASR) for converting speech to text, and natural language understanding (NLU) to recognize the intent of the text, to enable you to build applications with highly engaging user experiences and lifelike conversational interactions.
- d) **RASA:** Rasa provides infrastructure & tools necessary for high-performing, resilient, proprietary contextual assistants that work.
- e) **Wit.ai (Facebook):** Wit.ai makes it easy for developers to build applications and devices that you can talk or text to.

2. Google Dialogflow

2.1 Introduction:

Dialogflow is a natural language understanding platform that makes it easy to design and integrate a conversational user interface into your mobile app, web application, device, bot, interactive voice response system, and so on. Using Dialogflow, you can provide new and engaging ways for users to interact with your product.

Dialogflow can analyze multiple types of input from your customers, including text or audio inputs (like from a phone or voice recording). It can also respond to your customers in a couple of ways, either through text or with synthetic speech.

2.2 Signup For Dialogflow:

You need to have a google account to signup for Dialogflow.

a) Go to <https://dialogflow.com/> and click on 'Sign Up for Free' button.

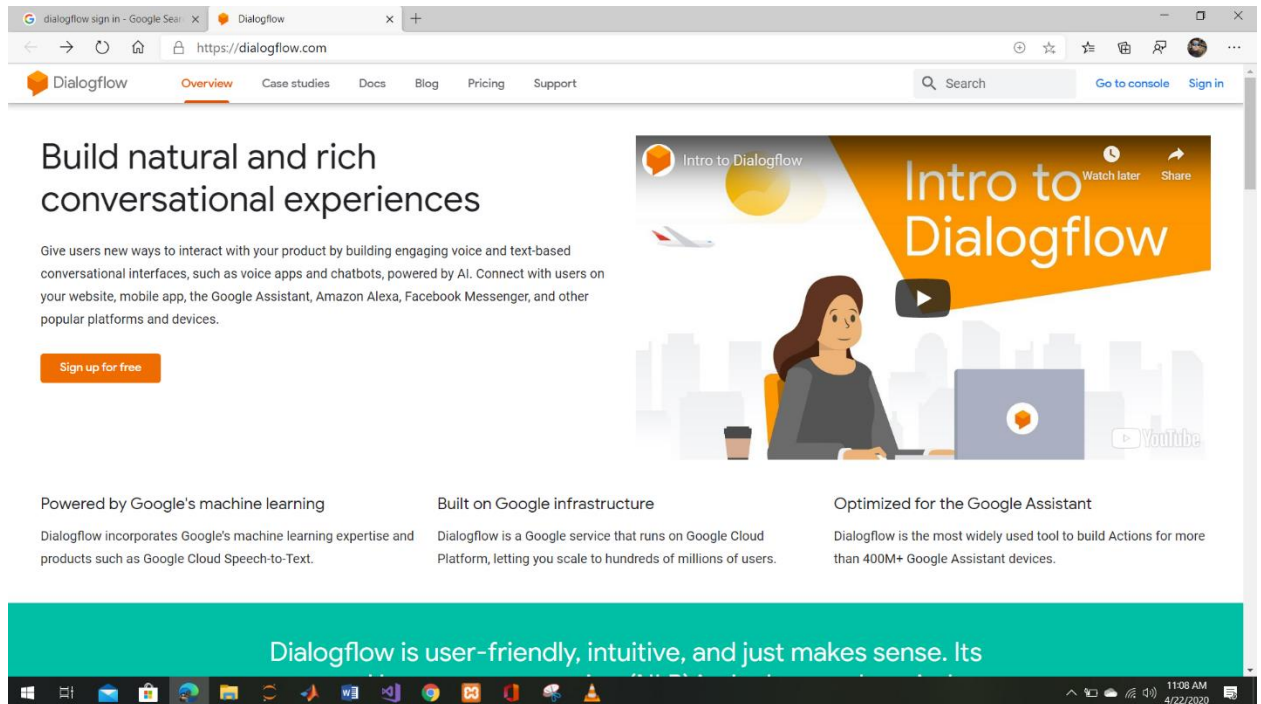


Figure 1: Sign up for Google Dialogflow

b) Click 'sign-in with Google'.

c) Select your google account and once you are redirected, click on 'Go To Console' on the upper right corner of the screen.

2.3 Dialogflow Console:

Dialogflow provides a web user interface called the *Dialogflow Console* ([open console](#)). You use this console to create, build, and test agents. The uses are:

- Create Agents
- Create Intents
- Create entities
- Fulfillment to connect to other APIs.
- Integrate the bot with other platforms
- Analyze agent performance
- Test the agent in conversation simulator.

3. Building a chatbot using google Dialogflow

3.1 The problem statement:

To build a chatbot which can answer all the queries of a customer and whenever a customer does an enquiry, it automatically send the customer the course details. Also an email is sent to the support team to assist the customer further with their queries.

3.2 Agent:

A Dialogflow *agent* is a virtual agent that handles conversations with your end-users. It is a natural language understanding module that understands the nuances of human language. A Dialogflow agent is similar to a human call center agent. You train them both to handle expected conversation scenarios, and your training does not need to be overly explicit.

3.2.1 Creating an Agent:

- a) Click on *Create Agent* from the left menu.
- b) Provide the name of the agent and click on the *SAVE* button to create the Agent.

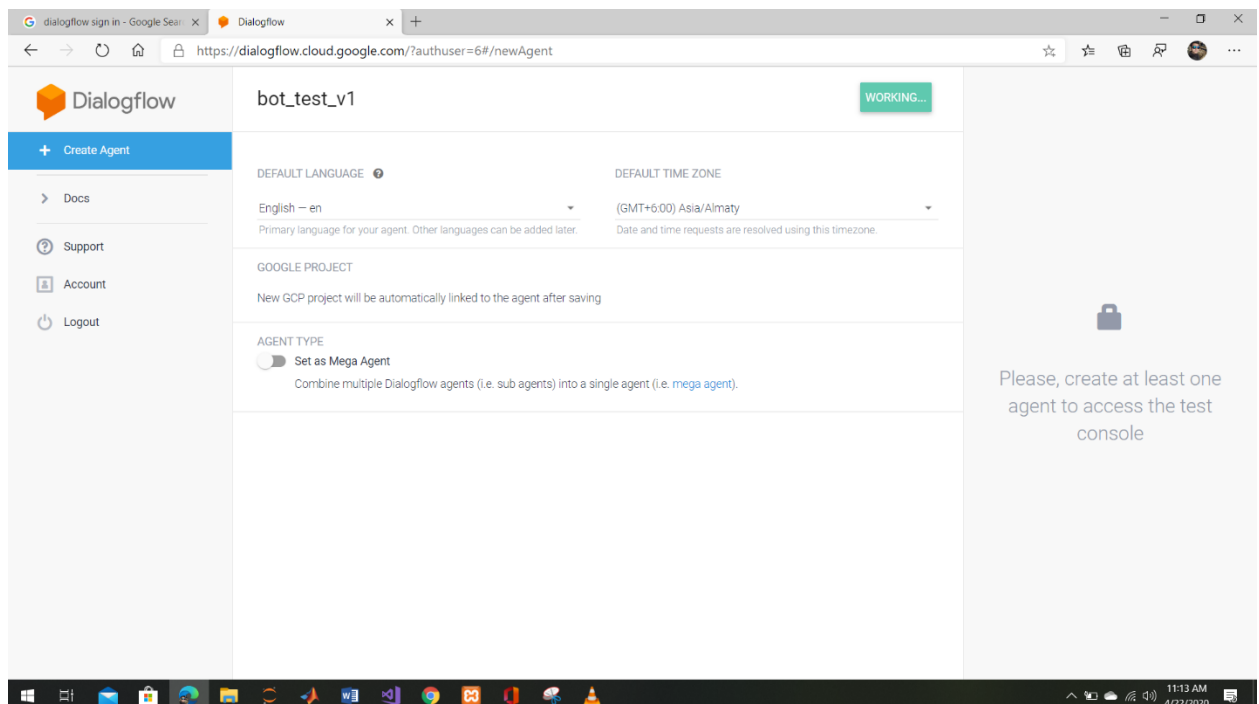


Figure 2: Creating an Agent for our ChatBot.

- c) After saving, the agent is shown in the left hand side of your console. You can click the gear icon to edit the agent settings.

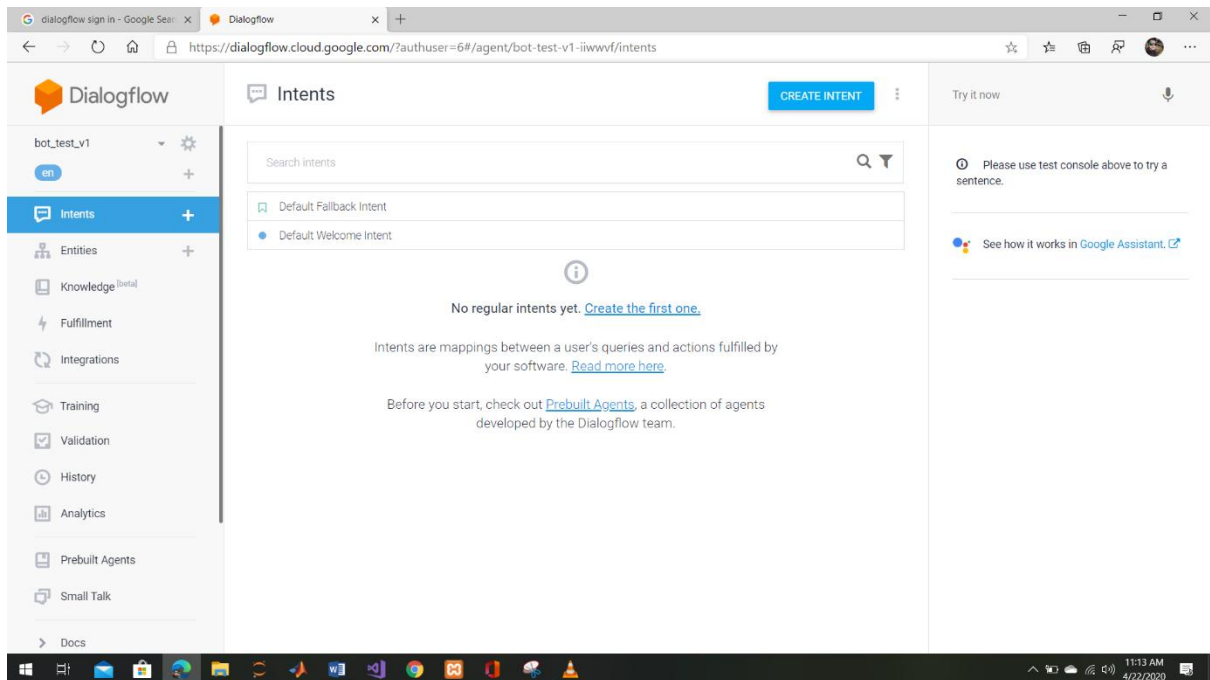


Figure 3: Gear icon to edit the Agent Settings.

3.3 Intent:

An *intent* categorizes an end-user's intention for one conversation turn. For each agent, you define many intents, where your combined intents can handle a complete conversation. When an end-user writes or says something, referred to as an *end-user expression*, Dialogflow matches the end-user expression to the best intent in your agent. Matching an intent is also known as *intent classification*.

The following diagram shows the basic flow for intent matching and responding to the end-user:

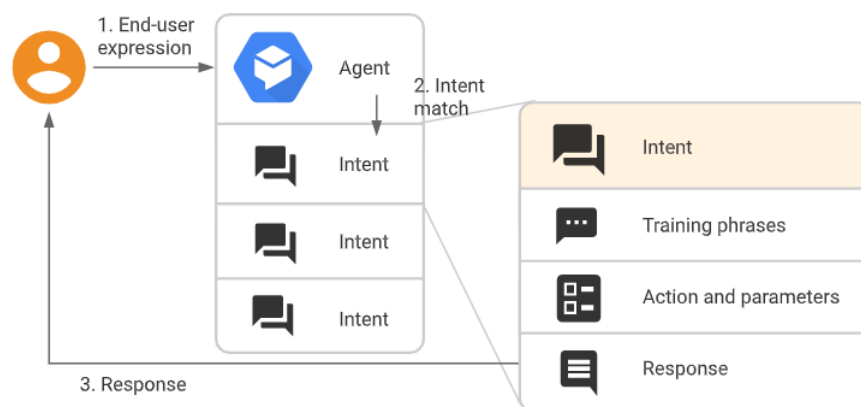


Figure 4: Flow-Chart of Response

3.3.1 Creating an Intent:

- a) Click the + add button next to Intents in the left sidebar menu. Enter a name for your intent. Your intent name should represent the end-user expressions it recognizes and click Save.

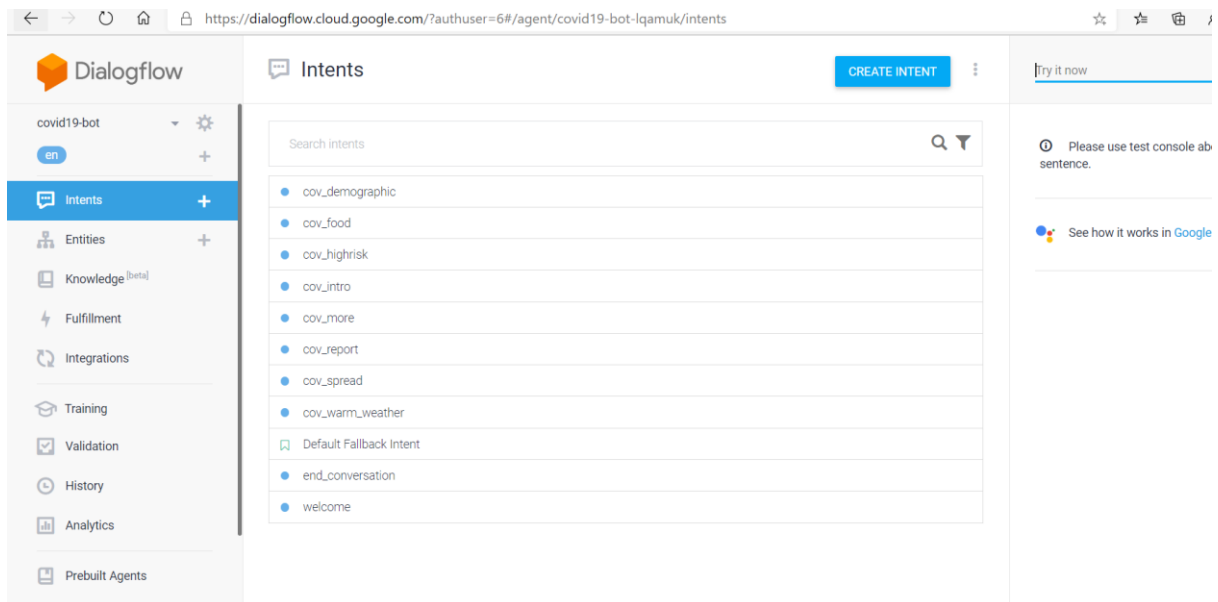


Figure 5: Add button to create an Intent.

3.3.1.1 Training Phrases:

Training phrases are example phrases for what end-users might type or say, referred to as *end-user expressions*. For each intent, you create many training phrases. When an end-user expression resembles one of these phrases, Dialogflow matches the intent.

Adding the training Phrases:

- a) Click the text field that shows "Add user expression".
- b) Type your training phrases and press the Enter key after each.

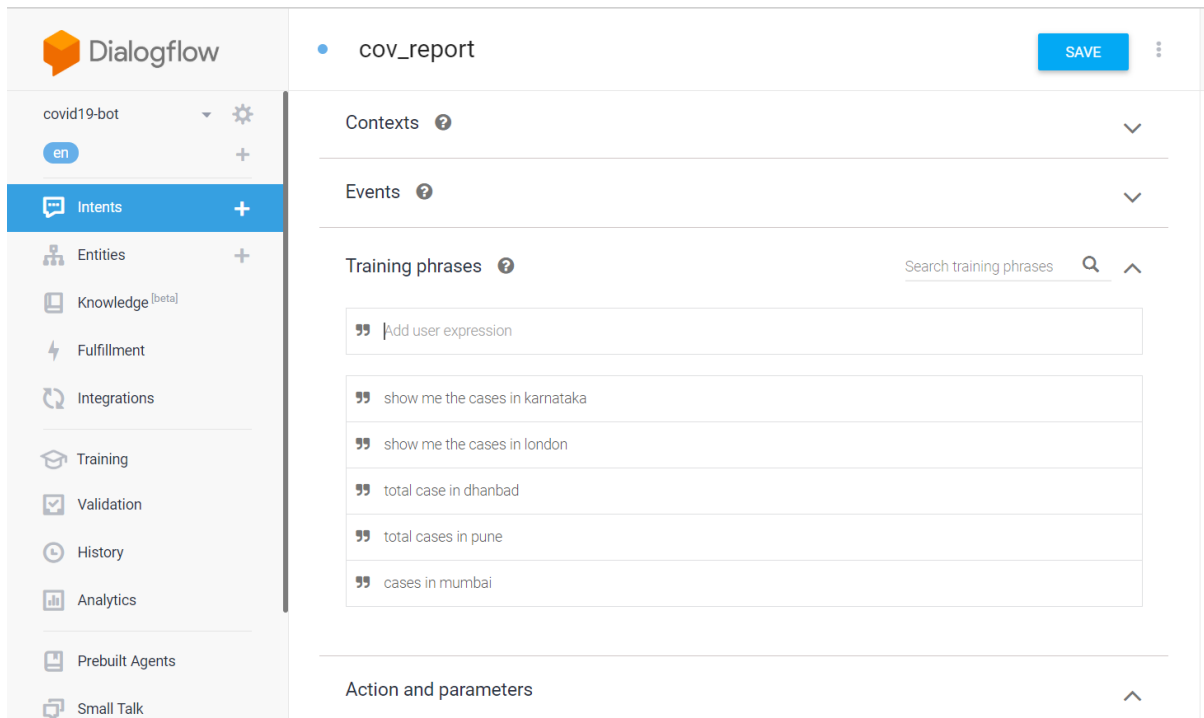


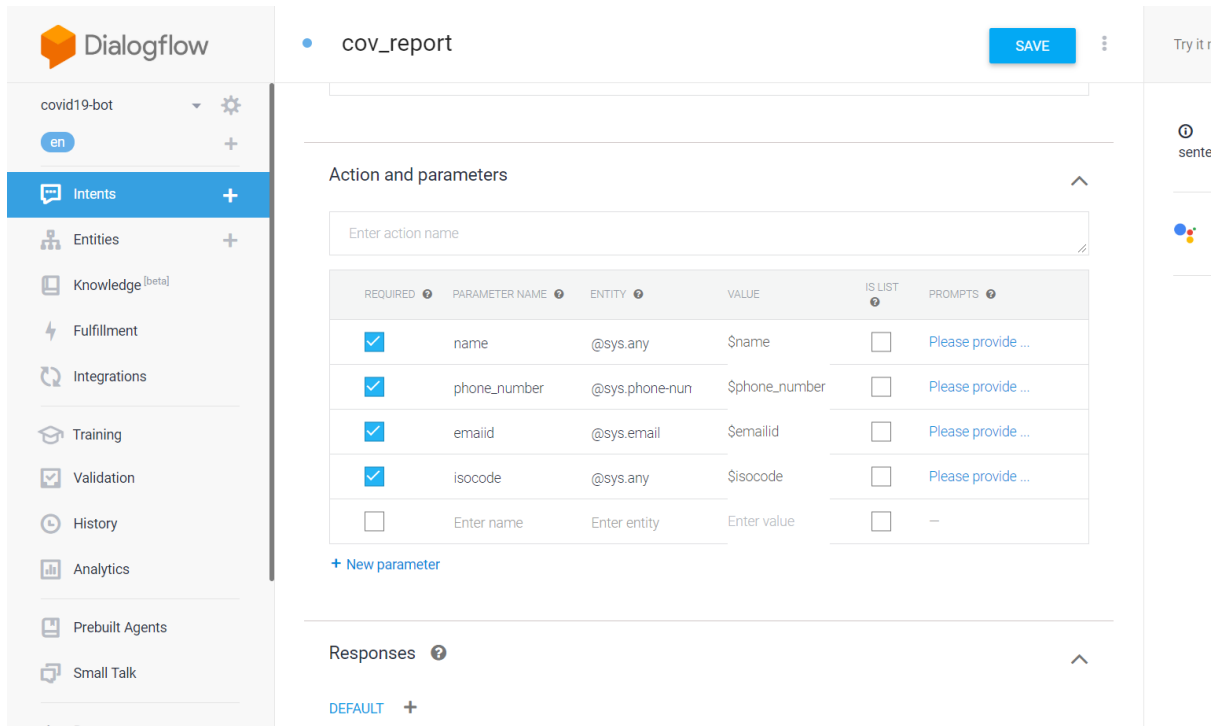
Figure 6: Listing out the training phases for the model.

3.3.1.2 Extracting the Entities:

When an intent is matched at runtime, Dialogflow provides the extracted values from the end-user expression as *parameters*. Each parameter has a type, called the *entity type*, which dictates exactly how the data is extracted. Unlike raw end-user input, parameters are structured data that can easily be used to perform some logic or generate responses.

Each intent parameter has a type, called the *entity type*, which dictates exactly how data from an end-user expression is extracted.

Dialogflow provides predefined system entities that can match many common types of data. For example, there are system entities for matching dates, times, colors, email addresses, and so on. You can also create your own custom entities for matching custom data.



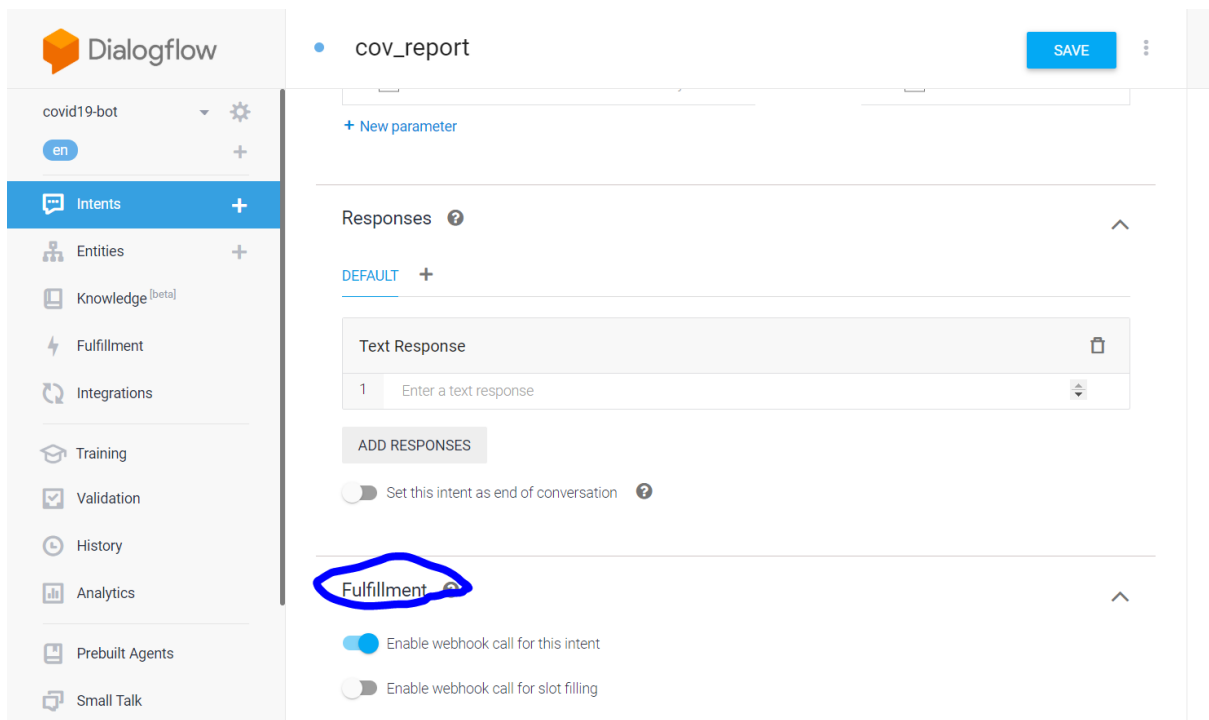
The screenshot shows the Dialogflow console interface. On the left is a sidebar with navigation options: Intents, Entities, Knowledge, Fulfillment, Integrations, Training, Validation, History, Analytics, Prebuilt Agents, and Small Talk. The main area is titled 'cov_report' and has a 'SAVE' button. The 'Action and parameters' section is expanded, showing a table of parameters. The 'Responses' section is also visible.

REQUIRED	PARAMETER NAME	ENTITY	VALUE	IS LIST	PROMPTS
<input checked="" type="checkbox"/>	name	@sys.any	\$name	<input type="checkbox"/>	Please provide ...
<input checked="" type="checkbox"/>	phone_number	@sys.phone-num	\$phone_number	<input type="checkbox"/>	Please provide ...
<input checked="" type="checkbox"/>	emailid	@sys.email	\$emailid	<input type="checkbox"/>	Please provide ...
<input checked="" type="checkbox"/>	isocode	@sys.any	\$isocode	<input type="checkbox"/>	Please provide ...
<input type="checkbox"/>	Enter name	Enter entity	Enter value	<input type="checkbox"/>	—

Below the table is a '+ New parameter' link. The 'Responses' section is also visible, showing a 'DEFAULT' response.

Figure 7: Specifying the Action and Parameters in Intents Tab.

3.3.1.3 Specifying custom Responses:



The screenshot shows the Dialogflow console interface. On the left is a sidebar with navigation options: Intents, Entities, Knowledge, Fulfillment, Integrations, Training, Validation, History, Analytics, Prebuilt Agents, and Small Talk. The main area is titled 'cov_report' and has a 'SAVE' button. The 'Responses' section is expanded, showing a 'Text Response' and a 'Fulfillment' section.

The 'Text Response' section shows a list of responses. The first response is 'Enter a text response'. Below the list is an 'ADD RESPONSES' button.

The 'Fulfillment' section is circled in blue. It contains two toggle switches: 'Enable webhook call for this intent' (which is turned on) and 'Enable webhook call for slot filling' (which is turned off).

Figure 8: Providing the responses.

Intents have a built-in response handler that can return responses after the intent is matched. This feature only supports static responses, though you can use parameter references in these responses to make them somewhat dynamic. This is helpful for recapping information provided by the end-user. For example, your intent response could look like: "Okay, I booked a room for you on date".

3.4 Fulfillment:

By default, your agent responds to a matched intent with a static response. If you're using one of the integration options, you can provide a more dynamic response by using *fulfillment*. When you enable fulfillment for an intent, Dialogflow responds to that intent by calling a service that you define. For example, if an end-user wants to schedule a haircut on Friday, your service can check your database and respond to the end-user with availability information for Friday.

Each intent has a setting to enable fulfillment. If an intent requires some action by your system or a dynamic response, you should enable fulfillment for the intent. If an intent without fulfillment enabled is matched, Dialogflow uses the static response you defined for the intent.

When an intent with fulfillment enabled is matched, Dialogflow sends a request to your *webhook* service with information about the matched intent. Your system can perform any required actions and respond to Dialogflow with information for how to proceed. The following diagram shows the processing flow for fulfillment.

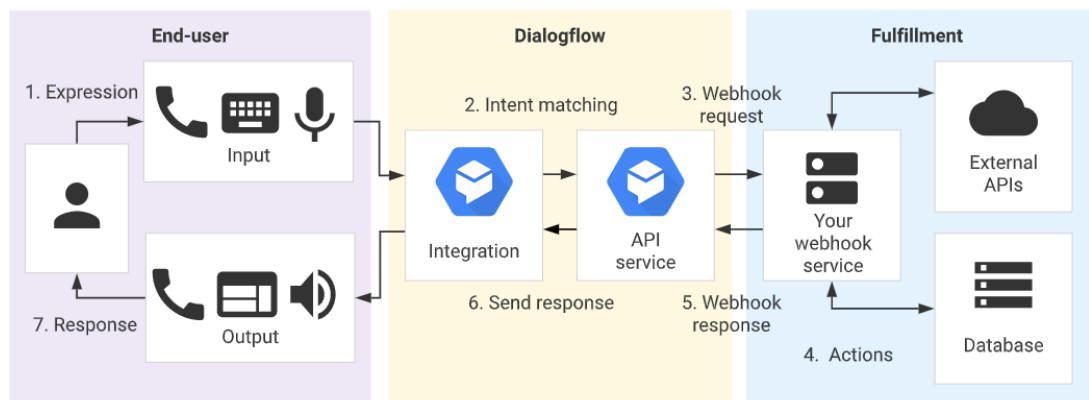


Figure 9: Overall Flow graph for operation in Google DialogFlow.

- The end-user types or speaks an expression.
- Dialogflow matches the end-user expression to an intent and extracts parameters.
- Dialogflow sends a webhook request message to your webhook service. This message contains information about the matched intent, the action, the parameters, and the response defined for the intent.
- Your service performs actions as needed, like database queries or external API calls.

- e) Your service sends a webhook response message to Dialogflow. This message contains the response that should be sent to the end-user.
- f) Dialogflow sends the response to the end-user.
- g) The end-user sees or hears the response.

The entire code for fulfilment can be found at https://github.com/viratsagar/iNeuron_academics_bot

3.4.1 Functionalities achieved in fulfilment:

- a) Sending an email to the customer with the syllabus and all the course details based on the course selected.
- b) Sending an email to the Support team to further contact the customer for further clarification.

3.4.2 Webhook for fulfilment:

- a) Select Fulfillment in the left sidebar menu.
- b) Toggle the Webhook field to Enabled.
- c) Provide the details for your webhook service in the form. If your webhook doesn't require authentication, leave the authentication fields blank.
- d) Click Save at the bottom of the page.

Dialogflow

covid19-bot

en

Intents

Entities

Knowledge [beta]

Fulfillment

Integrations

Training

Validation

History

Analytics

Prebuilt Agents

Small Talk

Fulfillment

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*

BASIC AUTH

HEADERS

[+ Add header](#)

SMALL TALK

Inline Editor (Powered by Google Cloud Functions) DISABLED

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

[index.js](#) [package.json](#)

Figure 10: Provide the details for your webhook service.

3.4.3 Fulfilment Request and Response:

```
app.py x
1  import smtplib
2  from email.mime.multipart import MIMEMultipart
3  from email.mime.text import MIMEText
4  from flask import Flask, request, make_response
5  import json
6
7  from flask_cors import cross_origin
8
9  import requests
10
11
12  app = Flask(__name__)
13
14  @app.route('/', methods=['GET'])
15  def home():
16      return '''<h1>Send Mail-ID</h1>
17      <p>A prototype API for sending Information of Covid-19 to Mail-ID.</p>'''
18
19  # getting and sending response to dialogflow
20  @app.route('/webhook', methods=['POST'])
21  @cross_origin()
22  def webhook():
23
24      req = request.get_json(silent=True, force=True)
25
26      #print("Request:")
27      #print(json.dumps(req, indent=4))
28
29      res = processRequest()
30
31      res = json.dumps(res, indent=4)
32      #print(res)
```

Figure 11

6The following code is written to send covid19 information to the E-mail ID according to their state ISO Code.

```
app.py x
29     res = processRequest()
30
31     res = json.dumps(res, indent=4)
32     #print(res)
33     r = make_response(res)
34     r.headers['Content-Type'] = 'application/json'
35     return r
36
37
38     # processing the request from dialogflow
39     def processRequest():
40
41         f1=request.json["queryResult"]
42         f2=f1['parameters']
43         print(f2)
44         name=f2["name"]
45         #print(cust_name)
46         mobile_number = f2["phone_number"]
47         toaddr=f2["emailid"]
48         isocode= f2["isocode"]
49         print(toaddr)
50
51         # email_sender=EmailSender()
52         #####
53         url = "https://covid19india.p.rapidapi.com/getStateData/" + isocode
54
55         headers = {
56             'x-rapidapi-host': "covid19india.p.rapidapi.com",
57             'x-rapidapi-key': "f8791938c1mshc8f41195c9870fep181866jsn733b0c0182f8"
58         }
59
60         response = requests.request("GET", url, headers=headers)
```

Figure 12

```

app.py x
59
60 response = requests.request("GET", url, headers=headers)
61 res1 = json.loads(response.text)
62 # res1['response']['active']
63 state = res1['response']['name']
64 confirmed = res1['response']['confirmed']
65 confirmed = str(confirmed)
66 recovered = res1['response']['recovered']
67 recovered = str(recovered)
68 deaths = res1['response']['deaths']
69 deaths = str(deaths)
70 active = res1['response']['active']
71 active = str(active)
72
73 body2="State {0} have total number of confirmed cases {1} where active cases are {2}." \
74      " Recovery count {3} and death toll rises to {4} due to covid-19 till present.".format(state, confirmed, active, recovered, deaths)
75 print(body2)
76 #####
77
78 contact_add = "6363636363@gmail.com" ## the email id of the support team
79 fromaddr = "6363636363@gmail.com" ## the email id from where we are going to send the mail
80 #####
81
82 # instance of MIMEMultipart
83 msg = MIMEMultipart()
84
85 # storing the senders email address
86 msg['From'] = fromaddr
87
88 # storing the receivers email address
89 msg['To'] = ",".join(toaddr)
90

```

```

app.py x
88 # storing the receivers email address
89 msg['To'] = ",".join(toaddr)
90
91 #####
92 # instance of MIMEMultipart
93 msg = MIMEMultipart()
94 # storing the subject
95 msg['Subject'] = "Query For Covid-19 Cases"
96
97 # attach the body with the msg instance
98 msg.attach(MIMEText(body2, 'plain'))
99
100 #####
101 # creates SMTP session
102 s = smtplib.SMTP('smtp.gmail.com', 587)
103
104 # start TLS for security
105 s.starttls()
106
107 # Authentication
108 s.login(fromaddr, "6363636363") # give your password here
109
110 # Converts the Multipart msg into a string
111 text = msg.as_string()
112 # sending the mail
113 s.sendmail(fromaddr, toaddr, text)
114
115 # send mail with username , mobile number and email id to concerned team
116
117 # instance of MIMEMultipart
118 msg1 = MIMEMultipart()
119 # storing the subject
120 msg1['Subject'] = "Query For Covid-19 Cases"
webhook()

```

```

117 # instance of MIME multipart
118 msg1 = MIME multipart()
119 # storing the subject
120 msg1[ 'Subject' ] = "Query For Covid-19 Cases"
121
122 # string to store the body of the mail
123 body1 = "person with name {0} have some queries regarding Covid-19 cases." \
124         "Please reach to {0} at his mobile number {1} and email id {2}.".format(name, mobile_number, toaddr)
125
126 # attach the body with the msg instance
127 msg1.attach(MIMEText(body1, 'plain'))
128
129 text = msg1.as_string()
130 # sending the mail
131 s.sendmail(fromaddr, contact_add, text)
132
133 # terminating the session
134 s.quit()
135
136 fulfillmentText = body2 + " An email has been sent to your contact information. Do you have further queries?"
137
138 return {
139     "fulfillmentText": fulfillmentText
140 }
141
142
143
144 if __name__ == '__main__':
145
146     app.run(debug=True)
147

```

Figure 15, (Fig 11 to 15Code to send information to customer E-mail ID)

```

app.py x requirements.txt x
Plugins supporting requirements.txt files found.
1 APScheduler==3.6.3
2 certifi==2019.11.28
3 Click==7.0
4 colorhash==1.0.2
5 configparser==4.0.2
6 cycycler==0.10.0
7 Flask==1.1.1
8 Flask-Cors==3.0.8
9 Flask-MonitoringDashboard==3.0.6
10 gunicorn==20.0.4
11 itsdangerous==1.1.0
12 Jinja2==2.11.0
13 kiwisolver==1.1.0
14 kneed==0.5.3
15 MarkupSafe==1.1.1
16 numpy==1.18.1
17 psutil==5.7.0
18 pyparsing==2.4.6
19 python-dateutil==2.8.1
20 six==1.14.0
21 SQLAlchemy==1.3.13
22 tzlocal==2.0.0
23 Werkzeug==0.16.1
24 wincertstore==0.2
25

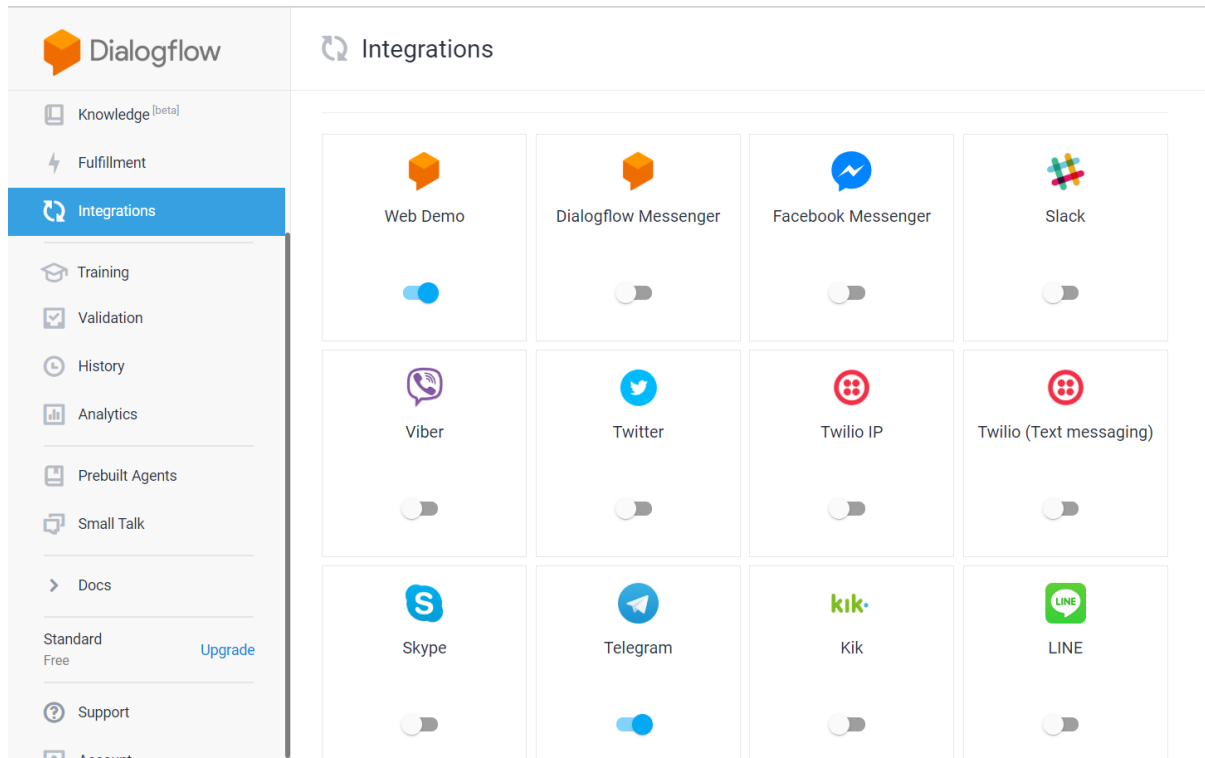
```

Figure 16: Following required plugins imported to do the project

3.5 Integration:

Dialogflow integrates with many popular conversation platforms like Google Assistant, Slack, and Facebook Messenger. If you want to build an agent for one of these platforms, you should use one of the many *integrations* options. Direct end-user interactions are handled for you, so you can focus on building your agent.

Figure 17: Intergration with other platforms.



3.5.1 Integration with Web Demo:

The following code directs the user to html file.

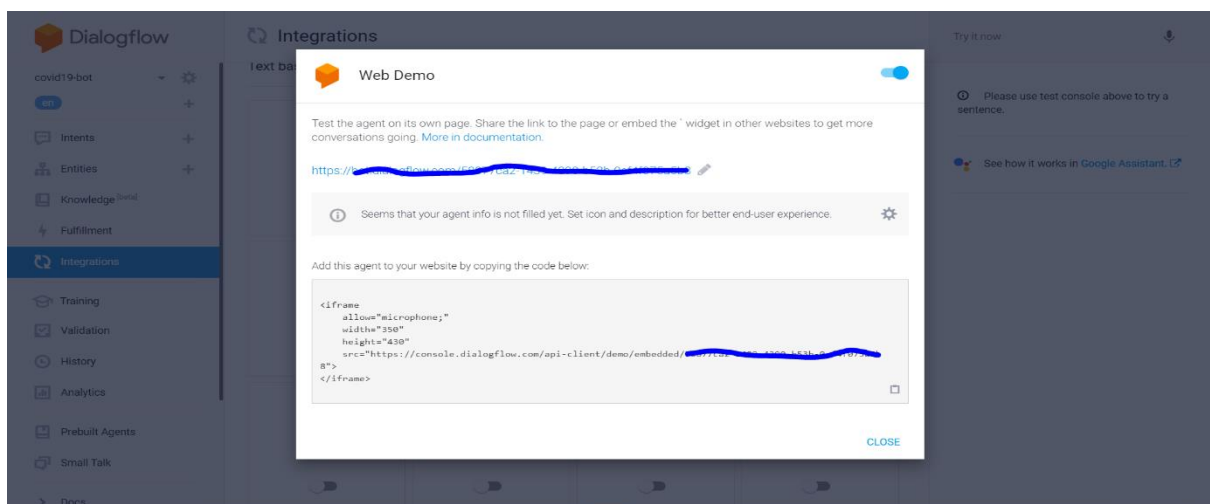


Figure 18: Web Demo intergration.

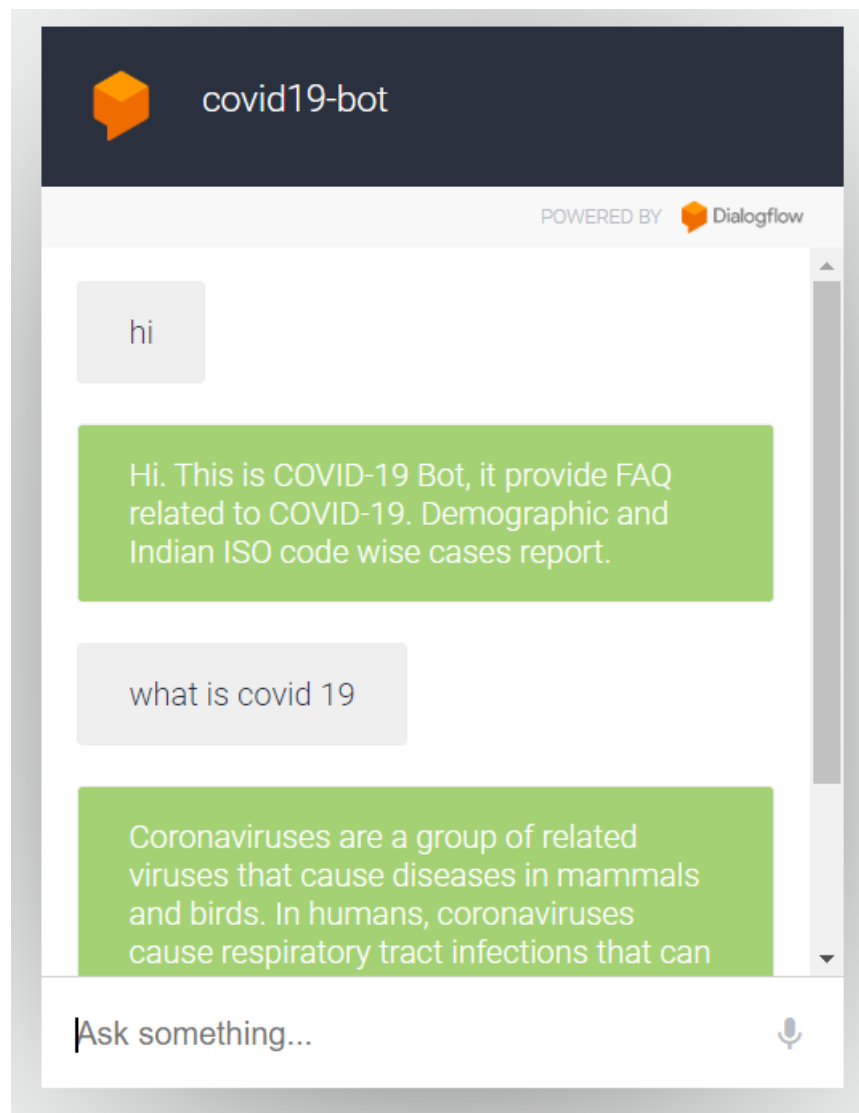


Figure 19: Web demo ChatBot

3.5.2 Integration with Telegram:

Dialogflow Telegram Integration allows you to easily create Telegram bots with natural language understanding based on the Dialogflow technology.

In order to set up the Telegram integration for your agent, you'll need the following:

- a) Telegram account.

3.7.1.2 Creating a Bot in Telegram

- a) Login to Telegram and go to <https://telegram.me/botfather>
- b) Click the Start button in the web interface or type /start

- c) Click on or type `/newbot` and enter a name
- d) Enter a username for the bot, ending in "bot" (e.g. garthswetherbot)
- e) Copy the generated access token

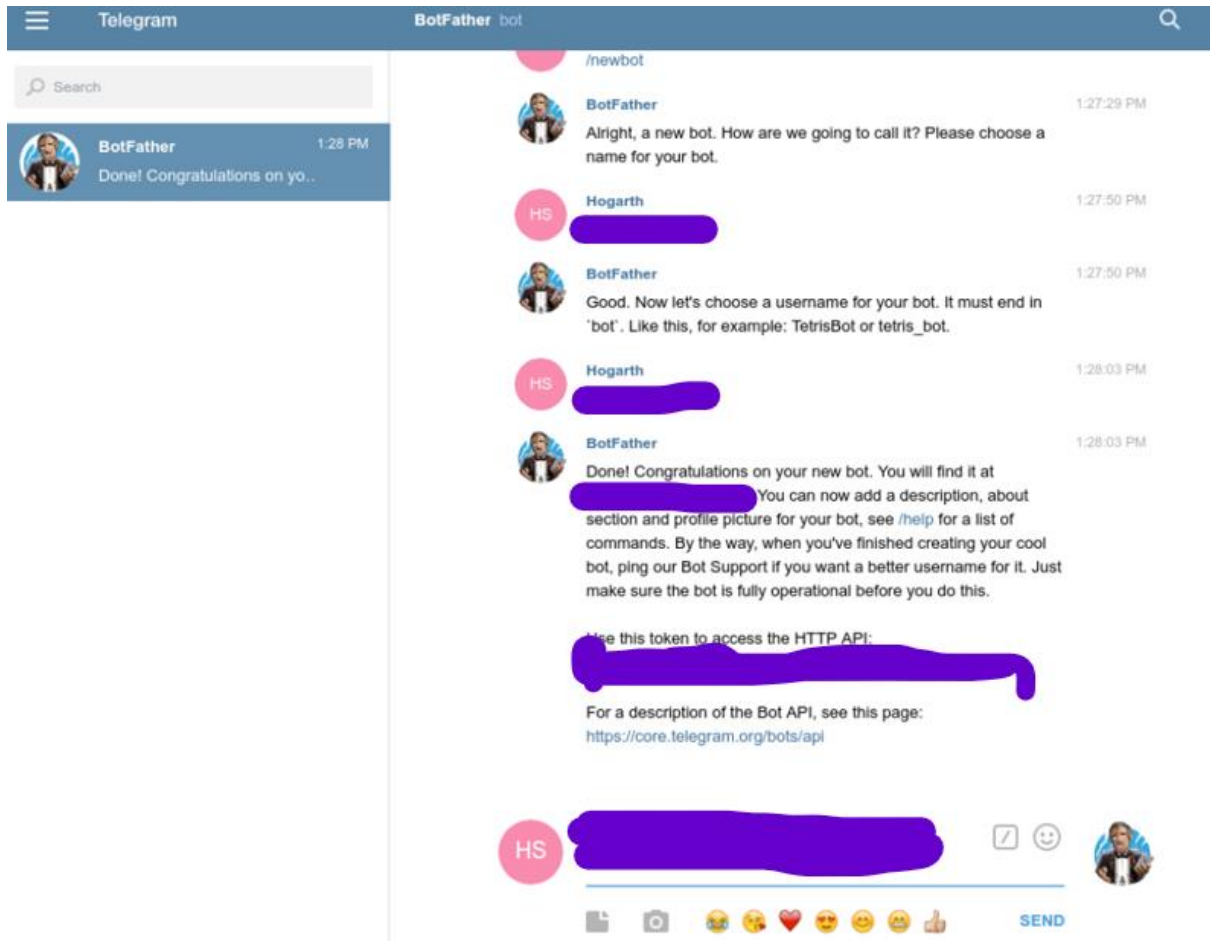
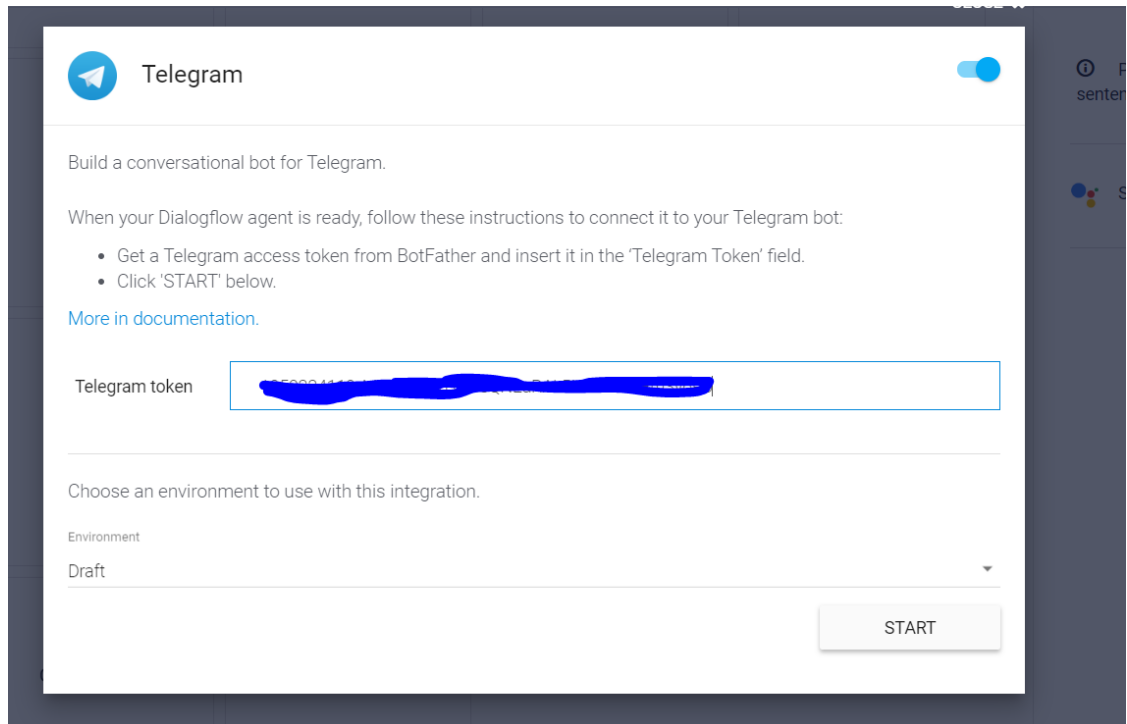


Figure 20: Creation of Bot in Telegram using the generated access token.

3.7.1.3 Setting Up Dialogflow

- a) **In Dialogflow, go to Integrations in the left hand menu**
- b) **Click on the Telegram tile**
- c) **Paste the Access Token into the related field**

d) Click the Start button



The screenshot shows the 'Telegram' integration setup interface in DialogFlow. At the top, there's a Telegram logo and a toggle switch. Below, instructions guide the user to build a conversational bot and connect it to a Telegram bot. A list of steps includes getting a Telegram access token from BotFather and clicking 'START'. A link for 'More in documentation' is provided. The 'Telegram token' field is highlighted with a blue box, containing a redacted token. Below this, the user is prompted to choose an environment, with 'Draft' selected. A 'START' button is located at the bottom right.

Telegram

Build a conversational bot for Telegram.

When your Dialogflow agent is ready, follow these instructions to connect it to your Telegram bot:

- Get a Telegram access token from BotFather and insert it in the 'Telegram Token' field.
- Click 'START' below.

[More in documentation.](#)

Telegram token

Choose an environment to use with this integration.

Environment

Draft

START

Figure 21: DialogFlow intergration with Telegram.

e) Now you can search your bot in telegram and you can chat with it.

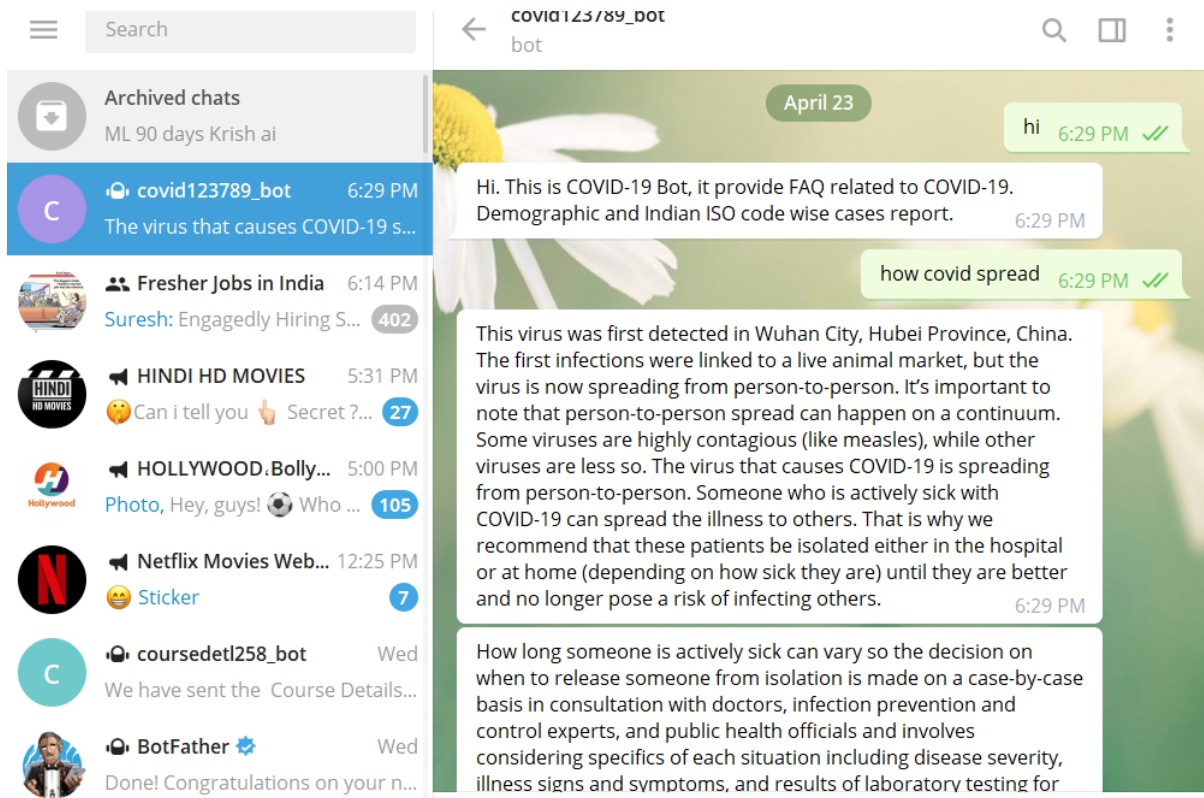
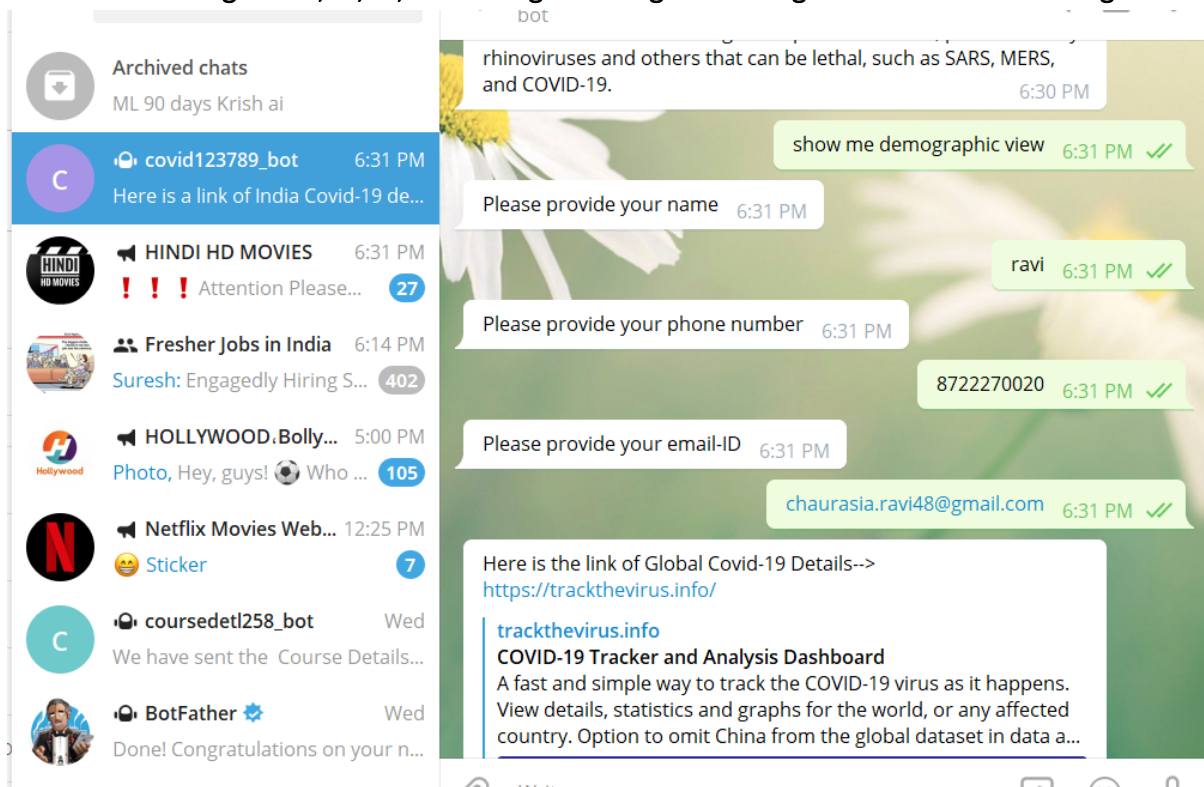
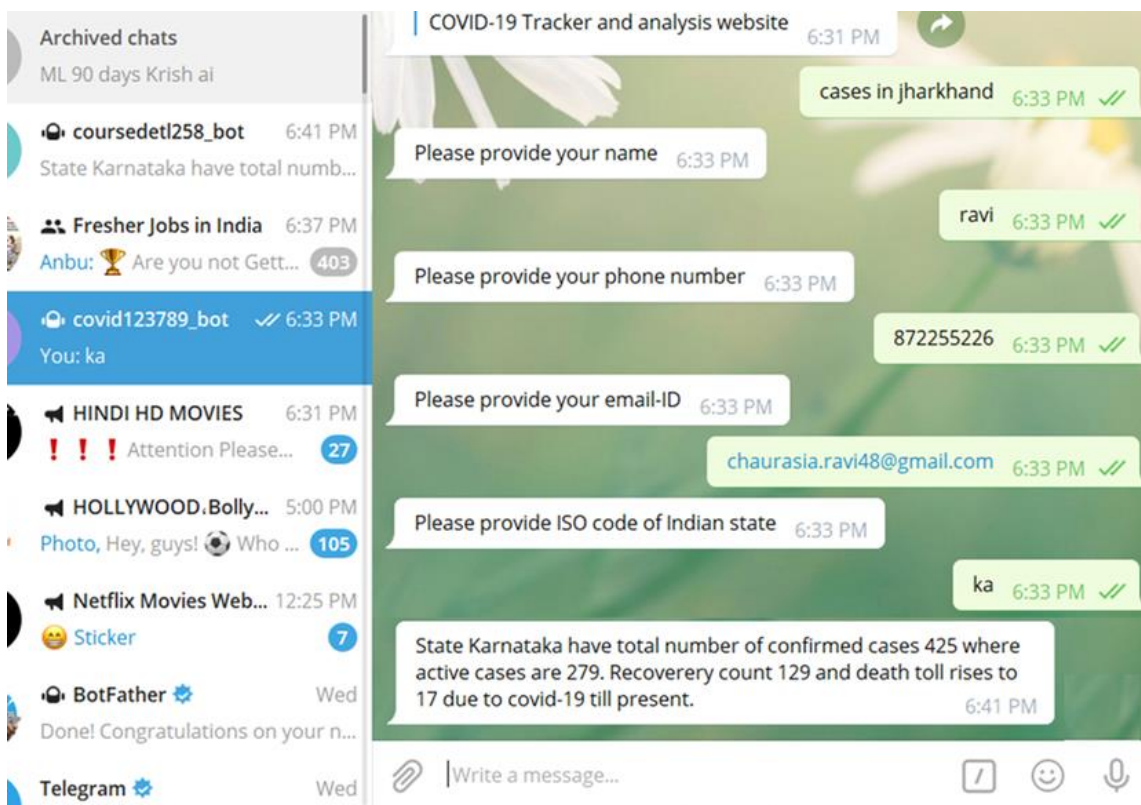
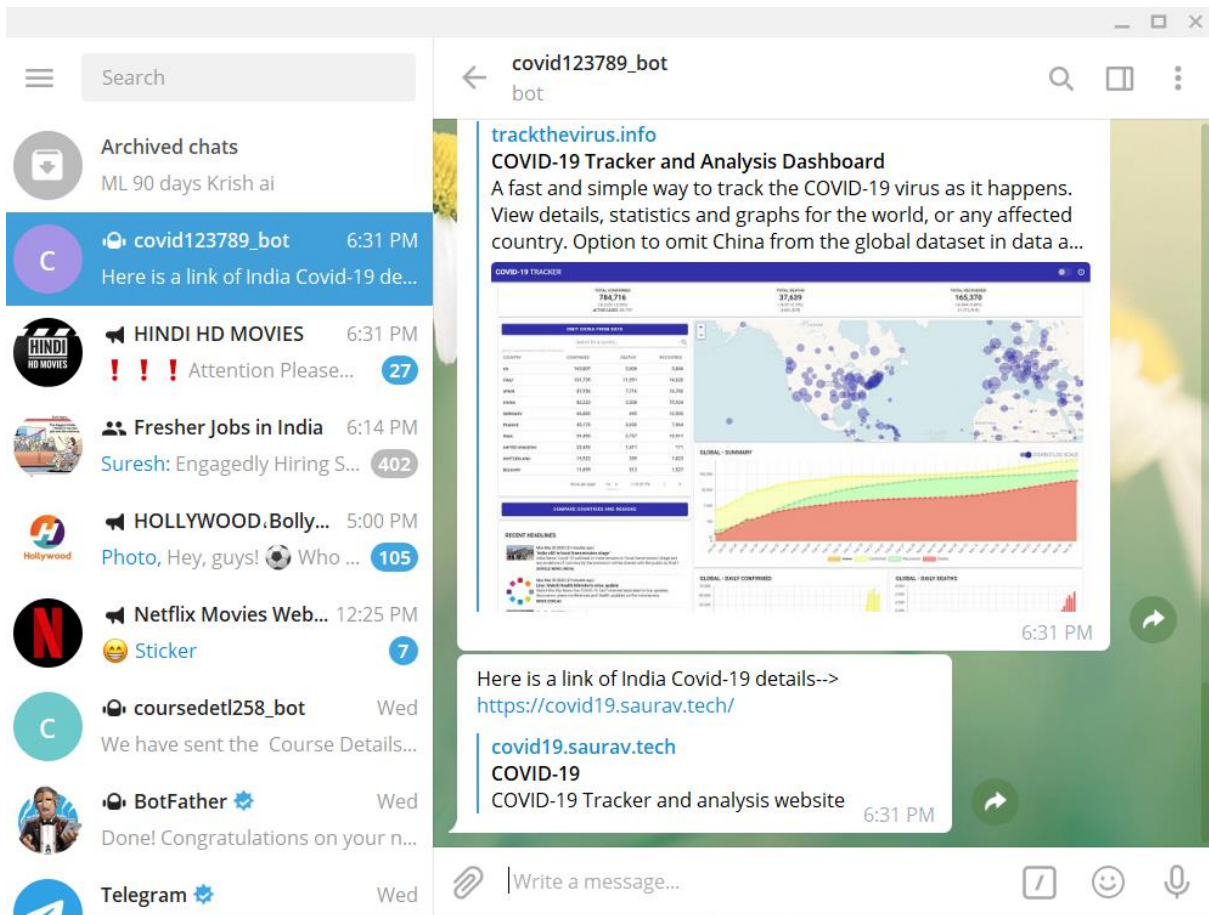


Figure 22,23,24,25: Testing the integrated Telegram ChatBot with DialogFlow.





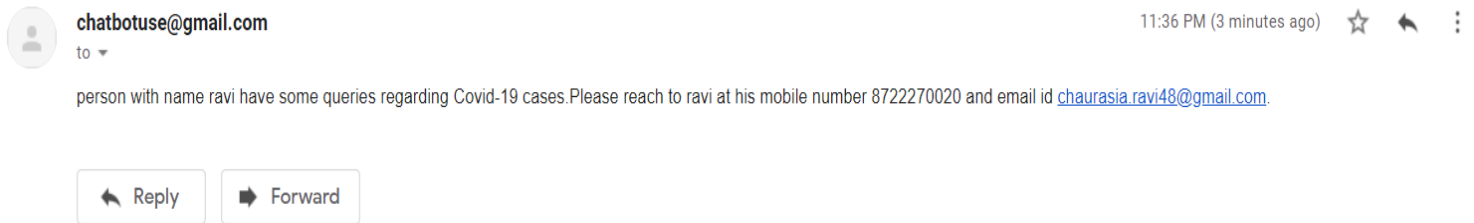


Figure 26: E-mail sent to customer support regarding customer details

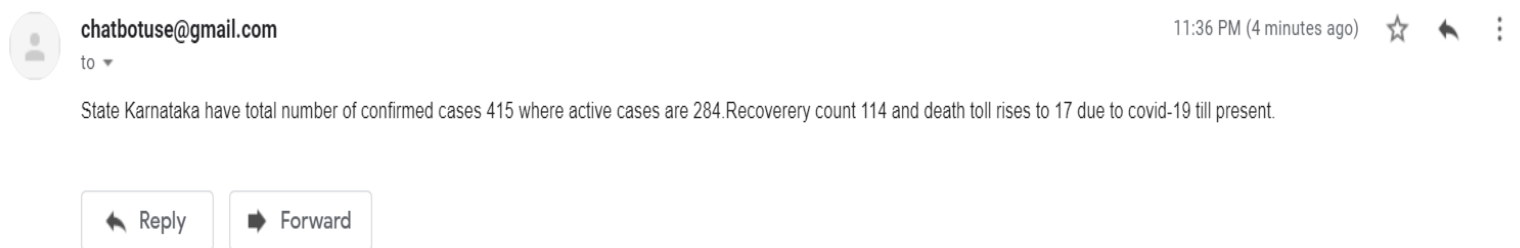


Figure 27: E-mail sent to customer regarding State Covid19 details.

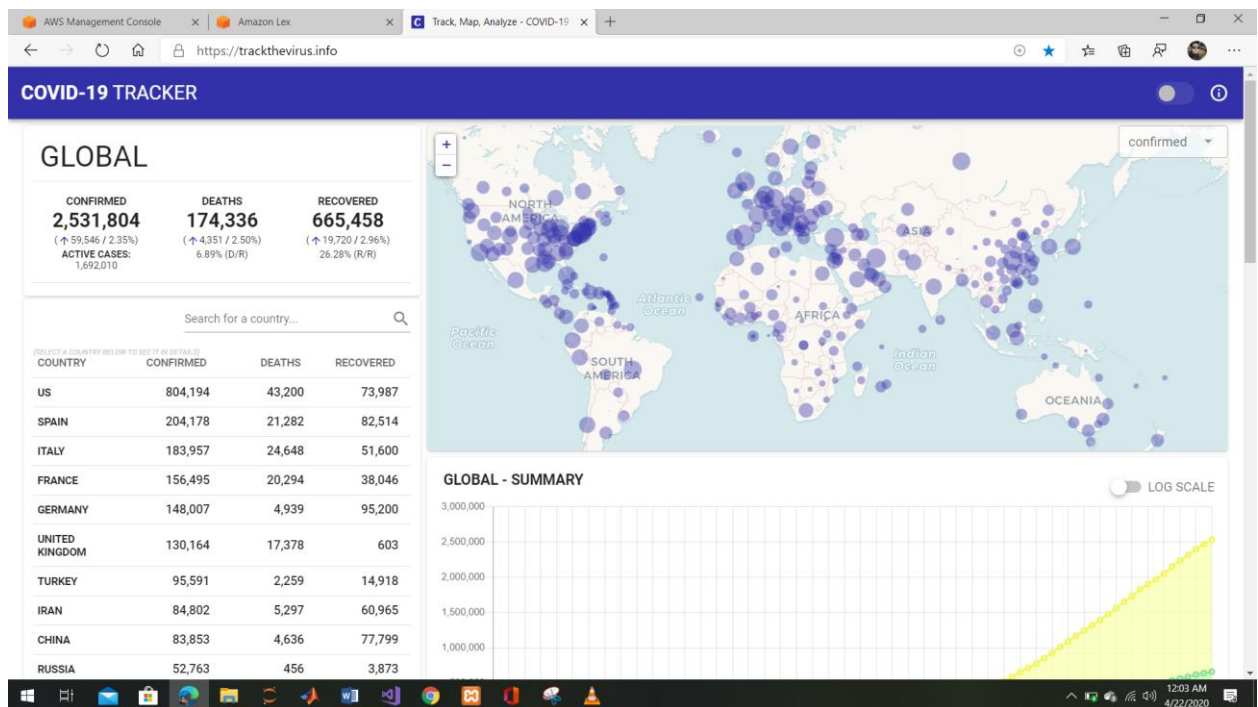


Figure 1: Global covid-19 demographic view

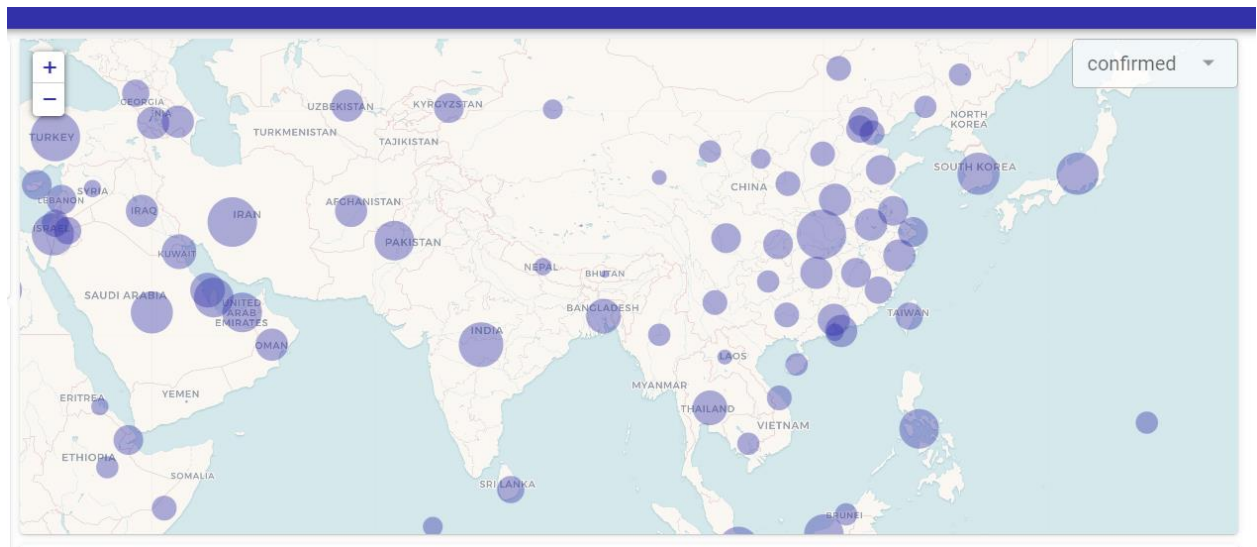


Figure 29: Zoom in to map

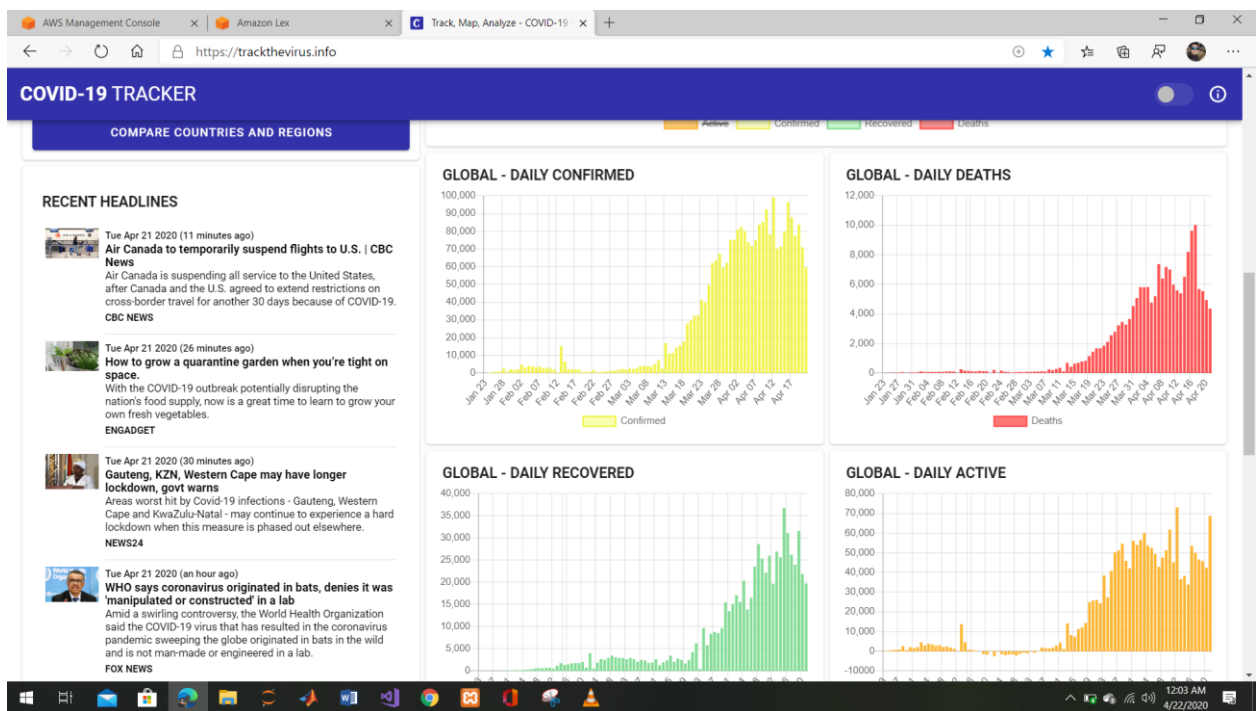


Figure 30: Global covid-19 demographic graph with news

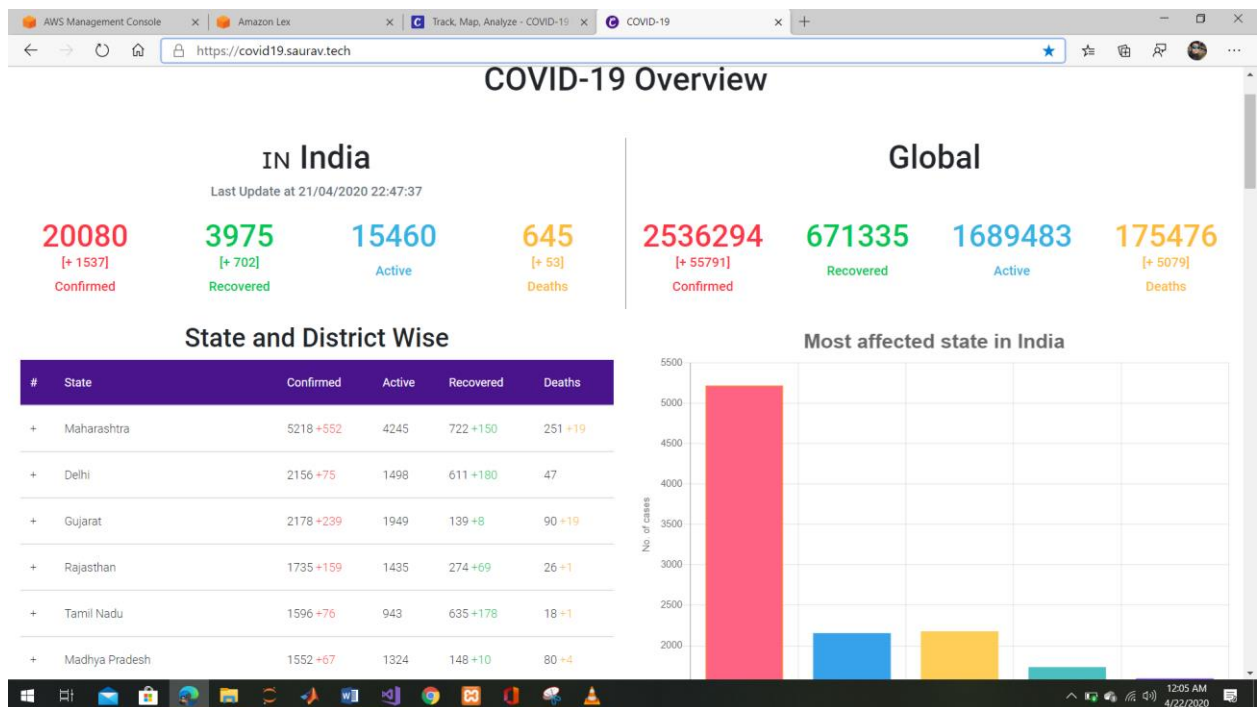


Figure 31: India covid-19 demographic view with numbers

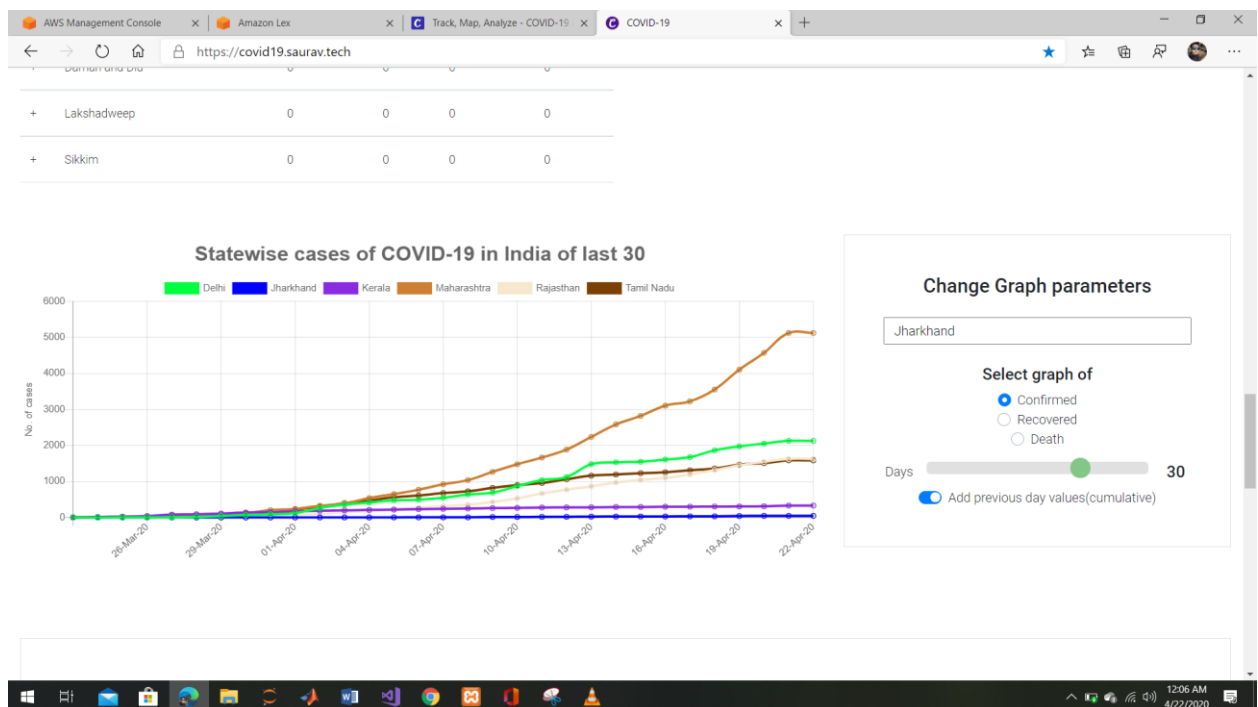


Figure 32 India covid-19 demographic view continues (State-wise)

Thanks for watch