Covid-19 Bot using Amazon AWS Lex (Procedure and Outcome)

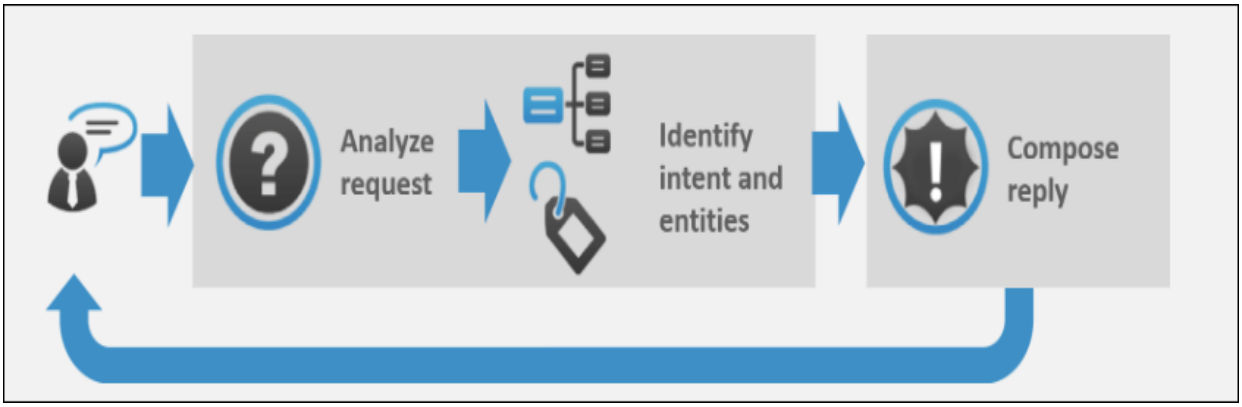
Contents

1. Chatbot Concept

What are chatbots?

A Chatbots is an intelligent software that can simulate a conversation (or a chat) with a user in natural language through messaging applications, websites, mobile phones or telephone. They can be programmed to respond to simple keywords or prompts to complex conversations.

A Chatbot has two different tasks at the core: user request analysis (identifying the intent of the user) and returning the response.



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

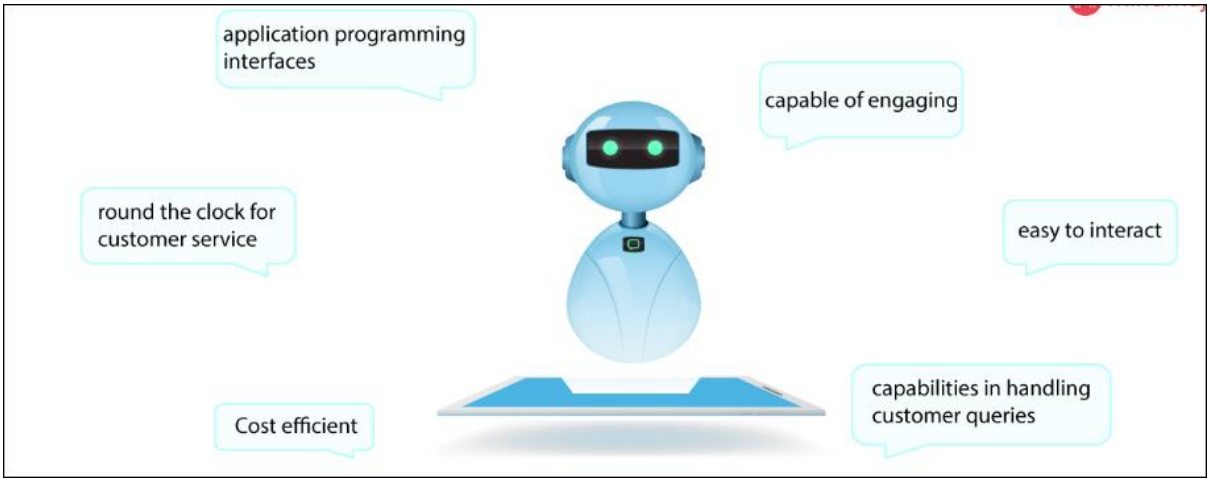
2. **Chatbots that function on rules**: It is easy to build. Though simplistic, it can get basics tasks done with limitations to specific commands. This bot is only as smart as it is programmed.
3. **Chatbots that function on machine learning**: It understands languages and not just commands and provides relevant responses. It communicates through speech or text relying on AI technologies like machine learning and NLP. It gets smarter as it learns from the conversations from users.

4. Use cases

A chatbot can be used in many different ways. Any business or industry having a website or an app can have a chatbot. Following are some of the chatbot applications, out of the infinite possibilities:

1. A takeaway restaurant that allows customers to order from their office or home.
2. A chatbot that answers customer service questions and provides help with different tasks.
3. A chatbot that allows the customer to book a flight and receive relevant information.
4. A chatbot that helps a customer with ecommerce purchases.
5. A marketing campaign chatbot that asks questions to the customer.
6. A health bot that provides services for booking a Doctor consultation.
7. A movie ticket booking bot.

5. Advantages of Chatbots



The advantages of chatbots are as follows:

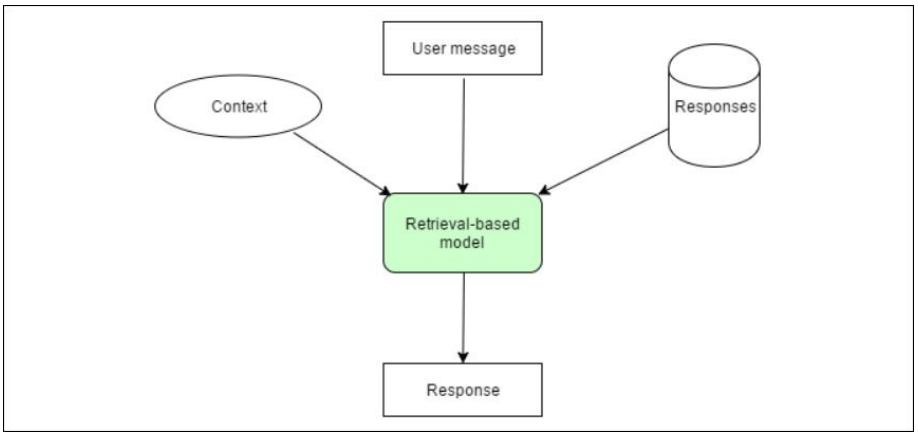
- They are capable of engaging customers in a friendly manner.
- A simple interface makes it easier to interact with.
- Interactions are made possible via social media platforms like Twitter, Facebook, etc. Chats are served through application programming interfaces (APIs).
- Chatbots have extraordinary capabilities in handling customer queries.
- Chatbots work efficiently round the clock for customer service.
- Cost-efficient and are easy to build.

6. Models for a Chatbot

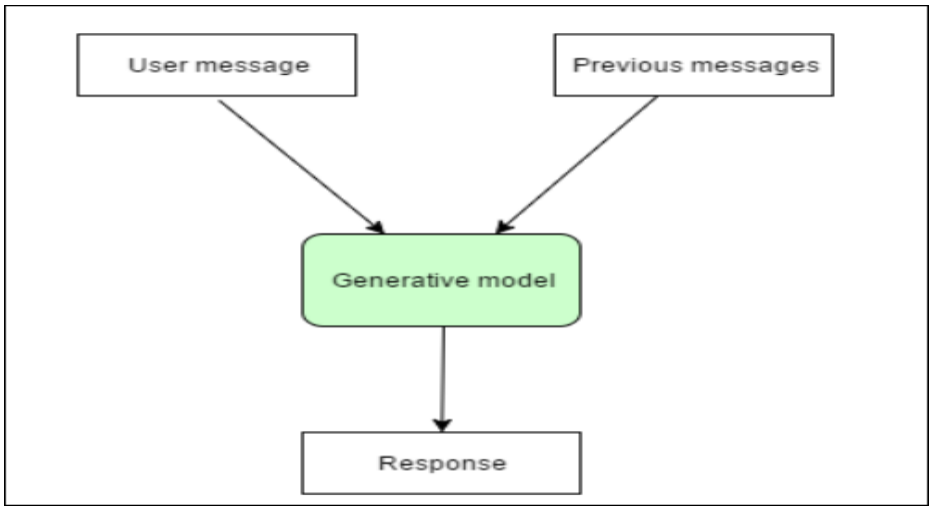
When creating a chatbot, the goal is to automate the process altogether and to reduce human intervention. The first step in creating a chatbot involves inputting thousands of existing interactions between customers and customer service representatives to teach the machine which words/phrases are sensitive to the business. The next step is preprocessing which involves incorporating grammar to identify any misspellings.

Then the next step is to identify the kind of chatbot we would like to implement. There mainly are two models:

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



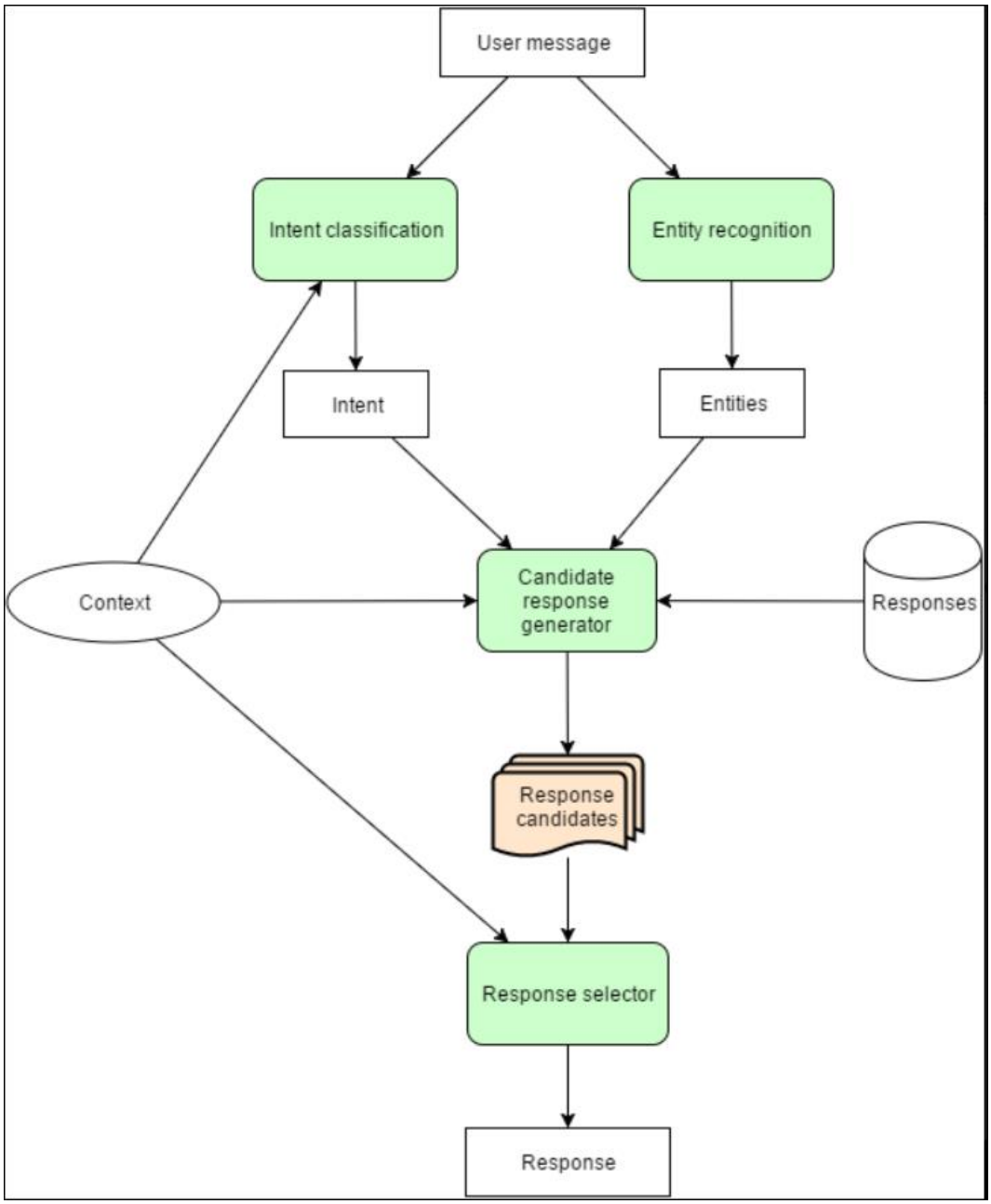
2. **Generative based model**: A generative model chatbot doesn't use any predefined repository. This kind of chatbot is more advanced because it learns from scratch using "Deep Learning."



Pros and Cons:

Generative models	Selective models
+ Can possibly generate arbitrary answer (more similar to general AI) + Can generate answer in correct grammar form (e.g. with correct speaker gender) - Can generate answer with incorrect grammar/syntax - Prone to "general answer" problem - Difficult to impose properties on model replies (e.g. no obscene words, speak like some specific person), but possible!	- Restricted pool of answers which can not cover all dialogue topics - For context "What is your name, girl?" can select "My name is Stephen." (inconsistency) + Predefined answers have good grammar/syntax + Less prone to "general answer" problems + You can customize answers for your own needs (without obscenities, kind answers)

7. Architecture of AI chatbot



Message processing begins with understanding what the user is talking about.

1. An intent classification module identifies the intent of the user messages. Typically, it is a selection of one out of many predefined intents, though more sophisticated bots can identify multiple intents from one message.
2. An entity recognition module extracts structured bits of information from the message. The weather bot can extract the location and date.
3. The candidate response generator is doing all the domain-specific calculations to process the user request. It can use different algorithms, call a few external APIs, or

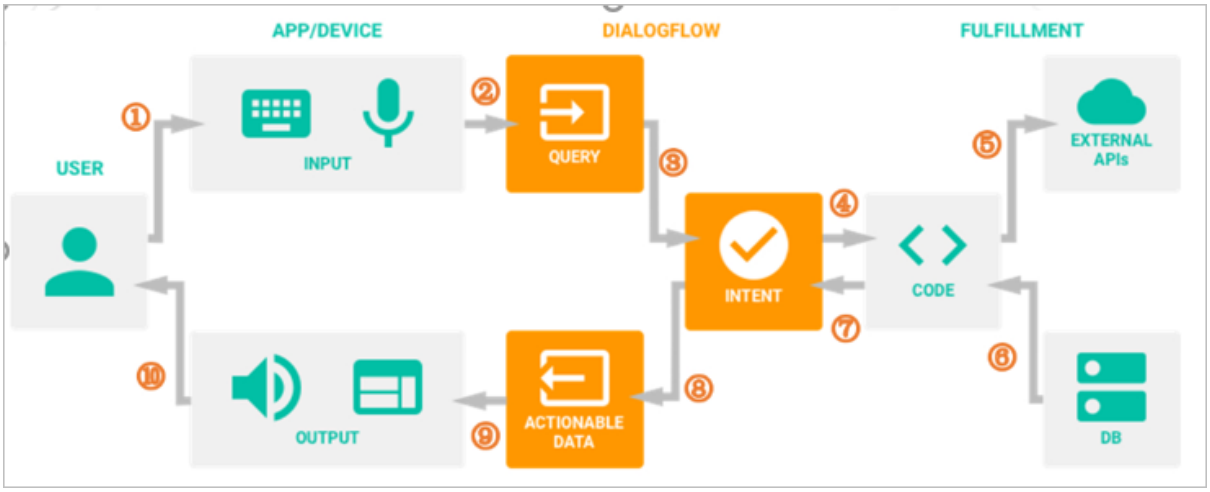
even ask a human to help with response generation. The result of these calculations is a list of response candidates. All these responses should be correct according to the domain-specific logic. They can't be just tons of random responses. The response generator must use the context of the conversation as well as intent and entities extracted from the last user message. Otherwise, it can't support multi-message conversations.

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

1 Introduction of Amazon Lex:

1.1 Amazon Lex is an AWS service for building conversational interfaces for applications using voice and text.

1.2 How do Chatbots work?



Detailed steps:

1. A user sends a text/voice message to a device or an App
2. The App/Device transfers the message to Lex
3. The message is categorized and matched to a corresponding intent (Intents are defined manually by developers in Lex)
4. We define actions for each intent in the corresponding fulfillment (Webhook).
5. When Lex finds a specific intent, the webhook will use external APIs to find a response from external databases.
6. The external databases send back the required information to the webhook.
7. Webhook sends a formatted response to the intent.

8. Intent generates actionable data according to different channels.
9. The actionable data go to output Apps/Devices.
10. The user gets a text/image/voice response.

1.3 What Is Amazon Lex?

Amazon Lex is an AWS service for building conversational interfaces for applications using voice and text. With Amazon Lex, the same conversational engine that powers Amazon Alexa is now available to any developer, enabling you to build sophisticated, natural language chatbots into your new and existing applications. Amazon Lex provides the deep functionality and flexibility of natural language understanding (NLU) and automatic speech recognition (ASR) so you can build highly engaging user experiences with lifelike, conversational interactions, and create new categories of products.

Amazon Lex enables any developer to build conversational chatbots quickly. With Amazon Lex, no deep learning expertise is necessary—to create a bot, you just specify the basic conversation flow in the Amazon Lex console. Amazon Lex manages the dialogue and dynamically adjusts the responses in the conversation. Using the console, you can build, test, and publish your text or voice chatbot. You can then add the conversational interfaces to bots on mobile devices, web applications, and chat platforms (for example, Facebook Messenger).

Amazon Lex provides pre-built integration with AWS Lambda, and you can easily integrate with many other services on the AWS platform, including Amazon Cognito, AWS Mobile Hub, Amazon CloudWatch, and Amazon DynamoDB. Integration with Lambda provides bots access to pre-built serverless enterprise connectors to link to data in SaaS applications, such as Salesforce, HubSpot, or Marketo.

Some of the benefits of using Amazon Lex include:

- **Simplicity** – Amazon Lex guides you through using the console to create your own chatbot in minutes. You supply just a few example phrases, and Amazon Lex builds a complete natural language model through which the bot can interact using voice and text to ask questions, get answers, and complete sophisticated tasks.
- **Democratized deep learning technologies** – Powered by the same technology as Alexa, Amazon Lex provides ASR and NLU technologies to create a Speech Language Understanding (SLU) system. Through SLU, Amazon Lex takes natural language speech and text input, understands the intent behind the input, and fulfills the user intent by invoking the appropriate business function.

Speech recognition and natural language understanding are some of the most challenging problems to solve in computer science, requiring sophisticated deep learning algorithms to be trained on massive amounts of data and infrastructure. Amazon Lex puts deep learning technologies within reach of all developers, powered by the same technology as Alexa. Amazon Lex chatbots convert incoming speech to text and understand the user intent to generate an intelligent response, so you can focus on building your bots with differentiated value-add for your customers, to define entirely new categories of products made possible through conversational interfaces.

- **Seamless deployment and scaling** – With Amazon Lex, you can build, test, and deploy your chatbots directly from the Amazon Lex console. Amazon Lex enables you to easily publish your voice or text chatbots for use on mobile devices, web apps, and chat services (for example, Facebook Messenger). Amazon Lex scales automatically so you don't need to worry about provisioning hardware and managing infrastructure to power your bot experience.
- **Built-in integration with the AWS platform** – Amazon Lex has native interoperability with other AWS services, such as Amazon Cognito, AWS Lambda, Amazon CloudWatch, and AWS Mobile Hub. You can take advantage of the power of the AWS platform for security, monitoring, user authentication, business logic, storage, and mobile app development.
- **Cost-effectiveness** – With Amazon Lex, there are no upfront costs or minimum fees. You are charged only for the text or speech requests that are made. The pay-as-you-go pricing and the low cost per request make the service a cost-effective way to build conversational interfaces. With the Amazon Lex free tier, you can easily try Amazon Lex without any initial investment.

2 Amazon Lex: How It Works

Amazon Lex enables you to build applications using a speech or text interface powered by the same technology that powers Amazon Alexa. Following are the typical steps you perform when working with Amazon Lex:

1. Create a bot and configure it with one or more intents that you want to support. Configure the bot so it understands the user's goal (intent), engages in conversation with the user to elicit information, and fulfills the user's intent.
2. Test the bot. You can use the test window client provided by the Amazon Lex console.
3. Publish a version and create an alias.
4. Deploy the bot. You can deploy the bot on platforms such as mobile applications or messaging platforms such as Facebook Messenger.

Before you get started, familiarize yourself with the following Amazon Lex core concepts and terminology:

- **Bot** – A bot performs automated tasks such as ordering a pizza, booking a hotel, ordering flowers, and so on. An Amazon Lex bot is powered by Automatic Speech Recognition (ASR) and Natural Language Understanding (NLU) capabilities, the same technology that powers Amazon Alexa.

Amazon Lex bots can understand user input provided with text or speech and converse in natural language. You can create Lambda functions and add them as code hooks in your intent configuration to perform user data validation and fulfillment tasks.

- **Intent** – An intent represents an action that the user wants to perform. You create a bot to support one or more related intents. For example, you might create a bot that orders pizza and drinks. For each intent, you provide the following required information:

    - Intent name– A descriptive name for the intent. For example, OrderPizza.
    - Sample utterances – How a user might convey the intent. For example, a user might say "Can I order a pizza please" or "I want to order a pizza".
    - How to fulfill the intent – How you want to fulfill the intent after the user provides the necessary information (for example, place order with a local pizza shop). We recommend that you create a Lambda function to fulfill the intent.

You can optionally configure the intent so Amazon Lex simply returns the information back to the client application to do the necessary fulfillment.

In addition to custom intents such as ordering a pizza, Amazon Lex also provides built-in intents to quickly set up your bot. For more information, see Built-in Intents and Slot Types.

- **Slot** – An intent can require zero or more slots or parameters. You add slots as part of the intent configuration. At runtime, Amazon Lex prompts the user for specific slot values. The user must provide values for all *required* slots before Amazon Lex can fulfill the intent.

For example, the OrderPizza intent requires slots such as pizza size, crust type, and number of pizzas. In the intent configuration, you add these slots. For each slot, you provide slot type and a prompt for Amazon Lex to send to the client to elicit data from the user. A user can reply with a slot value that includes additional words, such as "large pizza please" or "let's stick with small." Amazon Lex can still understand the intended slot value.

- **Slot type** – Each slot has a type. You can create your custom slot types or use built-in slot types. For example, you might create and use the following slot types for the OrderPizza intent:
    - Size – With enumeration values Small, Medium, and Large.
    - Crust – With enumeration values Thick and Thin.

Amazon Lex also provides built-in slot types. For example, AMAZON.NUMBER is a built-in slot type that you can use for the number of pizzas ordered. For more information, see Built-in Intents and Slot Types.

Currently, Amazon Lex supports only US English language. That is, Amazon Lex trains your bots to understand only US English.

(Source: https://docs.aws.amazon.com/lex/latest/dg/what-is.html)

3 Implementation of chat-bot

3.1 Creating a Lex App

1) Login to AWS portal https://signin.aws.amazon.com/

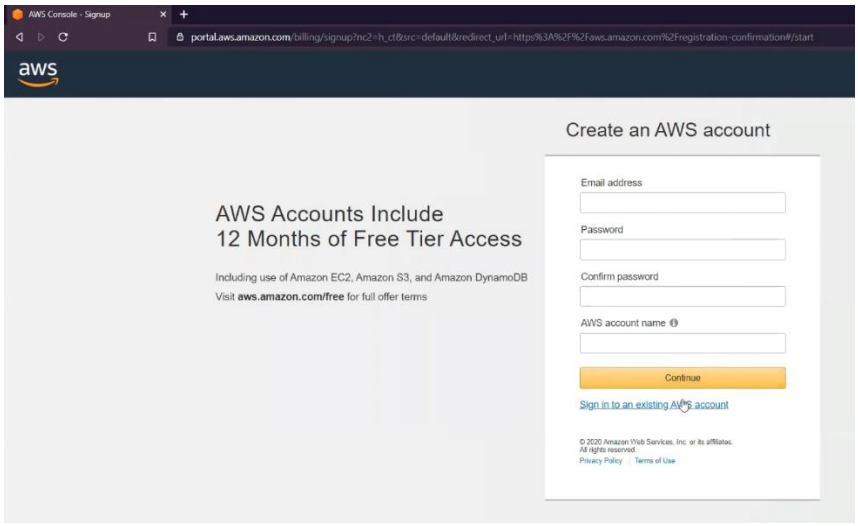


Figure 1: create your AWS account

- Once you are sign in go to amazon lex and click to create bottom

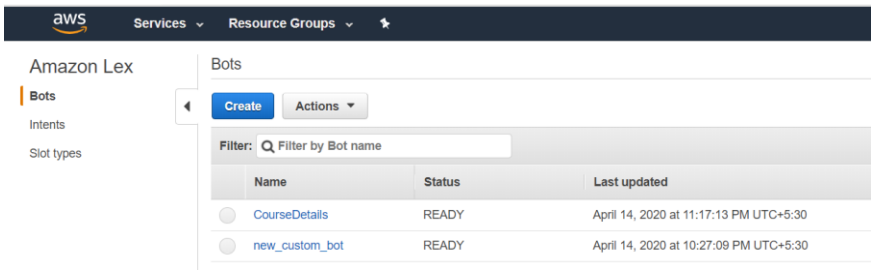


Figure 2: To create new bot

- Once you click the create button, you will see a window like this. Click on the "custom bot" to make your own bot. The other bots are example bots and you can go through them to get a glance on some higher details.

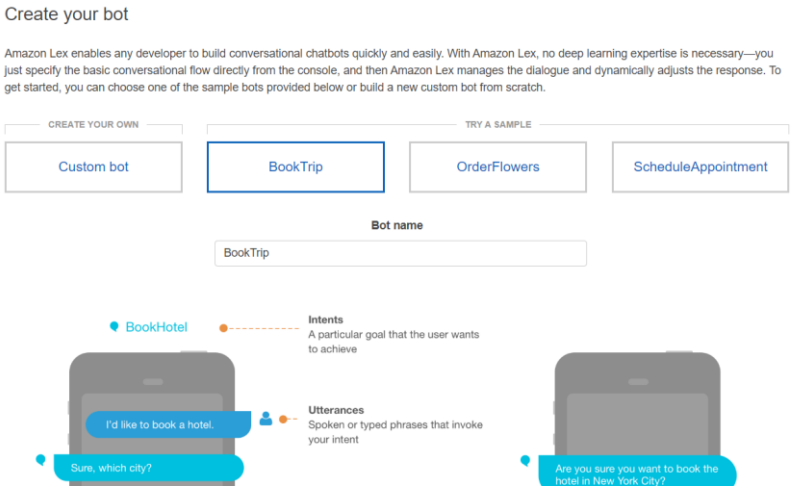


Figure 3: Different option of creating bot

- But here we use custom bot option to create from scratch

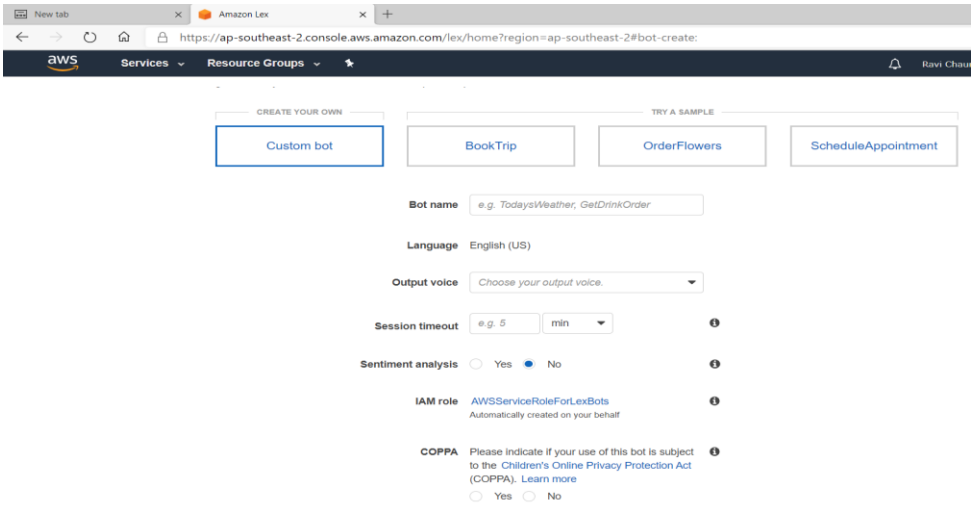


Figure 4: We need to fill-up all the things to create custom bot

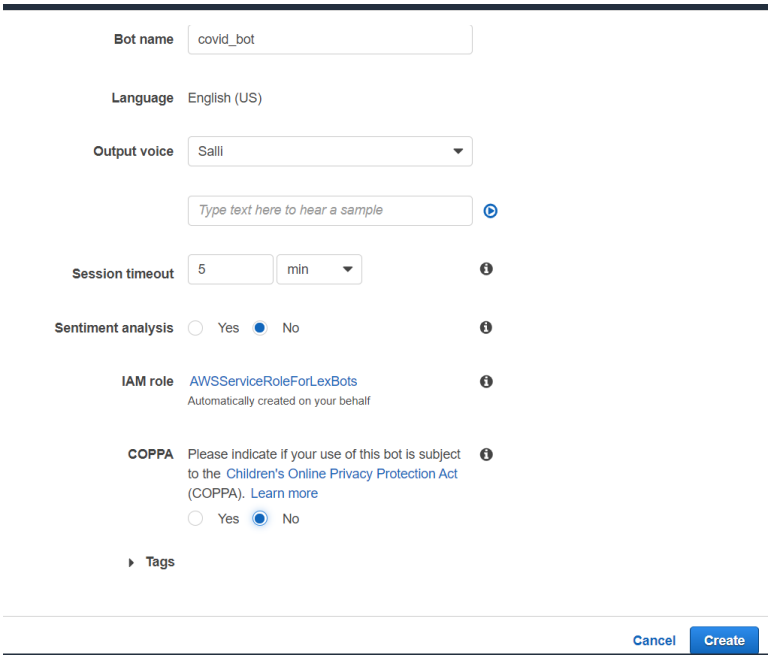


Figure 5: Bot name the respective required details has filled-up

- Now we will start by creating intents for our bot. Click on the plus button next to intents.

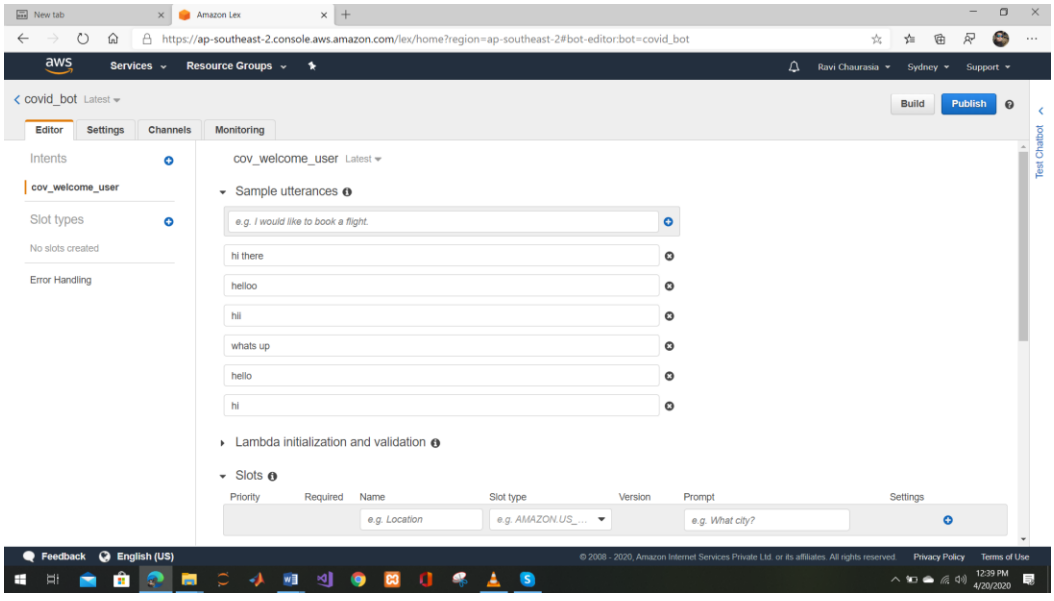


Figure 6 Intent name cov_welcome_user is given, where different welcome utterances can be seen in figure

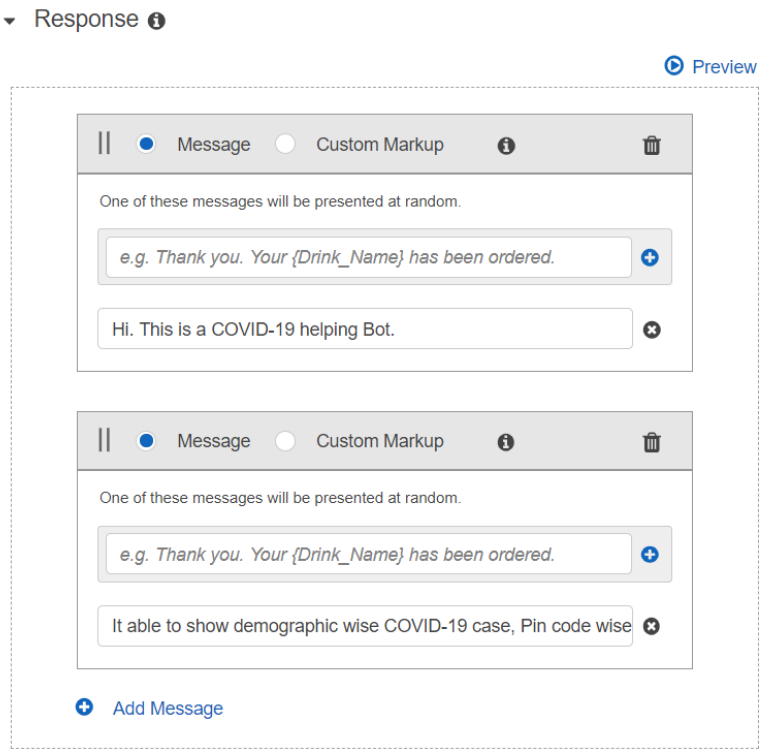


Figure 7 shows the responses in the form of messages

- One its done we save and click on build button

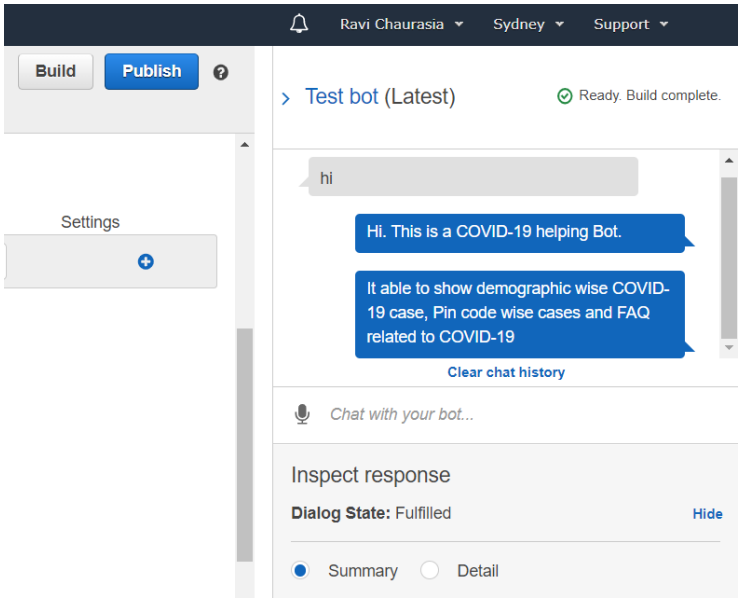


Figure 8: ones the first intent is done, we do testing

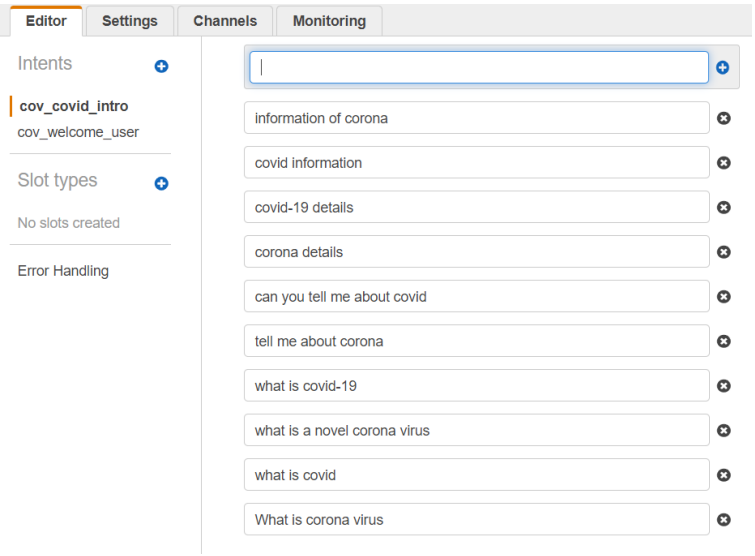


Figure 9 creating other intents and there various utterances

- Similarly we are creating new intents as per our requirements that we need the bot to respond to.

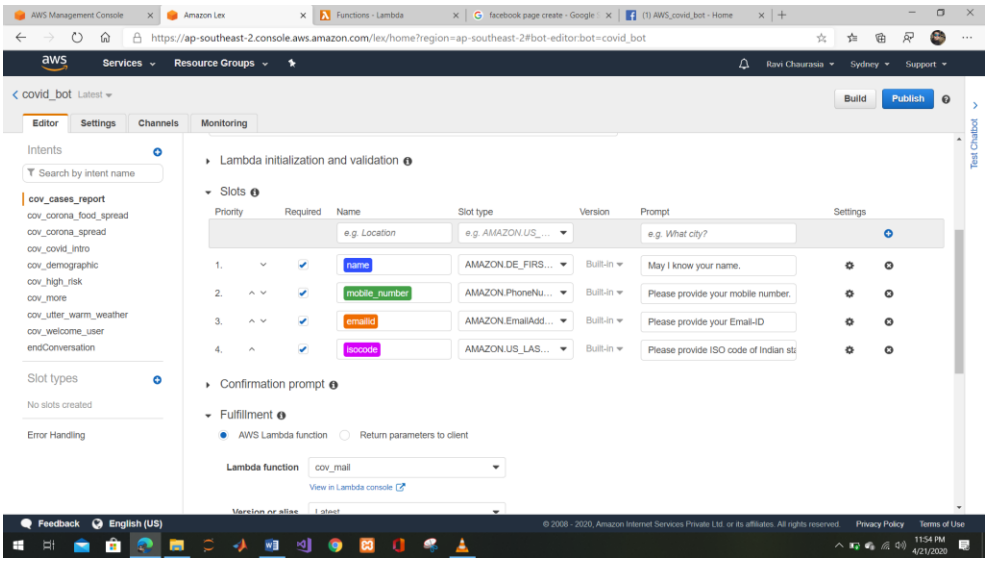


Figure 10: the use of slots and use of lambda function

- The use of slots is to take input from user one by one, afterwards we will use lambda function to take all inputs and responds the needful output.
- Let's see how to create the lamda function.
- Go to the AWS console and click on "Lambda" inside the "compute" section.

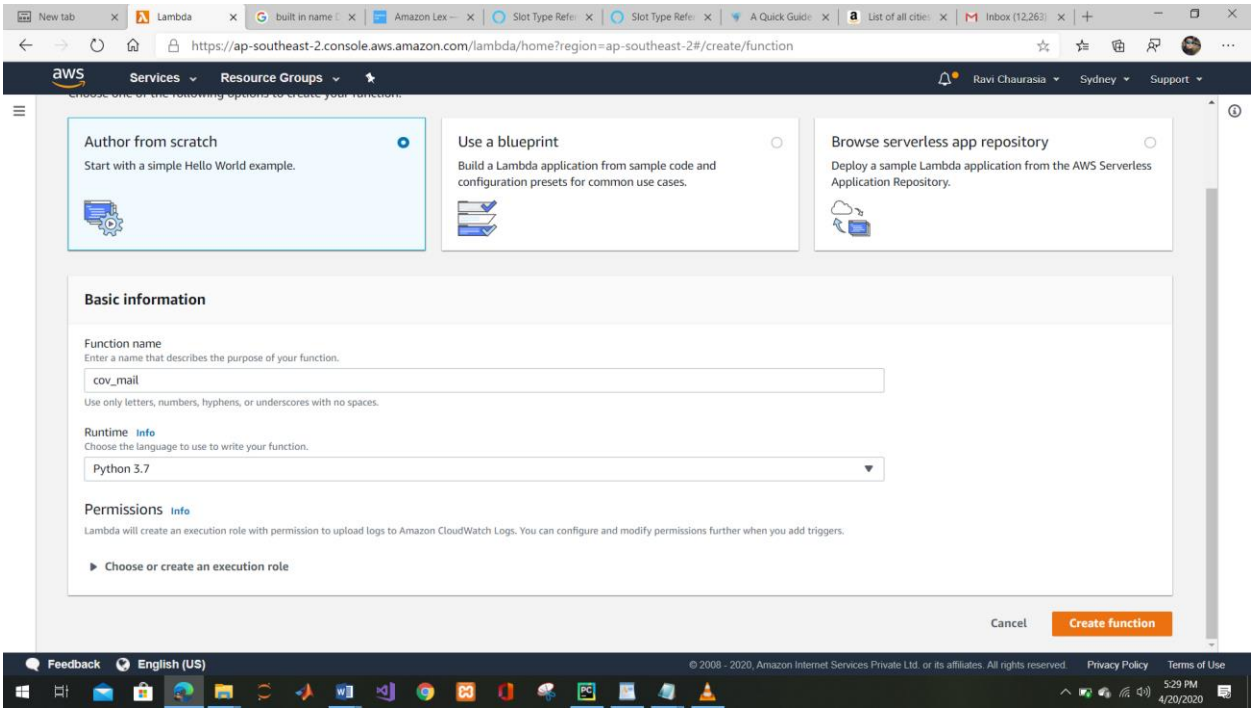


Figure 11 Providing the function name and select runtime

- We have created a function named “cov_mail”. After clicking the “create function” button, you will see a page like this

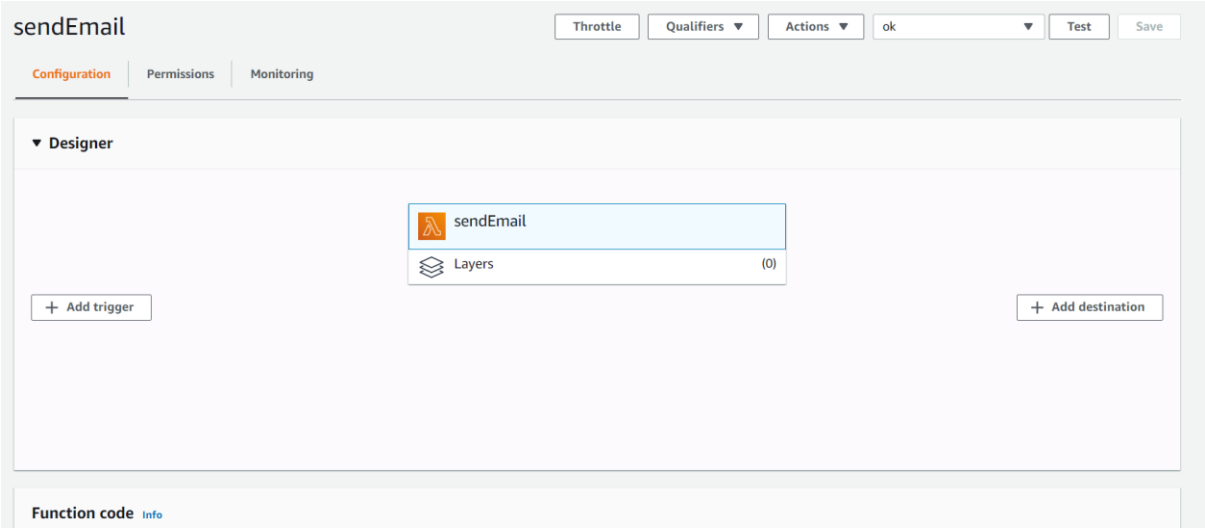


Figure 12: Lambda function page

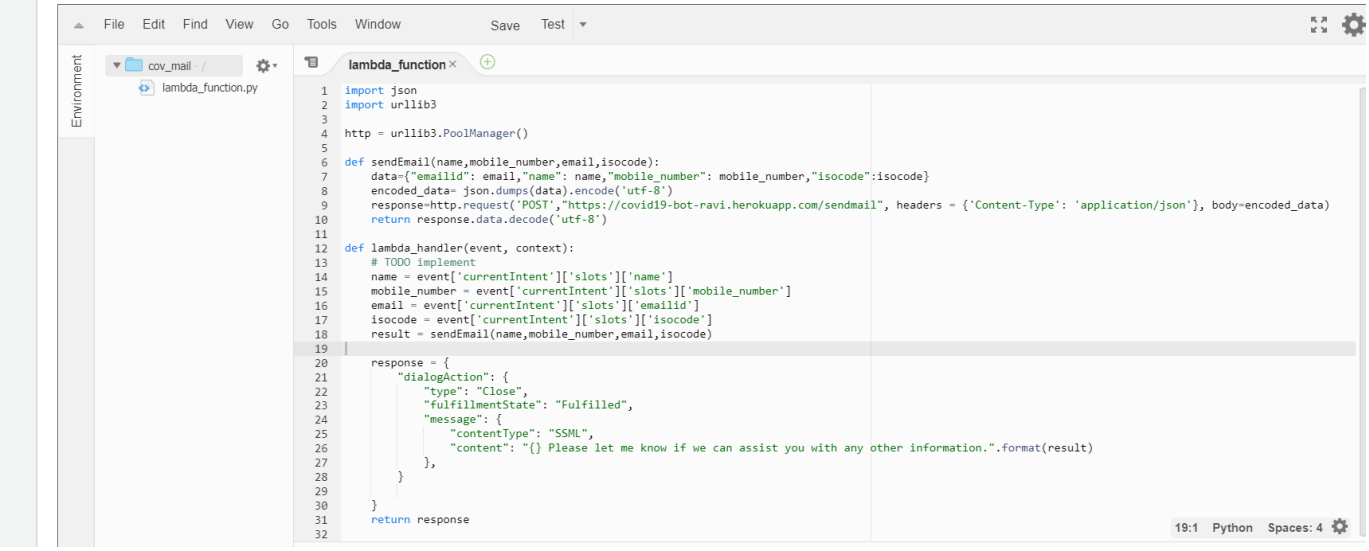


Figure 13: this is how the python code of lambda function will look like

- As you can, in 9th line we have provided the API

So, from where we are taking this API?

- API is generated using Flask

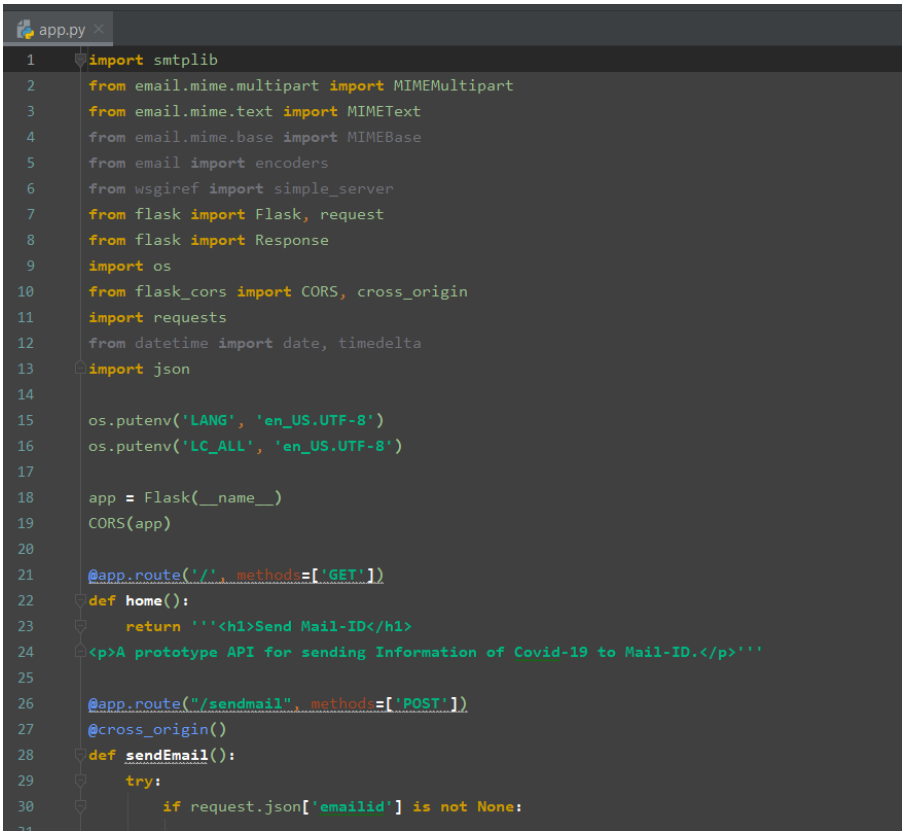


Figure 14: Python code to generate covid-details API

```
    return '''<h1>Send Mail-ID</h1>
<p>A prototype API for sending Information of Covid-19 to Mail-ID.</p>'''

@app.route("/sendmail", methods=['POST'])
@cross_origin()
def sendEmail():
    try:
        if request.json['emailid'] is not None:

            toaddr = request.json['emailid']  ## email id of the user
            contact_add = "[redacted]@gmail.com"   ## the email id of the support team
            fromaddr = "[redacted]@gmail.com"  ## the email id from where we are going to send the mail
            mobile_number = request.json['mobile_number']
            name = request.json['name']
            isocode = request.json['isocode']
##########################################################
            url = "https://covid19india.p.rapidapi.com/getStateData/"+isocode

            headers = {
                'x-rapidapi-host': "covid19india.p.rapidapi.com",
                'x-rapidapi-key': "f8791938c1mshc8f41195c9870fep181866jsn733b0c0182f8"
                }

            response = requests.request("GET", url, headers=headers)
            res1 = json.loads(response.text)
            # res1['response']['active']
            state= res1['response']['name']
            confirmed= res1['response']['confirmed']
            confirmed=str(confirmed)
            recovered= res1['response']['recovered']
            recovered=str(recovered)
            deaths=res1['response']['deaths']
```

Figure 15 Python code continue..

```
    confirmed=str(confirmed)
    recovered= res1['response']['recovered']
    recovered=str(recovered)
    deaths=res1['response']['deaths']
    deaths=str(deaths)
    active= res1['response']['active']
    active=str(active)

    body2 = "State {0} have total number of confirmed cases {1} where active cases are {2}." \
            "Recoverery count {3} and death toll rises to {4} due to covid-19 till present.".format(state, confirmed,active, recovered,deaths)

#####################################################

    # instance of MIMEMultipart
    msg = MIMEMultipart()

    # storing the senders email address
    msg['From'] = fromaddr

    # storing the receivers email address
    msg['To'] = ",".join(toaddr)

#########################################################
    # instance of MIMEMultipart
    msg = MIMEMultipart()
    # storing the subject
    msg['Subject'] = "Query For Covid-19 Cases"

    # attach the body with the msg instance
    msg.attach(MIMEText(body2, 'plain'))
```

Figure 16 Python code continue..

```
    msg = MIMEMultipart()
    # storing the subject
    msg['Subject'] = "Query For Covid-19 Cases"

    # attach the body with the msg instance
    msg.attach(MIMEText(body2, 'plain'))

##########################################
    # creates SMTP session
    s = smtplib.SMTP('smtp.gmail.com', 587)

    # start TLS for security
    s.starttls()

    # Authentication
    s.login(fromaddr, "[redacted]")# give your password here

    # Converts the Multipart msg into a string
    text = msg.as_string()
    # sending the mail
    s.sendmail(fromaddr, toaddr, text)

    # send mail with username , mobile number and email id to concerned team

    # instance of MIMEMultipart
    msg1 = MIMEMultipart()
    # storing the subject
    msg1['Subject'] = "Query For Covid-19 Cases"

    # string to store the body of the mail
    body1 = "person with name {0} have some queries regarding Covid-19 cases." \
            "Please reach to {0} at his mobile number {1} and email id {2}.".format(name,mobile_number,toaddr)
```

Figure 17 Python code continue..

app.py | requirements.txt

```
                "Please reach to {0} at his mobile number {1} and email id {2}.".format(name,mobile_number,toaddr)

                # attach the body with the msg instance
                msg1.attach(MIMEText(body1, 'plain'))

                text = msg1.as_string()
                # sending the mail
                s.sendmail(fromaddr, contact_add, text)

                # terminating the session
                s.quit()

                return Response(body2)

        except ValueError:
            return Response("Error Occurred! %s" %ValueError)
        except KeyError:
            return Response("Error Occurred! %s" %KeyError)
        except Exception as e:
            return Response("Error Occurred! %s" %e)

    #port = int(os.getenv("PORT"))
    if __name__ == "__main__":
        # host = '0.0.0.0'
        # port = 5000
        # httpd = simple_server.make_server(host, port, app)
        # print("Serving on %s %d" % (host, port))
        # httpd.serve_forever()
        app.run(debug=True)
```

Figure 18: Python code continue..

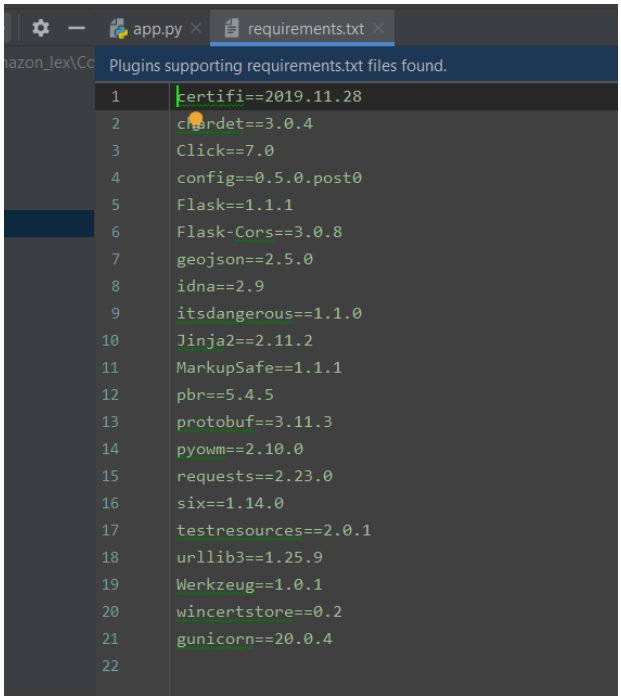


Figure 19: Requirements.txt file

- Ones API is generated, we are going to connect with the Facebook messenger

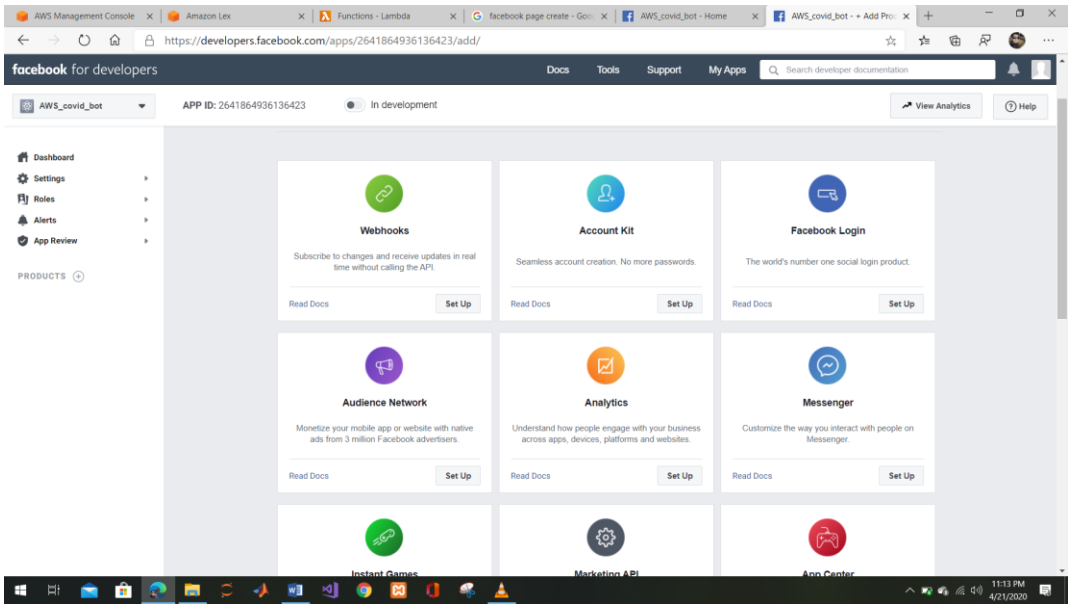


Figure 20: various option for channel integration

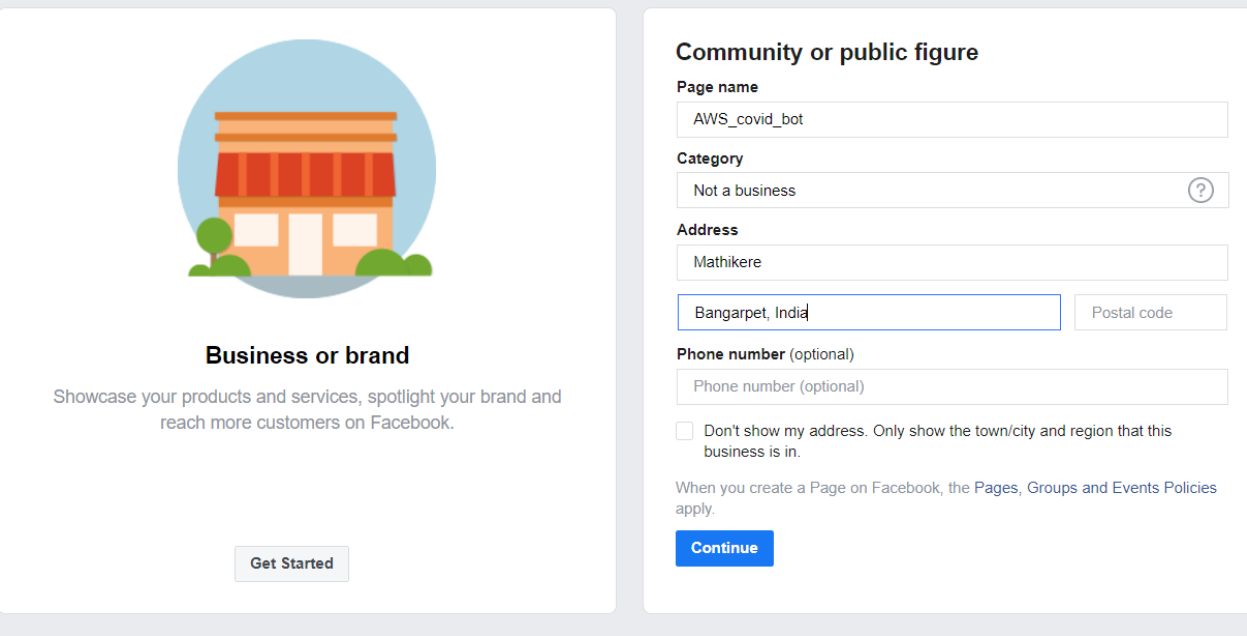


Figure 21: Creating page in Facebook

- Provide a relevant page information

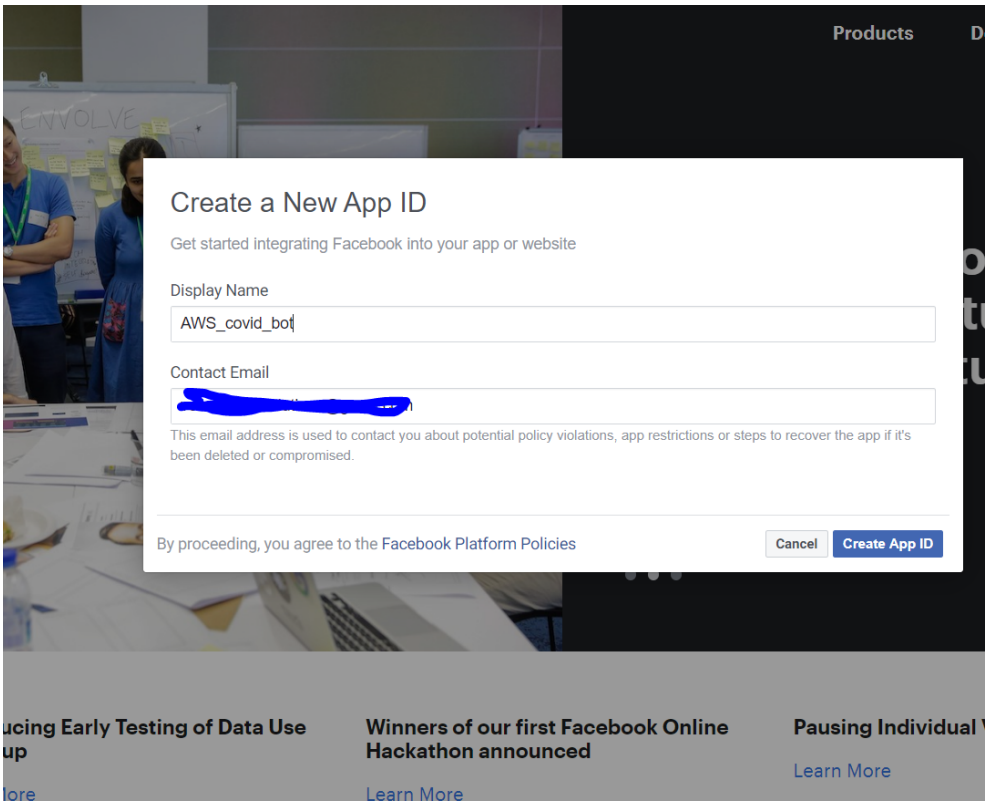


Figure 22: provide mail-id and click to create App Id

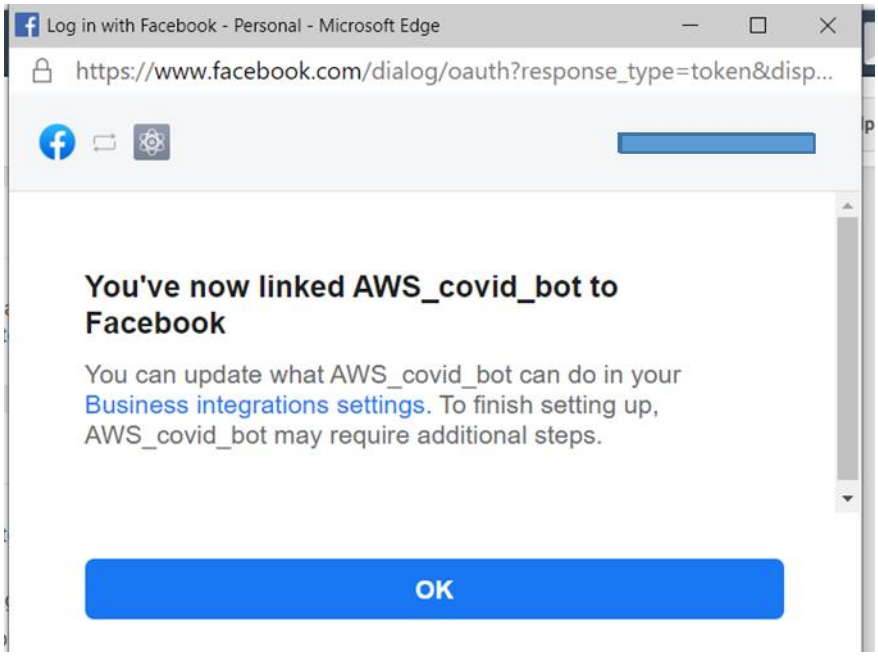


Figure 23: Now your Facebook profile is linked to created page

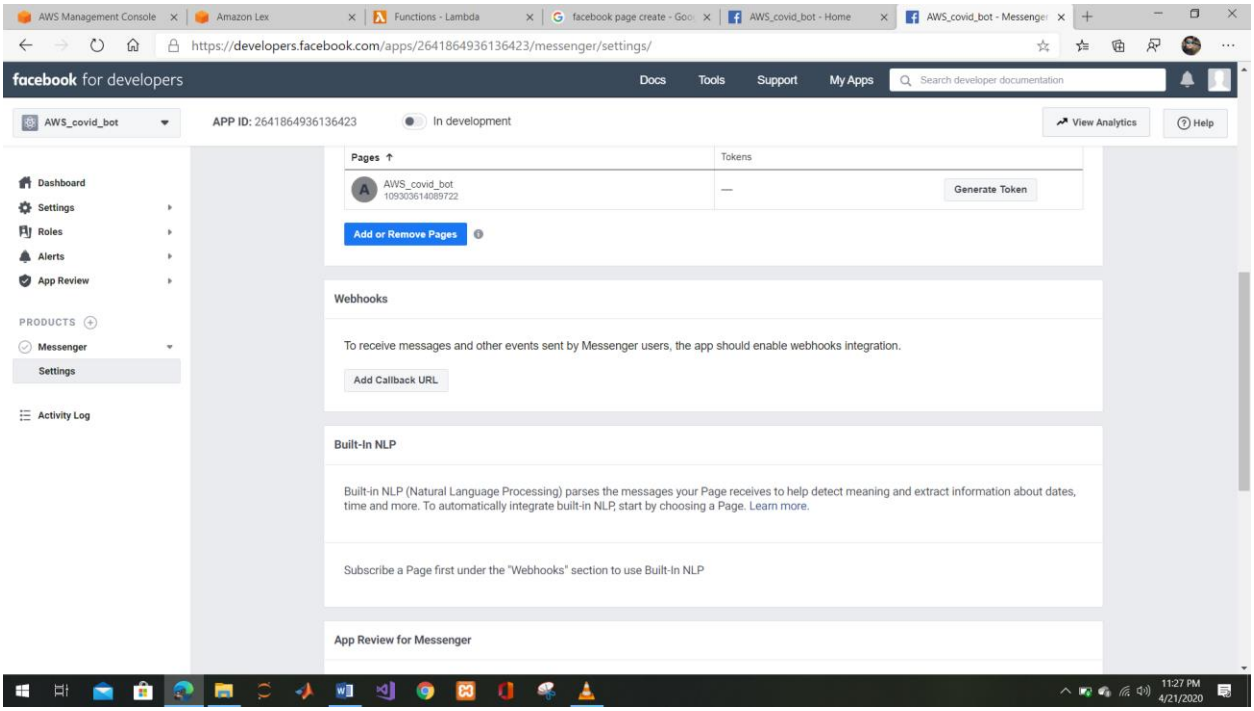


Figure 24: Go to setting folder in Facebook developer page and generate token

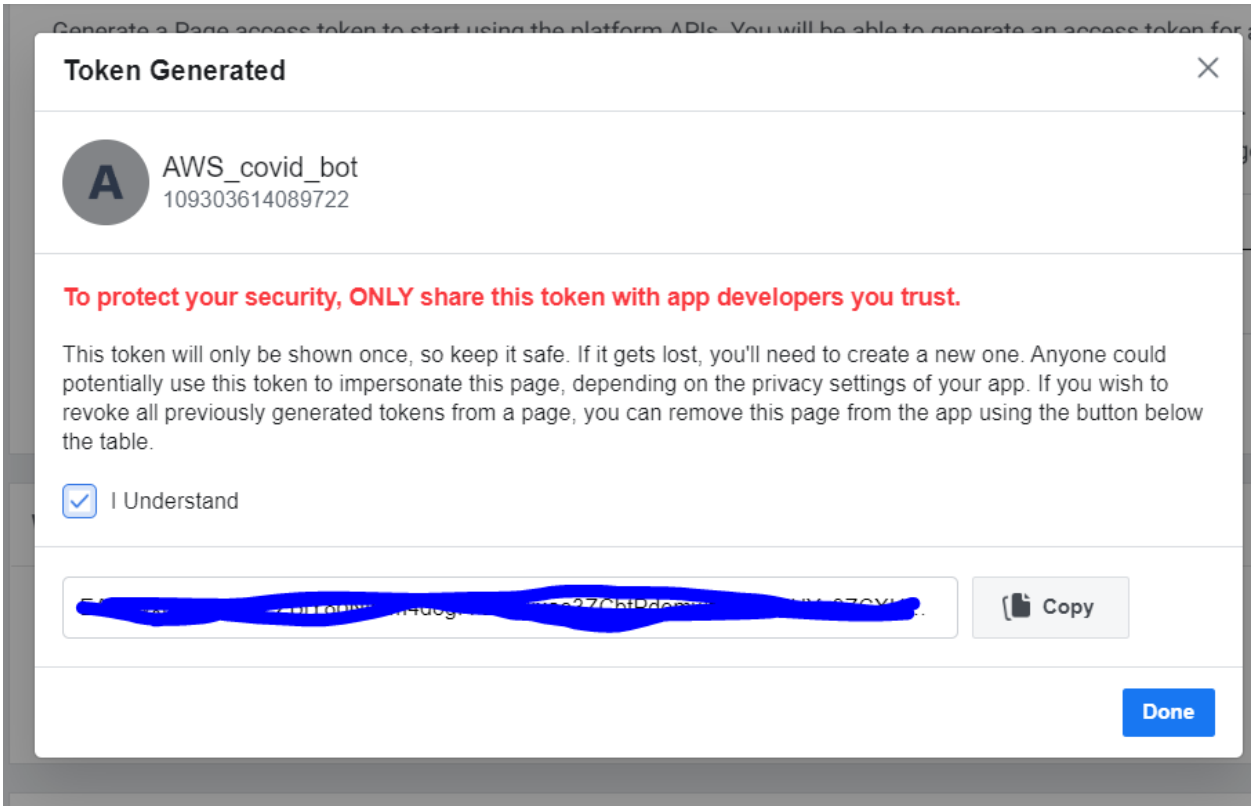


Figure 25: Copy the generated Token

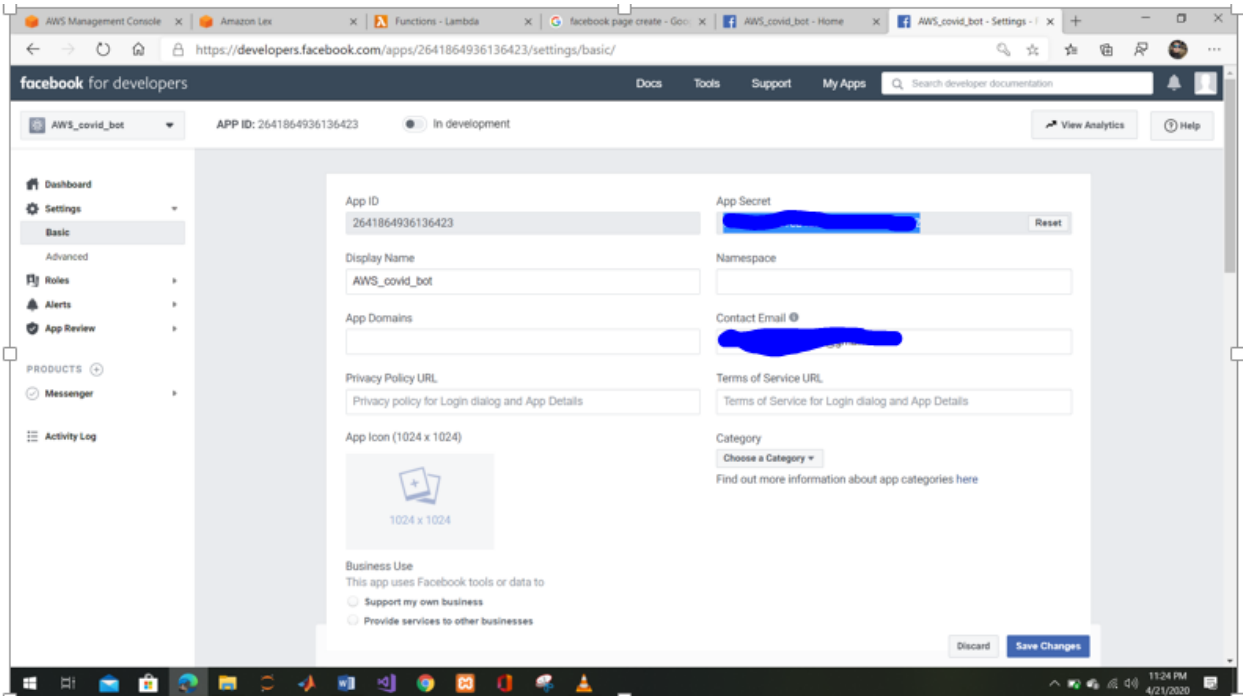


Figure 26: Copy app secret key from setting/basic

- Go to AWS bot channels page and paste both copied app generated Token and secret key

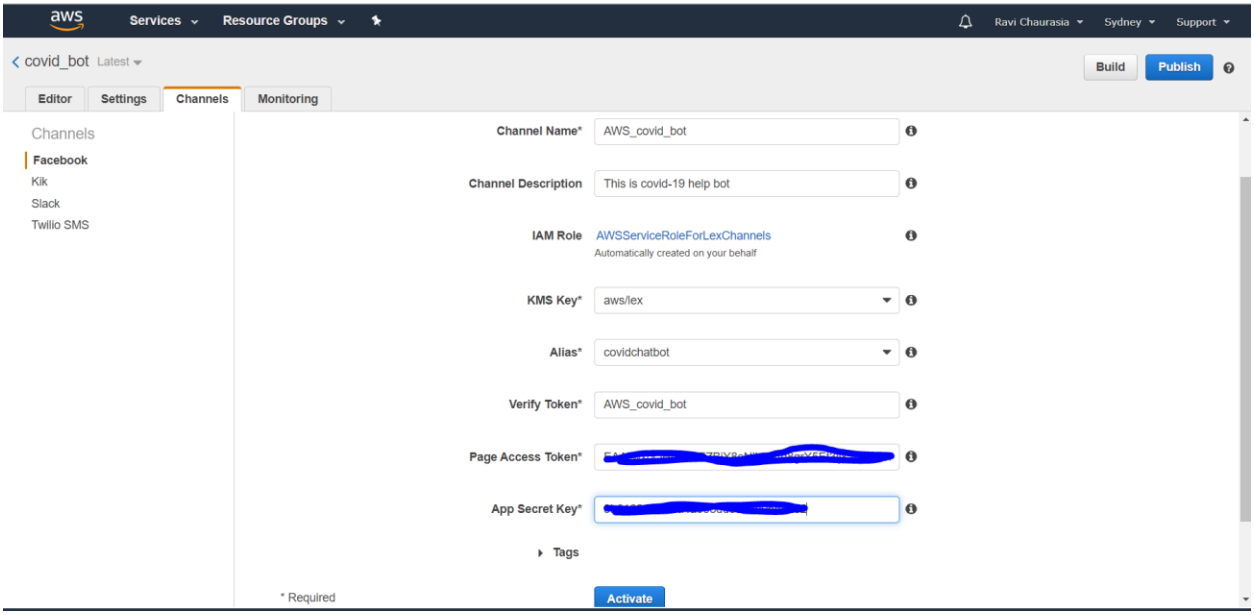


Figure 27: After filling credential, ***just remember alias name which we will going to use further***

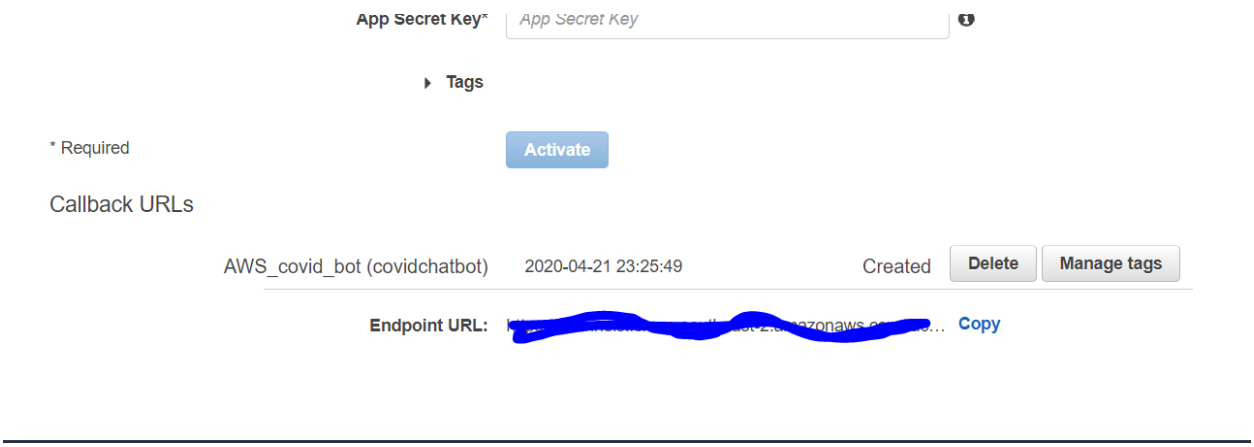


Figure 28: After activate, we will get the Endpoint URL

- Copy Endpoint URL and go back to the Facebook developer page as shown in figure 24, and click webhook url to connect

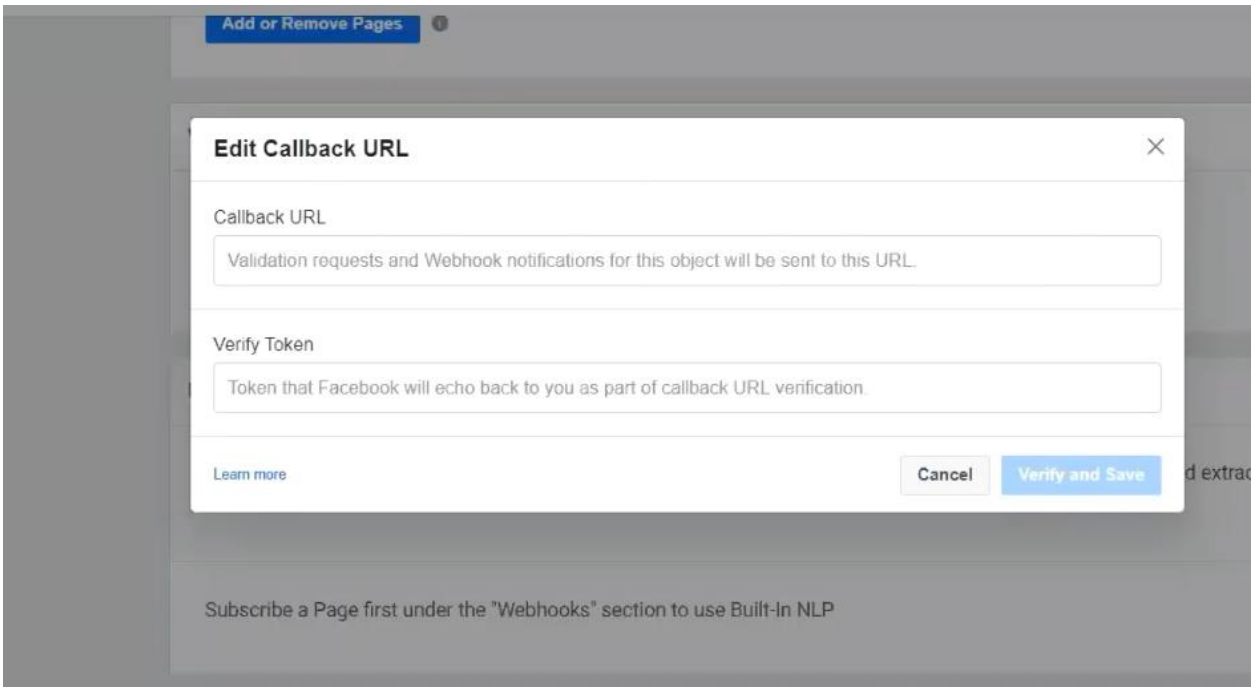


Figure 29: Provide the credential

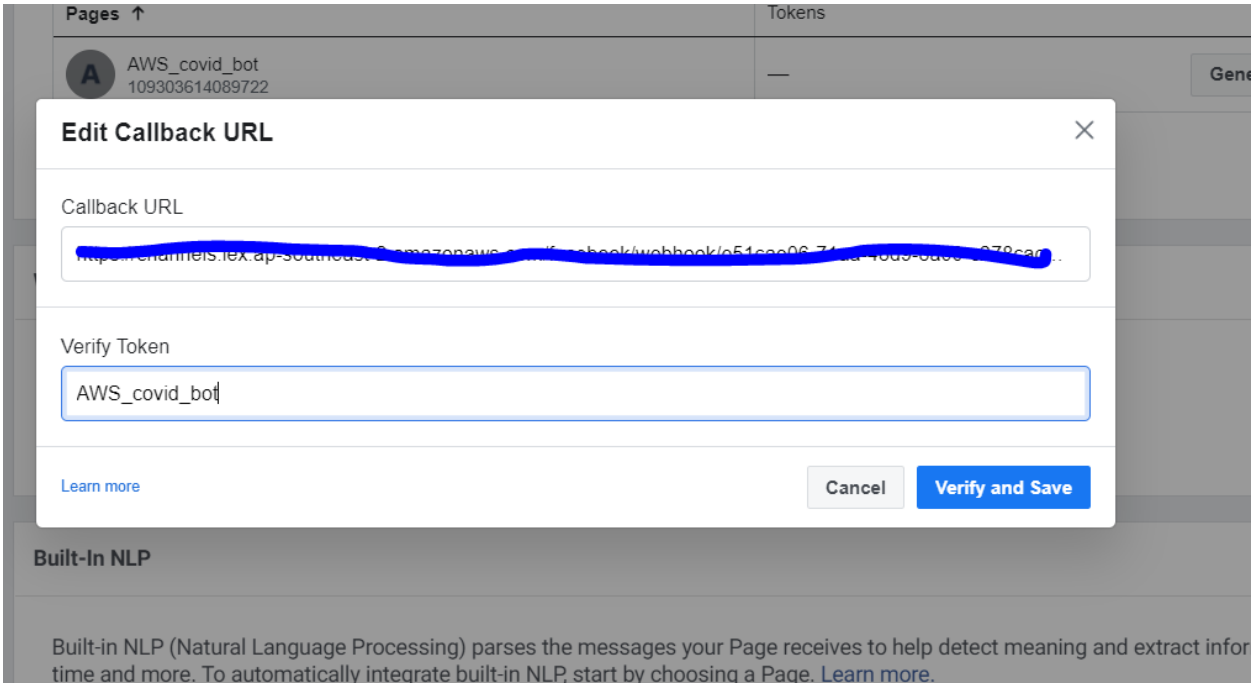


Figure 30: The URL and verify token is from AWS Lex page (Refer Fig.27)

- Ones its done go to the Facebook page and do add subscription

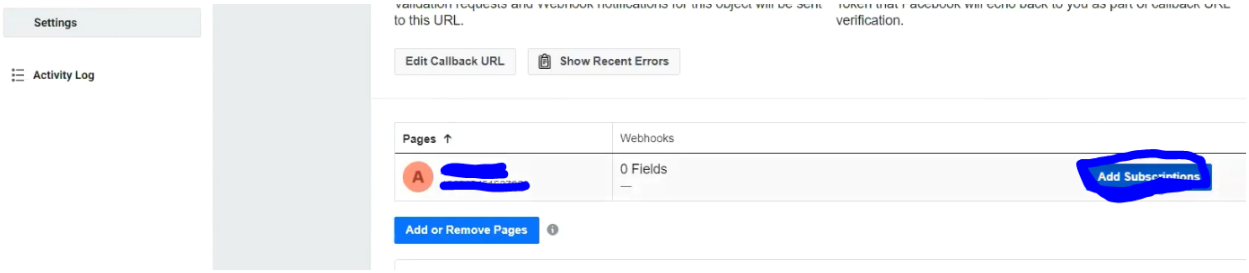


Figure 31: Click to Add subcriptions

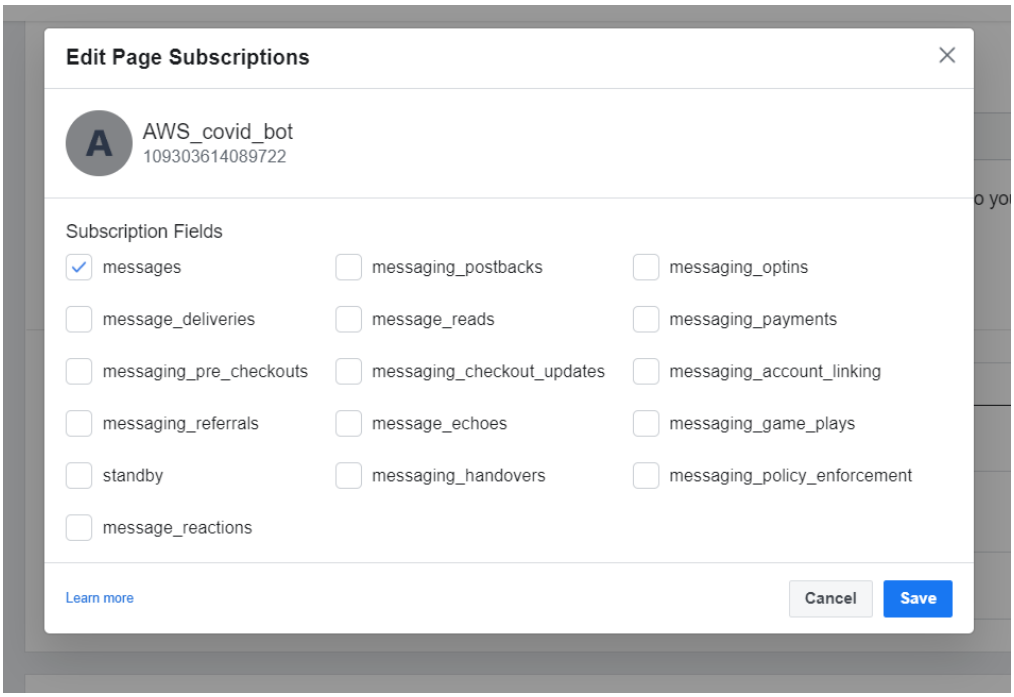


Figure 32: select messages and save

- Go to your Facebook page, you will see add bottom option, click on that

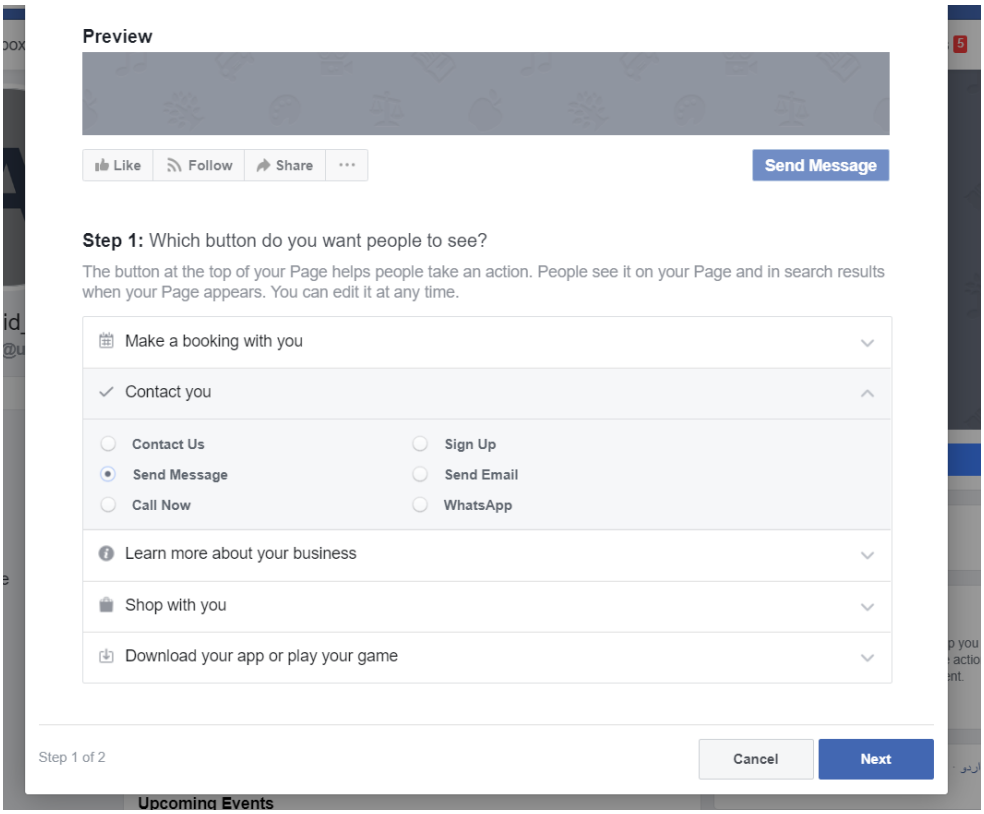


Figure 33: Select send messages and next

- Afterwards select messenger option and next

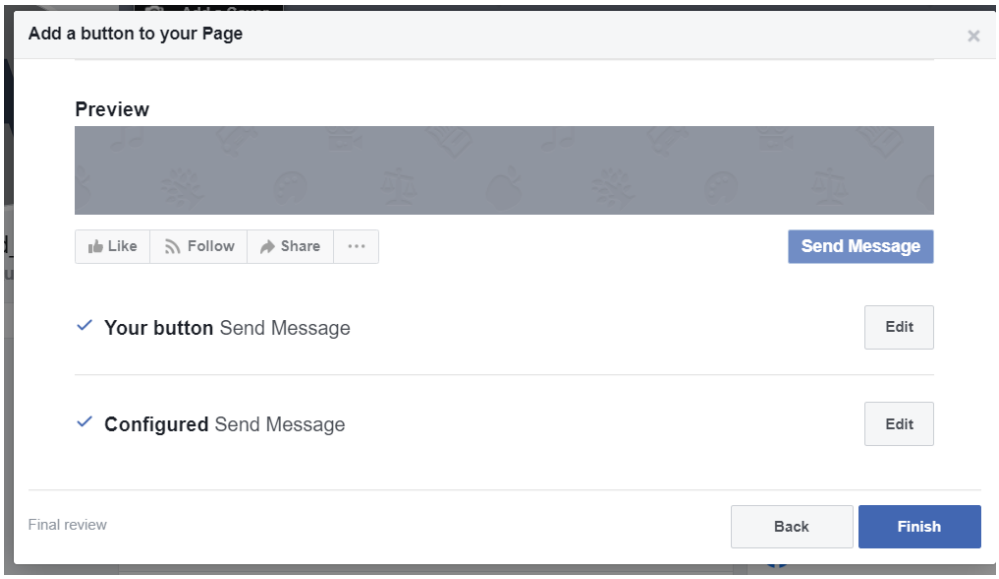


Figure 34 Click Finish

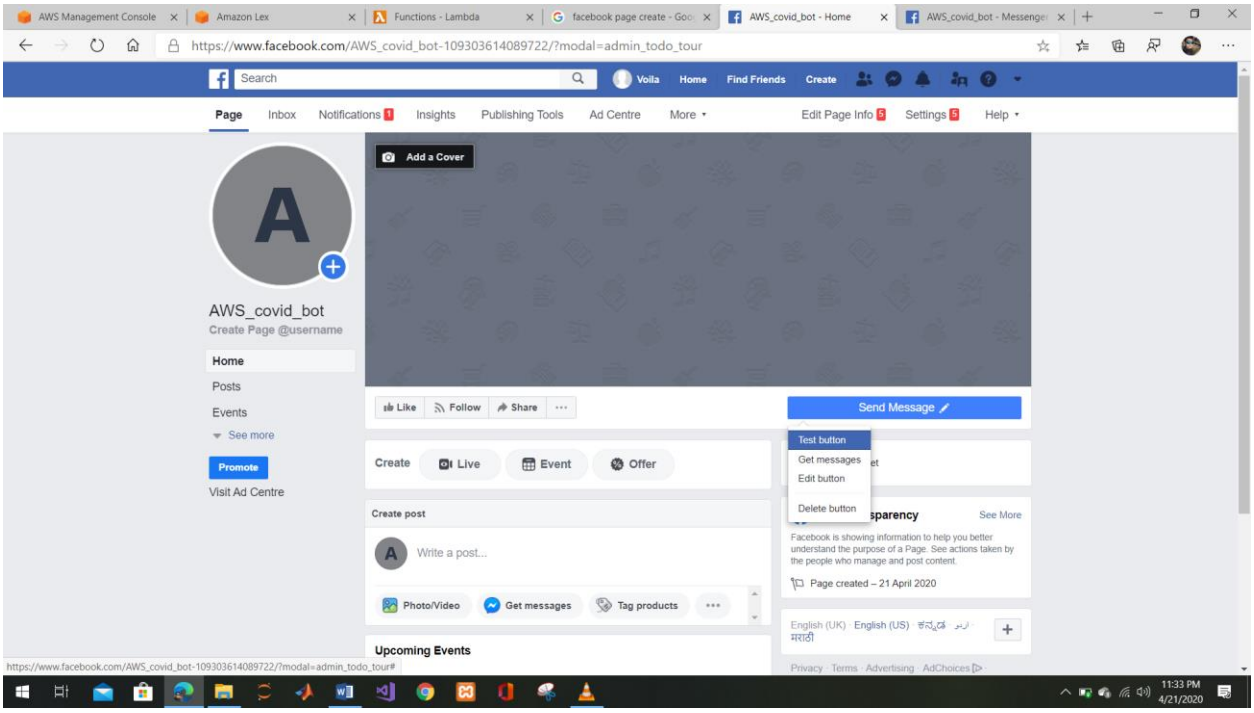


Figure 35: Now rover your mouse at send message button, you will see test button and click that

- Ohhh yaaa
- Your bot is ready for chat

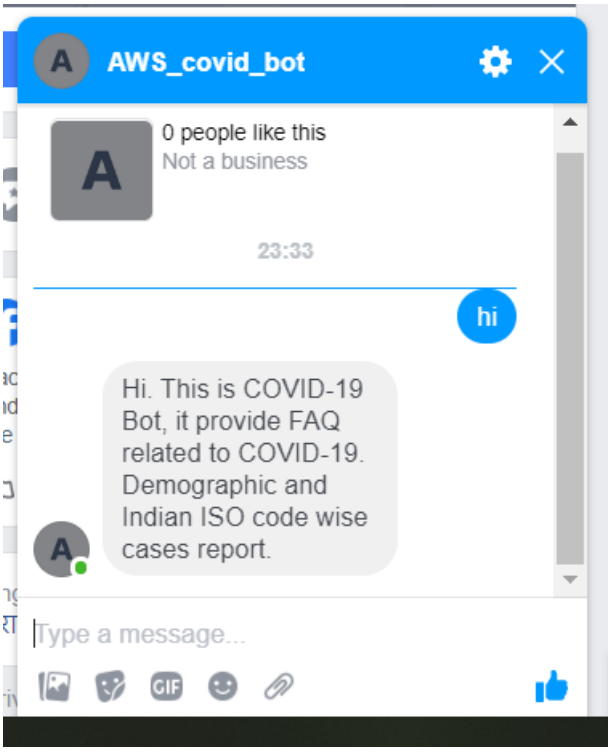


Figure 36: hi will consider as welcome intent

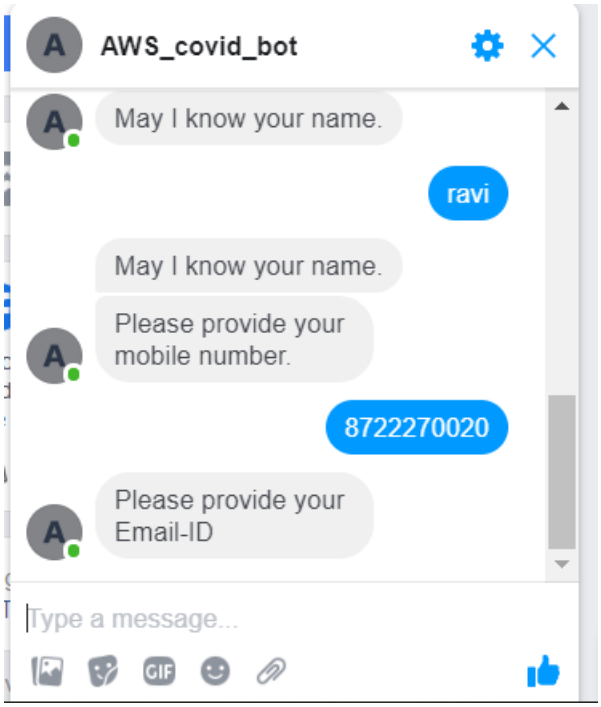


Figure 37 Chat continue..

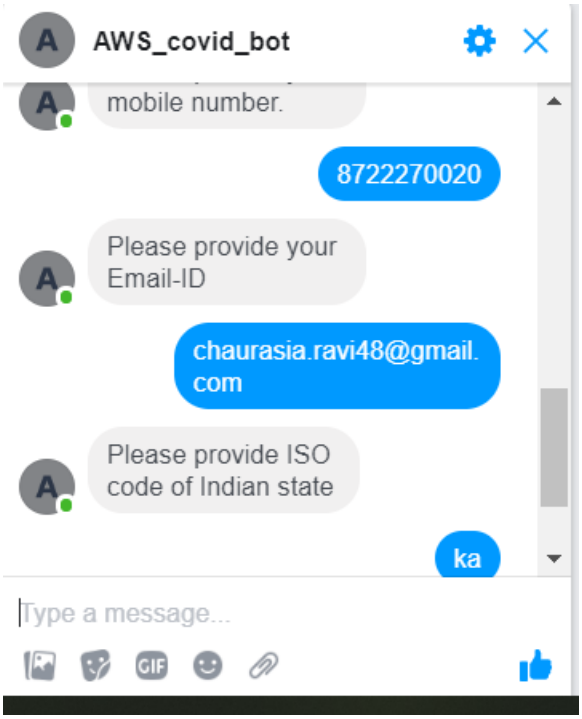


Figure 38 Chat continue..

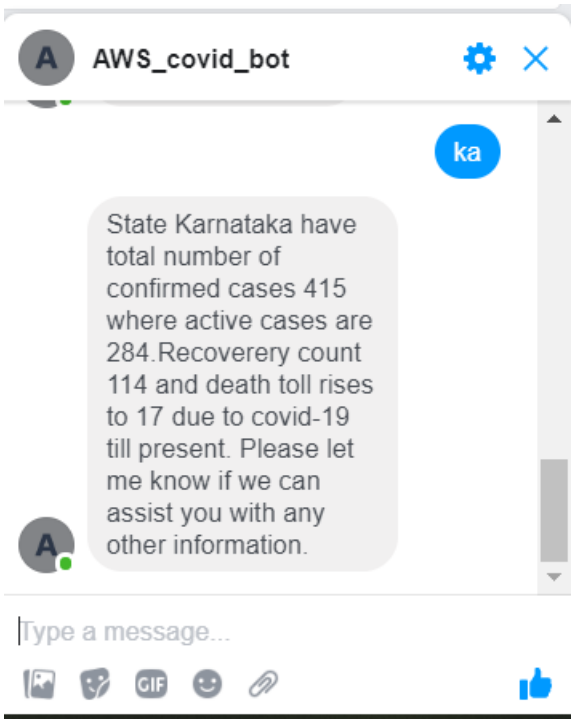


Figure 39: Its showing cases is Karnataka as I have given ka as ISO code

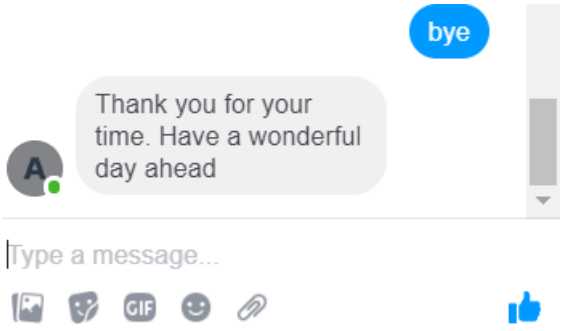


Figure 40: Bye captured as end conversation entent

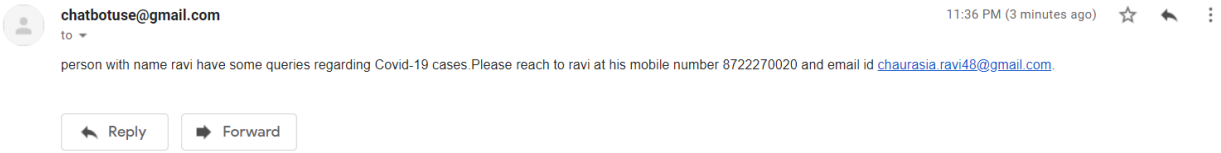


Figure 41: Token file has generated to customer support

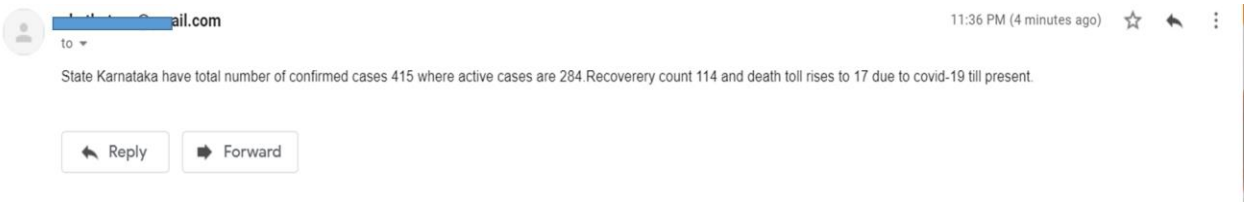


Figure 42: Query mail has sent to customer also

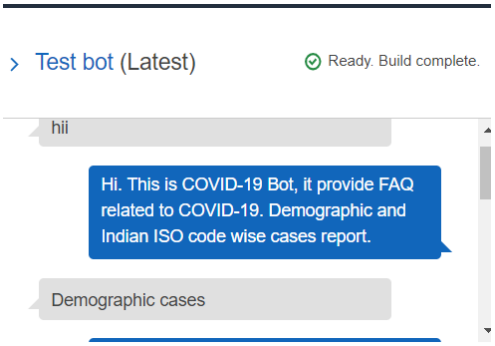


Figure 43: Some other example

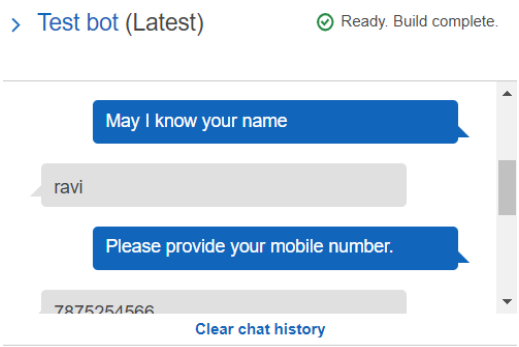


Figure 44 Examples Continue.

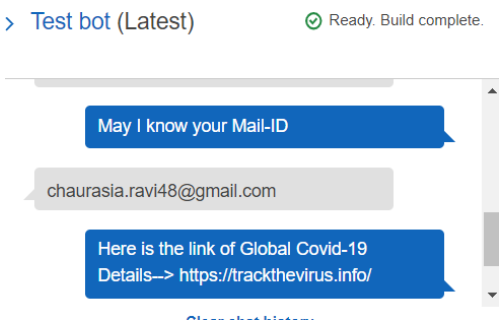


Figure 45 Global covid-19 link has provided

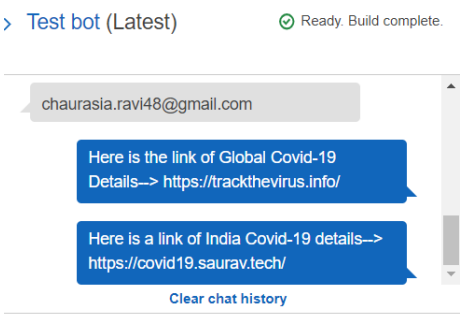


Figure 46: India demographic link

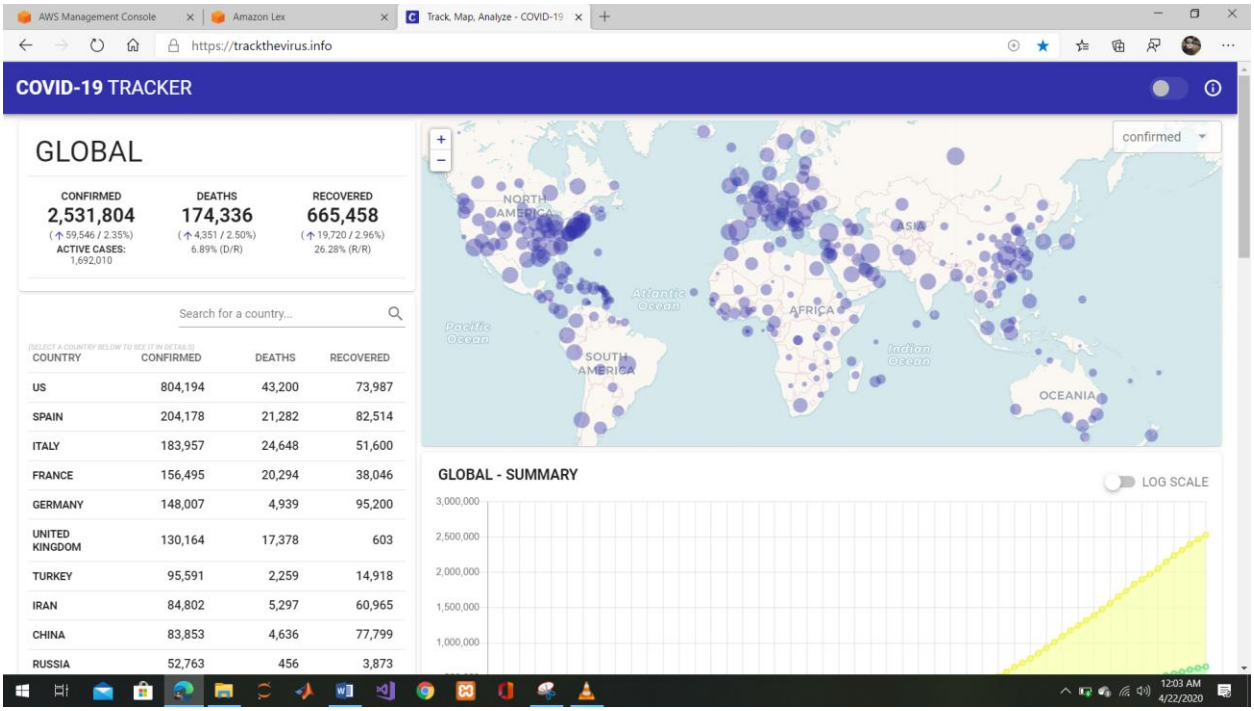


Figure 47: Global covid-19 demographic view

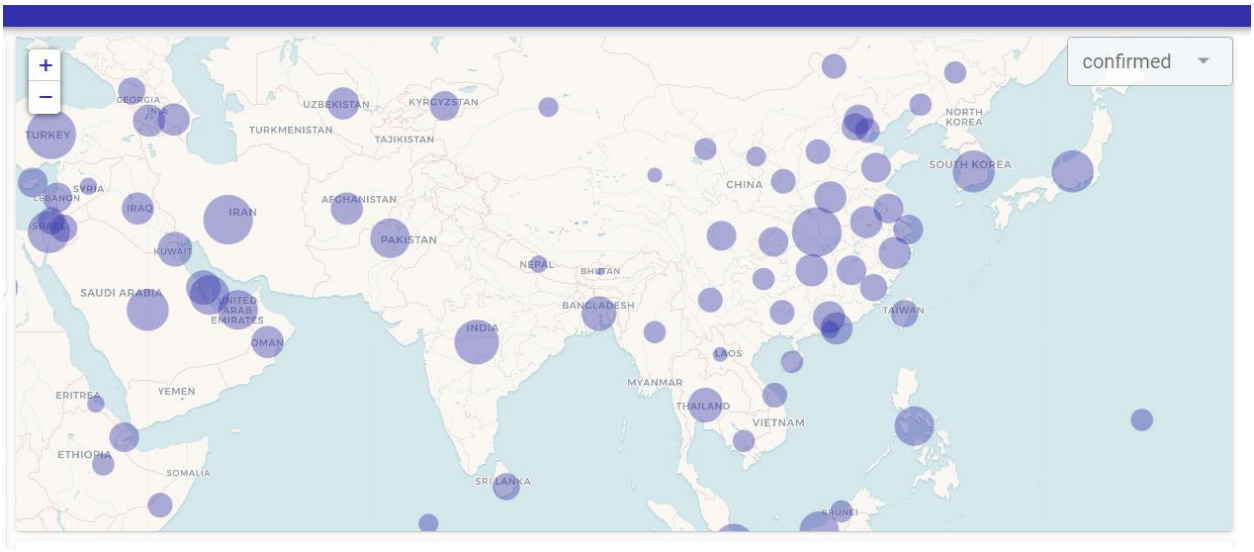


Figure 48: Zoom in to map

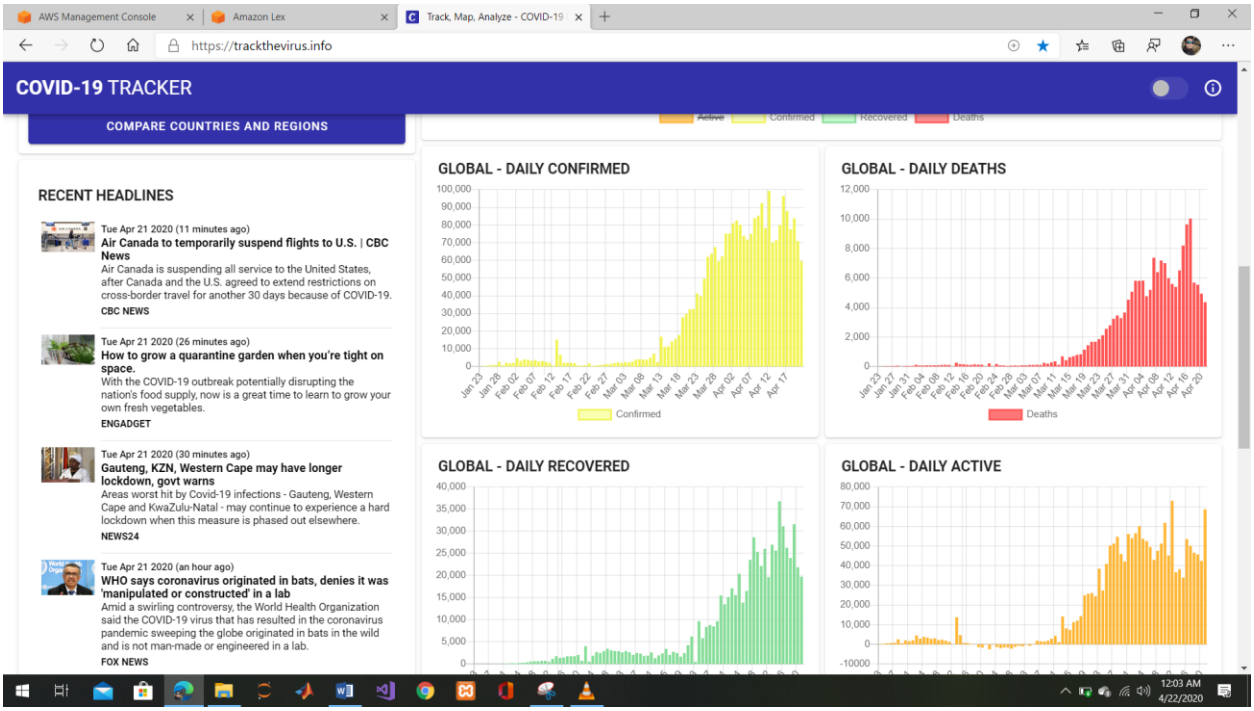


Figure 49: Global covid-19 demographic graph with news

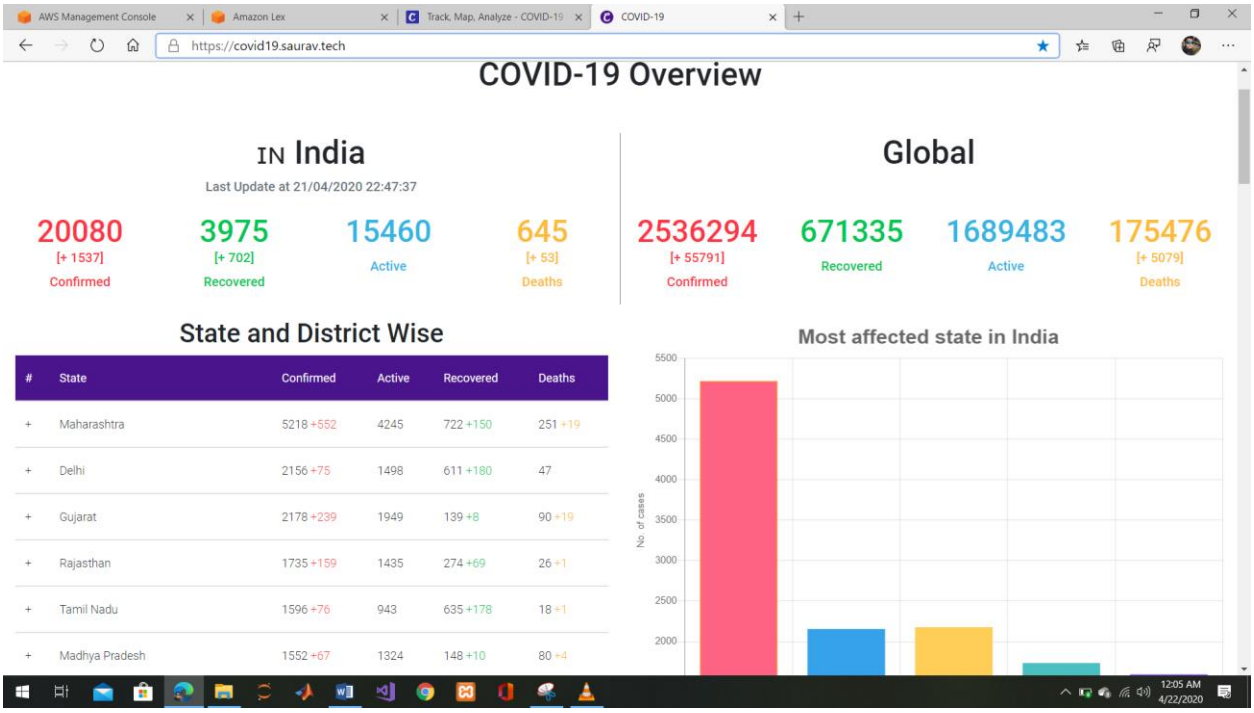


Figure 50: India covid-19 demographic view with numbers

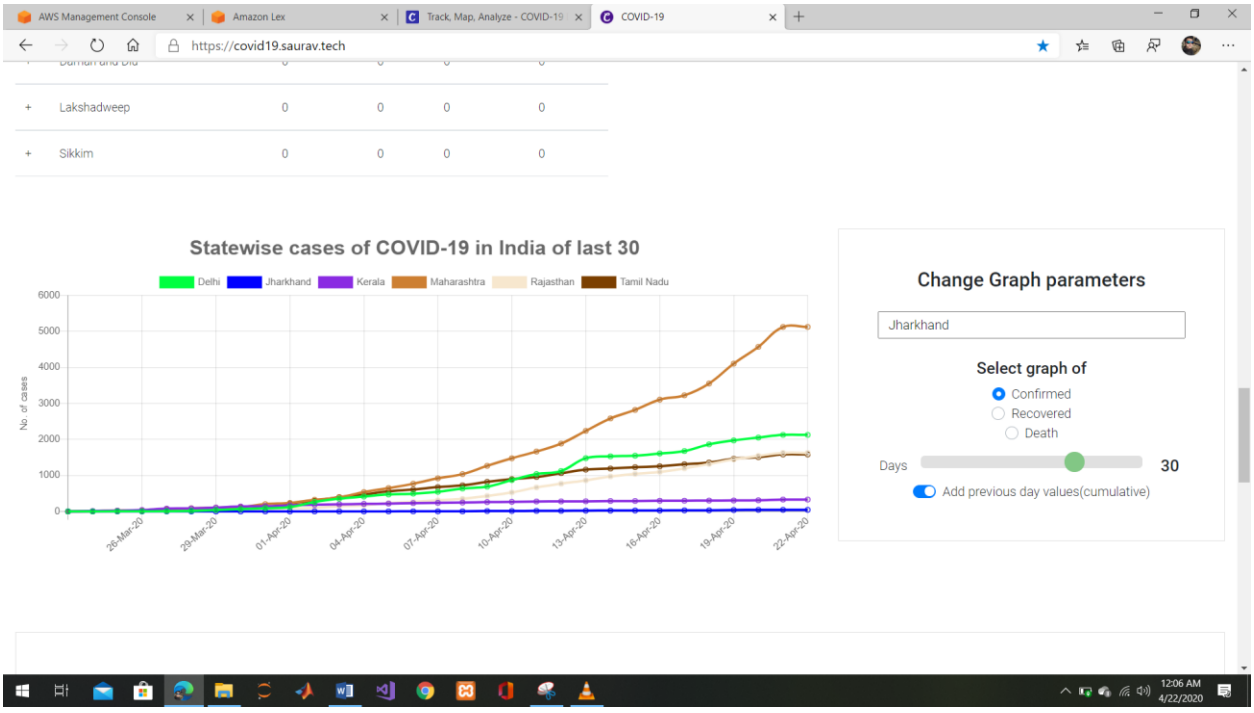


Figure 51 India covid-19 demographic view continues (State-wise)

Thanks for watch