iNeuron Course Selection Chatbot

-by Vignesh Krishnamurthy

**1.Objective**

To develop a chatbot which will interact with the user to provide the course details of iNeuron academy to the user through mail. Also shares the users details and the course that the user is interested to support team through mail for further support.

**1.1 Links**

* Telegram integrated chatbot link: <https://t.me/iNeuron_Coursebot>
* Web demo chatbot link: <https://bot.dialogflow.com/80b8071a-55fb-4b71-9063-41213d7d3542>

**2.Chatbot Concept  
2.1 What are chatbots?** A Chatbot is a software or an agent or a service which simulates human conversation  
in natural language through messaging applications, websites, mobile phones, or telephone.  
They can be programmed to respond to simple keywords or prompts to complex discussions.  
A Chatbot has two different tasks at the core: analyze the request (identifying the intent of the  
user) and providing the response.

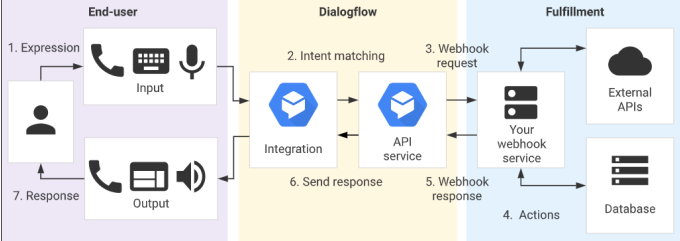


Currently, there are many chatbot frameworks are available in market. For our project, we have used Google Dialog Flow to develop the chatbot.

**2.2 Google Dialog Flow**

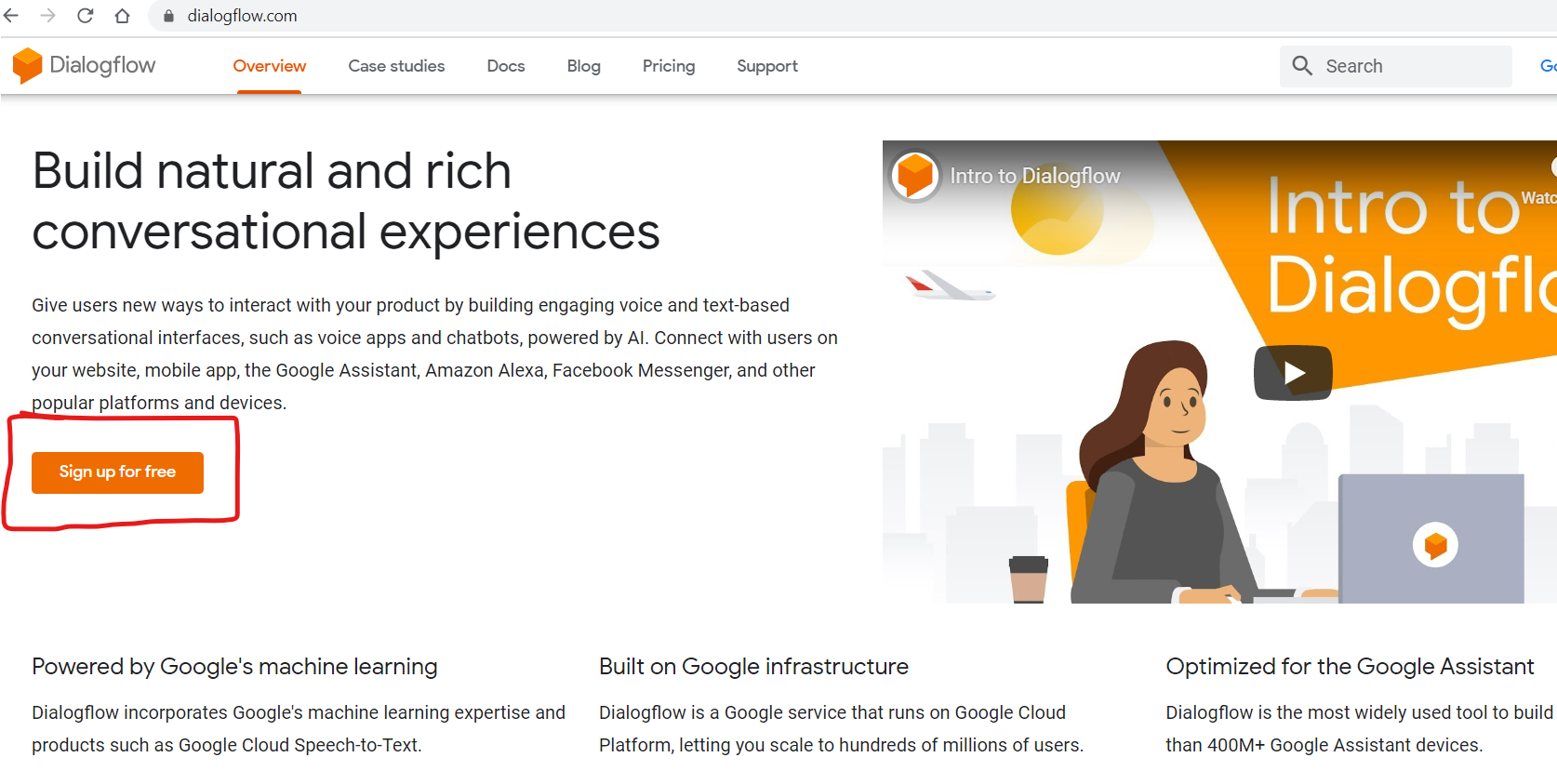
Google Dialogflow is a Google-owned developer of human-computer interaction  
technologies based on natural language conversations. It gives users a new option to interact  
with the product by building voice apps and chatbots powered by

**2.2.1 How do Chatbots work?**

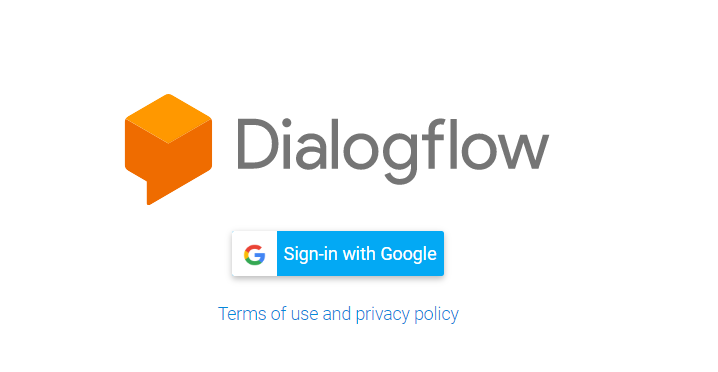


1. The user sends a text or voice message from a device or an App  
2. The App or the Device transfers the data to Dialogflow  
3. The message is categorized and matched to a corresponding intent  
4. We define actions for each intent in fulfillment (Webhook).  
5. When Dialogflow finds a specific intent, the webhook will use external APIs to find a  
response from external databases.  
6. The external databases send back the required information to the webhook.  
7. Webhook sends a formatted response to the intent.  
8. Intent generates actionable data according to different channels.  
9. Data go to output Apps or Devices attached  
10. The user would get a text/image/voice as a response.

**3. Build your first chatbot  
3.1 Signup for Dialogflow account**

**Prerequisite:** A Google account is required for connecting to Google Dialogflow.  
1. Create our Dialogflow account by using the link (https://dialogflow.com/).  
Click on **Sign for Free** and proceed with account creation. 

2. Click on **Sign-in with Google**

****

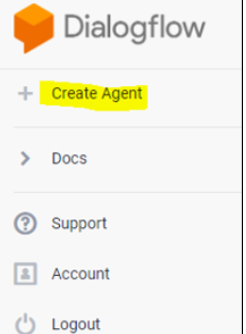
3.Select your google account, and it will direct to the Google Dialogflow home page. Click on Go to console in the upper right corner to navigate to the home page of Google Dialogflow.



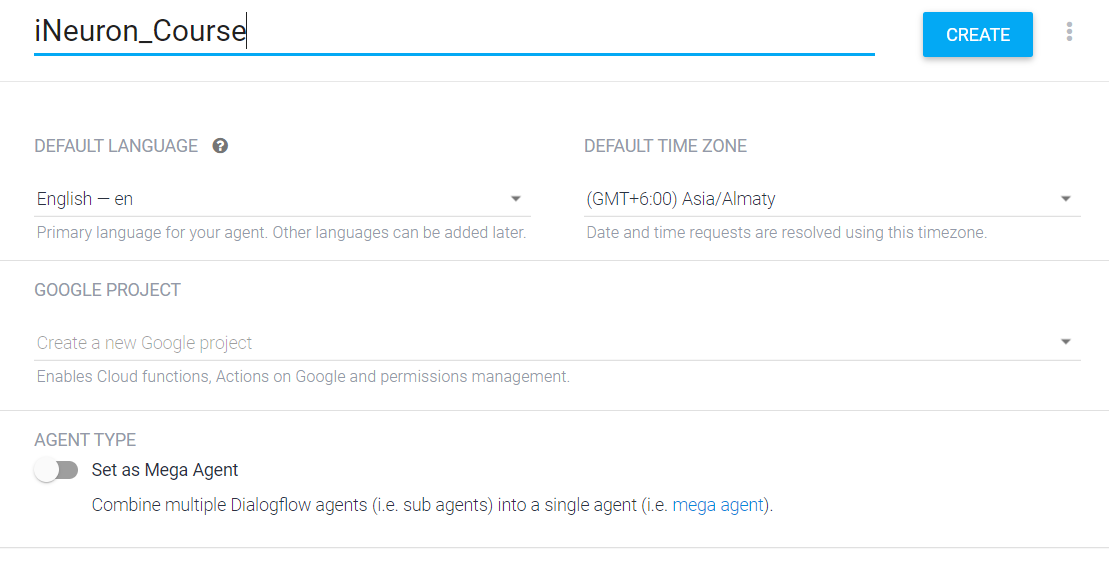
**4. Creating an Agent**

An agent is a virtual agent or bot that handles the conversation with end-users. We can design  
or build a Dialogflow agent to handle the different types of communications required by the system. These can be included in any app, product, or service and transform natural user requests into actionable data

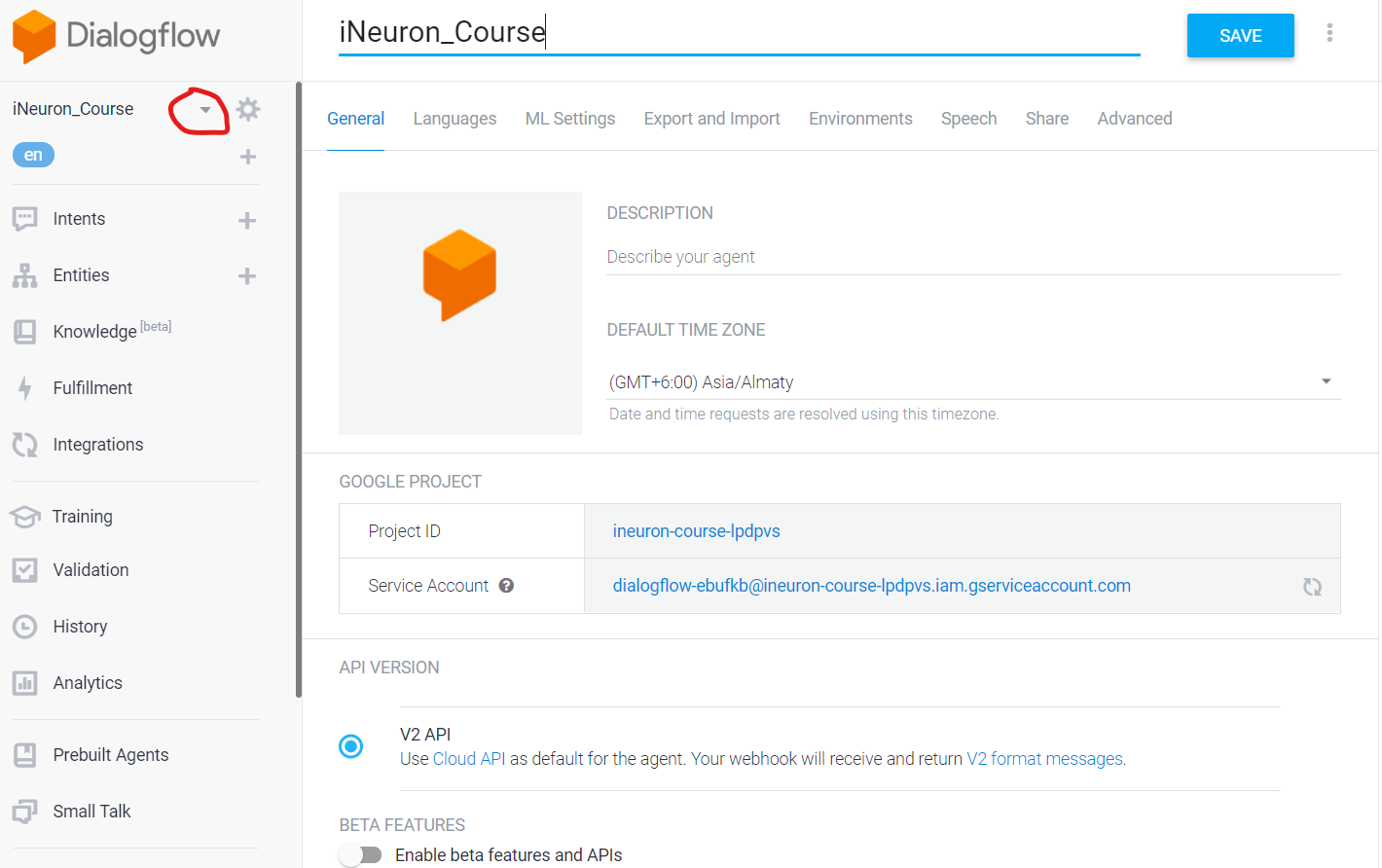
It incorporates Natural Language Processing to understand what the user meant and to figure out what “action” has to be carried out. Agents succeed in conversations with the user through intent,entities, contexts, and other building blocks.  
1. Log in to Dialogflow. Click on **Create Agent** from the left menu.



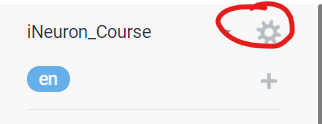
2. Provide the name of the agent and click on the **SAVE** button to create the  
Agent.



3. The Agent gets created and gets listed below the Dialogflow icon. If there are multiple agents, use the dropdown down button to select the Agent for editing or adding new data.



4. Edit the Agent details using the setting icon .

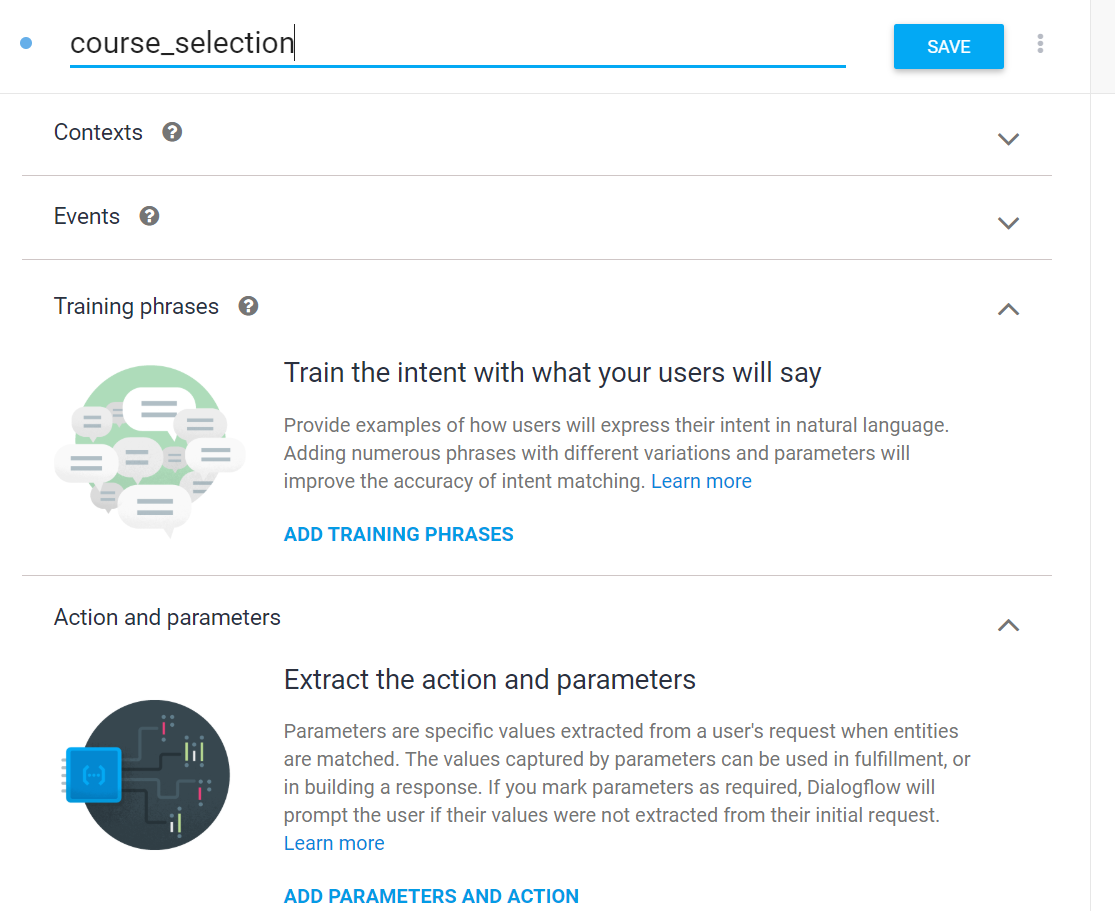


**5.Creating an Intent**

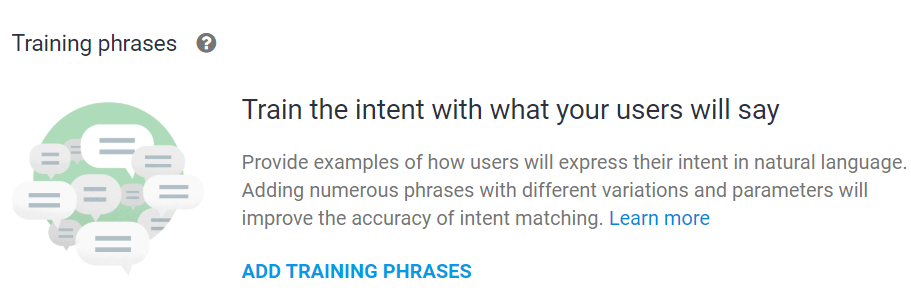
Intents are mappings between a user's queries and actions fulfilled by our software. Intents determine the action to be taken by the code. It is a mapping of what the user says and what our software should do with the user utterance.

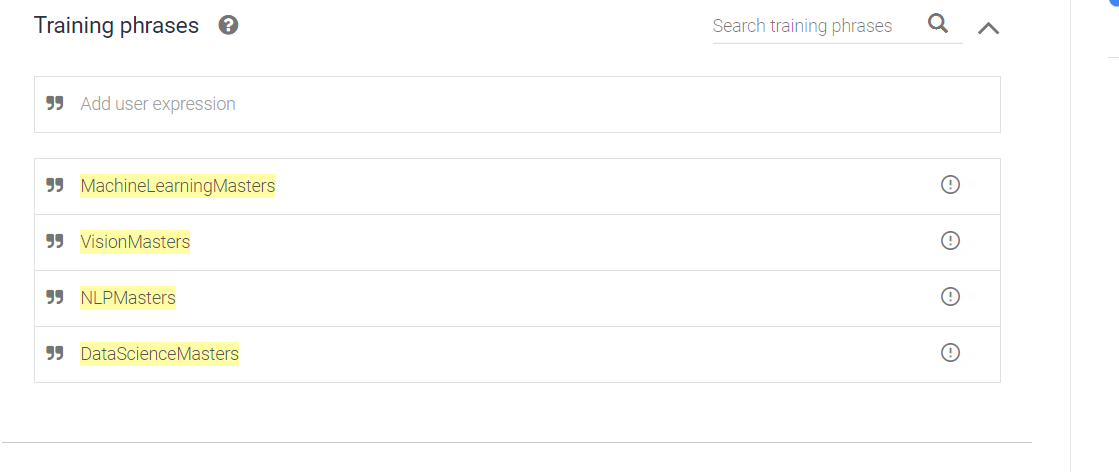
Two types of intent:  
**Default intent**: Dialogflow provides two default intents, namely default Welcome  
Intent (for greetings) and Default Fallback Intent (Default fall through intent when no  
other intents match).  
**Custom intent**: Dialogflow provides options to create customized Intents based on  
business requirements.

1. Click on the plus icon as specified in the below image to create a new intent, which allows you to map what our user says to what our agent responds. Provide a name as course selection and click on the SAVE button.

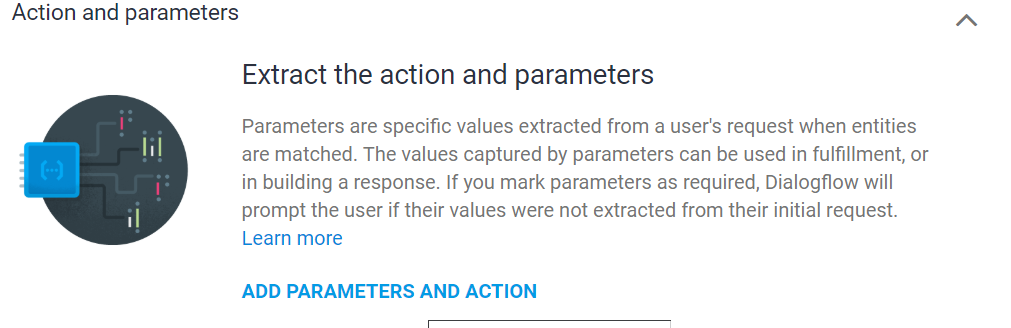


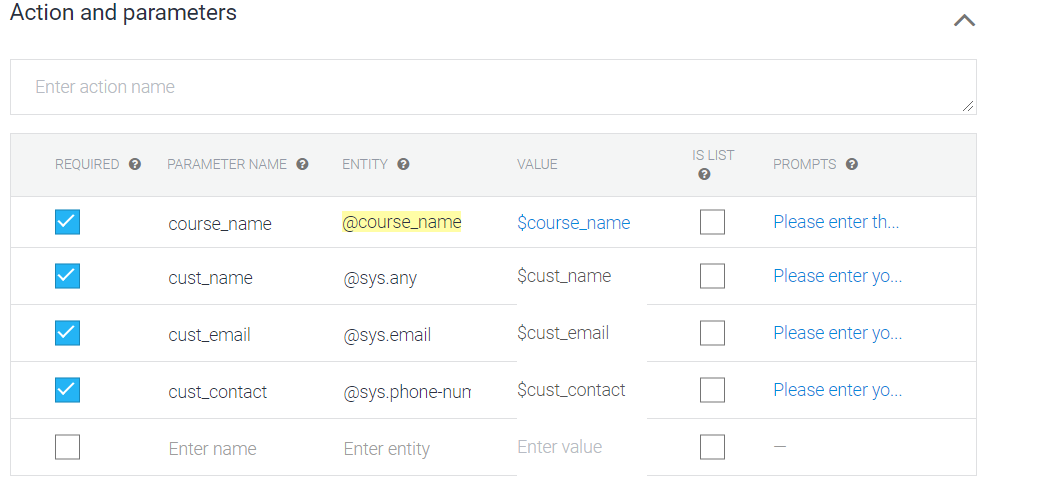
2. Click on **ADD TRAINING PHRASE** and then add **Training phrases**. Training phrases are examples of what users can say to match a particular intent. Adding numerous phrases with different variations and parameters will improve the accuracy of intent matching.





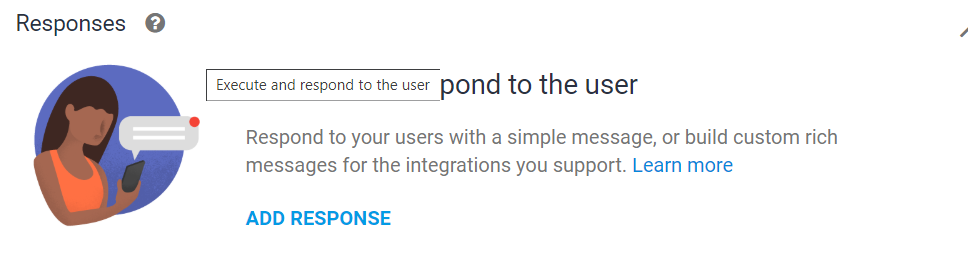
3.click on the ‘ADD PARAMETERS AND ACTION’. They are the inputs we need from the user to act on the user request. e.g., consultation. Examples are dates, times, names, places, and more.



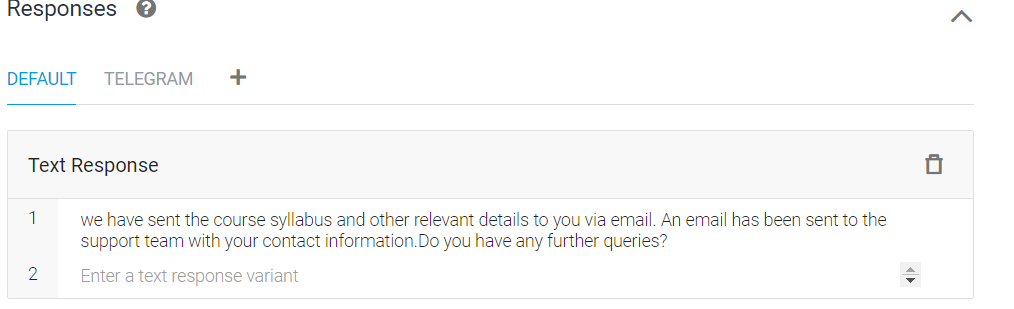


This step involves identifying the purpose of the user conversation and saving the data to improve communications.

4. Click on **ADD RESPONSE** to provide an appropriate reponse to user queries.



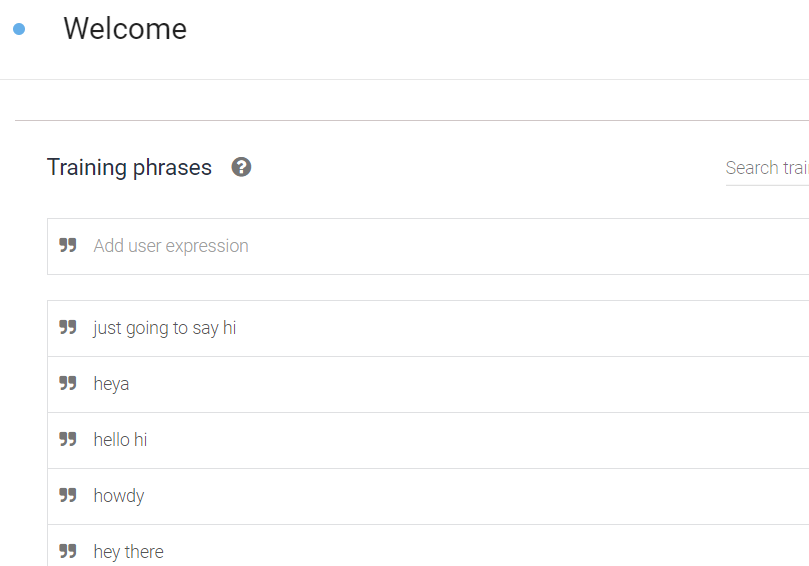
5. Add the below Responses:

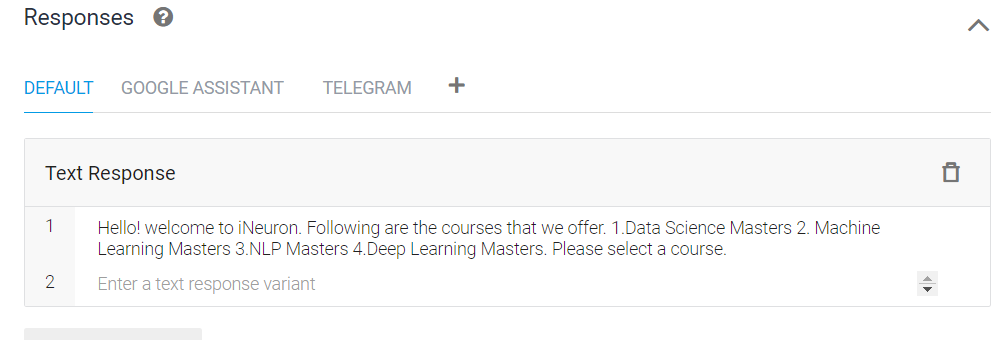


6. Click on the **SAVE** button which is available in the upper right corner to save  
all the details of the intent.

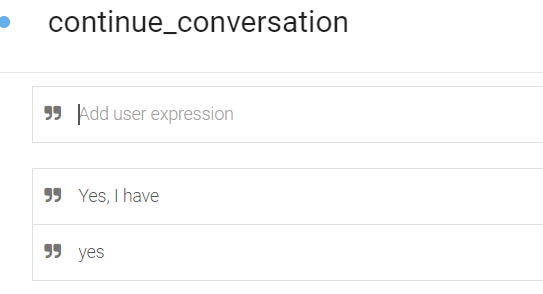
Smilarly add the below intents

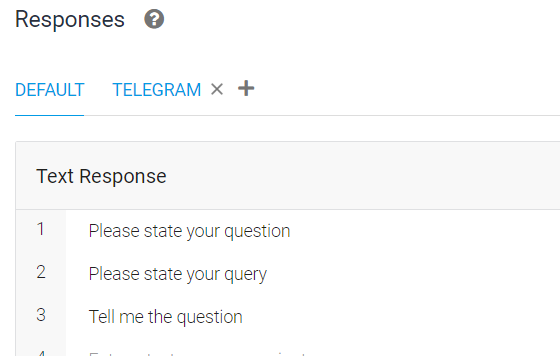
1. **Welcome intent**



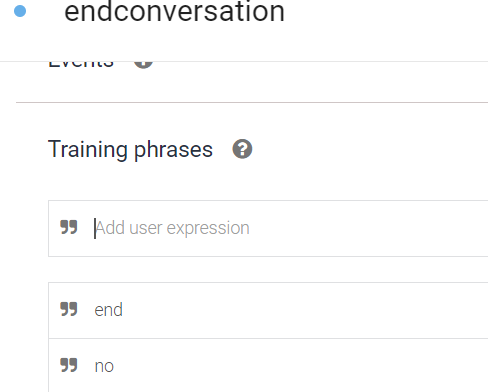


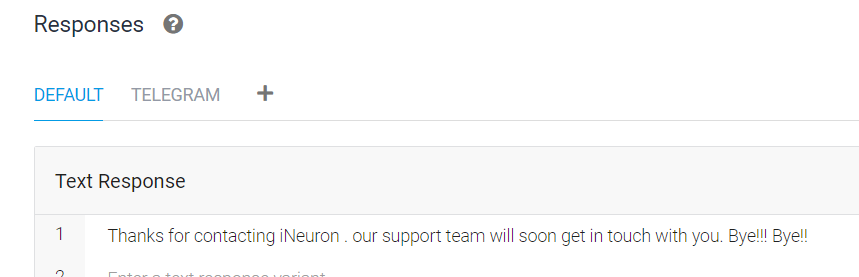
1. **Continue conversation**





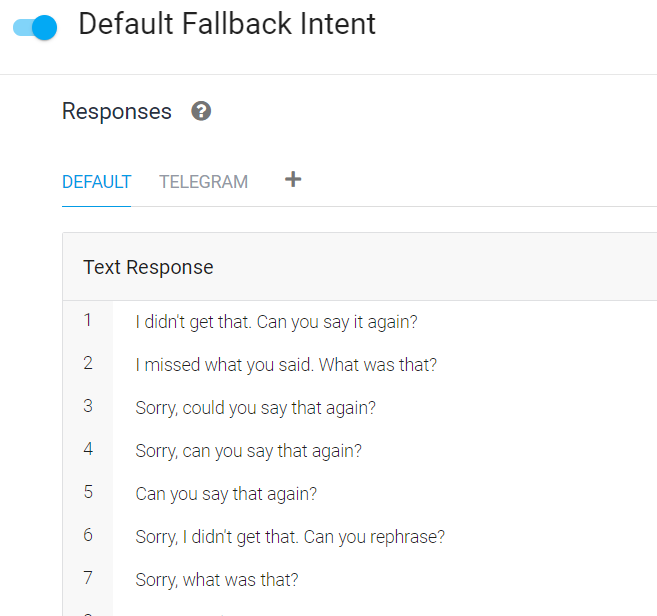
1. **End conversation**





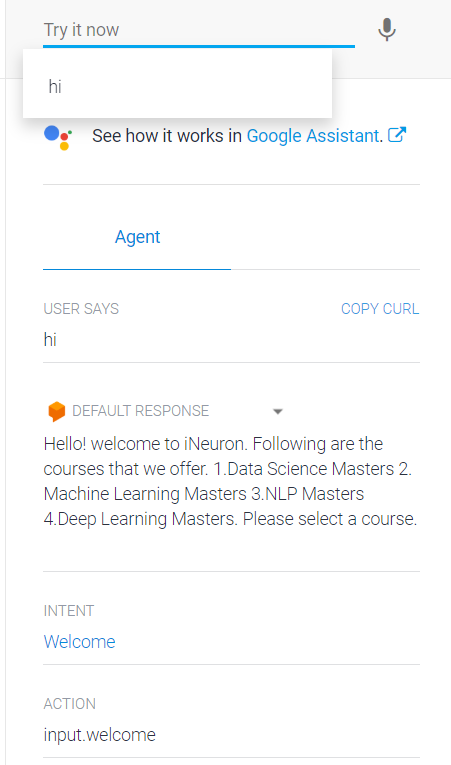
1. **Fallback intent**

Default fallback intent will be created during the agent creation. From this intent, the responses will be fetched if the user’s query does not match any of the created intents.



**e.Testing of the chatbot.**

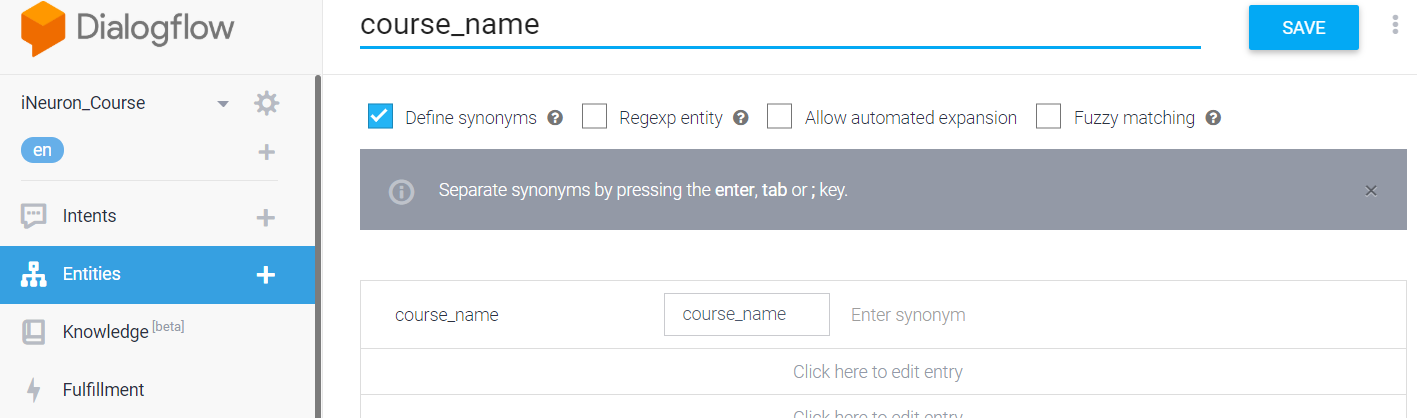
We can use the test console to test our requests and check the responses from  
Agent



**6. Entities**  
 Entities identify and extract useful data from user's inputs. While intents help to understand the motivation behind a particular user input, entities pick out specific pieces of information that the users mention.

**Two types of entities:**  
**a.System entities:** In-built entities provided by Google Dialogflow such as @sys.any, @sys.email, @sys.phone-number for customer name, email, phone number respectively.

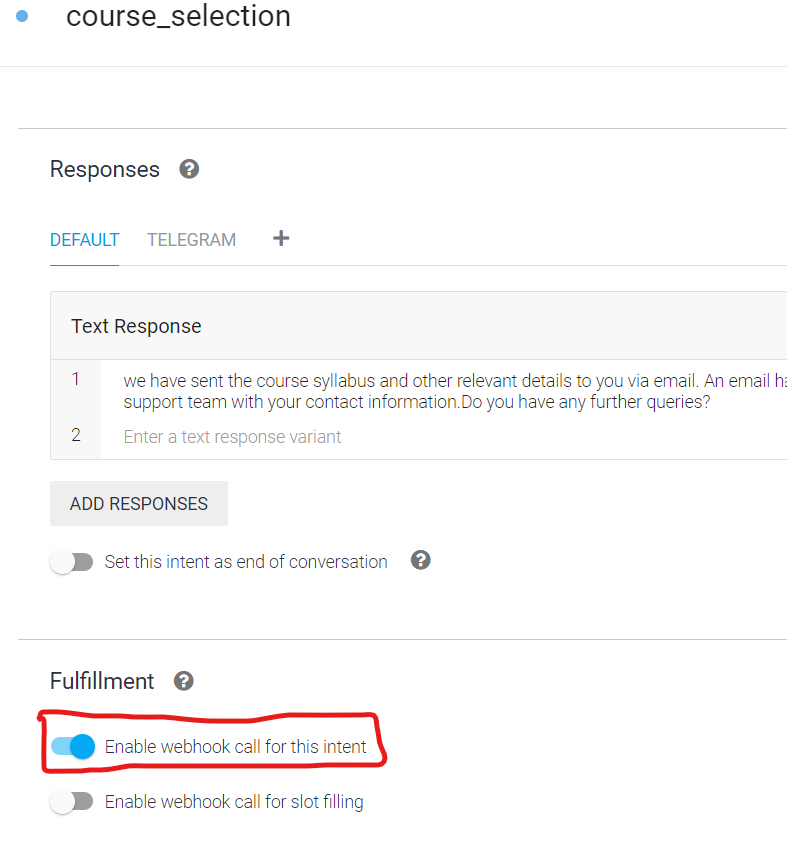
**b.Customer entities:** The developer creates customized entities. Here, we have created an Entity  
“course\_name” provided the synonyms, and saved it.



**7. Fulfilment:**

Fulfillment helps to use the information extracted by Dialogflow's natural language processing to generate responses dynamically or to trigger actions in the back-end.

**7.1 Configure fulfillment:**  
 Enable fulfillment by sliding **Enable webhook call for this intent** of course\_selection intent. So that when the user selects the course the back end logic will executed to sent the mail to support team and user with course details.



**7.2 External API integrations:**

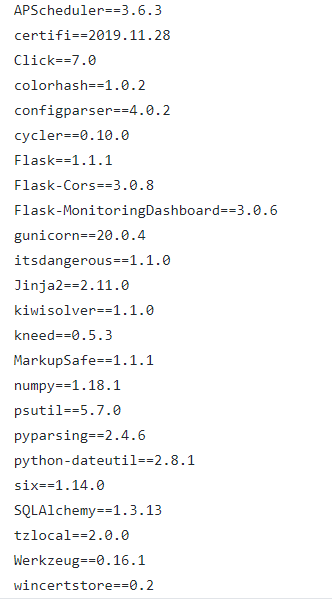
**7.2.1 Creating a Webhook**

a. Create a python file and then copy the below code and save it as app.py

from flask import Flask, request, make\_responseimport jsonimport osfrom flask\_cors import cross\_originfrom SendEmail.sendEmail import EmailSenderfrom logger import loggerfrom email\_templates import template\_readerapp = Flask(\_\_name\_\_)# geting and sending response to dialogflow@app.route('/webhook', methods=['POST'])@cross\_origin()def webhook(): req = request.get\_json(silent=True, force=True) #print("Request:") #print(json.dumps(req, indent=4)) res = processRequest(req) res = json.dumps(res, indent=4) #print(res) r = make\_response(res) r.headers['Content-Type'] = 'application/json' return r# processing the request from dialogflowdef processRequest(req): log = logger.Log() sessionID=req.get('responseId') result = req.get("queryResult") user\_says=result.get("queryText") log.write\_log(sessionID, "User Says: "+user\_says) parameters = result.get("parameters") cust\_name=parameters.get("cust\_name") #print(cust\_name) cust\_contact = parameters.get("cust\_contact") cust\_email=parameters.get("cust\_email") course\_name= parameters.get("course\_name") intent = result.get("intent").get('displayName') if (intent=='course\_selection'): email\_sender=EmailSender() template= template\_reader.TemplateReader() email\_message=template.read\_course\_template(course\_name) email\_sender.send\_email\_to\_student(cust\_email,email\_message) email\_file\_support = open("email\_templates/support\_team\_Template.html", "r") email\_message\_support = email\_file\_support.read() email\_sender.send\_email\_to\_support(cust\_name=cust\_name,cust\_contact=cust\_contact,cust\_email=cust\_email,course\_name=course\_name,body=email\_message\_support) fulfillmentText="We have sent the course syllabus and other relevant details to you via email. An email has been sent to the Support Team with your contact information, you'll be contacted soon. Do you have further queries?" log.write\_log(sessionID, "Bot Says: "+fulfillmentText) return { "fulfillmentText": fulfillmentText } else: log.write\_log(sessionID, "Bot Says: " + result.fulfillmentText)if \_\_name\_\_ == '\_\_main\_\_': port = int(os.getenv('PORT', 5000)) print("Starting app on port %d" % port) app.run(debug=False, port=port, host='0.0.0.0')

b. create sendEmail.py for sending the HTML email template using SMTP

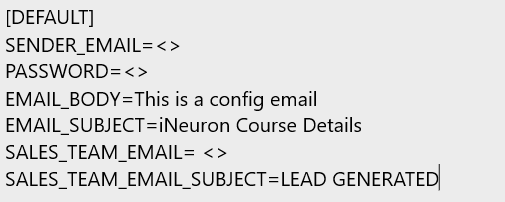
import smtplibfrom email.mime.multipart import MIMEMultipartfrom email.mime.text import MIMETextfrom email.mime.base import MIMEBasefrom config\_reader import ConfigReaderclass EmailSender: def send\_email\_to\_student(self, recepient\_email, message): try: self.config\_reader=ConfigReader() self.configuration=self.config\_reader.read\_config() # instance of MIMEMultipart self.msg = MIMEMultipart() # storing the senders email address self.msg['From'] = self.configuration['SENDER\_EMAIL'] # storing the receivers email address self.msg['To'] = ",".join(recepient\_email) # storing the subject self.msg['Subject'] = self.configuration['EMAIL\_SUBJECT'] # string to store the body of the mail #body = "This will contain attachment" body=message # attach the body with the msg instance self.msg.attach(MIMEText(body, 'html')) # instance of MIMEBase and named as p self.p = MIMEBase('application', 'octet-stream') # creates SMTP session self.smtp = smtplib.SMTP('smtp.gmail.com', 587) # start TLS for security self.smtp.starttls() # Authentication self.smtp.login(self.msg['From'], self.configuration['PASSWORD']) # Converts the Multipart msg into a string self.text = self.msg.as\_string() # sending the mail self.smtp.sendmail(self.msg['From'] , recepient\_email, self.text) # terminating the session self.smtp.quit() except Exception as e: print('the exception is '+str(e)) def send\_email\_to\_support(self,cust\_name,cust\_email,cust\_contact,course\_name,body): try: self.config\_reader = ConfigReader() self.configuration = self.config\_reader.read\_config() # instance of MIMEMultipart self.msg = MIMEMultipart() # storing the senders email address self.msg['From'] = self.configuration['SENDER\_EMAIL'] # storing the subject self.msg['Subject'] = self.configuration['SALES\_TEAM\_EMAIL\_SUBJECT'] # string to store the body of the mail # body = "This will contain attachment" body = body.replace('cust\_name',cust\_name) body = body.replace('cust\_contact', cust\_contact) body = body.replace('cust\_email', cust\_email) body = body.replace('course\_name', course\_name) # attach the body with the msg instance self.msg.attach(MIMEText(body, 'html')) # instance of MIMEBase and named as p self.p = MIMEBase('application', 'octet-stream') # creates SMTP session self.smtp = smtplib.SMTP('smtp.gmail.com', 587) # start TLS for security self.smtp.starttls() # Authentication self.smtp.login(self.msg['From'], self.configuration['PASSWORD']) # Converts the Multipart msg into a string self.text = self.msg.as\_string() # sending the mail self.support\_team\_email = self.configuration['SALES\_TEAM\_EMAIL'] self.smtp.sendmail(self.msg['From'], self.support\_team\_email, self.text) # terminating the session self.smtp.quit() except Exception as e: print('the exception is ' + str(e))

c. create a file requirements.txt and paste the following data  
.

d. Create the file Procfile and paste the following data.



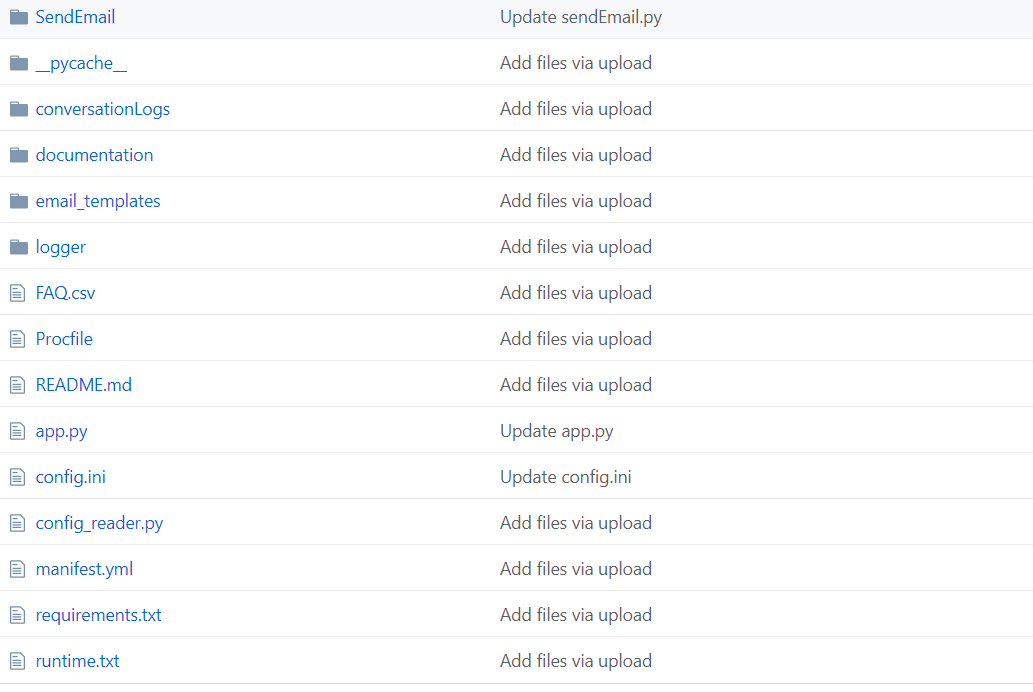
e. create config.ini with the mailing details.



f. create logger.py for logging all the conversation and saving it into a file

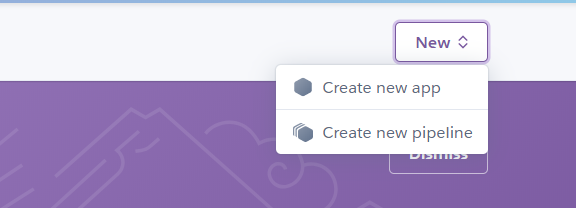
from datetime import datetimeclass Log: def \_\_init\_\_(self): pass def write\_log(self, sessionID, log\_message): self.file\_object = open("conversationLogs/"+sessionID+".txt", 'a+') self.now = datetime.now() self.date = self.now.date() self.current\_time = self.now.strftime("%H:%M:%S") self.file\_object.write( str(self.date) + "/" + str(self.current\_time) + "\t\t" + log\_message + "\n") self.file\_object.close()

g. Create a git repository and upload all the files

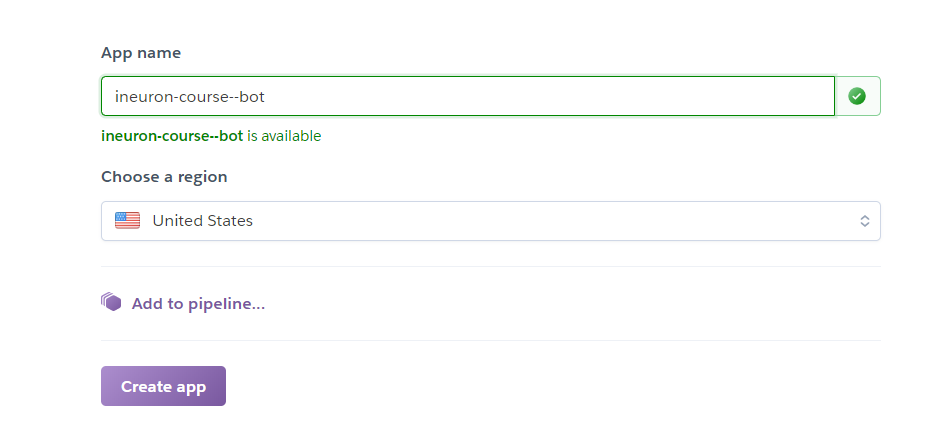


**7.2.2 Deployment in Heroku**

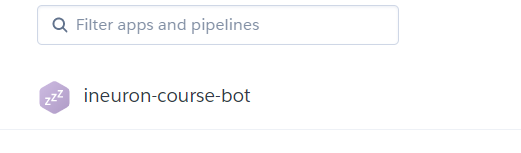
* Sign into the Heroku website and Create a new app



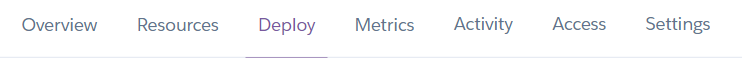
* Provide the name as ineuron-course-bot and then click on Create app



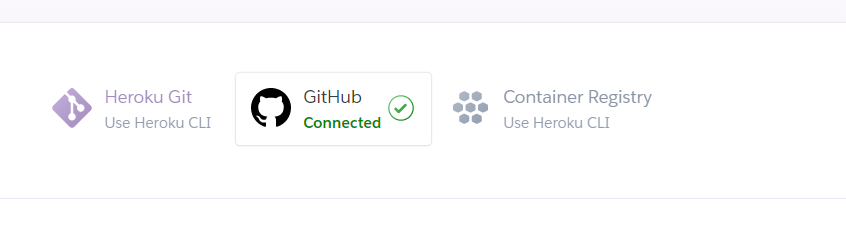
* Select the newly created app.



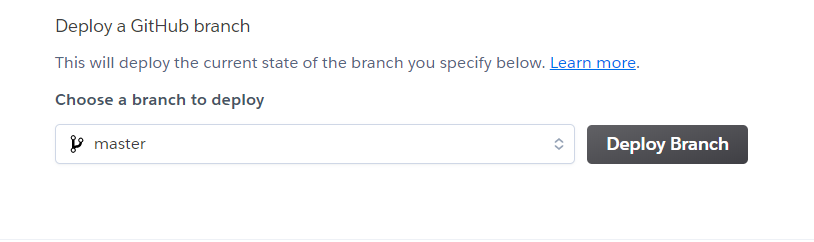
* Select Deploy



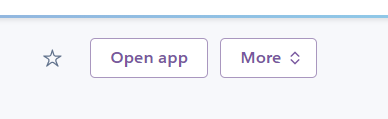
* Under the Deployment method, select “Github” and then search for the GIT repository and  
  then connect to it.



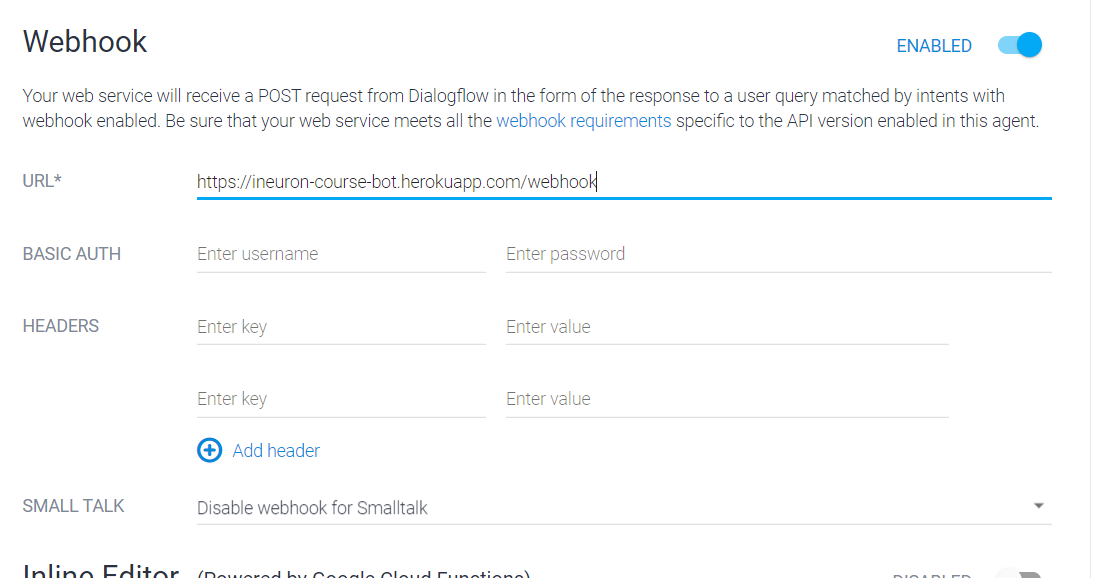
* Click deploy



* Click the open app button to get the webhook.

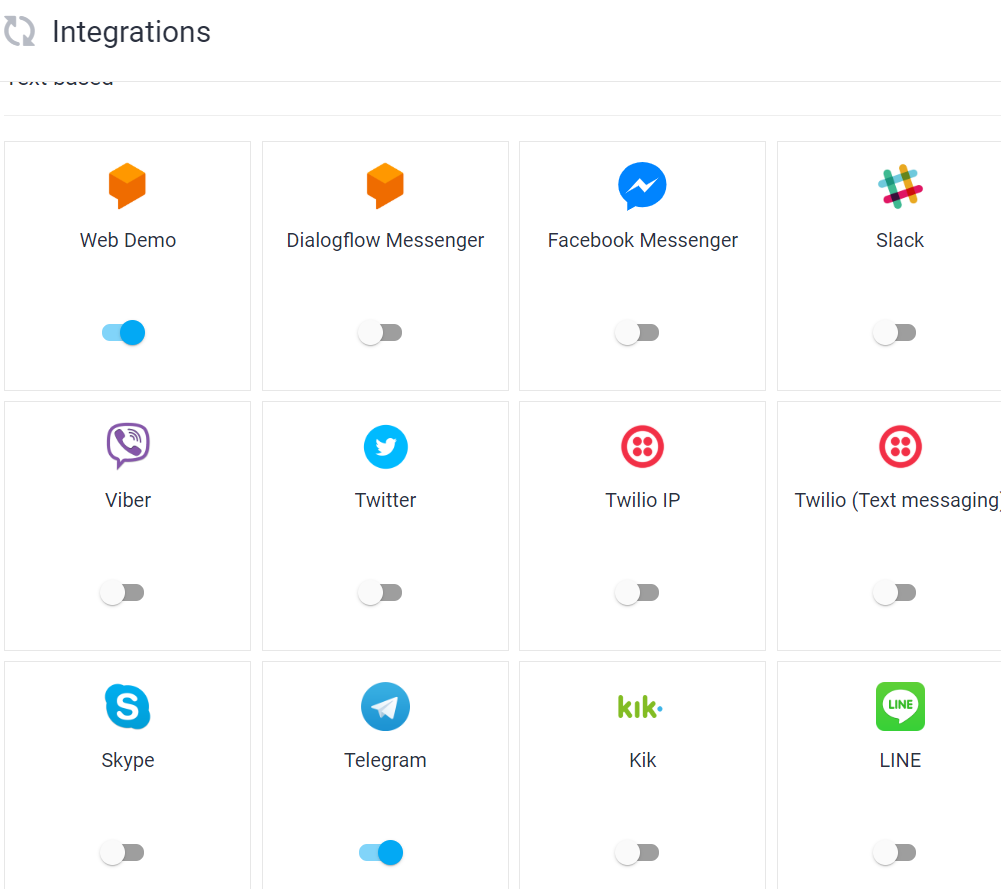


* Copy the URL and append /webhook to the url and go to the Dialogflow fulfillment and paste the URL under the webhook section of Fulfillment.

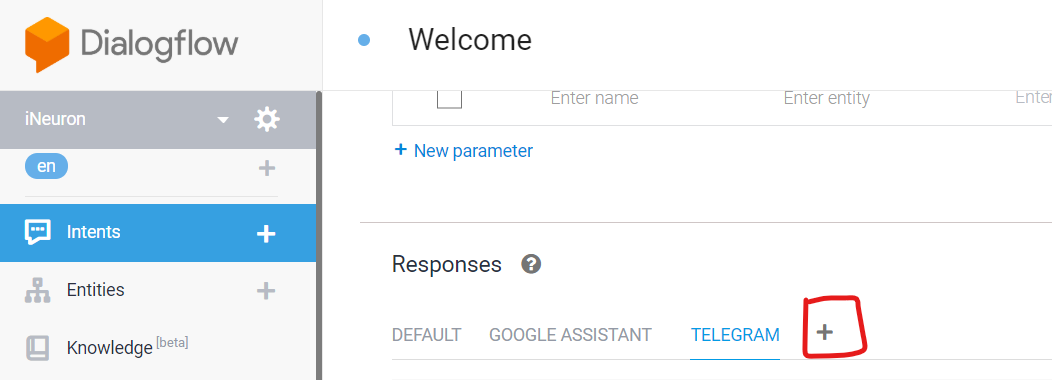


**8. Integrations**

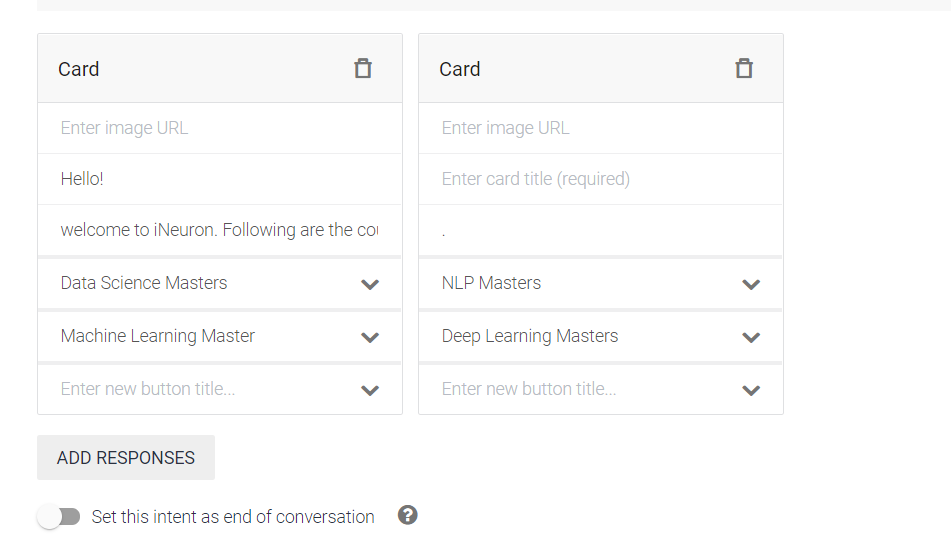
* Dialogflow integrates with many platforms like Slack, Google Assistant, and Telegram. These integrations provide platform-specific features for building productive responses.



* In telegram, there are different options available for adding the responses from intents. Select the welcome intent and add the telegram by clicking the plus symbol



* Click add ‘add response’, select card and enter the different courses



**8.1 Dialogflow Web Demo:**

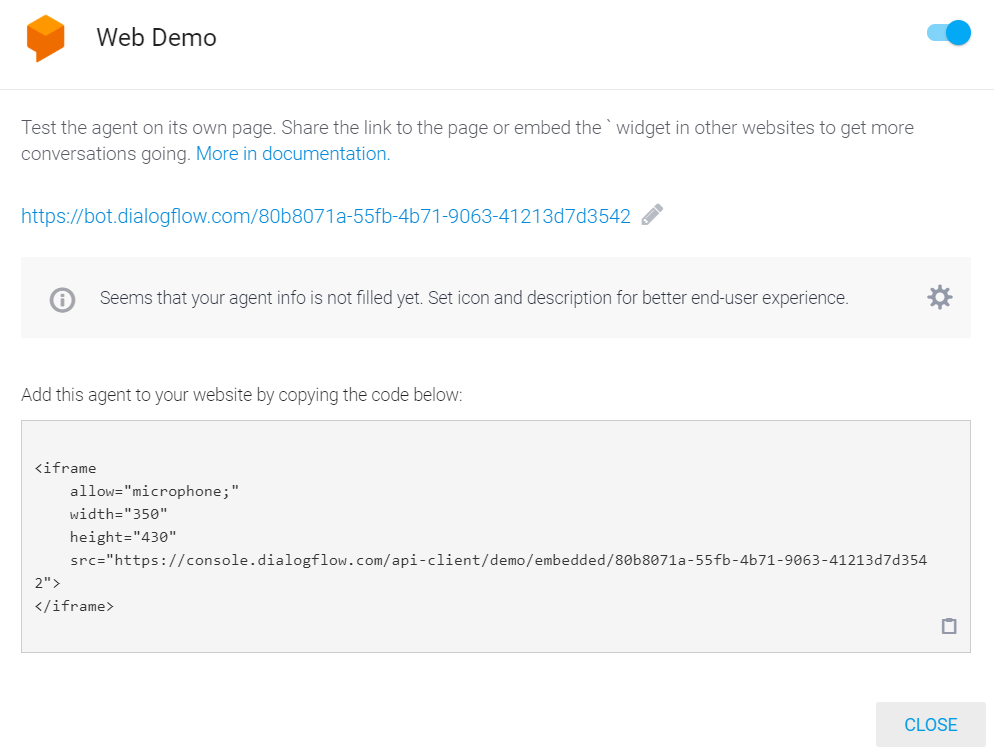
The Web demo allows us to publicly share our agent through a generated page or by  
embedding it in our current portal/website.

**8.1.2 Setting Up a Web Demo**

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

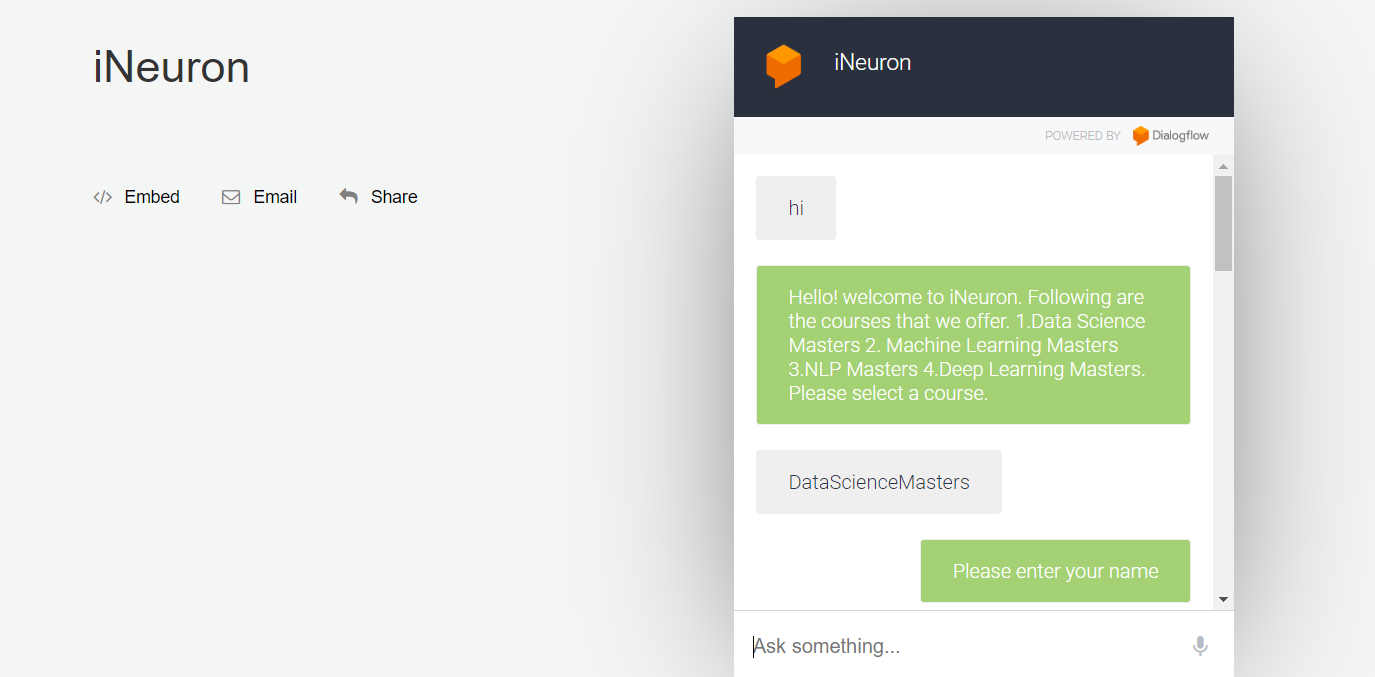
After Web Demo is enabled, a window will be displayed with the following info:  
• A URL to the generated webpage where our agent is hosted  
• A link to icon and description settings

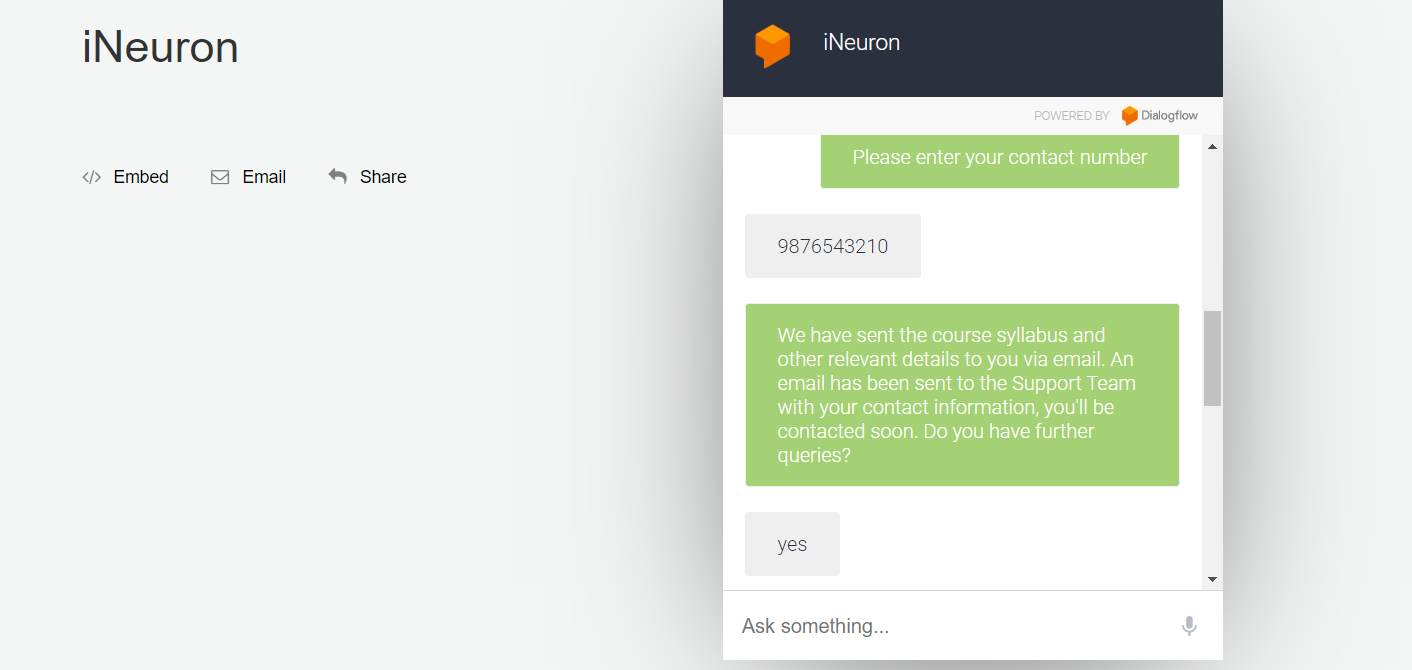
• Code to embed our agent in our website, via HTML

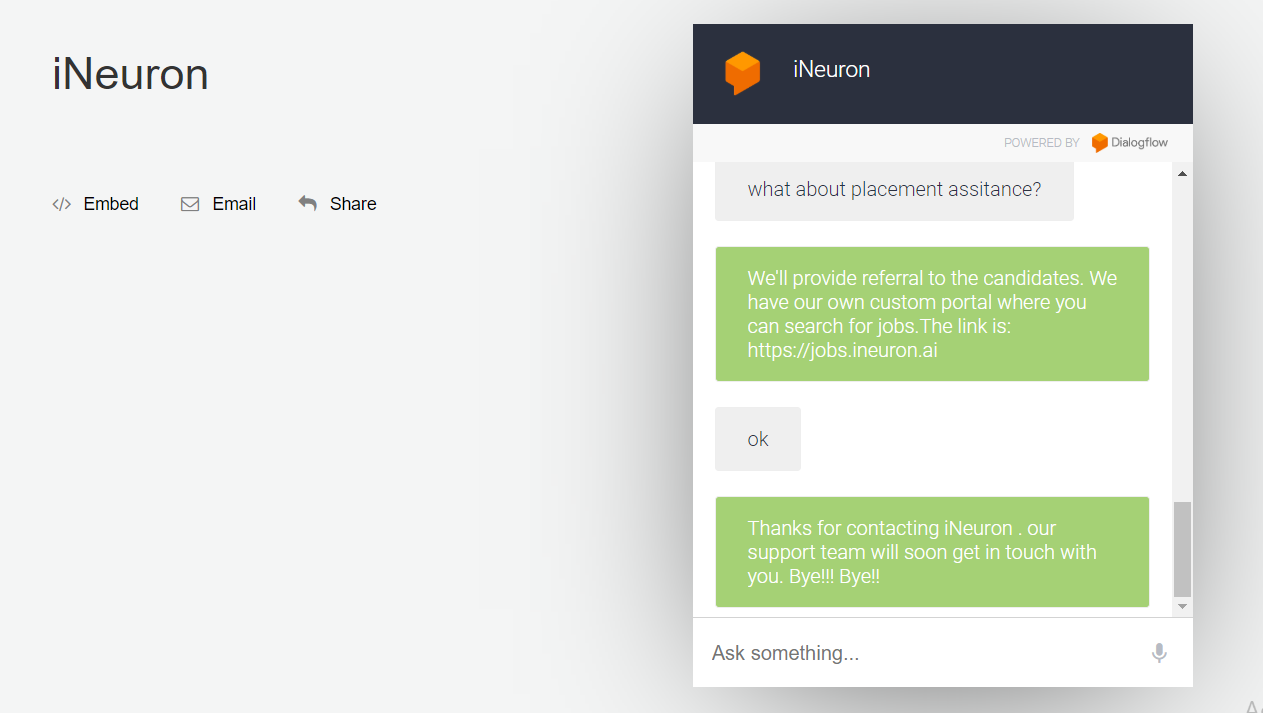


**8.1.2 Testing the web demo**

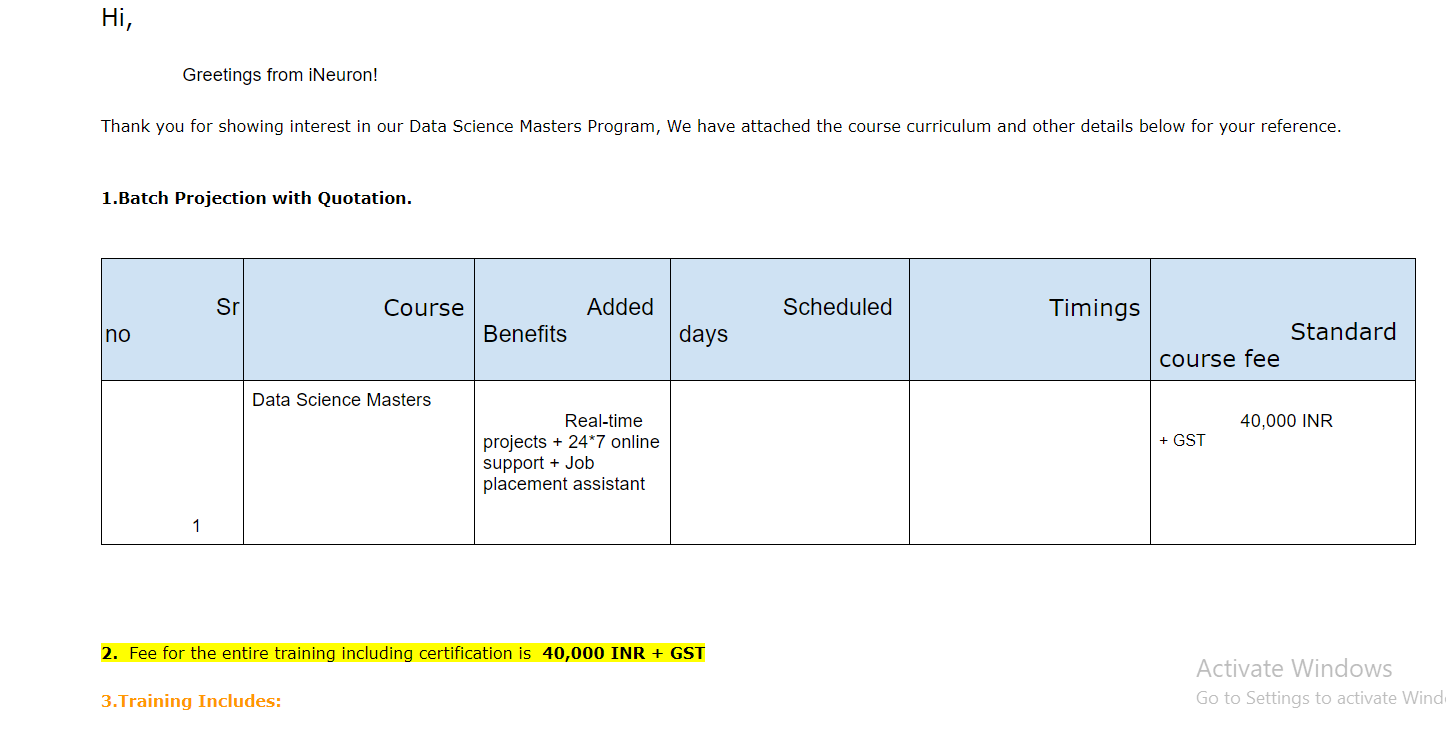
1. Testing the bot through web



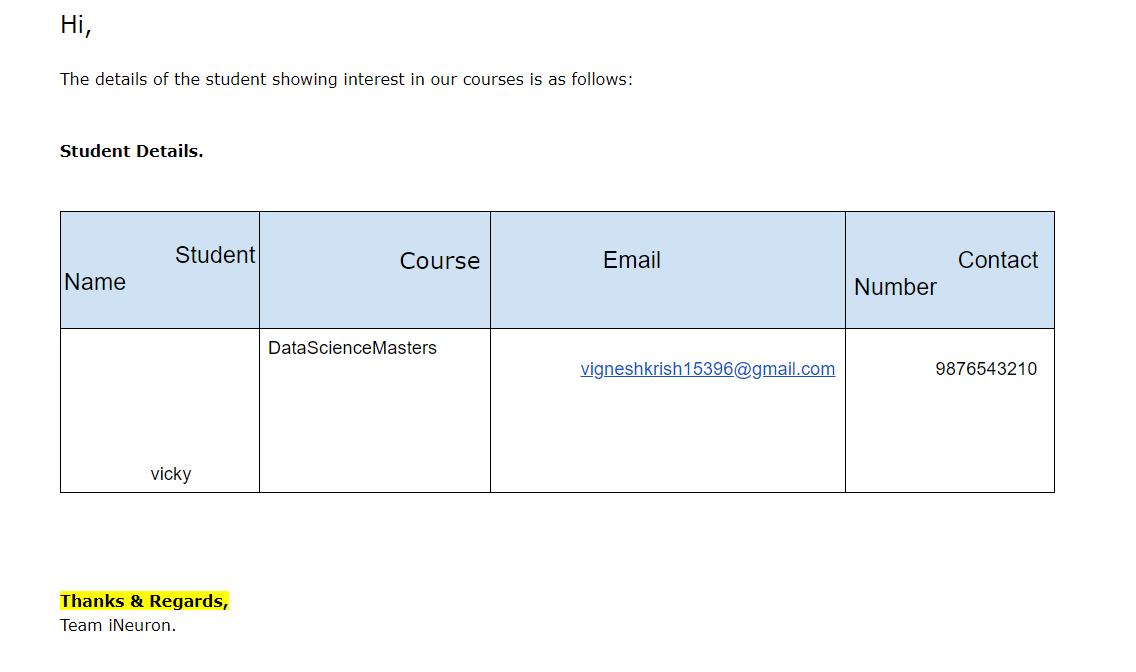




1. User Received a mail from bot with the course details

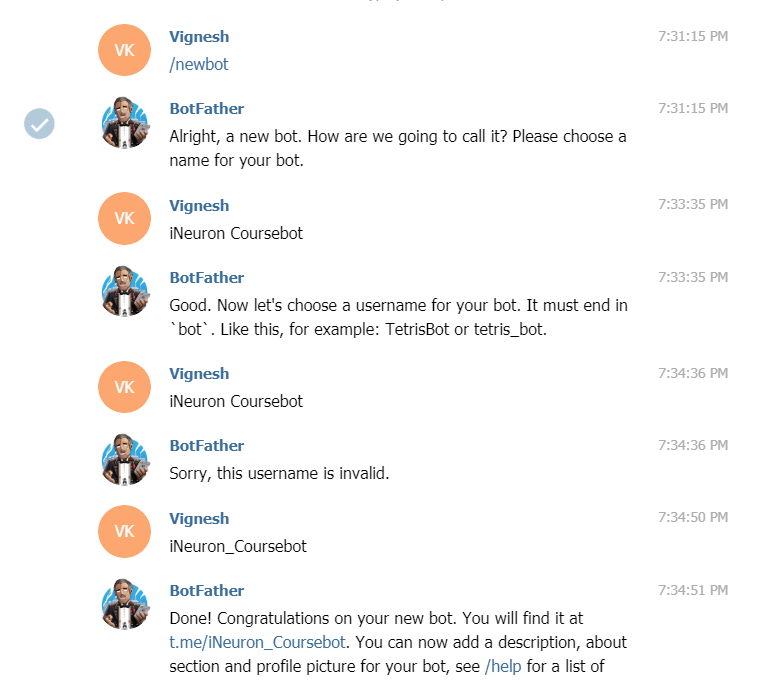


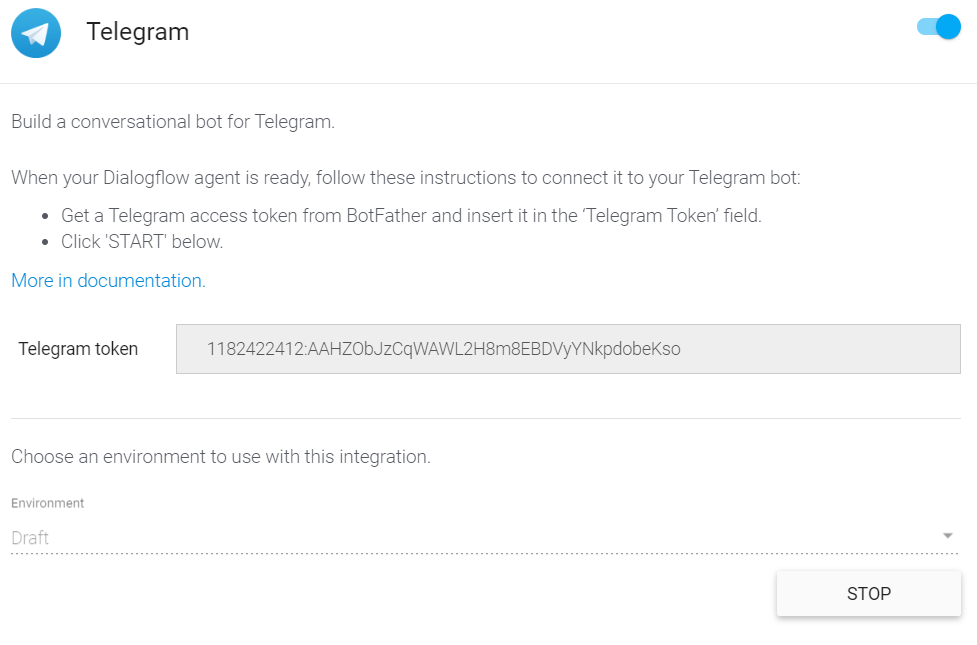
1. support team received a mail from bot with the user details



**8.2 Telegram:** To set up the Telegram integration for our agent, you'll need a Telegram account.  
**8.2.1Creating a Bot in Telegram**1. Log-in to Telegram using the link https://telegram.me/botfather  
2. Click the **Start** button in the web interface or type /start in Telegram  
3. Type **/newbot** and provide a name  
4. Provide a username for the bot, ending in "bot" (e.g.,iNeuron\_coursebot) Copy the generated access token

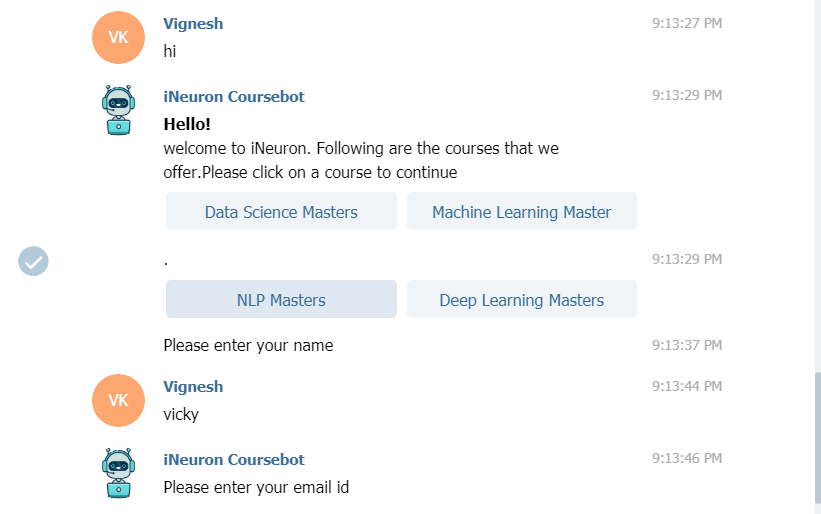
5. In Dialogflow, go to **Integrations** available in the left-hand menu  
6. Click on the **Telegram** tile to enable the Telegram integrations.  
7. Paste the Telegram **Access Token** into the related field. Click the **Start**button

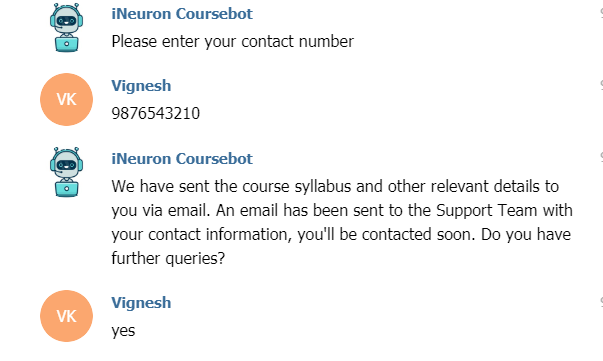


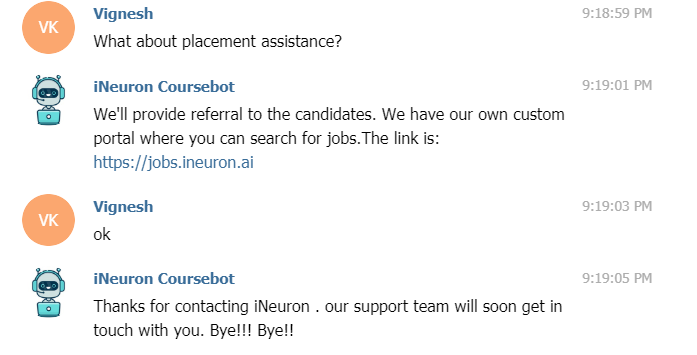


**8.2.2 Testing the Bot through telegram**

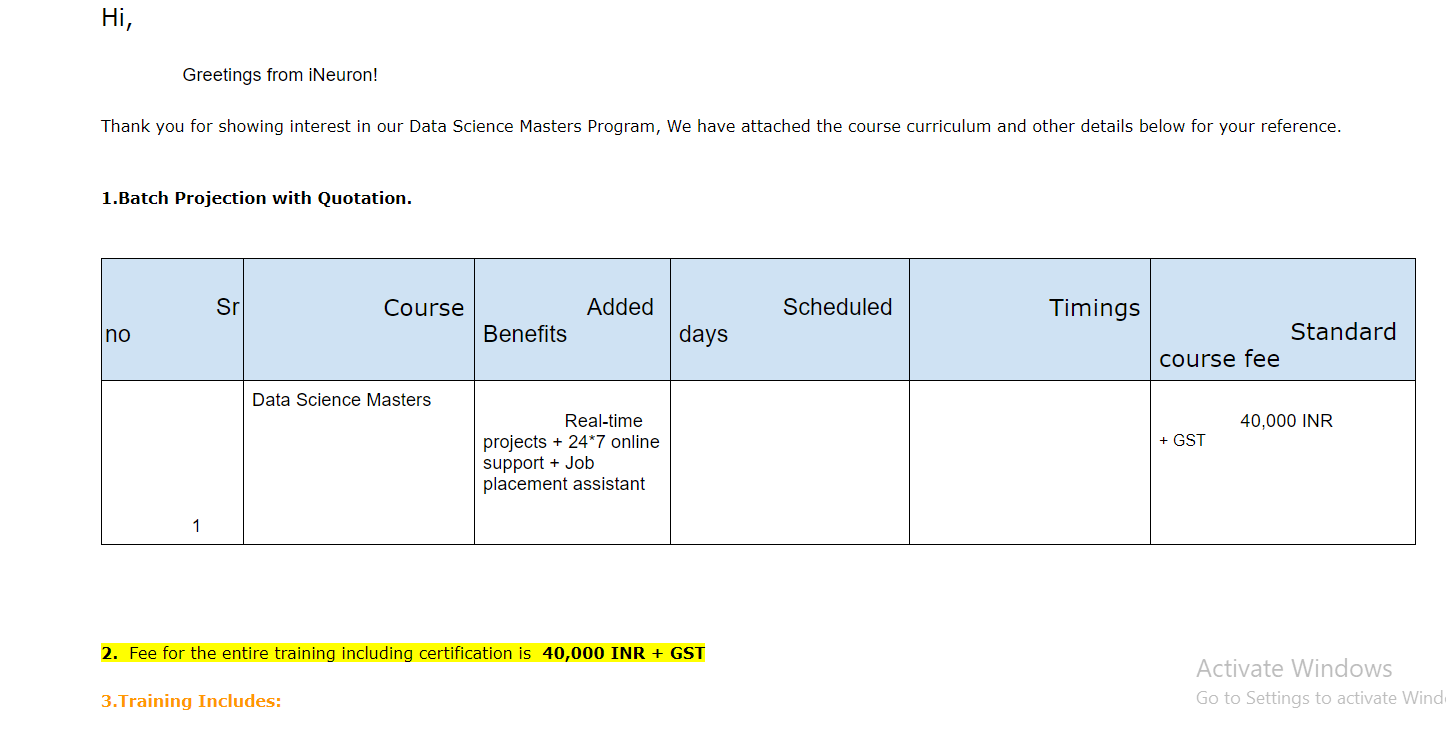
1. User chat with the bot.



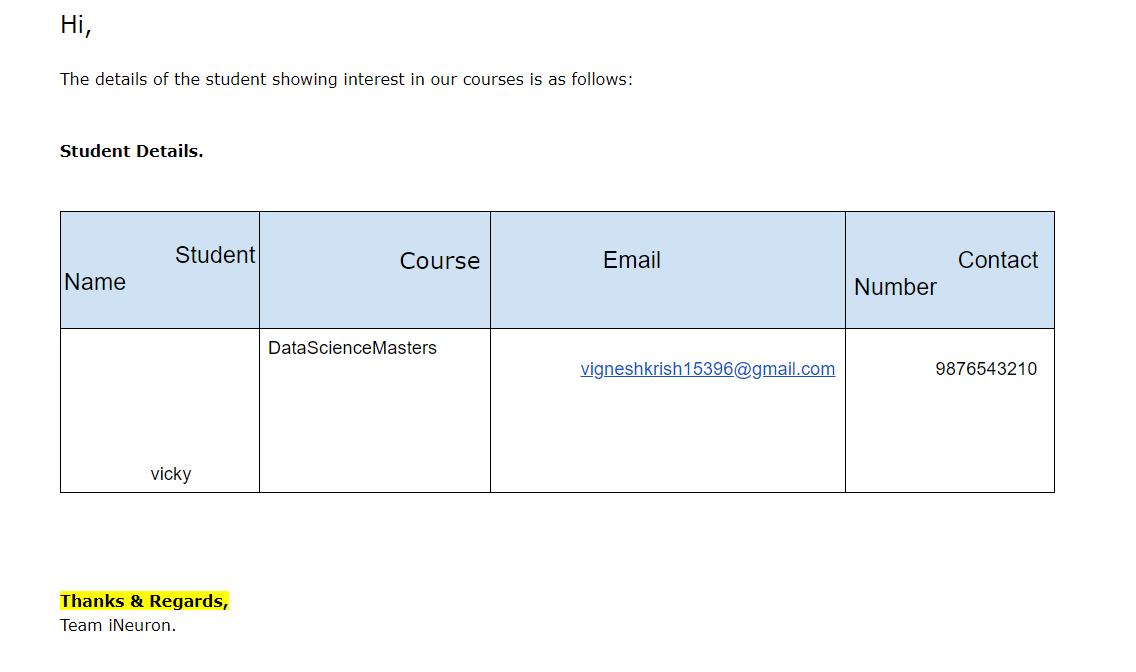




1. User Received a mail from bot with the course details



1. support team received a mail from bot with the user details

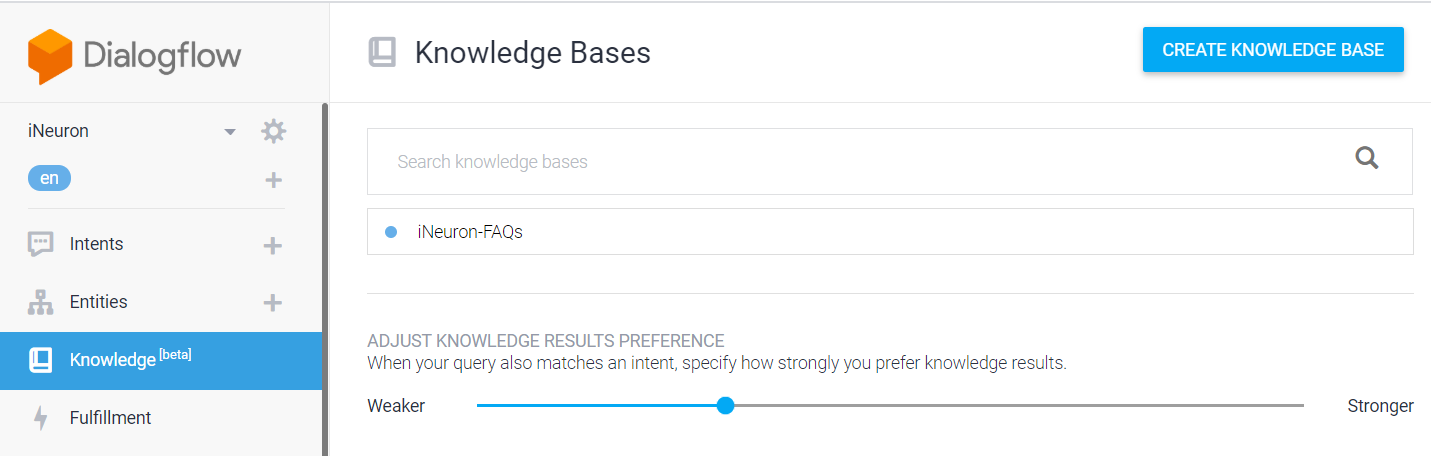


**9. Knowledge Bases**

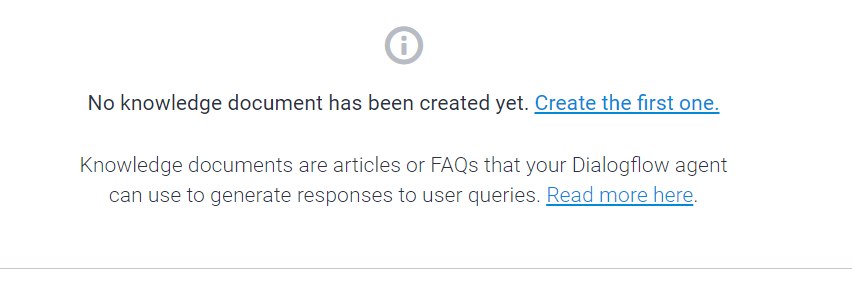
Knowledge is the repository of questions and answers. The knowledge base gets the data from various sources such as documents, cloud, url.

**9.1 Creating a knowledge bases**

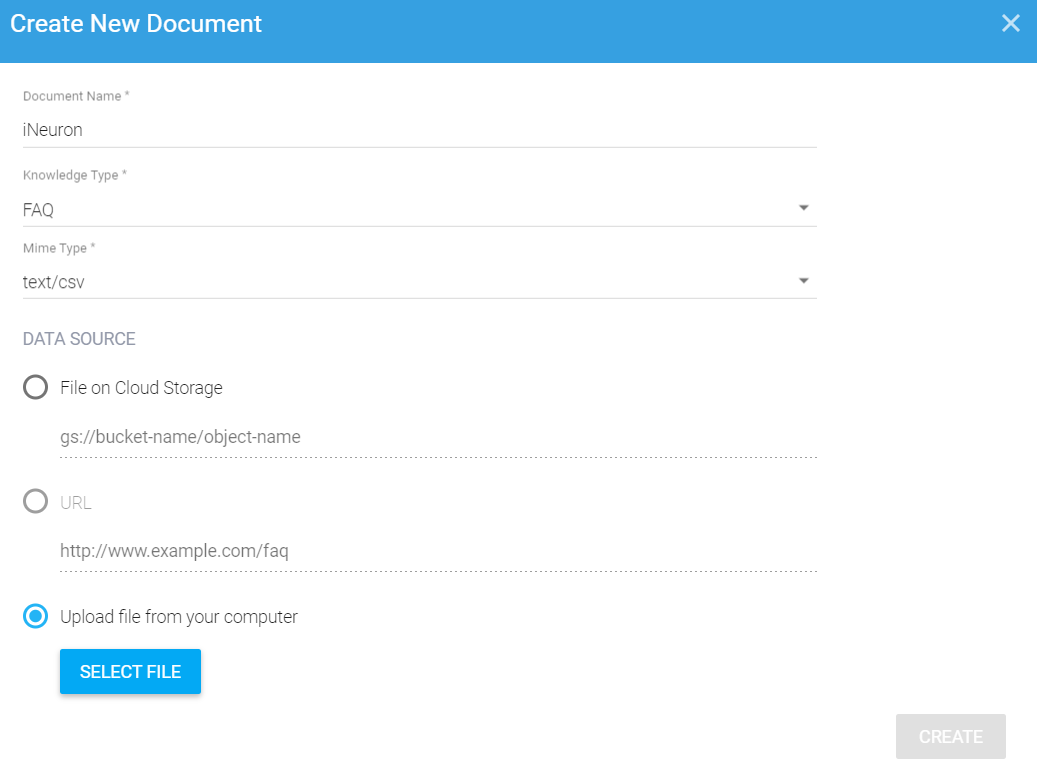
a.Click on the CREATE KNOWLEDGE BASE, enter a name and save



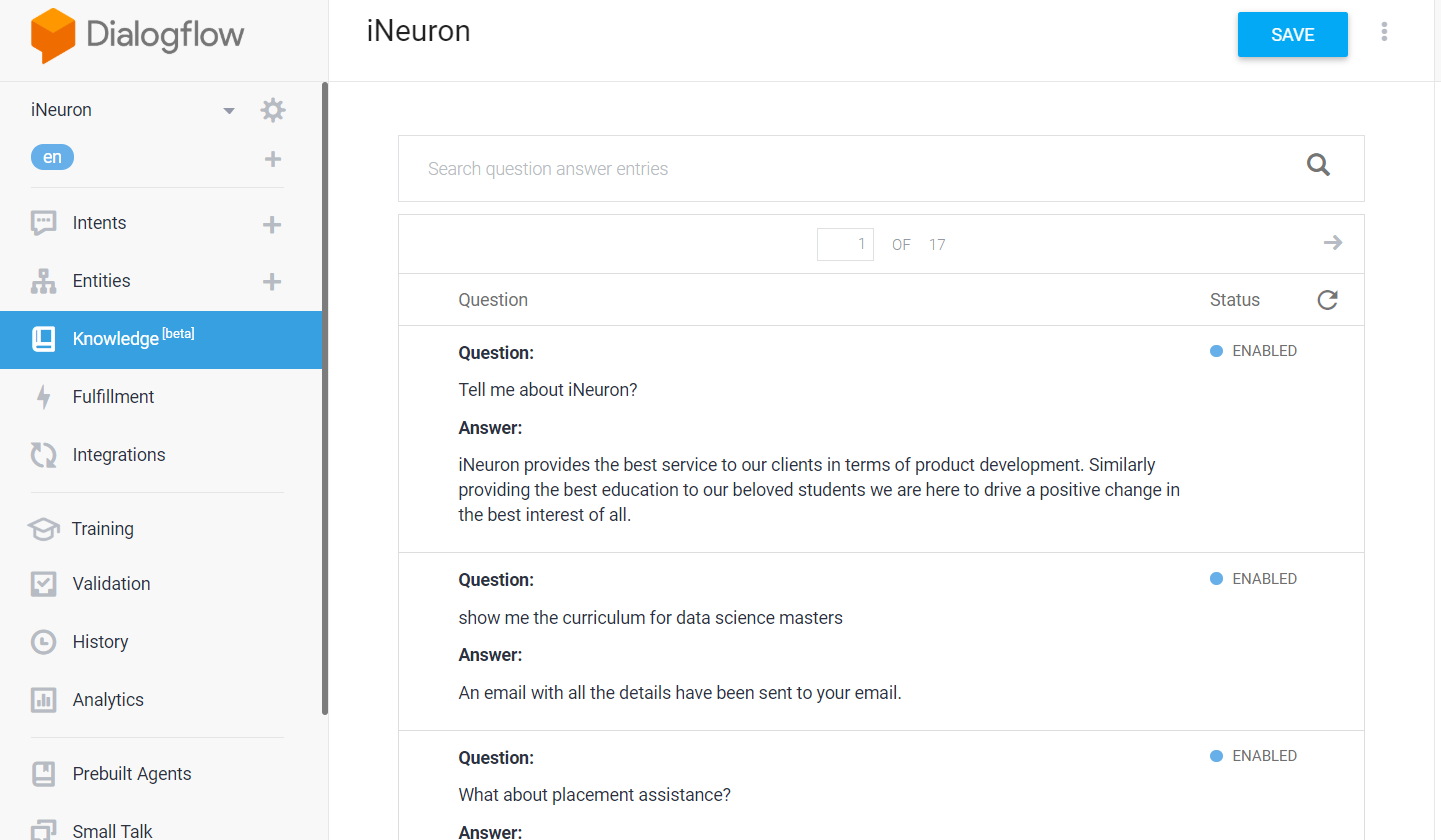
b. click on create the first one



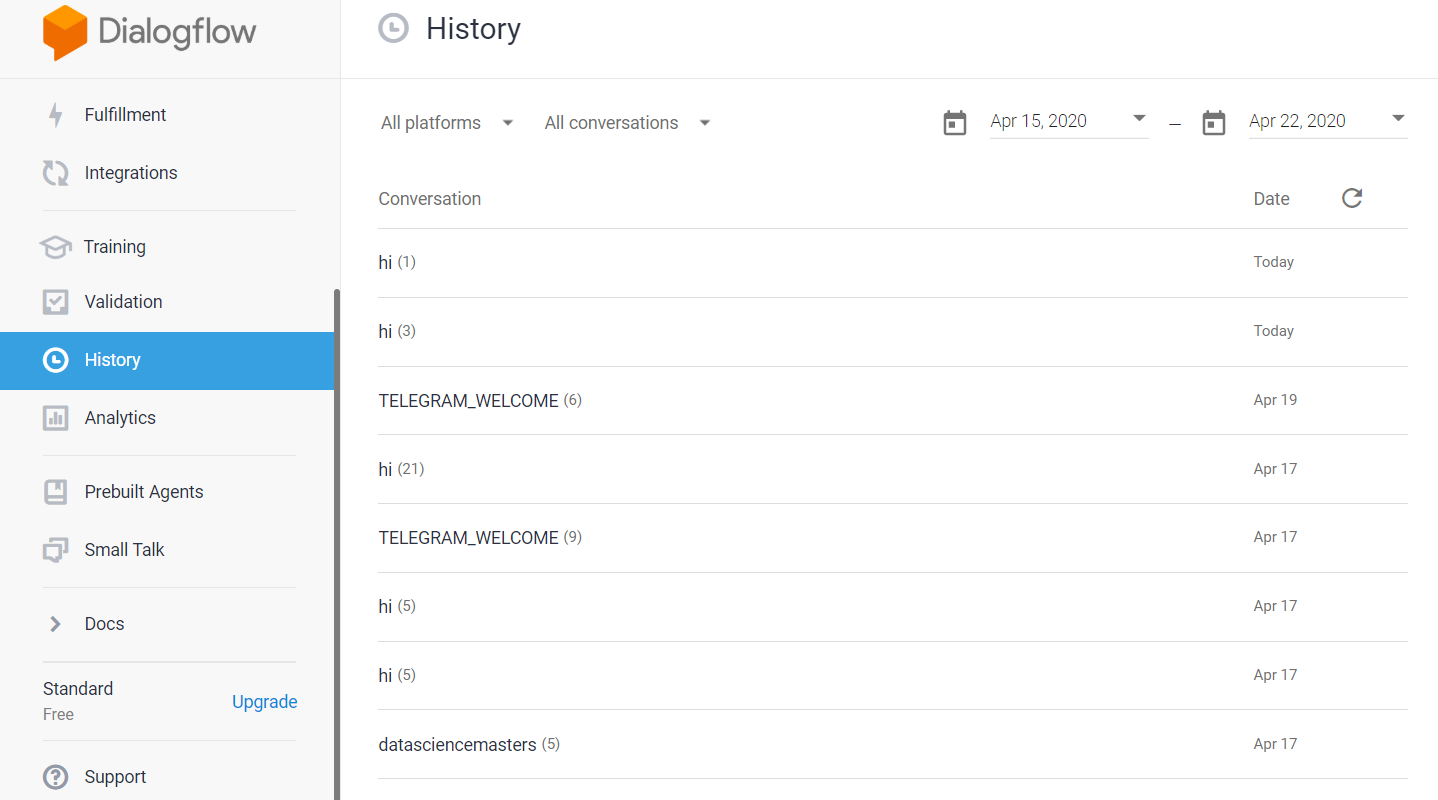
c.Enter the details and upload a.csv file with FAQs



d. Now the chatbot will display responses from the knowledge base if the query from the user is similar to that of the questions in the knowledge database.



**10. History** Dialogflow History section displays a simplified version of conversations the agent has  
engaged. The records provide an overview of how users interact with the agent



**11. Training:** Dialogflow's natural language processing built on machine learning. We can add training data that the agent learns from and uses to improve its performance. Dialogflow's training feature provides an interface for incorporating both external and internal customer interaction logs into an agent's training phrases.

