iNeuron Course Chatbot

**Using Google Dialogflow**

**Submitted by Harish Janagam**

iNeuron Course Chatbot Telegram Integration Link

[https://t.me/Dialogdemo\_bot](https://t.me/iNeuronCourseChatbot)

Overview

To build a chatbot which can answer all the queries of a customer and whenever a customer does an enquiry, it automatically sends the customer the course details. Also, an email is sent to the support team to assist the customer further with their queries. By using Google Dialogflow we will achieve this task.

Goals

1. **Welcome User:​**The bot should greet the user with a good quote and available courseswhich we provide whenever the user starts a conversation.
2. **Get User Details:​**By prompting options to the user it should collect user details such asname, email contact number.
3. **Send Email:** By**​** calling webhook it should send an email of course details to the user anda trailing mail to support team about the lead generated.
4. **FAQs:** Bot**​** should be able to answer common user queries by the FAQs set by us.



2

Specifications

Chatbot automates the process of receiving user data and sends all the course details to the user in a friendly way. Generating leads and will notify with an email that a new lead is generated. Solving user FAQ’s regarding iNeuron and its courses.

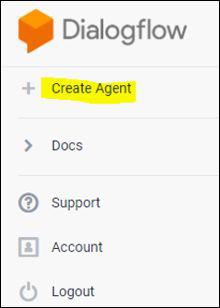
Milestones

1. 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 centre agent. You train them both to handle expected conversation scenarios, and your training does not need to be overly explicit.

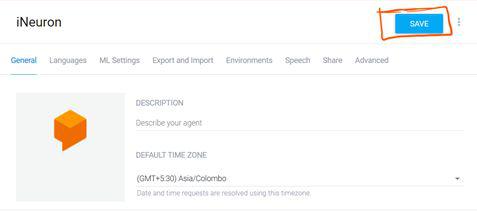
1.1 Creating an Agent

1. Click on Create Agent from the left menu.

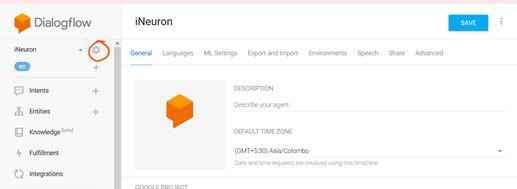


3

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



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



1. Intent

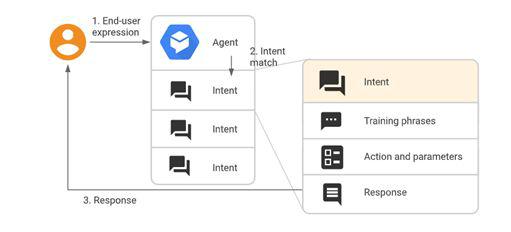
An intent categorizes an end-user's intention for one conversation turn. For each agent, you define many intents, where your combined intents can handle a complete conversation. When an end-user writes or says something, referred to as an end-user expression, Dialogflow



4

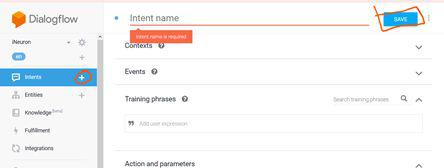
matches the end-user expression to the best intent in your agent. Matching an intent is also known as intent classification.

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



2.1 Creating an Intent:

1. 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.



2.1.1 Training Phrases:

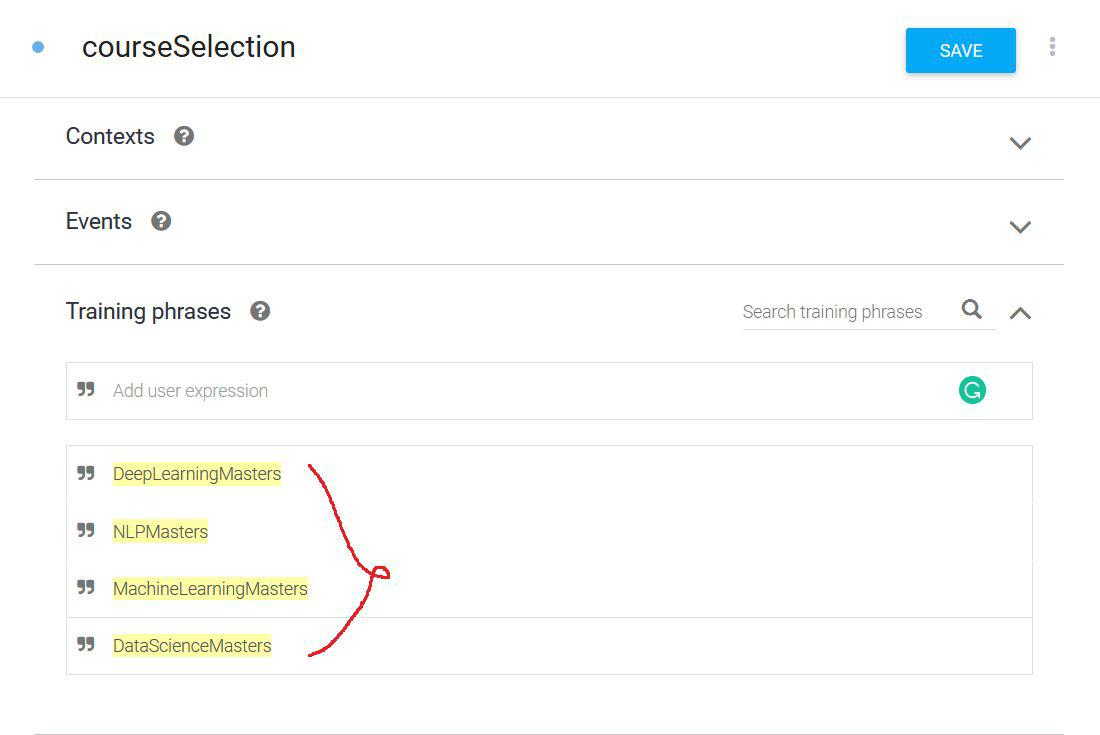
Training phrases are example phrases for what end-users might type or say, referred to as end-user expressions. For each intent, you create many training phrases. When an end-user expression resembles one of these phrases, Dialogflow matches the intent.



5

Adding the training Phrases:

1. Click the text field that shows "Add user expression".
2. Type your training phrases and press the Enter key after each.



2.1.2 Extracting the Entities:

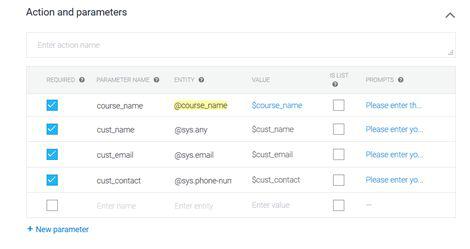
When an intent is matched at runtime, Dialogflow provides the extracted values from the end-user expression as parameters. Each parameter has a type, called the entity type, which dictates exactly how the data is extracted. Unlike raw end-user input, parameters are structured data that can easily be 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, colours, email addresses, and so on. You can also create your own custom entities for matching custom data.

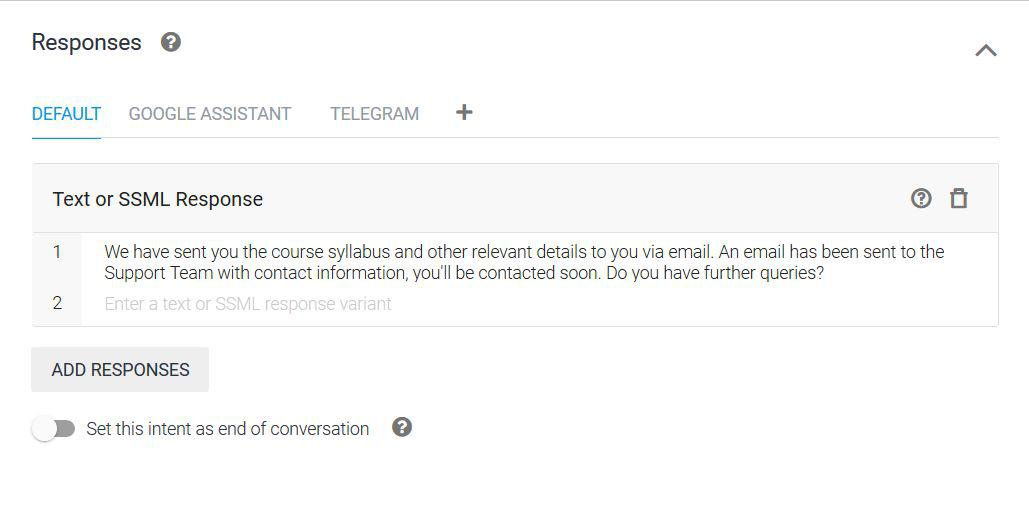


6



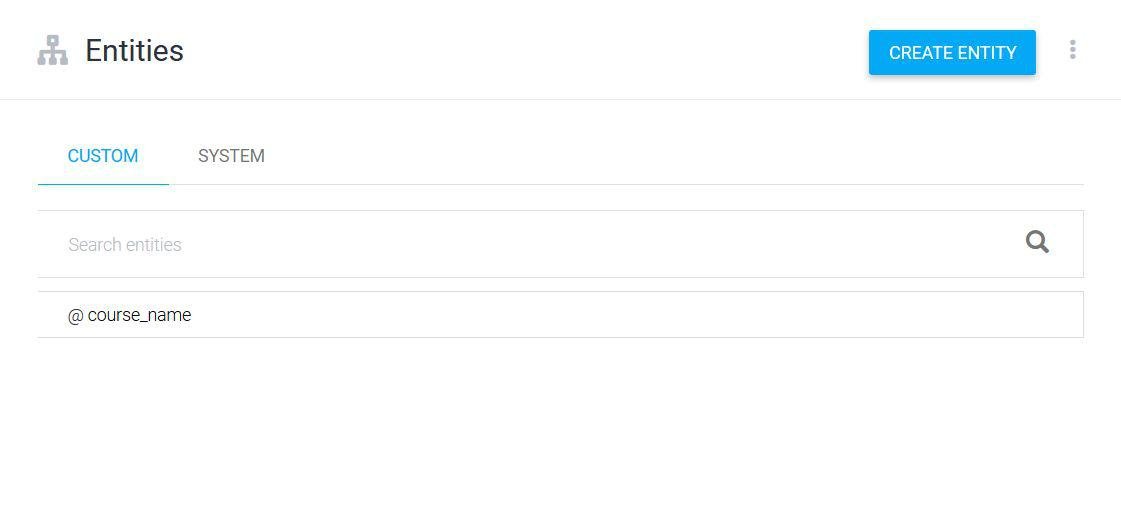
2.1.3 Specifying custom Responses:

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".



7

1. Entities

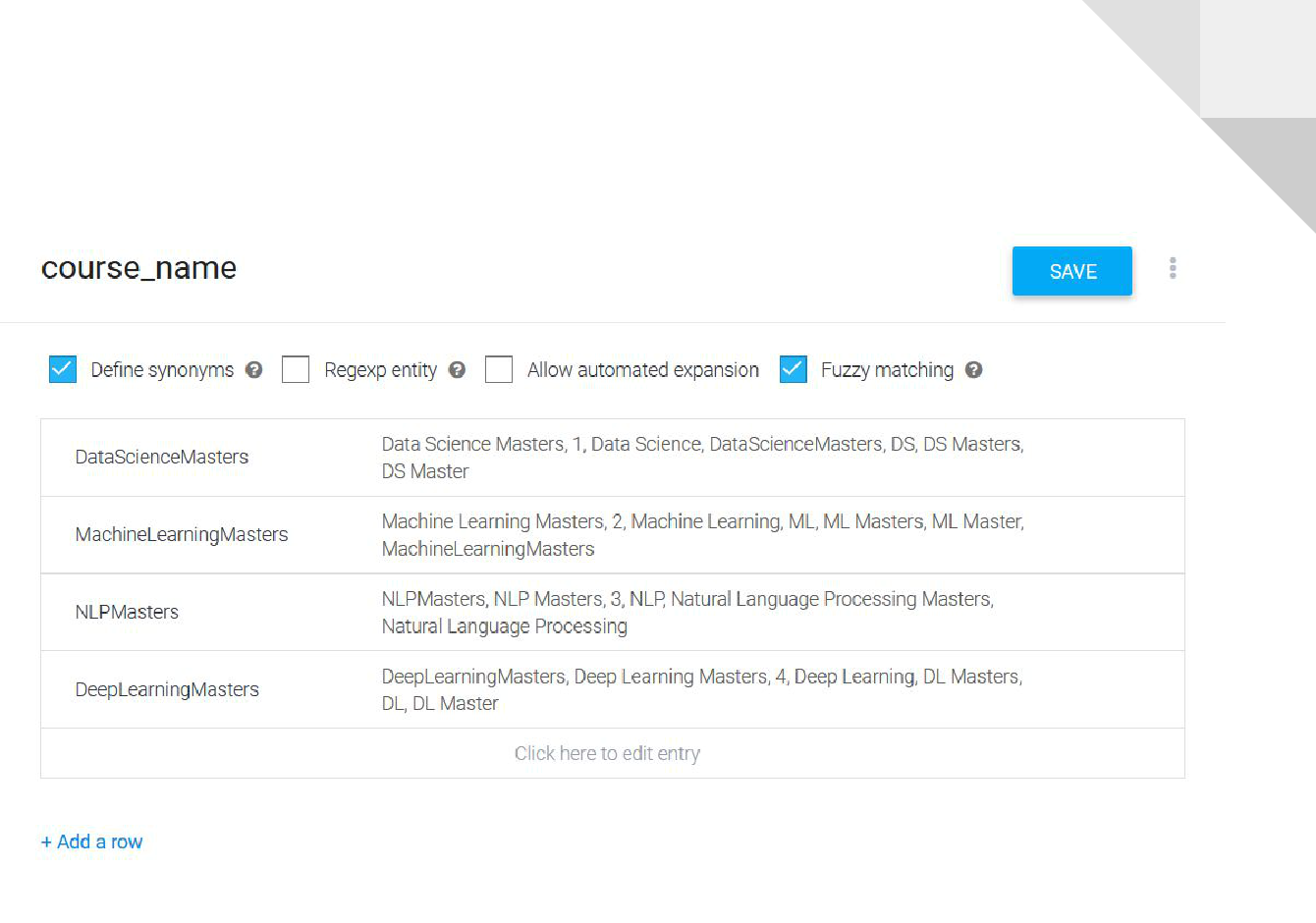


Entities identify and extract useful data from user's inputs. While intents help to understand the motivation behind a particular user input, entities pick out specific pieces of information that the users mention. Two types of entities:

System entities: In-built entities provided by Google Dialogflow.

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



8

1. Knowledge Bases

A knowledgebase represents a collection of knowledge documents that you provide to Dialogflow. Your knowledge documents contain information that may be useful during conversations with end-users. Some Dialogflow features use knowledge bases when looking for a response to an end-user expression. This guide describes how to create and manage knowledge bases.

4.1 Create Knowledge Base

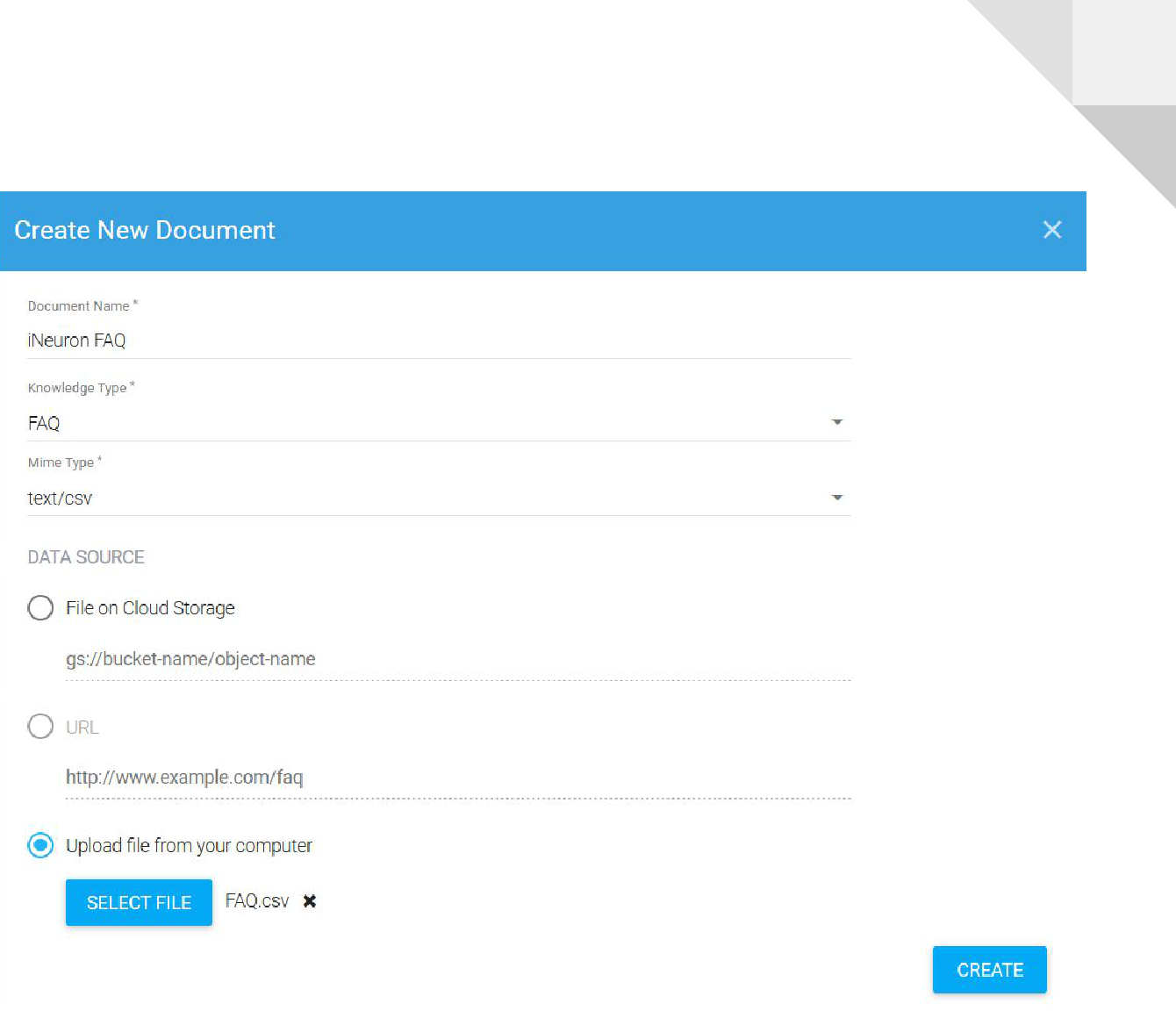
Click on Knowledge in the left menu. Click button Create Knowledge Base.



4.1.1 Create Knowledge Document

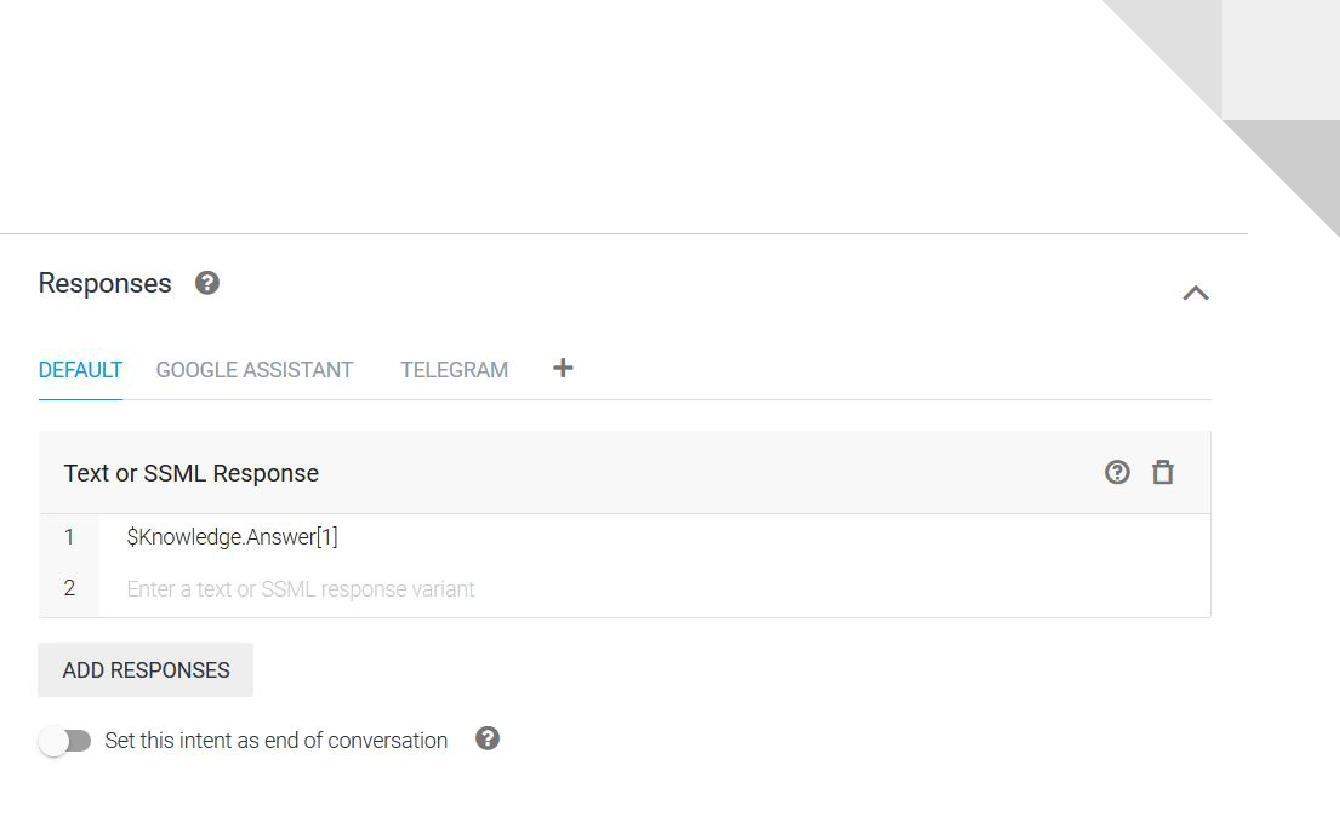
Click on Create the first one to create a Knowledge document and add your knowledge base.



9

4.2 Click on Responses to enable responses for Knowledge.



10

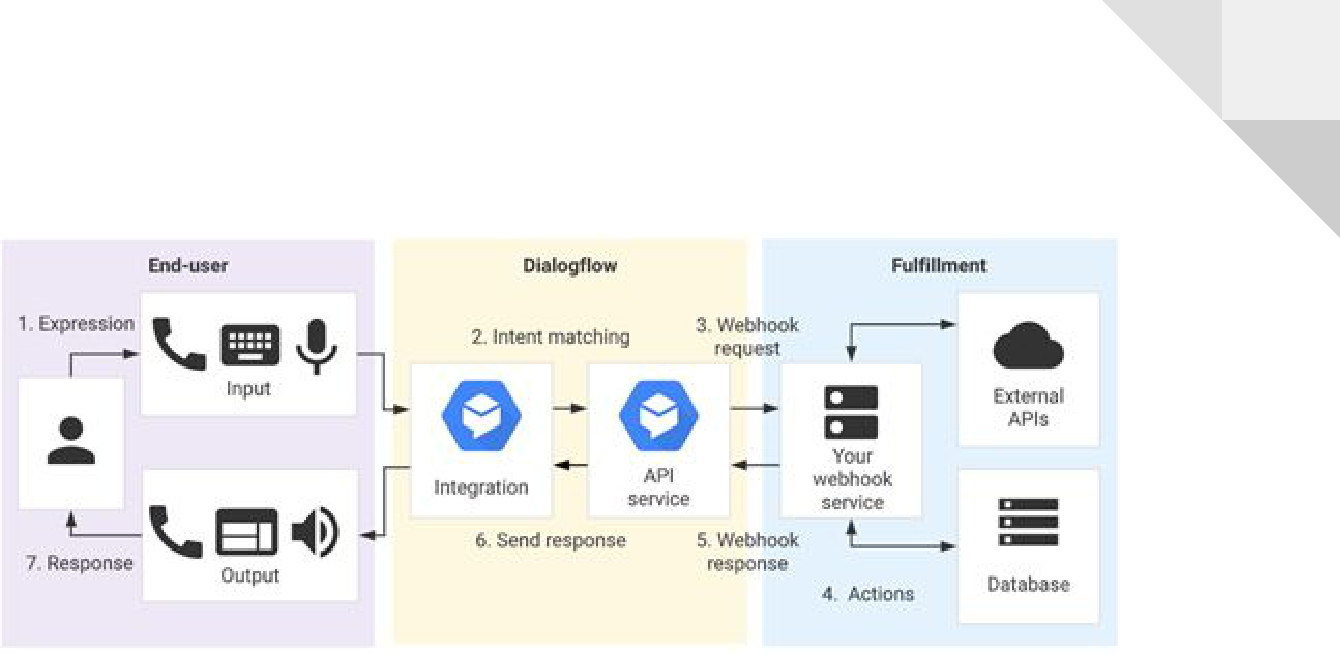
1. Fulfillment

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.



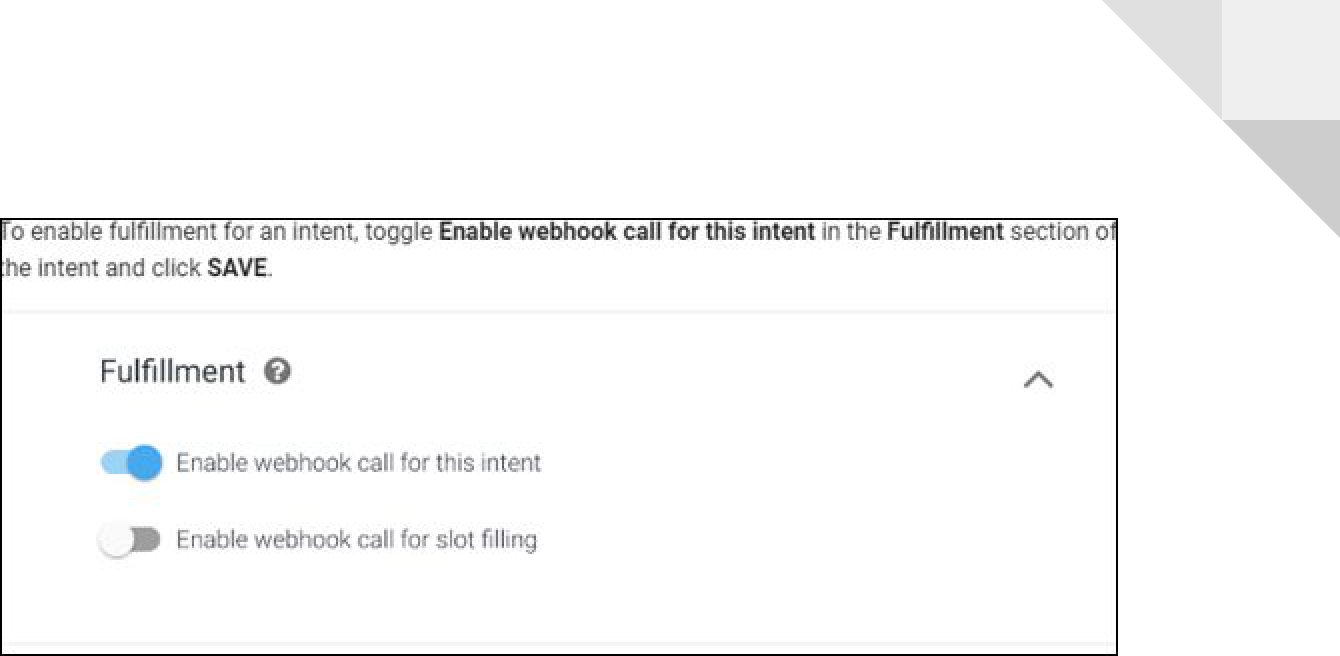
11

1. The end-user types or speaks an expression.
2. Dialogflow matches the end-user expression to an intent and extracts parameters.
3. 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.
4. Your service performs actions as needed, like database queries or external API calls.
5. Your service sends a webhook response message to Dialogflow. This message contains the response that should be sent to the end-user.
6. Dialogflow sends the response to the end-user.
7. The end-user sees or hears the response.

5.1 Configure fulfillment:

1. Enable fulfillment by sliding Enable webhook call for this intent to the intents, which involve some logic building or to trigger actions.

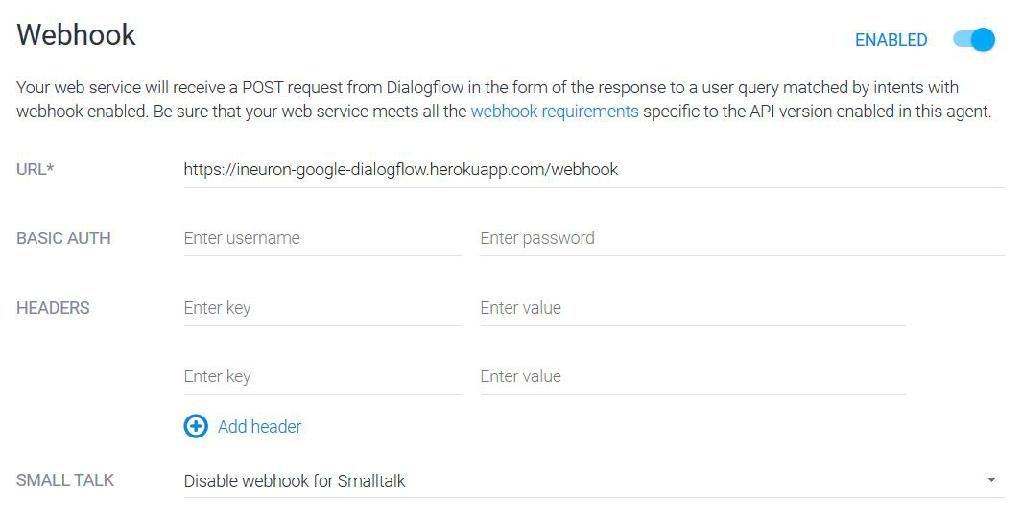


12

1. To enable the Webhook, Click on Fulfillment in the left menu. Click the switch for

Webhook.

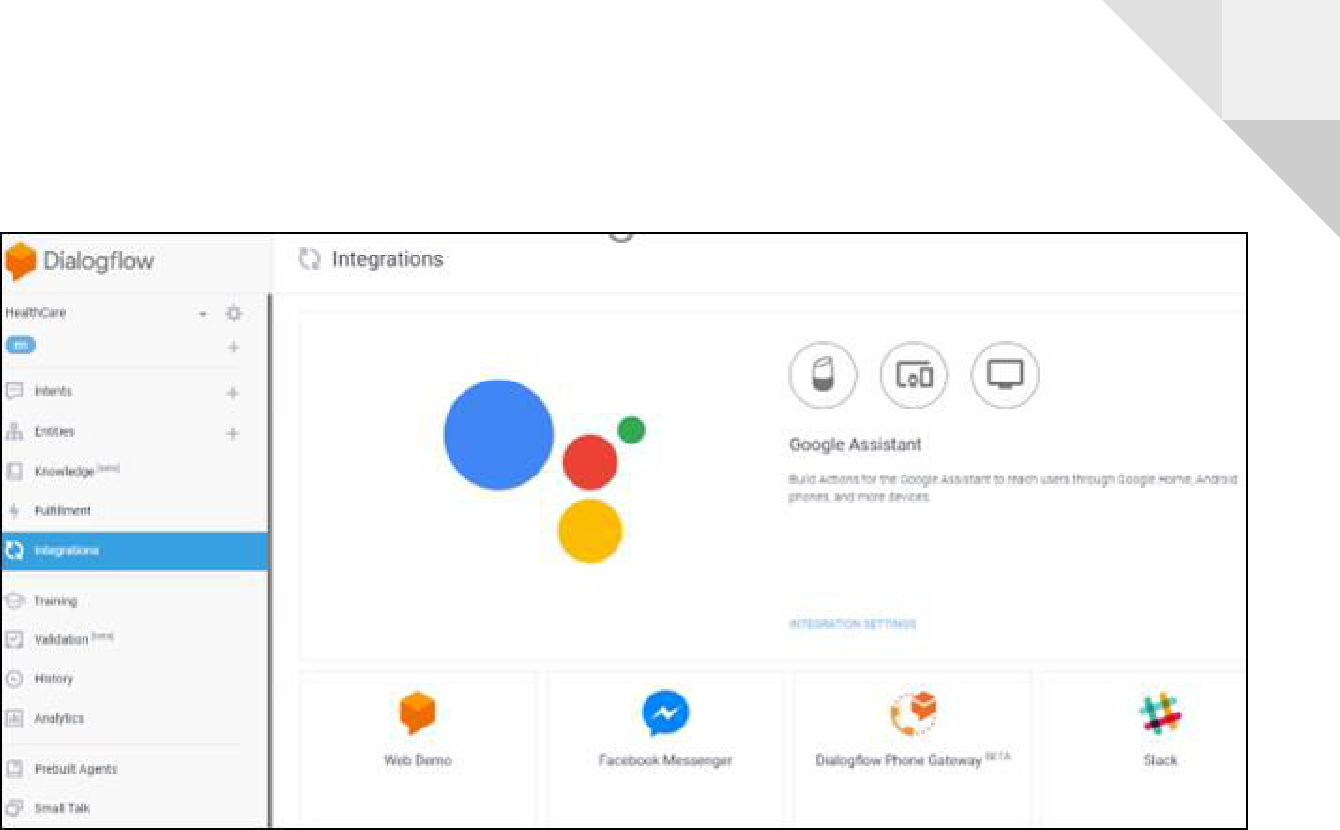
1. Now add the URL from where the webhook call is to be intended.



1. Integration

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



13

6.1 Dialogflow Web Demo

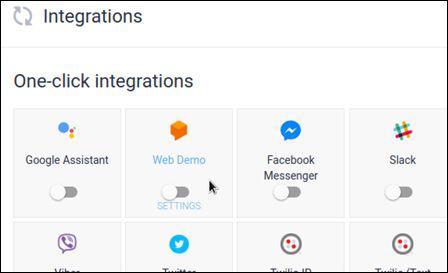
The Web demo allows us to publicly share our agent through a generated page or by embedding it in our current portal/website.

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.

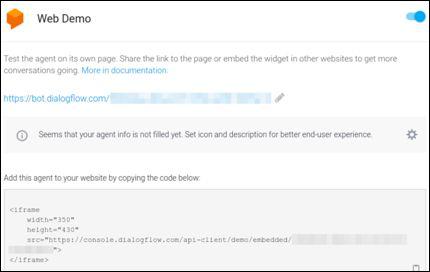


14



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, which affects our agent on the generated webpage
* Code to embed our agent in our website, via HTML

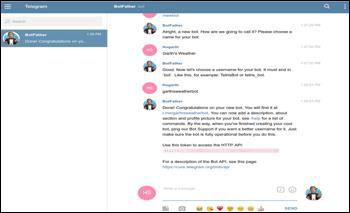


15

6.2 Telegram

To set up the Telegram integration for our agent, you'll need a Telegram account. Creating 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., iNeuronCourseChatbot)
5. Copy the generated access token

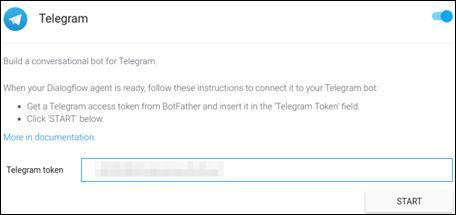


Setting Up Dialogflow

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

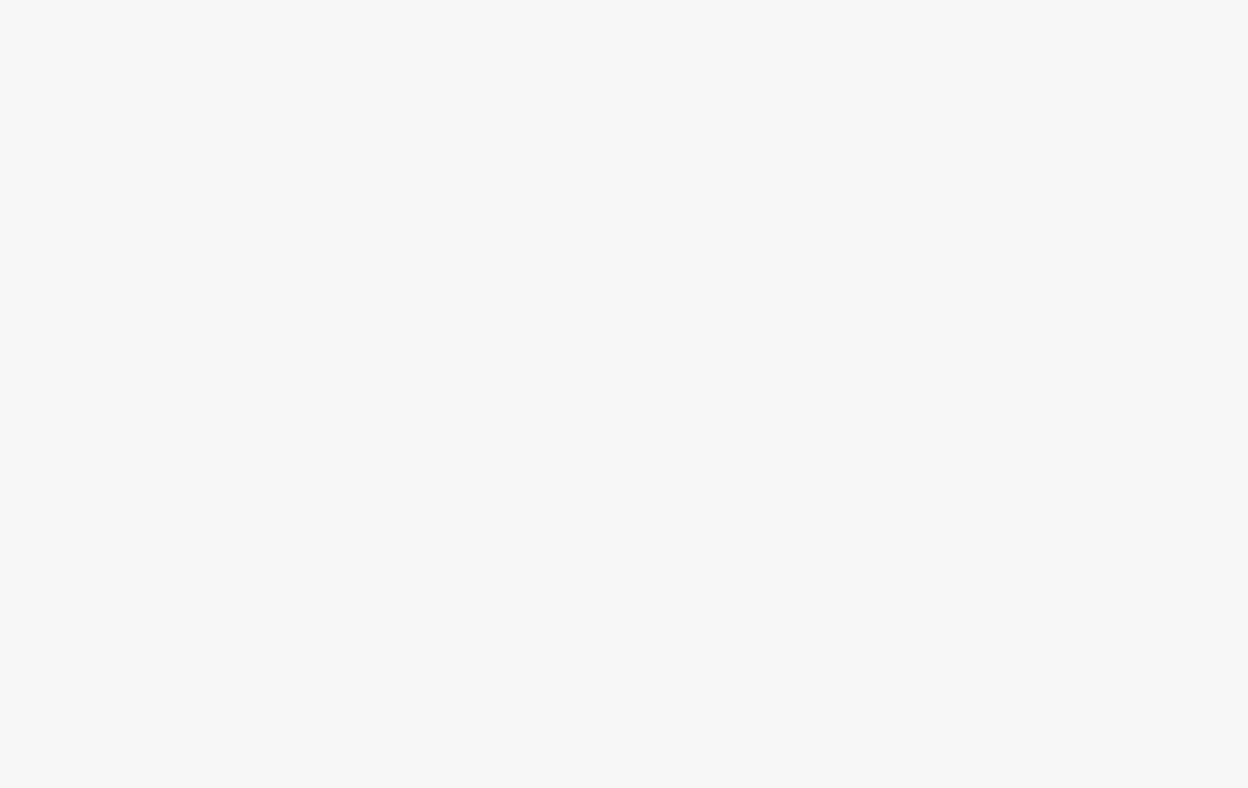


16



1. Email Integration

Fulfilment code for Email Generation and sending HTML template as an attachment File app.py (For initializing the webhook call and handling request parameters)



**from flask** ​**import**​ **Flask, request, make\_response**

**import**​ **json**

**import**​ **os**

**from flask\_cors** ​**import**​ **cross\_origin**

**from SendEmail.sendEmail** ​**import**​ **EmailSender from logger** ​**import**​ **logger**

**from email\_templates** ​**import**​ **template\_reader**

**app = Flask(\_\_name\_\_)**

* ***geting and sending response to dialogflow* @app.route('/webhook', methods=['POST']) @cross\_origin()**

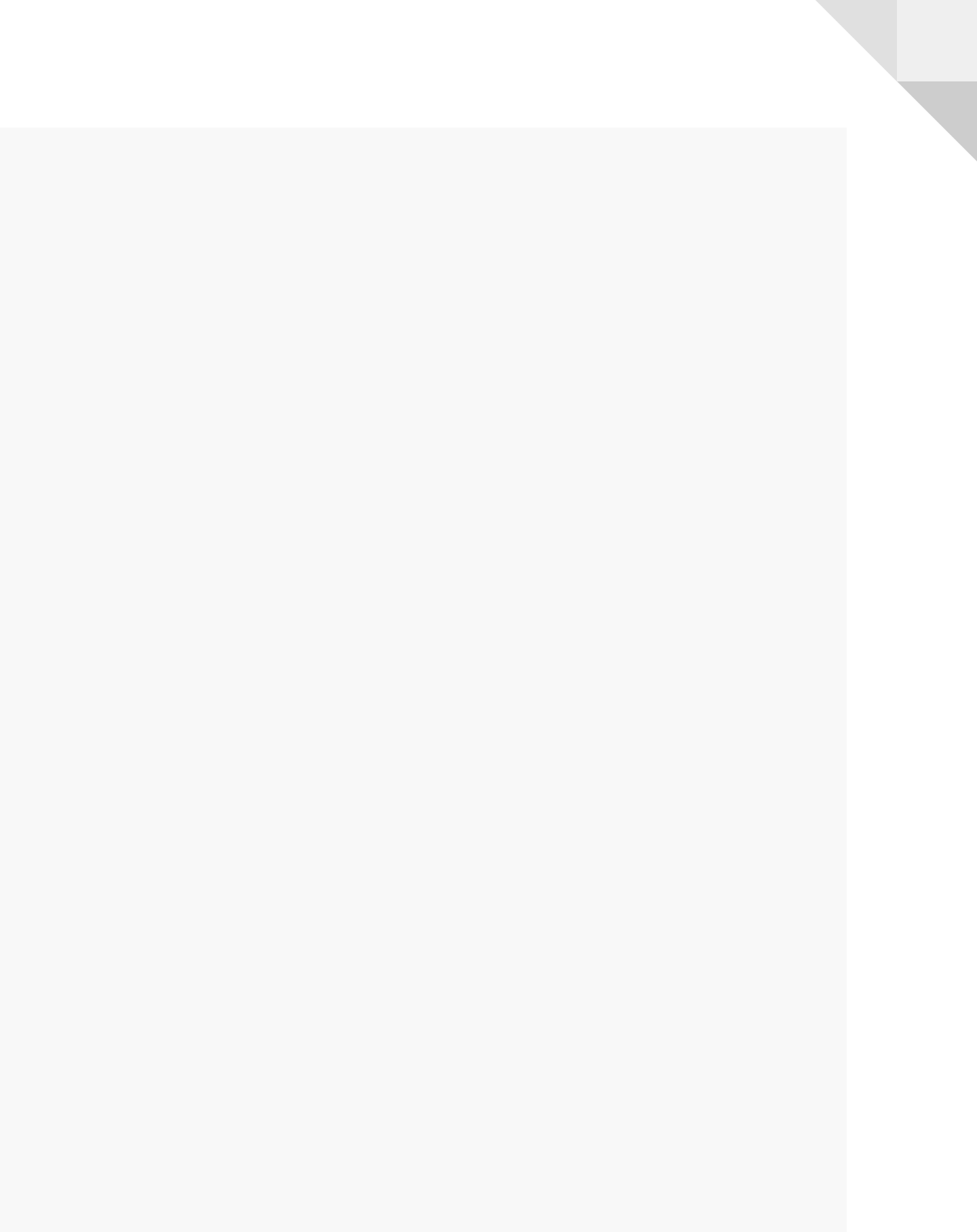
**def webhook():**

**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)**



17

**print(res)**

**r = make\_response(res)**

**r.headers['Content-Type'] = 'application/json'**

**return r**

* ***processing the request from dialogflow* def 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")**​

**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 == 'courseSelection'):**

**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)**

**log.write\_log(sessionID, "Bot**​ **Says: "**​ **+ req.get("fulfillmentText"**​**))**​

**else:**

**log.write\_log(sessionID,** ​**"Bot Says: "**​ **+ req.get(**​**"fulfillmentText"**​**))**

**if \_\_name\_\_ == '\_\_main\_\_':**

**port = int(os.getenv('PORT',** ​**5000**​**))**

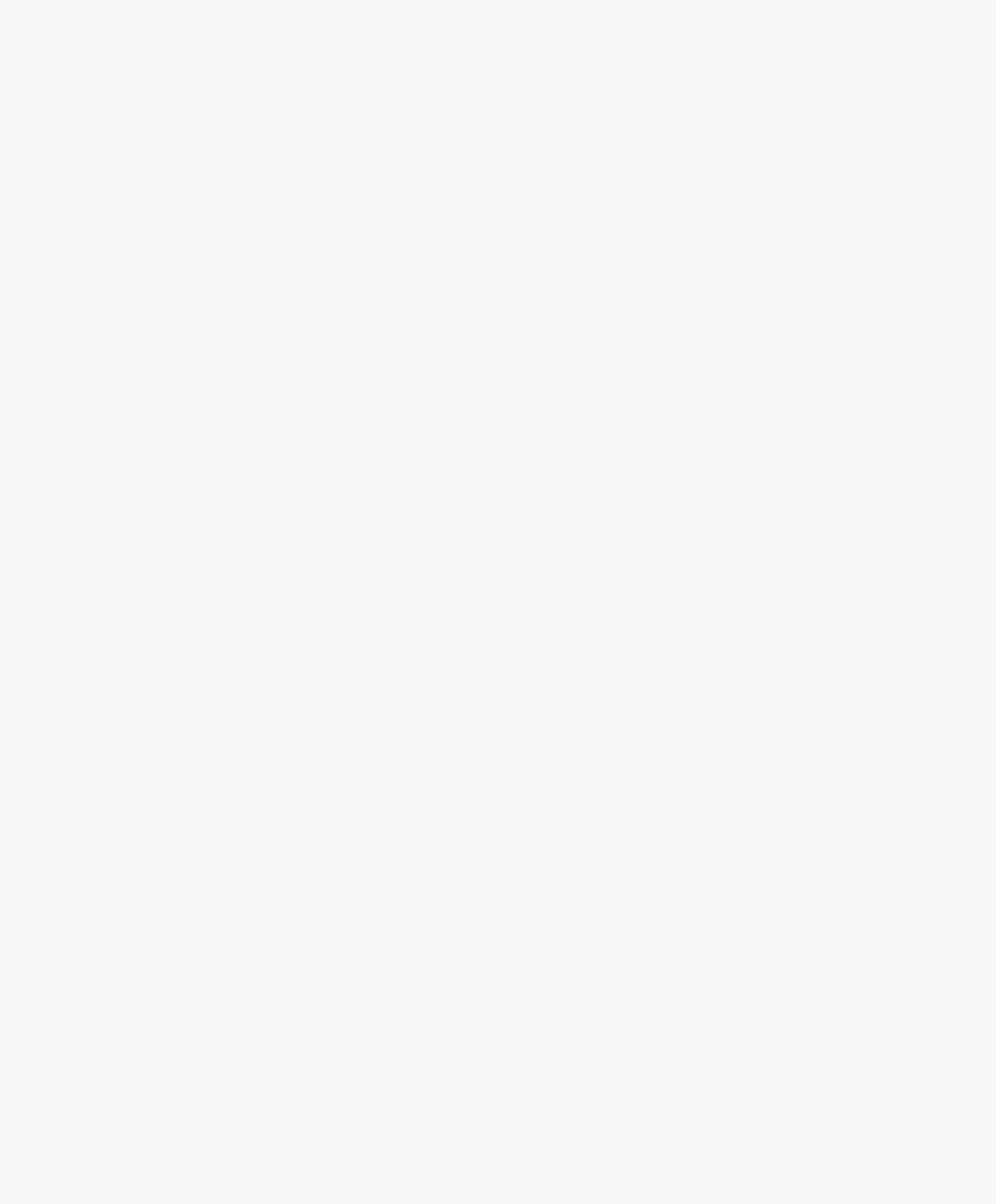
**print("Starting**​ **app on port %d"**​ **% port)**

**app.run(debug=False, port=port, host="0.0.0.0")**



18

File sendEmail.py (For sending the HTML email template by configuring mail settings)



**import smtplib**

**from email.mime.multipart import MIMEMultipart from email.mime.text import MIMEText from email.mime.base import MIMEBase**

**from config\_reader import ConfigReader**

**class** ​**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'**​**]**​ **= 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**

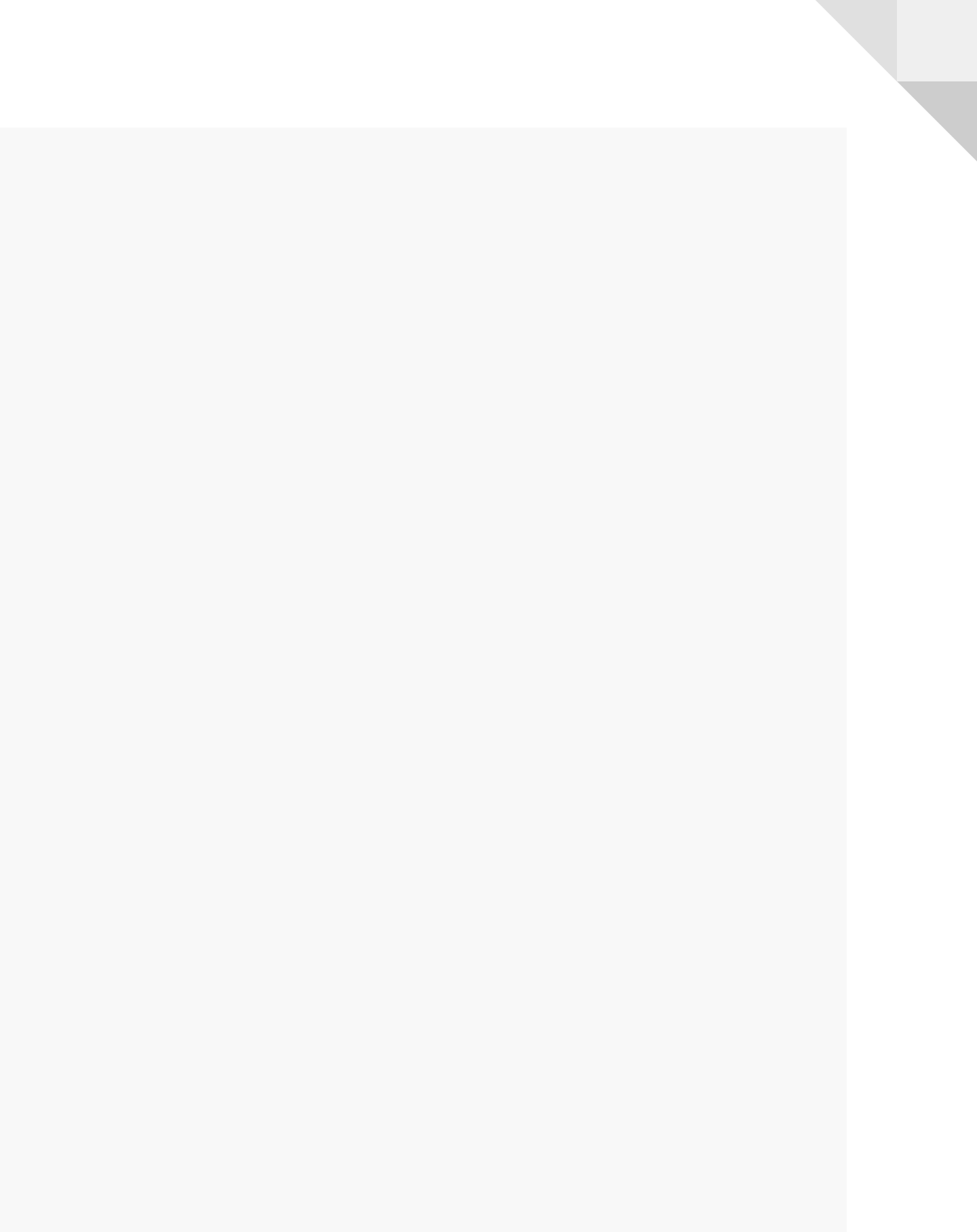
​***# attach the body with the msg instance*** **self.msg.attach(MIMEText(body,** ​**'html'**​**))**

​***# instance of MIMEBase and named as p***

**self.p = MIMEBase(**​**'application'**​**,** ​**'octet-stream'**​**)**

​***# creates SMTP session***



19

**self.smtp = smtplib.SMTP\_SSL(**​**'smtp.gmail.com'**​**,** ​**465**​**)**

​***# 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)**

**print(**​**"Email sent to User!"**​**)**

​***# 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 receivers email address***

**self.msg[**​**'To'**​**] = self.configuration[**​**'SALES\_TEAM\_EMAIL'**​**]**

​***# storing the subject***

**self.msg['Subject'**​**]**​ **=**

**self.configuration['SALES**​**\_TEAM\_EMAIL\_SUBJECT']**​

**body = body.replace(**​**'cust\_name'**​**, cust\_name)**

**body = body.replace(**​**'cust\_contact'**​**, cust\_contact)**



20

**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\_SSL(**​**'smtp.gmail.com'**​**,** ​**465**​**)**

​***# 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)**

**print("Email**​ **sent to Sales!")**​

​***# terminating the session***

**self.smtp.quit()**

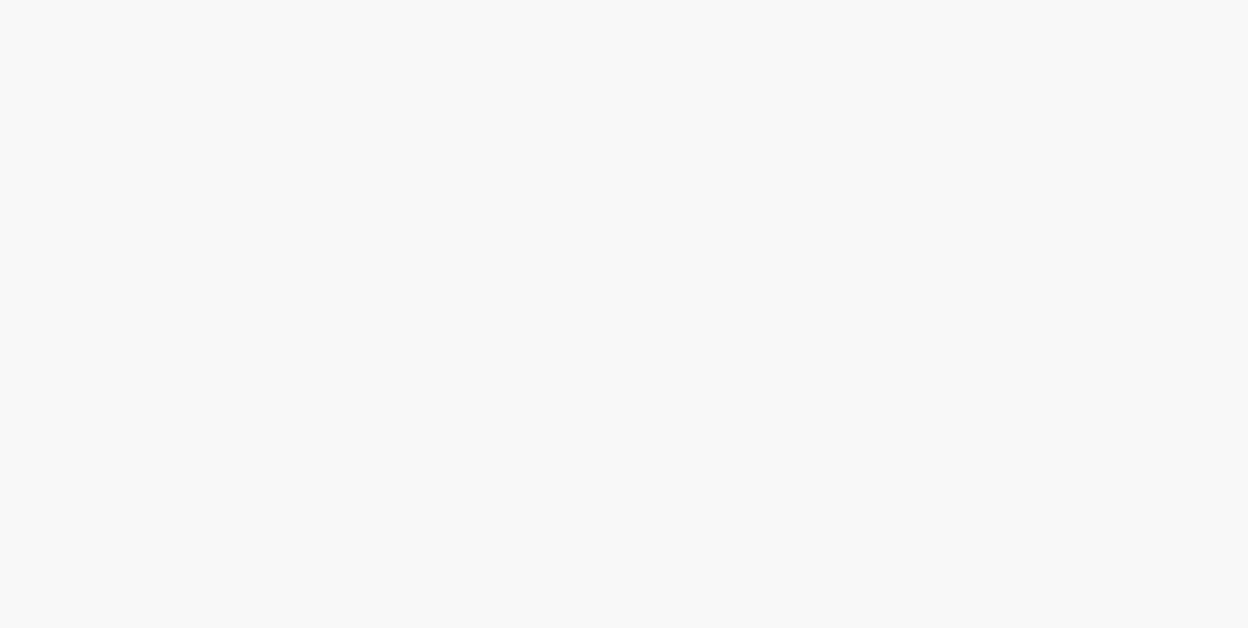
**except Exception as** ​**e:**

**print(**​**'the exception is '**​ **+ str(e))**



21

File logger.py (For logging all the conversation and saving it into a file)



**from datetime import datetime**

**class** ​**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()**

1. Deployment

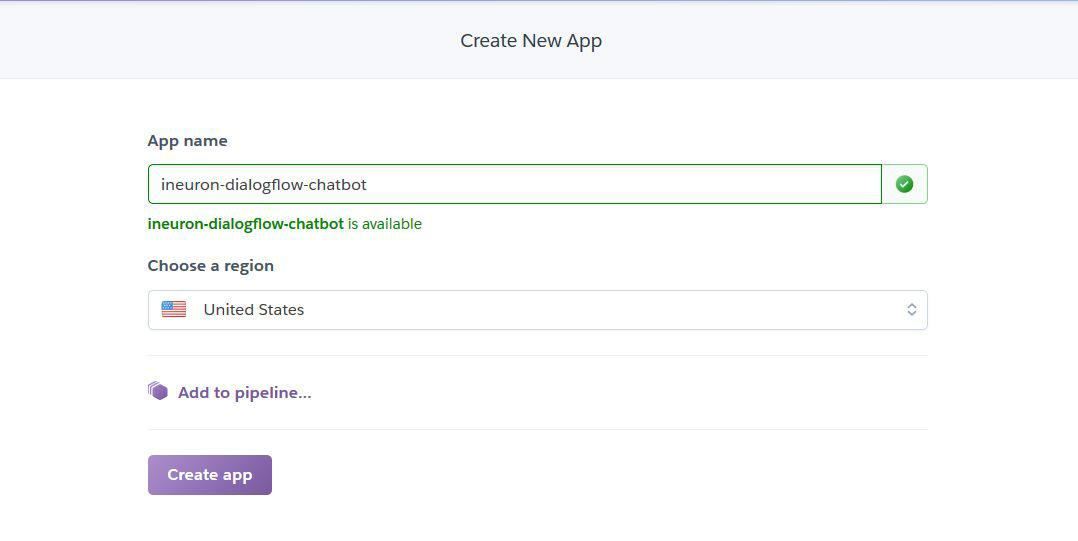
We would be deploying our project to Heroku.

8.1After Signing In into your account click new to create a new app.



22

8.2 Provide name ineuron-dialogflow-chatbot and click create app.



8.3 Select the created app.



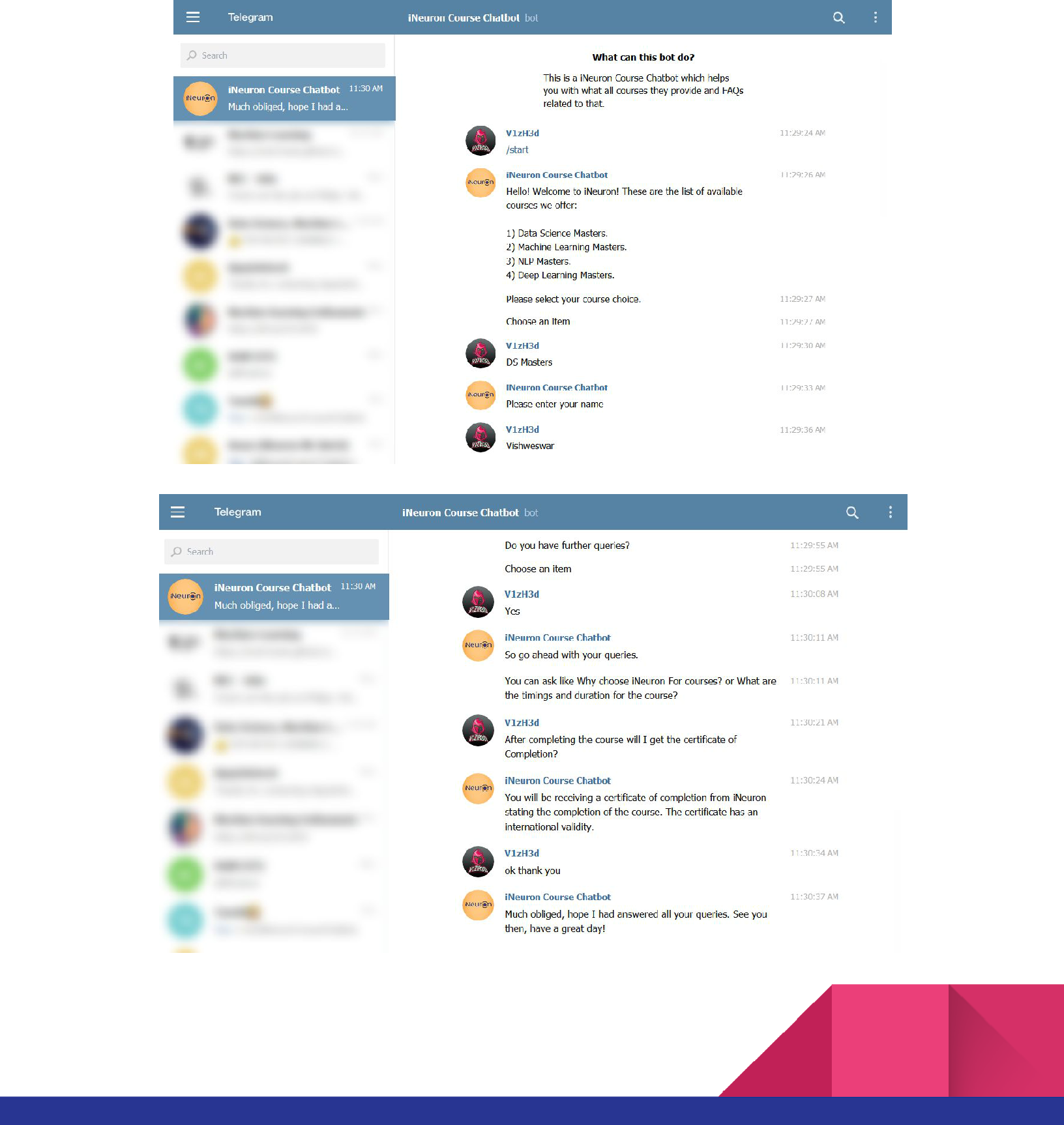
8.4 Follow the steps provided to deploy your app.



23

1. Chatbot Testing

Testing chatbot over Telegram.



24

