



Topic	WEB APP DEVELOPMENT: UNDERSTANDING jQuery AJAX	
Class Description	The student will learn to send and receive data on a webpage using AJAX and jQuery.	
Class	PRO C117	
Class time	45 mins	
Goal	<ul style="list-style-type: none"> • Role of AJAX in rendering HTML pages. • AJAX call using jQuery. • Use of Flask API and Machine Learning model for emotion prediction on the HTML page. 	
Resources Required	<ul style="list-style-type: none"> • Teacher Resources: <ul style="list-style-type: none"> ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ○ Smartphone • Student Resources: <ul style="list-style-type: none"> ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen 	
Class structure	Warm-Up Teacher-led Activity 1 Student-led Activity 1 Wrap-Up	10 mins 10 mins 20 mins 05 mins
Credit	jQuery by John Resig Flask by Armin Ronacher and contributors	
WARM-UP SESSION - 10 mins		
<div>  </div> <p>Teacher Starts Slideshow</p> <p>Slide 1 to 5</p>		

Refer to speaker notes and follow the instructions on each slide.	
<p>Hey <student's name>. How are you? It's great to see you! Are you excited to learn something new today?</p> <p>Following are the WARM-UP session deliverables:</p> <ul style="list-style-type: none"> • Greet the student. • Revision of previous class activities. • Quizzes. 	<p>ESR: Hi, thanks! Yes I am excited about it!</p> <p>Click on the slide show tab and present the slides</p>
<p align="center">WARM-UP QUIZ Click on In-Class Quiz</p>	
<p align="center">Continue WARM-UP Session Slide 6 to 18</p> 	
<p>Following are the session deliverables:</p> <ul style="list-style-type: none"> • Appreciate the student. • Narrate the story by using hand gestures and voice modulation methods to bring in more interest in students. 	
Teacher Action	Student Action
<p>Uptil now, we had seen that we can use Flask to render an HTML page. Today we'll be integrating the Machine Learning model for Text Sentiment Analysis in our webpage for prediction.</p> <p>We would take the user input and predict the emotion related to the text entered. Thus, the user is sending the request to predict the sentiment and in return, the emotions will be displayed. So, for example when a user enters 'It was a great day', the result will be a Happy or smiling emoji.</p>	

To process this request and make it reach the Python file with API, an intermediary is used which is known as **AJAX**.

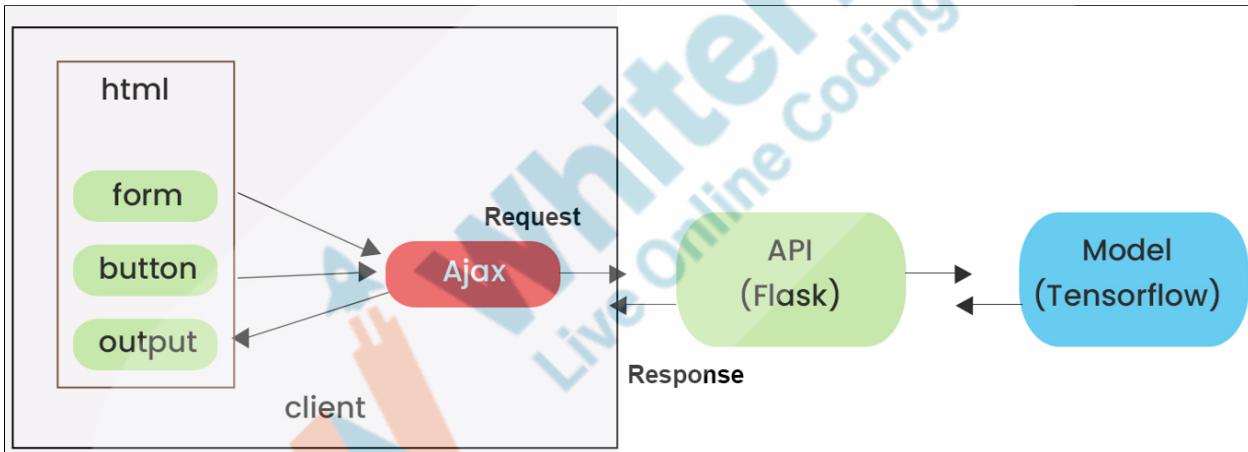
ESR: Yes.

Do you use social media websites?

Whenever someone sends you a message, it pops up directly without reloading the page right?

AJAX helps in refreshing a **part of the HTML page** rather than reloading the whole page again and again.

So, today we'll be learning about **AJAX**. and how it can be used to send requests and receive responses from HTML pages.



When we search for something on a web browser and the time when a web page is completely loaded:

- The browser sends a **request**.
- The server **responds** and sends back the requested information as an HTML file.

Now, these requests are processed by APIs we have written using **Flask**. To render these pages APIs are used. To call these APIs we'll use AJAX today.

Isn't it interesting?

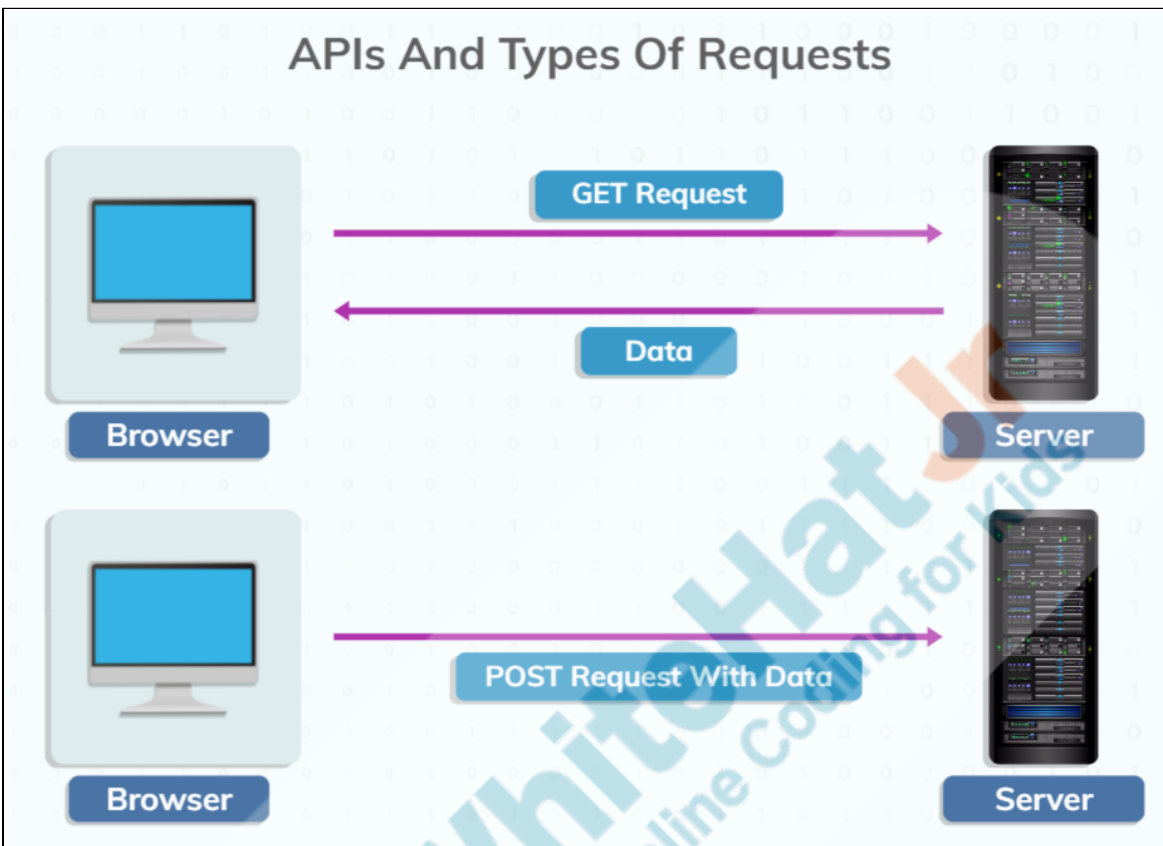
Let's get started.

APIs:

An API (Application Programming Interface) is a software intermediary that allows two applications to talk to each other. They are written to make requests to get or send some information from the server.

Types of Request:

The most common types of request methods are GET and POST. In GET, the client (browser) sends a request to get some data from the server. Whereas, in POST, the client sends the data to the server.




What is AJAX?

AJAX stands for **Asynchronous JavaScript And XML**. **AJAX** is not a programming language but it uses JavaScript.

AJAX is used to send and receive data from HTML to Python files. We'll be using **jQuery** to create the AJAX call.

jQuery is a **JavaScript Library**. It uses CSS style selectors to select elements of an HTML page such as buttons, text inputs, etc, and uses AJAX to change the state of these elements.

<p>How is jQuery used?</p> <p>jQuery is a JavaScript library. The purpose of jQuery is to make it much easier to use JavaScript on websites.</p> <p>jQuery is used to shorten the JavaScript code to accomplish and wrap them into methods that you can call with a single line of code.</p> <p>Selectors are used to select one or more HTML elements using jQuery. Once an element is selected then we can perform various operations on that selected element using AJAX.</p> <p><i>Note: Refer Teacher Activity 1 for jQuery.</i></p>	
<div>  Teacher Ends Slideshow </div>	
TEACHER-LED ACTIVITY - 10 mins	
Teacher Initiates Screen Share	
<p align="center"><u>ACTIVITY</u></p> <ul style="list-style-type: none"> • Creating Flask API for sentiment prediction • Creating Ajax call to send and receive the request 	
Teacher Action	Student Action
<p>Remember in the last class I had mentioned integrating the text sentiment model on our webpage?</p> <p>So today we'll be integrating the model to predict the emotion on the webpage.</p> <p><i>Note: Open Teacher activity 2 for Boilerplate code. Create a folder and save all the files. Check the directory structure. It has all the required files.</i></p>	<p>ESR: Yes.</p>

We'll start with an **HTML** page in the **templates** folder.
This page has elements in the following manner:

1. A text box for displaying the date.
2. A text area for entering text.
3. A submit button for sending the request.
4. A paragraph for displaying the emotion as a result.
Since we need to hide it initially, the display is kept as none.
5. An image (emoji) will be shown according to the emotion. Initially, this too remains hidden.

```
templates > index.html > html > body
1  <!DOCTYPE html>
2  <html>
3
4  <head>
5    <title>Student Activity</title>
6  </head>
7
8  <body>
9    <!--Date-->
10   <div>
11     <h4 id='display_date'>DATE</h4>
12   </div>
13
14   <!--Input Text -->
15   <div>
16     <h4>Enter Text</h4>
17     <textarea id="text" name="text_input" rows="4" maxlength="200"
18       placeholder="How was your day?"></textarea>
19     <br><br>
20   </div>
```

```
21
22     <!--Predict Button-->
23     <div>
24         <button id="predict_button" type="submit">Predict Emotion</button><br><br>
25     </div>
26
27     <!--Predicted Emotion-->
28     <div>
29         <p id="prediction" style="display:none"></p>
30         <img id="emo_img_url" src="" width="55" min-height="50" alt=""
31             style="display:none">
32     </div>
33
34 </body>
35
36 </html>
```

Now to use jQuery, it is embedded into the **<script>** tag of HTML file between the **<head>** tag and the **<title>** tag. The source is specified as Content Delivery Network given by ["https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"](https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js) .

Since we'll be using AJAX, we have to create a JavaScript file by the name **index.js** in the **static** folder. Also indicate the same in the HTML file using the **<script>** tag.

To use jquery we have to include the **jQuery** library in the HTML file. Also, **index.js(js file with AJAX call)** must be included to send and receive data.


```
templates > index.html > html > body > div
1  <!DOCTYPE html>
2  <html>
3
4  <head>
5      <title>Student Activity</title>
6      <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.0/jquery.min.js">
7      <script src="./static/index.js"></script>
8
9  </head>
```

Now, we have to create a Python file to render this HTML page.

Also, to predict the result we will use the model that we had used for sentiment analysis. There will be two Python files.

1. **app.py** for the API.
2. **text_sentiment_prediction.py** for the sentiment analysis.

Let's run this file to check our HTML webpage.

```
app.py >
1  from flask import Flask, render_template
2
3  app = Flask(__name__)
4
5  @app.route('/')
6  def index():
7      return render_template('index.html')
```

DATE

Enter Text

How was your day?

Predict Emotion

Further, let's check the **text_sentiment_prediction.py** file. This has the model trained in it. These are the same operations that we performed previously.

Dictionary is included to display the predicted emotion in the form of emoticons. These emoticons are provided to you in the static folder.

```
text_sentiment_prediction.py > predict
1 import pandas as pd
2 import numpy as np
3
4 import tensorflow
5 from tensorflow.keras.preprocessing.text import Tokenizer
6 from tensorflow.keras.preprocessing.sequence import pad_sequences
7 from tensorflow.keras.models import load_model
8
9
10 train_data = pd.read_csv("./static/assets/data_files/tweet_emotions.csv")
11 training_sentences = []
12
13 for i in range(len(train_data)):
14     sentence = train_data.loc[i, "content"]
15     training_sentences.append(sentence)
16
17 model = load_model("./static/assets/model_files/Tweet_Emotion.h5")
18
19 vocab_size = 40000
20 max_length = 100
21 trunc_type = "post"
22 padding_type = "post"
23 oov_tok = "<OOV>"
24
25 tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)
26 tokenizer.fit_on_texts(training_sentences)
27
28 emo_code_url = {
29     "empty": [0, "./static/assets/emoticons/Empty.png"],
30     "sadness": [1, "./static/assets/emoticons/Sadness.png"],
31     "enthusiasm": [2, "./static/assets/emoticons/Enthusiasm.png"],
32     "neutral": [3, "./static/assets/emoticons/Neutral.png"],
33     "worry": [4, "./static/assets/emoticons/Worry.png"],
34     "surprise": [5, "./static/assets/emoticons/Surprise.png"],
35     "love": [6, "./static/assets/emoticons/Love.png"],
36     "fun": [7, "./static/assets/emoticons/fun.png"],
37     "hate": [8, "./static/assets/emoticons/hate.png"],
38     "happiness": [9, "./static/assets/emoticons/happiness.png"],
39     "boredom": [10, "./static/assets/emoticons/boredom.png"],
40     "relief": [11, "./static/assets/emoticons/relief.png"],
41     "anger": [12, "./static/assets/emoticons/anger.png"]
42 }
43 }
```

Now, we'll have to write a **function** to predict the emotion.

Let's write this function in the

text_sentiment_prediction.py file.

This function takes the text from **Flask API** and returns the **predicted emotion** and **emoji associated** with the emotion.

```

37 def predict(text):
38     predicted_emotion=""
39     predicted_emotion_img_url=""
40
41     if text!="":
42         sentence = []
43         sentence.append(text)
44
45         sequences = tokenizer.texts_to_sequences(sentence)
46
47         padded = pad_sequences(
48             sequences, maxlen=max_length, padding=padding_type,
49             truncating=trunc_type)
50
51         testing_padded = np.array(padded)
52
53         predicted_class_label = np.argmax(model.predict(testing_padded), axis=1)
54         print(predicted_class_label)
55         for key, value in emo_code_url.items():
56             if value[0]==predicted_class_label:
57                 predicted_emotion_img_url=value[1]
58                 predicted_emotion=key
59         return predicted_emotion, predicted_emotion_img_url

```

To get the request from the user (**index.html**), we have to send it to the flask through AJAX. Let's create a JavaScript file by name **index.js** in the **static** folder.

Assign the **Date()** function to variable **date**. The **display_date** variable is used to display the current date on the HTML page using this function.

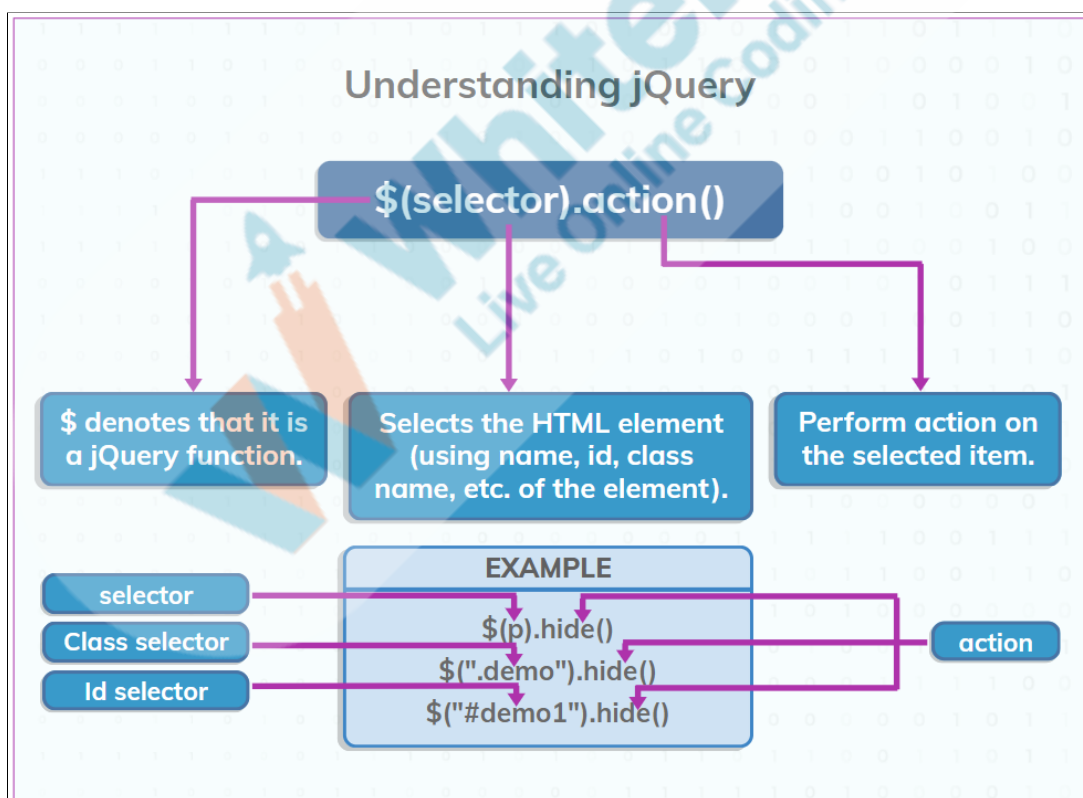
Next, we'll be using the **jQuery selector** for displaying the date in place of the heading of the HTML page.

“\$” denotes that we are using a **jQuery selector**. It is followed by the action to be performed on the HTML element.

Inside selector, we give:

1. The element to be selected. For e.g., **p** stands for paragraph. e.g. **\$(p)**
2. Class selector by giving class name followed by ‘.’
e.g. **\$(".demo")**
3. Id selector by writing id of the HTML element followed by #. e.g. **\$("#demo")**

[Teacher Activity3: jQuery.AJAX API](#)



The **\$(document).ready()** function is used to initialize the document. It is used to make sure that HTML page is rendered first

```
$(document).ready(function(){
    //jQuery methods go here...
});
```

The **\$("#display_date).html()** function is used to select the **display_date** heading element of an HTML page and display the date.

```
static > JS index.js > {} callback
1 var date = new Date()
2 let display_date= "Date:" + date.toLocaleDateString()
3
4 //Load HTML DOM
5 $(document).ready(function () {
6     $("#display_date").html(display_date)
7 })
```

To accept the data from the HTML page and send a response, the following steps are followed:

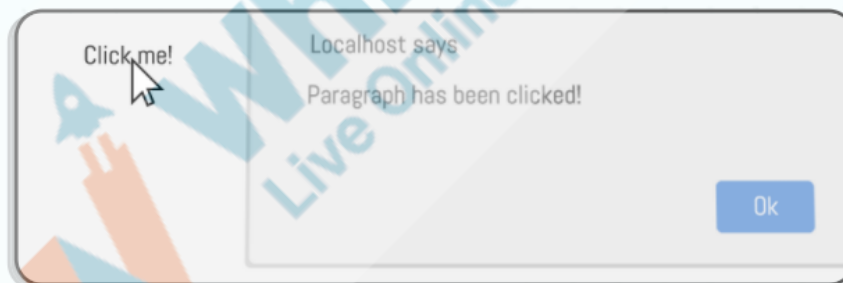
1. Define a variable **predicted_emotion** to store the result of **predict()** function.
2. Let's create a function for sending **request** through **AJAX call** and receiving the **response**.

- When the '**Predict Emotion**' button is clicked, the text should be passed to the API through AJAX. So we'll use a selector action which is '**click()**'.

Common Actions - click()

```
$(selector).click (function () {  
    //some click action  
});
```

```
$(p).click(function(){  
    alert("Paragraph has been clicked!")  
});
```



- As soon as the button is clicked, the text is stored in a variable called **input_data**.
- Selector is used to select the text input and the value of the text input is given to the variable **input_data** by creating a '**text**' object.

```
static > JS index.js > ($) callback > click() callback > data
1  var date = new Date()
2  let display_date= "Date:" + date.toLocaleDateString()
3
4  //Load HTML DOM
5  $(document).ready(function () {
6      $("#display_date").html(display_date)
7  })
8
9  let predicted_emotion;
10 //HTML-->JavaScript-->Flask
11 //Flask-->JavaScript-->HTML
12
13 $(function () {
14     $("#predict_button").click(function () {
15
16         let input_data = {
17             "text": $("#text").val()
18         }
19         console.log(input_data)
20     })
21 })
```

6. In this function, we'll create the AJAX function. This method requires parameters in the form of key-value pairs. Some common key-value pairs that need to be passed are:

- i. **url**: URL of the page where we want to send the request.
- ii. **type**: type of the request, GET or POST.
- iii. **data**: actual data that needs to be sent to the server.
- iv. **dataType**: datatype of the response.
- v. **contentType**: type of content used while sending a request to the server.
- vi. **success**: a function after the request is successful.
- vii. **error**: a function after the request fails.

Calling APIs Through jQuery AJAX

Syntax:

```
$.ajax({  
  
  name:value,  
  
  name:value,  
  ....  
  ....  
})
```

url

type

data

dataType

contentType

success

error

Therefore, when a request is sent using the **'POST'** method and response is received in the variable **result**.

The request is sent in **JSON** format. **JSON** stands for **JavaScript Object Notation**. It is used to store human-readable text and transmit data.

On success, the function is called with the **result** received from Flask.

```
13  $(function () {  
14      $("#predict_button").click(function () {  
15  
16          let input_data = {  
17              "text": $("#text").val()  
18          }  
19          console.log(input_data)  
20  
21          $.ajax({  
22              type: 'POST',  
23              url: "/predict-emotion",  
24              data: JSON.stringify(input_data),  
25              dataType: "json",  
26              contentType: 'application/json',  
27              success: function (result) {
```

The responses we get from Flask are **the emotion** and **the emoticon** related to the emotion.

After getting the response the results are displayed on the HTML page using jQuery selectors.

Also, errors such are also displayed. These errors occurs when AJAX call is no able to find the required data. Such as the Python file for sentiment processing, emoticons are missing.

```
$.ajax({
  type: 'POST',
  url: "/predict-emotion",
  data: JSON.stringify(input_data),
  dataType: "json",
  contentType: 'application/json',
  success: function (result) {

    // Result Received From Flask ----->JavaScript
    predicted_emotion = result.data.predicted_emotion
    emo_url = result.data.predicted_emotion_img_url

    // Display Result Using JavaScript----->HTML
    $("#prediction").html(predicted_emotion)
    $('#prediction').css("display", "block");

    $("#emo_img_url").attr('src', emo_url);
    $('#emo_img_url').css("display", "block");

  },
  error: function (result) {
    alert(result.responseJSON.message)
  }
});
```

Let's write the code for Flask to receive the request and process it. The steps are:

1. Create an **API** to receive the request in **POST** format.
2. When the request is received, the function **predict_emotion ()** is called.
3. This function will receive the data in JASON format from the '**text**' object created in AJAX.
4. Now, there are two possibilities.
 - a. The user clicks the **Predict Emotion** button without entering text. In this case, the error must be generated and sent back to AJAX.
 - b. The user enters text for prediction.
5. When no text is entered the response is sent with two parameters:

- a. **status:** 'error'
This shows that the request is not successful.
- b. **Message:** If an error occurs, then the message is to be sent to the HTML page.

These errors are displayed on command prompt.

```

app.py > predict_emotion
1 from flask import Flask, render_template, url_for, request, jsonify
2 from text_sentiment_prediction import *
3
4 app = Flask(__name__)
5 @app.route('/')
6 def index():
7     return render_template('index.html')
8
9 @app.route('/predict-emotion', methods=["POST"])
10 def predict_emotion():
11
12     # Get Input Text from POST Request
13     input_text = request.json.get("text")
14
15     if not input_text:
16         # Response to send if the input_text is undefined
17         response = {
18             "status": "error",
19             "message": "Please enter some text to predict emotion!"
20         }
21     return jsonify(response)
  
```


6. **Input_text** is given to the **predict** function which returns the results in **predicted_emotion** and **predicted_emotion_img_url**.
7. Now, for the successful response, these two variables are defined again in the **data**.
8. Now the data is sent in JSON format using the **jsonify()** function.



```

app.py > ...
15     if not input_text:
16         # Response to send if the input_text is undefined
17         response = {
18             "status": "error",
19             "message": "Please enter some text to predict emotion!"
20         }
21         return jsonify(response)
22     else:
23         predicted_emotion, predicted_emotion_img_url = predict(input_text)
24
25         # Response to send if the input_text is not undefined
26         response = {
27             "status": "success",
28             "data": {
29                 "predicted_emotion": predicted_emotion,
30                 "predicted_emotion_img_url": predicted_emotion_img_url
31             }
32         }
33
34         # Send Response
35         return jsonify(response)
36
37 app.run(debug=True)



```

Let's run the code using command prompt and check the prediction.

<p>Date:1/17/2022</p> <p>Enter Text</p> <p>It was a warm evening, I went for a walk with my father</p> <p>Predict Emotion</p> <p>neutral</p> 	<p>Date:1/17/2022</p> <p>Enter Text</p> <p>he was glad to hear the news of promotion</p> <p>Predict Emotion</p> <p>happiness</p> 	<p>Date:1/17/2022</p> <p>Enter Text</p> <p>I met with an accident , broke my knee</p> <p>Predict Emotion</p> <p>worry</p> 
--	--	--

This way we can deploy our ML model on the webpage. That was interesting!	
Teacher Stops Screen Share	
So now it's your turn. Please share your screen with me.	
<div style="text-align: center;">  <p>Teacher Starts Slideshow Slide 19 to 20 <Note: Only Applicable for Classes with VA> Refer to speaker notes and follow the instructions on each slide.</p> </div>	
We have one more class challenge for you. Can you solve it? Let's try. I will guide you through it.	ESR: Yes
<div style="text-align: center;">  <p>Teacher Ends Slideshow</p> </div>	
STUDENT-LED ACTIVITY - 20 mins	
<ul style="list-style-type: none"> • Ask the student to press the ESC key to come back to the panel. • Guide the student to start Screen Share. • The teacher gets into Full Screen. 	
Student Initiates Screen Share	
<div style="text-align: center;"> <p><u>ACTIVITY</u></p> <ul style="list-style-type: none"> • Create a predict function in text_sentiment_prediction file • Write AJAX call to send request and receive response from Flask API • Write API to process the request sent by AJAX </div>	

Teacher Action	Student Action
<p>Note: Guide the student to open Student Activity 1 and make a copy of the file.</p> <p>Note 1: The student will perform the same activities as done by the teacher.</p> <p>Note 2: Please refer to Teacher Activity to follow through the steps and guide the student to complete the activities</p>	
<p>Show the directory structure and guide the student to perform the activities.</p> <ol style="list-style-type: none"> 1. Go to the command prompt and traverse the working folder as we did in class 110. 2. Create a virtual environment (Windows/Mac) using python -m venv <name_of_the_environment> for testing the model. 3. Activate the virtual environment using following command: <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <code><name_of_the_environment>\Scripts\activate</code> </div> <p><i>Run the command to create a virtual environment with the name "Model_Deployment". The environment is user-defined. We can keep the name as we want relevant to our project.</i></p>	
<ol style="list-style-type: none"> 4. Now, run the app.py file and check the output in localhost. 5. Now, you are provided with the text_sentiment_prediction file. In this file, create a predict function to take the text, predict the emotion and emoticons associated with the emotion. 6. Next, you are provided with the JavaScript file in the static folder. 7. Create the AJAX call for providing the request in 	

<p>the 'POST' method and receive the response.</p> <p>8. Next, we'll be creating the API for receiving the text data in JSON format from the index.js file. Process it and send the response again in JSON format to index.js</p> <p>9. After writing the code, let's run it and check the output on the HTML page.</p>	
<p>Great work!!!</p> <p>Thus, we have integrated the ML model on our webpage to check the output on localhost.</p> <p>You did amazing work today!</p>	
Teacher Guides Student to Stop Screen Share	
WRAP-UP SESSION - 05 mins	
<div style="text-align: center;">  Teacher Starts Slideshow Slide 21 to 26 <Note: Only Applicable for Classes with VA> </div>	
<p>Activity details</p> <p>You performed very well today.</p> <p>Following are the WRAP-UP session deliverables:</p> <ul style="list-style-type: none"> • Appreciate the student. • Revise the current class activities. • Discuss the quizzes. 	
<div style="text-align: center;"> WRAP-UP QUIZ Click on In-Class Quiz </div>	
<div style="text-align: center;">  Continue WRAP-UP Session Slide 27 to 32 </div>	

<Note: Only Applicable for Classes with VA>

Activity Details

Following are the session deliverables:

- Explain the facts and trivia
- Next class challenge
- Project for the day
- Additional Activity (Optional)

FEEDBACK

- Appreciate and compliment the student for trying to learn a difficult concept.
- Get to know how they are feeling after the session.
- Review and check their understanding.

Teacher Action

You get Hats off for your excellent work!

In the next class, we will learn about web app deployment using Python.

Student Action

Make sure you have given at least 2 Hats Off during the class for:

Creatively Solved Activities  +10

Great Question  +10

Strong Concentration  +10

PROJECT OVERVIEW DISCUSSION

Refer the document below in Activity Links Sections

Teacher Clicks

✕ End Class

ACTIVITY LINKS

Activity Name	Description	Links
Teacher Activity 1	SIMPLIFYING JavaScript - BASICS OF jQuery	https://s3-whjr-curriculum-uploads.whjr.online/35abdd8f-3d70-4cef-a2c2-dc82053efd1b.pdf
Teacher Activity 2	Boilerplate Code	https://github.com/procodingclass/PRO-C117-Boilerplate-Code
Teacher Activity 3	jQuery.AJAX API	https://api.jquery.com/jquery.ajax/
Teacher Activity 4	Reference Code	https://github.com/procodingclass/PRO-C117-Reference-Code
Teacher Reference 1	Project	https://s3-whjr-curriculum-uploads.whjr.online/be14c569-e416-4360-a836-0f4238cb1a63.pdf
Teacher Reference 2	Project Solution	https://github.com/procodingclass/PRO-C117-Project-Solution.git
Teacher Reference 3	Visual-Aid	https://s3-whjr-curriculum-uploads.whjr.online/2688434b-5eb2-4d71-8628-e914047c0843.html
Teacher Reference 4	In-Class Quiz	https://s3-whjr-curriculum-uploads.whjr.online/0521b1a4-1d3c-47ff-a89e-fd95bea76847.pdf
Student Activity 1	Boilerplate Code	https://github.com/procodingclass/PRO-C117-Boilerplate-Code