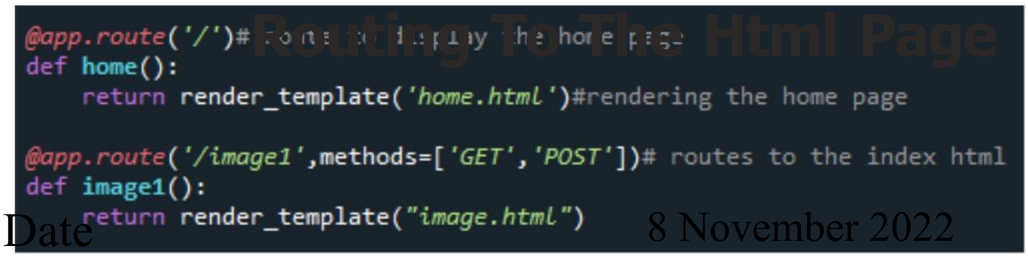
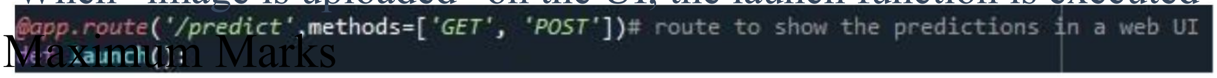
Routing To The Html Page



```
@app.route('/')# Route to display the home page
def home():
    return render_template('home.html')#rendering the home page

@app.route('/image1',methods=['GET','POST'])# routes to the index html
def image1():
    return render_template("image.html")
```

Date 8 November 2022

When “image is uploaded “on the UI, the launch function is executed



```
@app.route('/predict',methods=['GET', 'POST'])# route to show the predictions in a web UI
def launch():
```

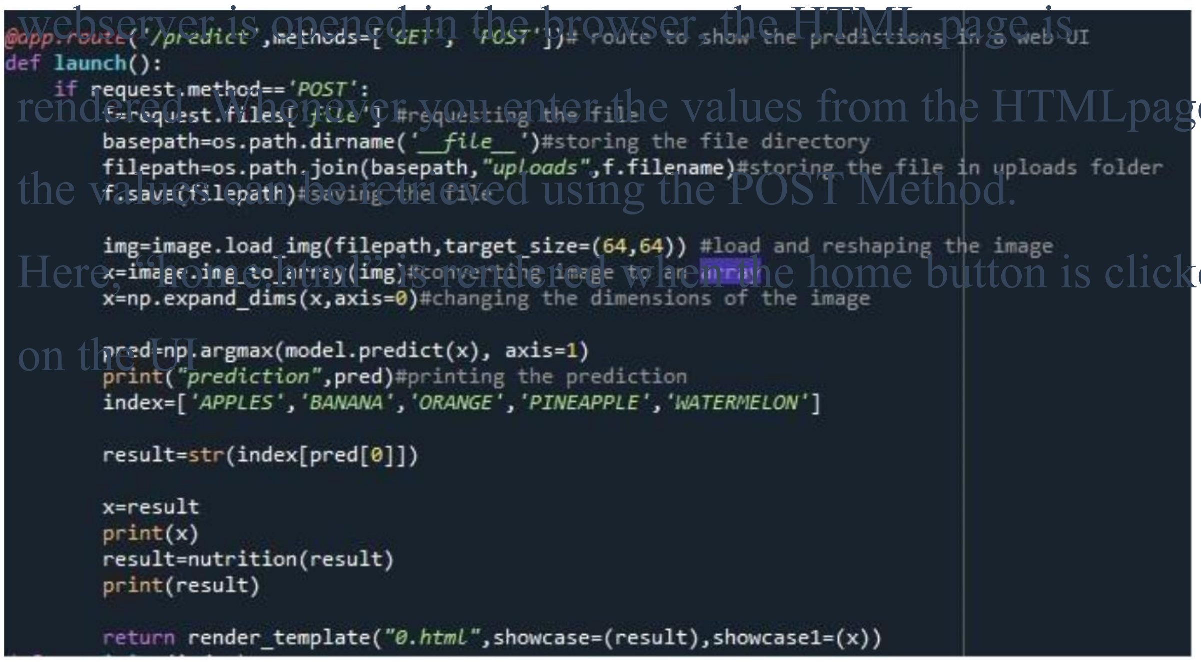
Maximum Marks

It will take the image request and we will be storing that image in our localsystem then we will convert the image into our required size and finally, we will be predicting the results with the help of our model which we trained and depending upon the class identified we will showcase the class name and its properties by rendering the respective html pages.

Here, the declared constructor is used to route to the HTML page createdearlier.

In the above example, the ‘/’ URL is bound with the home.html function. Hence, when the home page of the webserver is opened in the browser, the HTML page is rendered. Whenever you enter the values from the HTMLpage the values can be retrieved using the POST Method.

Here, “home.html” is rendered when the home button is clicked on the UI



```
@app.route('/predict',methods=['GET', 'POST'])# route to show the predictions in a web UI
def launch():
    if request.method=='POST':
        f=request.files['file'] #requesting the file
        basepath=os.path.dirname('__file__')#storing the file directory
        filepath=os.path.join(basepath,"uploads",f.filename)#storing the file in uploads folder
        f.save(filepath)#saving the file

        img=image.load_img(filepath,target_size=(64,64)) #load and reshaping the image
        x=image.img_to_array(img)#converting image to an array
        x=np.expand_dims(x,axis=0)#changing the dimensions of the image

        pred=np.argmax(model.predict(x), axis=1)
        print("prediction",pred)#printing the prediction
        index=['APPLES','BANANA','ORANGE','PINEAPPLE','WATERMELON']

        result=str(index[pred[0]])

        x=result
        print(x)
        result=nutrition(result)
        print(result)

        return render_template("0.html",showcase=(result),showcase1=(x))
```

API Integration:

Here we will be using Rapid API

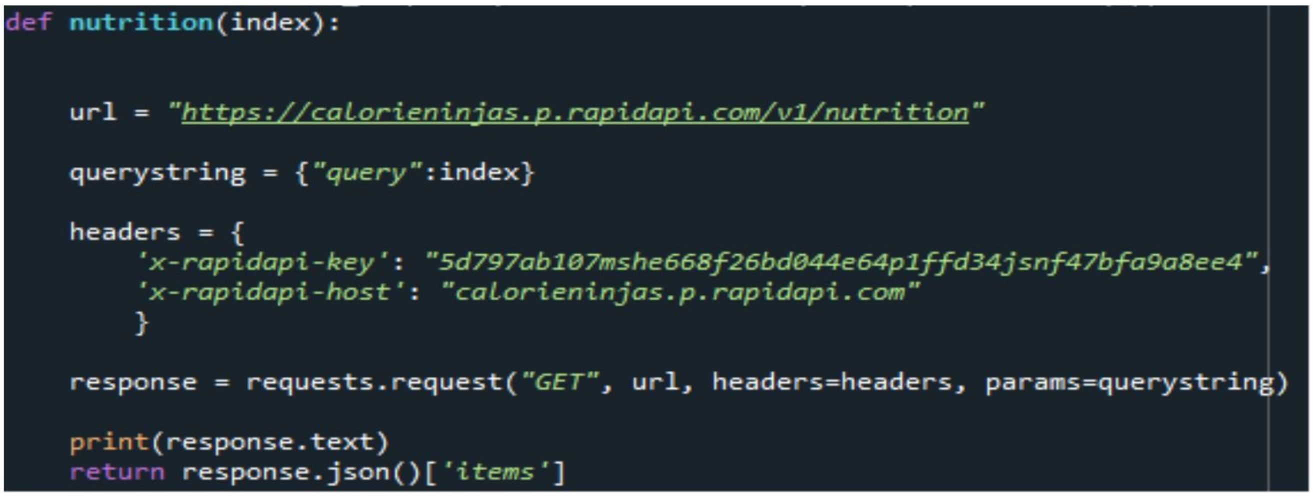
Using RapidAPI, developers can search and test the APIs, subscribe, and connect to the APIs — all with a single account, single API key and singleSDK. Engineering teams also use RapidAPI to share internal APIs and microservice documentation.

Reference link

API used: Link

The link above will allow us to test the food item and will result thenutrition content present in the food item.
NOTE: When we keep hitting the API the limit of it might expire. So making a smartuse of it will be an efficient way.

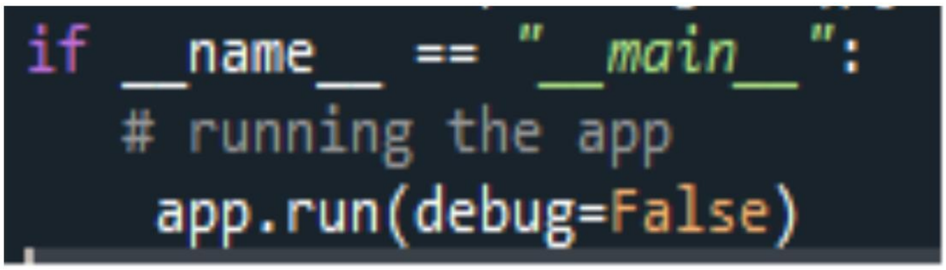
How to access and use the API will be shown in this video



```python
def nutrition(index):


    url = "https://calorieninjas.p.rapidapi.com/v1/nutrition"

    querystring = {"query":index}

    headers = {
        'x-rapidapi-key': "5d797ab107mshe668f26bd044e64p1ffd34jsnf47bfa9a8ee4",
        'x-rapidapi-host': "calorieninjas.p.rapidapi.com"
        }

    response = requests.request("GET", url, headers=headers, params=querystring)

    print(response.text)
    return response.json()['items']
```

Finally, Run the application

This is used to run the application in a localhost. The local host runs on portnumber 5000.(We can give different port numbers)



```python
if __name__ == "__main__":
    # running the app
     app.run(debug=False)
```