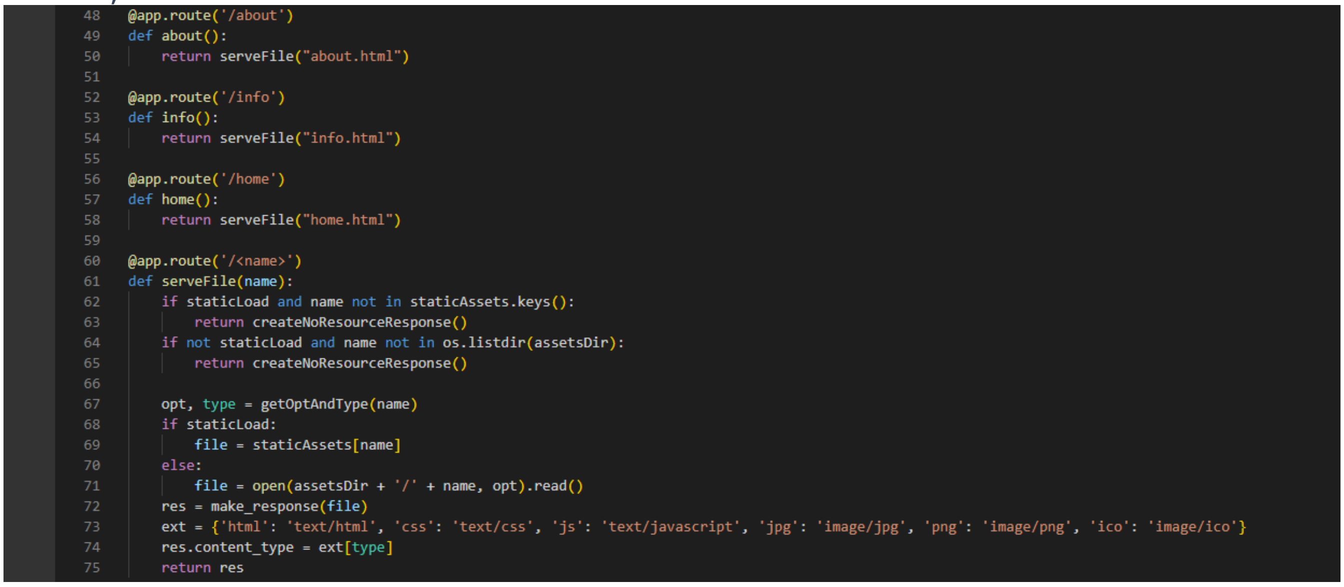
TOPIC : AI powered nutrition analyzer for fitness enthusiast

Team id : PNT2022TMID14641

Routing To The Html Page

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

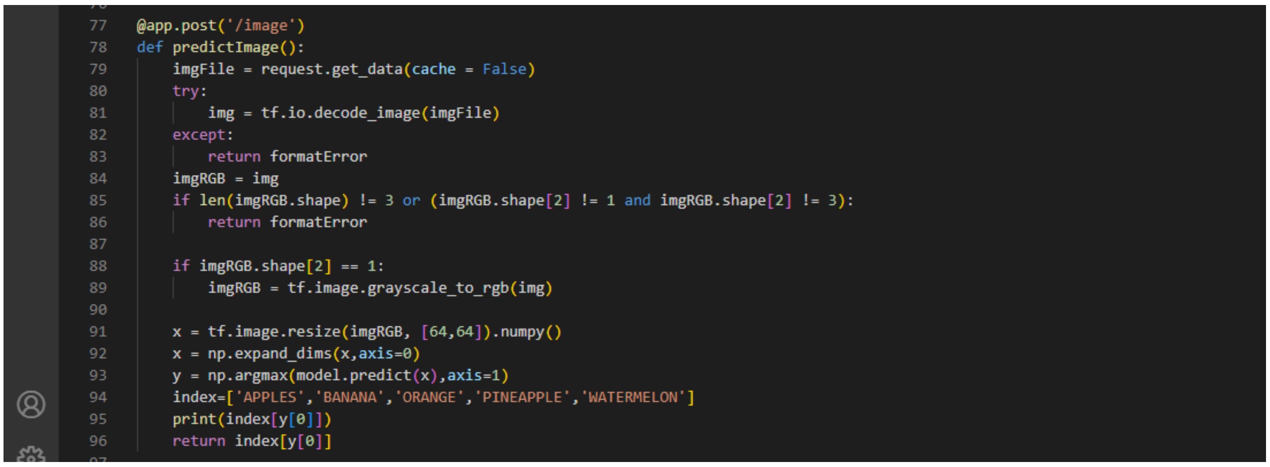
In the above example, the '/' URL is bound with the home.html function. Hence, when the home page of the webserver is opened in the browser, the HTML page is rendered. Whenever you enter the values from the HTML page the values can be retrieved using the POST Method.
Here, "home.html" is rendered when the home button is clicked on the UI



```python
@app.route('/about')
def about():
    return serveFile("about.html")

@app.route('/info')
def info():
    return serveFile("info.html")

@app.route('/home')
def home():
    return serveFile("home.html")

@app.route('/<name>')
def serveFile(name):
    if staticLoad and name not in staticAssets.keys():
        return createNoResourceResponse()
    if not staticLoad and name not in os.listdir(assetsDir):
        return createNoResourceResponse()

    opt, type = getOptAndType(name)
    if staticLoad:
        file = staticAssets[name]
    else:
        file = open(assetsDir + '/' + name, opt).read()
    res = make_response(file)
    ext = {'html': 'text/html', 'css': 'text/css', 'js': 'text/javascript', 'jpg': 'image/jpg', 'png': 'image/png', 'ico': 'image/ico'}
    res.content_type = ext[type]
    return res
```

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

It will take the image request and we will be storing that image in our local system 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.



```python
@app.post('/image')
def predictImage():
    imgFile = request.get_data(cache = False)
    try:
        img = tf.io.decode_image(imgFile)
    except:
        return formatError
    imgRGB = img
    if len(imgRGB.shape) != 3 or (imgRGB.shape[2] != 1 and imgRGB.shape[2] != 3):
        return formatError

    if imgRGB.shape[2] == 1:
        imgRGB = tf.image.grayscale_to_rgb(img)

    x = tf.image.resize(imgRGB, [64,64]).numpy()
    x = np.expand_dims(x,axis=0)
    y = np.argmax(model.predict(x),axis=1)
    index=['APPLES','BANANA','ORANGE','PINEAPPLE','WATERMELON']
    print(index[y[0]])
    return index[y[0]]
```

API Integration:

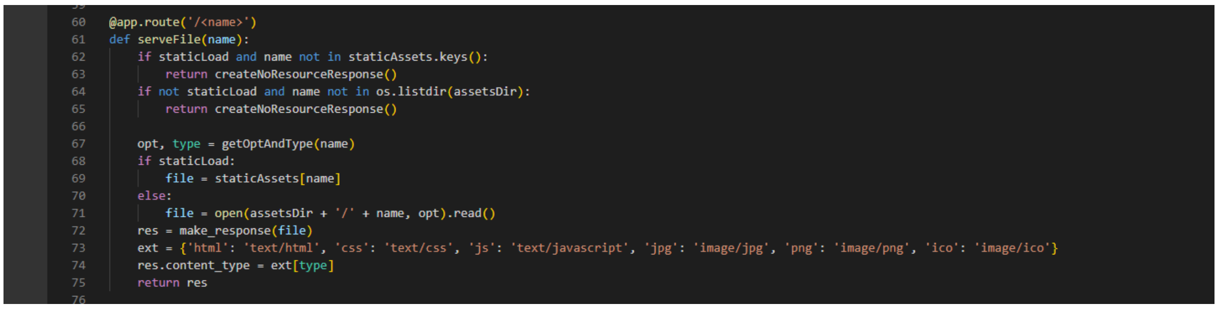
Here we will be using Rapid API

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

Reference link

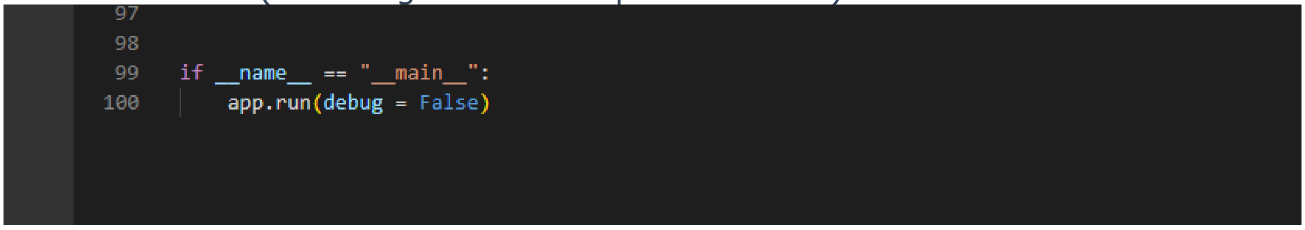
API used: Link
The link above will allow us to test the food item and will result the nutrition content present in the food item.
NOTE: When we keep hitting the API the limit of it might expire. So making a smart use of it will be an efficient way.

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



```python
@app.route('/<name>')
def serveFile(name):
    if staticLoad and name not in staticAssets.keys():
        return createNoResourceResponse()
    if not staticLoad and name not in os.listdir(assetsDir):
        return createNoResourceResponse()

    opt, type = getOptAndType(name)
    if staticLoad:
        file = staticAssets[name]
    else:
        file = open(assetsDir + '/' + name, opt).read()
    res = make_response(file)
    ext = {'html': 'text/html', 'css': 'text/css', 'js': 'text/javascript', 'jpg': 'image/jpg', 'png': 'image/png', 'ico': 'image/ico'}
    res.content_type = ext[type]
    return res
```

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



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