



PRIFYSGOL CYMRU
Y Drindod Dewi Sant
UNIVERSITY OF WALES
Trinity Saint David
SWANSEA - ABERTAWE

Applied Computing

Wales Institute of Science and Art (WISA)

Network Programmability (ACCB5029)

Lab 6: Integrate REST API in a Python Application

ICONS Used in this document:



Do research and complete the task. Need referencing/ in-text citations. Shall be included as a part of your coursework.



Record your Reflection.

 **Networking
Academy**

Task 1: (Integrate REST API in a Python Application)**Objectives**

1. Explore API Documentation (MAPQUEST)
2. Integrate MapQuest Directions API into your Application.
3. Discuss with your group and record your reflections.

Task 2: (Research and Propose Feature Enhancement)

(Note: this part is to work with your Group and submit as a part of Component 1 Assessment)

Objectives

4. Discuss with your group the possible feature enhancement.
5. Upgrade the MapQuest Directions Application with additional Features.

Task 3: (Research and Propose new application integration)**Objectives**

6. Do research similar mapping and navigation service that you could integrate with your application.
7. As a group, list similar mapping service, Compare and Contrast by implementing the similar one.

Required Resources

Host computer with DEVASC VM.

This lab will demonstrate integrating APIs in an application. You will create an application in Visual Studio Code (VS Code) that retrieves JSON data from the MapQuest Directions API, parse the data, and format it for output to the user. You will use the GET Route request from the MapQuest Directions API

Instructions

Launch the DEVASC VM

1. **Get a MapQuest API Key.** Before building the application, you need to complete the following steps to get a MapQuest API key.
 - 1.1. Go to: <https://developer.mapquest.com/>.
 - 1.2. Click Sign Up at the top of the page.
 - 1.3. Fill out the form to create a new account. For Company, enter Cisco Networking Academy Student.
 - 1.4. After clicking Sign Me Up, Click Manage Keys from the list of options on the left.
 - 1.5. Create a New Key and Click Approve Key.
 - 1.6. Expand My Application.
 - 1.7. Copy your Consumer Key to a text file for future use. This will be the key you use for the rest of this lab.

2. Build the Basic MapQuest Direction Application

In this Part, you will create a Python script to send a URL request to the MapQuest directions API. You will then test your API call. Throughout the rest of this lab, you will build your script in parts, saving the file with a new name each time. This will help with learning the pieces of the application as well as provide you a series of scripts that you can return to if you run into any problems in the current version of your application.

2.1. Create a New File in VS Code.

Note: You can use any tools you want to enter in Python commands and execute the Python code. As a Group work, I would recommend deciding your groups preference. However, document the steps while using an alternate tool.

2.2. Open VS Code.

2.2.1. Select File > Open Folder...

2.2.2. Navigate to the ~/labs/devnet-src/mapquest directory.

2.2.3. This directory should be empty. Use this directory to store each iteration of your application.

2.2.4. Open a new file and save as mapquest_parse-json_1.py.

2.3. Importing modules for the application.

To begin your script for parsing JSON data, you will need to import two modules from the Python library:

requests and urllib.parse.

- The requests module provides functions for retrieving JSON data from a URL.
- The urllib.parse module provides a variety of functions that will enable you to parse and manipulate the JSON data you receive from a request to a URL.

2.3.1. Add the following import statements at the top of your script.

```
import urllib.parse
```

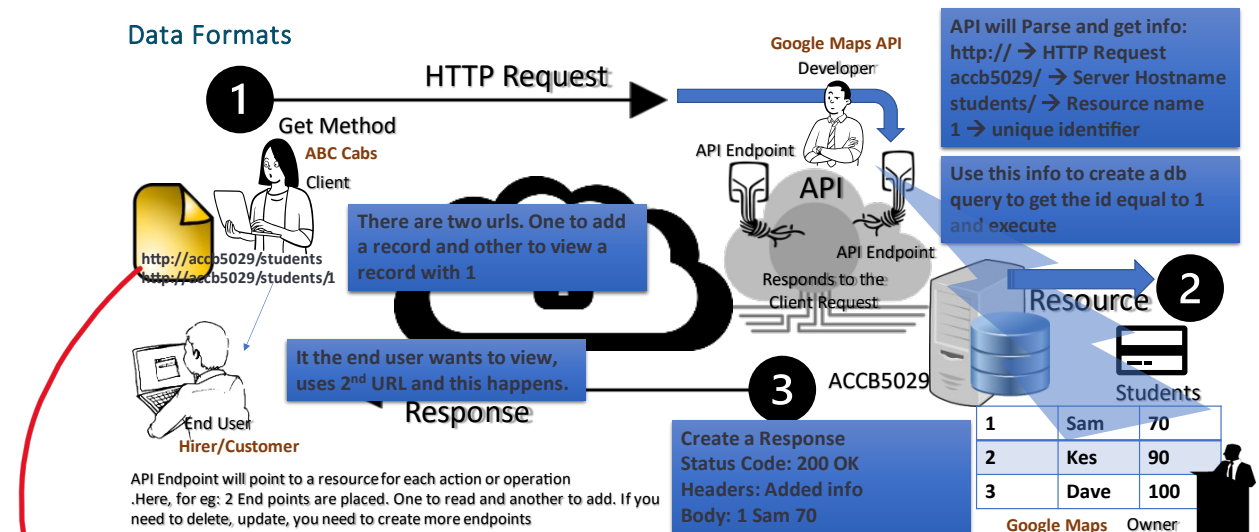
```
import requests
```

2.3.2. Open a Terminal inside VS Code. Save and run your script.

You should get no errors. You should save and run your scripts often to test code functionality.

```
devasc@labvm:~/labs/devnet-src/mapquest$ python3 mapquest_parse-  
json_1.py
```

2.4. Build the URL for the request to the MapQuest directions API.



Refer to our earlier lecture, the first step in creating your API request is to construct the URL that your application will use to make the call. Initially, the URL will be the combination of the following variables:

- main_api - the main URL that you are accessing.
- orig - the parameter to specify your point of origin.
- dest - the parameter to specify your destination.
- key - the MapQuest API key you retrieved from the developer website.

2.4.1. Create variables to build the URL that will be sent in the request. In the following code, replace your_api_key with the Consumer Key you copied from your MapQuest developer account.

```
main_api = "https://www.mapquestapi.com/directions/v2/route?"
```

```
orig = "Swansea, UK"
```

```
dest = "London, UK"
```

```
key = "your_api_key"
```

2.4.2. Combine the four variables main_api, orig, dest, and key to format the requested URL. Use the urlencode method to properly format the address value. This function builds the parameters part of the URL and converts possible special characters in the address value into acceptable characters (e.g. space into "+" and a comma into "%2C").

```
url = main_api + urllib.parse.urlencode({"key":key, "from":orig, "to":dest})
```

- 2.4.3. Create a variable to hold the reply of the requested URL and print the returned JSON data. The `json_data` variable holds a Python's Dictionary representation of the json reply of the get method of the requests module. The `requests.get` will make the API call to the MapQuest API.

```
json_data = requests.get(url).json()

print(json_data)
```

Your final code should look like this, but with a different value for the key.

```
import urllib.parse

import requests

main_api = "https://www.mapquestapi.com/directions/v2/route?"

orig = "Swansea, UK"

dest = "London, UK"

key = "fZadaFOY22VIEEmZcBFfxl5vjSXIPpZ"

url = main_api + urllib.parse.urlencode({"key":key, "from":orig, "to":dest})

json_data = requests.get(url).json()

print(json_data)
```

- 2.4.4. Test the URL request. Save and run your `mapquest_parse-json_1.py` script and verify it works. Troubleshoot your code, if necessary.

```
devasc@labvm:~/labs/devnet-src/mapquest$ python3 mapquest_parse-
json_1.py
```

2.4.5. What is your output?

```
{'route': {'hasTollRoad': False, 'hasBridge': True, 'boundingBox':
```

```
<<<<< >>>>> output omitted <<<<< >>>>>
```

```
'locationSequence': [0, 1], 'hasFerry': False}, 'info': {'statusCode': 0,
'copyright': {'imageAltText': '© 2019 MapQuest, Inc.', 'imageUrl':
'http://api.mqcdn.com/res/mqlogo.gif', 'text': '© 2019 MapQuest, Inc.'},
'messages': []}}
```



- 2.4.6. Did you notice that ***the output is a dictionary with two key/value pairs***. The value for the key route is another dictionary that includes additional dictionaries and lists.



- 2.4.7. What are Key/Value pairs? Justify your output.

- 2.4.8. Change the orig and dest variables. Rerun the script to get different results.

- 2.4.9. Print the URL and check the status of the JSON request.

- 2.5. Now that you know the JSON request is working, you can add some more functionality to the application. Save your script as mapquest_parse-json_2.py. Delete the print(json_data) statement as you no longer need to test that the request is properly formatted.
- 2.6. Create statements to do the following tasks:
- Print the constructed URL so that the user can see the exact request made by the application.
 - Parse the JSON data to obtain the statuscode value.
 - Start an if loop that checks for a successful call, which is indicated by a returned value of 0. Add a print statement to display the statuscode value and its meaning. The \n adds a blank line below the output.

```
print("URL: " + (url))
json_data = requests.get(url).json()
json_status = json_data["info"]["statuscode"]
if json_status == 0:
    print("API Status: " + str(json_status) + " = A successful route call.\n")
```

- 2.7. Test status and URL print commands.

The example here uses the following parameters.

orig = "Swansea, UK"

dest = "London, UK"

Save and run your mapquest_parse-json_2.py script and verify it works.

Troubleshoot your code, if necessary.

2.7.1. What is the output?



2.8. Add User input for starting location and destination. So far, you have used static values for the location variables. Now, make necessary changes that requires the user input these and save your script as `mapquest_parse-json_3.py`.

- Delete the current orig and dest variables.
- Rewrite the orig and dest to be within a while loop in which it requests user input for the starting location and destination. The while loop allows the user to continue to make requests for different directions. Be sure all the remaining code is correctly indented within the while loop.

```
import urllib.parse
import requests

main_api = "https://www.mapquestapi.com/directions/v2/route?"
key = "Mr10jQeogjS7dRHSL8up8CywI3MT4MU6"

while True:
    orig = input("Starting Location: ")
    dest = input("Destination: ")
```

2.8.1. Test user input functionality. Run your `mapquest_parse-json_3.py` script and verify it works. Troubleshoot your code, if necessary.

2.8.2. What is the output?

```
Traceback (most recent call last):
  File "/home/devasc/labs/devnet-src/mapquest/tEMPL.py", line 12, in <module>
    json_data = requests.get(url).json()
  File "/usr/lib/python3/dist-packages/requests/models.py", line 897, in json
    return complexjson.loads(self.text, **kwargs)
  File "/usr/lib/python3/dist-packages/simplejson/_init_.py", line 518, in loads
    return _default_decoder.decode(s)
```

Troubleshoot if any error. (Like above)

```
devasc@labvm:~/labs/devnet-src/mapquest$ cd /home/devasc/labs/devnet-src/mapquest ; /usr/bin/env /bin/python3 /home/devasc/.vscode/extensions/ms-python.python-2023.2.0/pythonFiles/lib/python/debugpy/adapters/..../debugpy/launcher 56199 -- /home/devasc/labs/devnet-src/mapquest/tEMPL.py
Starting Location: Swansea, UK
Destination: London, UK
URL: https://www.mapquestapi.com/directions/v2/route?key=Mr10jQeogjS7dRHSL8up8CywI3MT4MU6&from=Swansea%2C+UK&to=London%2C+UK
API Status: 0 = A successful route call.
Starting Location: 
```

You must get an output like this.

```
Starting Location: ^CTraceback (most recent call last):
  File "/home/devasc/labs/devnet-src/mapquest/tEMPL.py", line 8, in <module>
    orig = input("Starting Location: ")
KeyboardInterrupt
devasc@labvm:~/labs/devnet-src/mapquest$
```

Interrupt and continue.

- 2.9. Add quit functionality to the application. Instead of forcing the application to quit with a keyboard interrupt, you will add the ability for the user to enter q or quit as keywords to quit the application. Save your script as mapquest_parse-json_4.py.

Add an if statement after each location variable to check if the user enters q or quit, as shown below. To do that, use the following if statement appropriately:

```
if orig == "quit" or orig == "q":
    break
```

Do the same for Destination as well.

- 2.9.1. Test the quit functionality. Run your mapquest_parse-json_4.py script four times to test each location variable. Verify that both quit, and q will end the application. Troubleshoot your code, if necessary.

You must get an output like below.

```
devasc@labvm:~/labs/devnet-src/mapquest$ cd /home/d
ome/devasc/.vscode/extensions/ms-python.python-2023.
cher 53747 -- /home/devasc/labs/devnet-src/mapquest/
Starting Location: Swansea, UK
Destination: London, UK
URL: https://www.mapquestapi.com/directions/v2/route
ondon%2C+UK
API Status: 0 = A successful route call.

Starting Location: q
devasc@labvm:~/labs/devnet-src/mapquest$
```

- 2.10. Display the JSON data in JSONView. (The Chromium browser in the DEVASC VM includes the JSONView extension. You can use this to view a JSON object in a readable, coloured, and collapsible format.)

- 2.10.1. Run your mapquest_parse-json_4.py again and copy the code returned for the URL.

```
https://www.mapquestapi.com/directions/v2/route?key=Mr10jQeogjS7dRHSL
8up8Cywl3MT4MU6&from=Swansea%2C+UK&to=London%2C+UK
```

2.10.2. Paste the URL in the Chromium browser address field. Collapse the JSONView data by selecting the dash "-" before route, you will see that there are two root dictionaries: route and info.

```
{
  route:{
    hasTollRoad: false,
    hasBridge: true,
    <output omitted>
```

You will see that there are two root dictionaries: route and info. Notice that info has the statuscode key/value paired used in your code.

```
{
+ route: {},
- info: {
  statuscode: 0,
- copyright: {
  imageAltText: "© 2019 MapQuest, Inc.",
  imageUrl: "http://api.mqcdn.com/res/mqlogo.gif",
  text: "© 2019 MapQuest, Inc."
},
  messages: [ ]
}
}
```

3. Expand the route dictionary (click on the plus sign "+" before route) and investigate the rich data. What values do you see? Do research, discuss with your Group to implement new features to your code.



Record your understanding/ reflection so far in integrating APIs.



- 3.1. Continue to print the trip summary information. Here, Duration and distance are considered.

```
if json_status == 0:
    print("API Status: " + str(json_status) + " = A successful route call.\n")
    print("=====")
    print("Directions from " + (orig) + " to " + (dest))
    print("Trip Duration: " + (json_data["route"]["formattedTime"]))
    print("Miles: " + str(json_data["route"]["distance"]))
```

The expected output:

```
devasc@labvm:~/labs/devnet-src/mapquest$ cd /home/devasc/labs/devnet-src
/home/devasc/.vscode/extensions/ms-python.python-2023.2.0/pythonFiles/lib/p
cher 36391 -- /home/devasc/labs/devnet-src/mapquest/tEMP1.py
Starting Location: Swansea, UK
Destination: London, UK
URL: https://www.mapquestapi.com/directions/v2/route?key=Mr10jQeogjS7dRHS
ondon%2C+UK
API Status: 0 = A successful route call.

=====
Directions from Swansea, UK to London, UK
Trip Duration: 03:19:25
Miles: 186.4667
Starting Location: Swansea, UK
Destination: Cardiff, UK
URL: https://www.mapquestapi.com/directions/v2/route?key=Mr10jQeogjS7dRHS
ardiff%2C+UK
API Status: 0 = A successful route call.

=====
Directions from Swansea, UK to Cardiff, UK
Trip Duration: 00:48:58
Miles: 41.5579
Starting Location: █
```

Task 2: (Research and Propose Feature Enhancement)

(Note: this part is to work with your Group and submit as a part of Component 1 Assessment)

4. Discuss with your group the possible feature enhancement using MapQuest.

5. Upgrade the MapQuest Directions Application with additional Features.

➤ Note: Questions to be addressed

- + What features (as a Group) are you planning to enhance?
- + Does MapQuest support all the planned features?
- + If not, what other API would you use to integrate?
- + Note that, you must write clean code, use version control system to collaborate.

Task 3: (Research and Propose new application integration)

6. Do research on similar mapping and navigation service that you could integrate with your application.

6.1. List similar mapping and navigation services

6.2. Compare and contrast their features. (Refer to their documentation)

7. As a group, implement one similar mapping service in your application with adequate evidence. Note that, you must write clean code, use version control system to collaborate.

End of document.