## Module: EGT307 AI Application Development

Practical 1: RESTful API using Python Flask

In today’s lesson, we will learn how to use Python code to set up RESTful API using Python. A few terms to understand before we start today’s hands-on.

**Application Programming Interface (APIs)**

* Allows applications to communicate with each other
* Allow people to send and retrieve data using code

**RESTful APIs**

* API built according to Representational State Transfer (REST) pattern
* Enables client and server to be implemented independently of one another
* Data exposed to the internet through a URL, accessed by sending an HTTP request

**HTTP Methods**

A screenshot of a phone

Description automatically generated

Prerequisites –

Before we start, we must ensure that a couple of prerequisites are completed. To follow along and run this tutorial, you will need to:

1. Have Python installed
2. Have an IDE to edit your code

Activity1: Using Python to Consume APIs

The Python requests API enables developers to write code to interact with REST APIs. It allows them to send HTTP requests using Python without having to worry about the complexities.

The most straightforward way to install the requests module is with pip:



Activity2: Making a GET request

* GET is a read-only operation.
* Only suitable for accessing existing resources but should not be used to modify them.

Create a python file named “Prac1GetRequests.py” and paste the following code in the file.

import requests

# The API endpoint

url = "https://jsonplaceholder.typicode.com/users"

# A GET request to the API

response = requests.get(url)

# Print the response

response\_json = response.json()

print(response\_json)

In the code above, we carried out the following:

1. Defined the API endpoint to retrieve data from
2. Used the  method to retrieve the data from the defined endpoint.
3. We used the  method to store the response data in a dictionary object.
4. The last step is to print the JSON response data.

We can also check the status code returned from the API like this:

# Print status code from original response (not JSON)

print(response.status\_code)

Run the python code using command “*python Prac1GetRequests.py*”.

You should get result similar to the following output.

A screen shot of a computer screen

Description automatically generated

Activity3: Making a POST request

POST requests allow you to create new data.

Create a python file named “Prac1PostRequests.py” and paste the following code in the file.

import requests

# Define new data to create

new\_data = {

"id": 11,

"name": "Clementina DuBuque",

"username": "Moriah.Stanton",

"email": "Rey.Padberg@karina.biz",

"address": {

"street": "Kattie Turnpike",

"suite": "Suite 198",

"city": "Lebsackbury",

"zipcode": "31428-2261",

"geo": {

"lat": "-38.2386",

"lng": "57.2232"

}

},

"phone": "024-648-3804",

"website": "ambrose.net",

"company": {

"name": "Hoeger LLC",

"catchPhrase": "Centralized empowering task-force",

"bs": "target end-to-end models"

}

}

# The API endpoint to communicate with

url\_post = "https://jsonplaceholder.typicode.com/users"

# A POST request to tthe API

post\_response = requests.post(url\_post, json=new\_data)

# Print the response

post\_response\_json = post\_response.json()

print(post\_response\_json)

In the code above, we performed the following:

1. Created a new resource we wanted to add to the JSONPlaceholder API
2. Defined the endpoint to POST the new data
3. Sent a POST request using the requests.post() method.
4. Used the response.json() method to store the response data in a dictionary object
5. The last step is to print the JSON response data and status code.

Again, the output should be similar to the following.

A computer screen with white text

Description automatically generated

Now you know how to use Python to send requests to interact with RESTful APIs.

Next we will learn how to use Python to set up APIs using Flask.

**Python Flask**

Python Flask is a popular framework for building web applications and APIs in Python. It provides developers with a quick and easy way to create RESTful APIs that can be used by other software applications.

Flask is lightweight and requires minimal setup, making it a great choice for building small to medium-sized APIs. This makes Flask an ideal choice for developers looking to build robust and scalable APIs in Python.

Activity4: Write the Code to set up APIs

First, we will use pip to install Flask in the project directory. To do this, run the command below.



Create a python file named “Prac1api.py”. In our first line of code, we will import the modules for json, Flask, jsonify, and request.

import json

from flask import Flask, jsonify, request

Next, we will create a new Flask application by adding the following code just below our import statements.

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

Next, to give our API a little bit of data to work with, we will define an array of employee objects with an ID and name.

employees = [ { 'id': 1, 'name': 'Ashley' }, { 'id': 2, 'name': 'Kate' }, { 'id': 3, 'name': 'Joe' }]

nextEmployeeId = 4

To define our API endpoint, we will now add code to define a route for GET requests to the ‘/employees’ endpoint. This will return all employees (from our employees array defined above) in JSON format.

@app.route('/employees', methods=['GET'])

def get\_employees():

return jsonify(employees)

On top of our GET method, we will also define a route for POST, PUT, and DELETE methods as well. These functions can be used to create a new employee and update or delete the employee based on their given ID.

def get\_employee(id):

return next((e for e in employees if e['id'] == id), None)

def employee\_is\_valid(employee):

for key in employee.keys():

if key != 'name':

return False

return True

@app.route('/employees', methods=['POST'])

def create\_employee():

global nextEmployeeId

employee = json.loads(request.data)

if not employee\_is\_valid(employee):

return jsonify({ 'error': 'Invalid employee properties.' }), 400

employee['id'] = nextEmployeeId

nextEmployeeId += 1

employees.append(employee)

return jsonify(employees), 201

@app.route('/employees/<int:id>', methods=['PUT'])

def update\_employee(id: int):

employee = get\_employee(id)

if employee is None:

return jsonify({ 'error': 'Employee does not exist.' }), 404

updated\_employee = json.loads(request.data)

if not employee\_is\_valid(updated\_employee):

return jsonify({ 'error': 'Invalid employee properties.' }), 400

employee.update(updated\_employee)

return jsonify(employee)

@app.route('/employees/<int:id>', methods=['DELETE'])

def delete\_employee(id: int):

global employees

employee = get\_employee(id)

if employee is None:

return jsonify({ 'error': 'Employee does not exist.' }), 404

employees = [e for e in employees if e['id'] != id]

return jsonify(employee), 200

Lastly, we will add a line of code to run our Flask app. As you can see, we call the run method and get the Flask app running on port 5000.

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

app.run(port=5000)

With our code written and saved, we can start the app up by running *python Prac1api.py*

A screen shot of a computer error

Description automatically generated

The above output shows that you have a running API at http://127.0.0.1:5000. You need to keep it running while doing the next activity.

Activity 5: Send HTTP request to your APIs

Now, it’s your turn to write your own Python code to talk to the APIs you have just set up. You can refer to Activity 2 and 3:

Hint: The url should be http://127.0.0.1:5000/employees

1. Create a new python file and import necessary dependency
2. Get the list of employees (GET request)
3. Create a new employee with the following details and add into the list (POST request)

{ 'name': 'Bob' }

Hint for following questions:  
The url should be http://127.0.0.1:5000/employees/<id>  
Replace <id> with the employee id

1. Update the name of employee (id: 1) to Tom (PUT request)
2. Delete employee (id:3) (DELETE request)