**Table of Contents**

[**Introduction 1**](#_gjdgxs)

[Lab Overview 1](#_1fob9te)

[Lab Objectives 1](#_2et92p0)

[Lab Prerequisites 1](#_tyjcwt)

[**Instructions 2**](#_3dy6vkm)

[Lab Tutorial 2](#_4d34og8)

[**Additional Resources 10**](#_35nkun2)

# Introduction

Testing RESTful API Services Using Request and Postman

## Lab Overview

When building an API, it is essential to be able to test its functionality and ensure that it is working properly. There are a number of methods available to you to test your API. In this lab we will explore two different ways to test your API. You will learn how to use the Requests python library and the Postman application to test the functionality of a basic API.

## Lab Objectives

* Become proficient at using the Requests library to test the functionality of RESTful API services.
* Gain proficiency in using Postman to test the functionality of RESTful API services.
* Learn to write test suites in Python using the Requests library.
* Use both the Requests library and Postman for quality assurance purposes to ensure that all REST API services are functioning correctly.

## Lab Prerequisites

* Basic knowledge of the Python programming language.
* Understanding of what a REST API is and how it functions.
* Familiarity with GET, POST, PUT, and DELETE methods.
* Understanding of API endpoints, and how they are used in testing.
* The ability to use pip to install the required Python libraries.
* Installation of the Flask and Requests libraries
* Installation of the Postman application on your machine.

# Instructions

## Lab Tutorial

**Assignment Setup:**

1. Download and unzip the project files to your preferred IDE. This contains two python files that you will be using for this lab. First is ‘restful\_api.py’, which when run will use flask to create a basic API on your machine. The second is ‘request\_testing.py’, which we will use to write and execute tests on the API using the requests library.
2. In your terminal enter these two commands to install the flask and requests libraries.

*pip install Flask*

*pip install requests*

1. Go to <https://www.postman.com/>. Download Postman for your OS and install it.

**Step 1: Run the File**

1. Run the ‘restful\_api.py’ file. You should get output similar to this:

A screen shot of a computer

Description automatically generated

**Step 2: Verify Results**

1. To verify that the program is running correctly go to <http://127.0.0.1:5000/colleges>. Your output should look like this:

A computer screen shot of a program

Description automatically generated

**Step 3: Test POST**

1. The first method we will test is POST. This allows us to insert new data into our database. In the ‘request\_testing.py’ file insert the following code:

imports the request library

import requests

determines where the POST request will be sent  
ENDPOINT = <http://127.0.0.1:5000/colleges>

the data that will be sent to the API  
data\_to\_post = {'name': 'North Carolina Central University', 'abbreviation': 'NCCU', 'year\_founded': 1910, 'nickname': 'Eagles'}

sends the POST request and prints the response from the server  
response = requests.post(ENDPOINT, json=data\_to\_post)  
print(response)

prints the json data info that was sent back from the server  
data = response.json()  
print(data)

**Step 4: Execute the Code**

1. With the Flask application running, execute the code you typed in the previous step. If everything is working correctly you should receive this response:

<Response [200]>

{'abbreviation': 'NCCU', 'name': 'North Carolina Central University', 'nickname': 'Eagles', 'year\_founded': 1910}

The first line is the result of ‘print(response).’ The code ‘200’ tells us that the request was processed correctly. If you received a different code the request was not processed correctly. The second line is the result of ‘print(data).’ This shows us the JSON data that was sent back from the Flask server.

**Step 5: Review the Data**

1. Let’s look at our data that should now include the data sent in step 3. To do that we will use a GET request. This retrieves all of the data found at a certain endpoint. This is another way of doing what we did in step 2. Except this time it is being done using the requests library. To do this insert the following code into ‘request\_testing.py’:

Sends the GET request to the server and prints the response

response = requests.get(ENDPOINT)

print(response)

Prints the JSON data sent back from the Flask server

data = response.json()

print(data)

**Step 6: Check Response from the POST Request**

1. This time when you execute the program you should see the response from the POST request along with our new data from the GET request:

<Response [200]>

{'Colleges': [{'abbreviation': 'NCSU', 'name': 'North Carolina State University', 'nickname': 'Wolfpack', 'year Founded': 1887}, {'abbreviation': 'UNC', 'name': 'University of North Carolina Chapel Hill', 'nickname': 'Tar heels', 'year Founded': 1789}, {'abbreviation': 'DUKE', 'name': 'Duke University', 'nickname': 'Blue Devils', 'year Founded': 1838}, {'abbreviation': 'ECU', 'name': 'East Carolina University', 'nickname': 'Pirates', 'year Founded': 1907}, {'abbreviation': 'WF', 'name': 'Wake Forest University', 'nickname': 'Demon Deacons', 'year Founded': 1834}, {'abbreviation': 'NCCU', 'name': 'North Carolina Central University', 'nickname': 'Eagles', 'year\_founded': 1910}]}

Note that the data sent back now includes the data sent in our POST request! This verifies that not only has the POST request been sent successfully, but that it has been correctly added to the JSON data on our Flask server.

**Step 7: Test the PUT method**

1. Next we will test the PUT method on our API. This allows us to update information that is already in our database, instead of adding completely new information with the POST request. Like all respectable people you strongly dislike UNC. So let’s change the ‘nickname’ for UNC in our database to something more appropriate. To do this, insert the code below into your program:

Select a new endpoint to send our PUT request to

ENDPOINT = <http://127.0.0.1:5000/colleges/UNC>

Input the data we want to send

data\_to\_put = {"nickname": "Tar Holes"}

Send PUT request to API

response = requests.put(ENDPOINT, json=data\_to\_put)

Print the response from the server

print(response)

Print the JSON data that we received back from the server

data = response.content

print(data)

**Step 8: Run the Program Again**

1. Let’s run our program once again. You should see the output from our previous tests along with output from what we added in step 7. It should look something like this:

<Response [200]>

b'{"abbreviation":"UNC","name":"University of North Carolina Chapel Hill","nickname":"Tar Holes","year Founded":1789}\n'

As you can see the ‘nickname’ for UNC has been changed from ‘Tar Heels’ to ‘Tar Holes’ which is exactly the result we were looking for!

**Step 9: Test the DELETE Method**

1. Lastly, we should test the DELETE method on our API. Let’s test it out on our old friend UNC. So we can pretend that they don’t exist. To do this insert the following code into your program:

Deletes UNC from the database and prints the server response

response = requests.delete(ENDPOINT)

print(response)

Sends GET request to server and prints response

ENDPOINT = "http://127.0.0.1:5000/colleges"

response = requests.get(ENDPOINT)

print(response)

Prints the JSON data received from the GET request to verify that UNC has successfully been deleted

data = response.json()

print(data)

**Step 10: Run the Program Again**

1. Let’s run our program one more time. We should see the output from all our previous steps and now step 9. Your output should look something like this:

Response from server shows that the DELETE request has been sent successfully

<Response [200]>

Shows that get request has been sent successfully

<Response [200]>

JSON data shows that UNC has successfully been deleted from the database

{'Colleges': [{'abbreviation': 'NCSU', 'name': 'North Carolina State University', 'nickname': 'Wolfpack', 'year Founded': 1887}, {'abbreviation': 'DUKE', 'name': 'Duke University', 'nickname': 'Blue Devils', 'year Founded': 1838}, {'abbreviation': 'ECU', 'name': 'East Carolina University', 'nickname': 'Pirates', 'year Founded': 1907}, {'abbreviation': 'WF', 'name': 'Wake Forest University', 'nickname': 'Demon Deacons', 'year Founded': 1834}, {'abbreviation': 'NCCU', 'name': 'North Carolina Central University', 'nickname': 'Eagles', 'year\_founded': 1910}]}

As you can see the DELETE request worked, and UNC has been banished from our database!

**Step 11: Test API using Postman**

1. Now that we have tested our API using the Requests library, we can move on to testing our API with Postman. Postman allows you to test your API using a GUI. To get started open Postman. You should be greeted by a screen that looks something like this:

A screenshot of a computer

Description automatically generated

**Step 12: Using a POST Request**

1. Let’s start with a POST request. In the address bar type <http://127.0.0.1:5000/colleges>. Then select ‘POST’ from the list of methods to the left of your http address. Now select the tab that says ‘Body’ to input what we would like to send in our post request. In the ‘Body’ tab type:

{"name": "North Carolina Central University", "abbreviation":"NCCU", "year\_founded": 1910, "nickname": "Eagles"}

Above the data you just typed, make sure to select ‘raw’ and ‘JSON’ so that our data is sent correctly. This should be done for all subsequent steps where JSON data is being sent to the API. Now that this is done press the ‘Send’ button in the upper right corner. Your screen should look like this:

A screenshot of a computer

Description automatically generated

Notice that you received JSON data back from the server. This shows that the request was sent successfully

**Step 13: Submitting a PUT Request**

1. Next we will use Postman to submit a PUT request to the API. Press the ‘New’ button in the upper left corner to make a new request. In the address bar type <http://127.0.0.1:5000/colleges/UNC>. Then to the left select ‘PUT’ from the dropdown list of methods. Now select the ‘Body’ tab and type the following:

{"nickname": "Tar Holes"}

Click the ‘Send’ button to send your request. Your screen should look like this:

A screenshot of a computer

Description automatically generated

In the data you received back you should see that ‘nickname’ has successfully been changed.

**Step 14: Test the DELETE Method**

1. Now we will test the DELETE method. Press the ‘New’ button in the upper left corner to make a new request. Type <http://127.0.0.1:5000/colleges/UNC> into the address bar and select the ‘DELETE’ method from the dropdown list. That is all that we need to do for this method, so click the ‘Send’ button and you should see a screen like this one:

A screenshot of a computer

Description automatically generated

The data that has been returned indicates what has been deleted.

**Step 15: Test the GET Method**

1. Finally let’s test the GET method using Postman. Press the ‘New’ button in the upper left corner to make a new request. In the address bar type <http://127.0.0.1:5000/colleges>. Select ‘GET’ from the dropdown list and press the ‘Send’ button. Your screen should look like this:

A screenshot of a computer

Description automatically generated

The data you received back should include every school in our database except for the one we just deleted, UNC.

# Additional Resources

API:

<https://docs.github.com/en/rest/guides/getting-started-with-the-rest-api?apiVersion=2022-11-28>

Requests:

<https://docs.python-requests.org/en/latest/index.html>

Postman:

<https://www.postman.com/api-platform/api-documentation/>