Rest API | Complete Guide on Rest API with Python and Flask

ADVANCED MACHINE LEARNING PYTHON

Introduction

Welcome to the complete guide on creating a REST API using Flask (Flask-Restful). In one of our previous articles, we learned the basics of web development using Flask and how to set it up. Flask is a popular micro framework that is used for building web applications. I have made one complete end-to-end machine learning project with Flask applications. If you are a beginner to flask, then you can access the previous article from here.

People generally have many doubts about REST API, how can we create it for a different use case, or perform some task like validation, converting code, etc. In this tutorial, we will practically learn to create Flask API and make working our REST APIs more powerful. Before starting this, I assume that you are familiar with the basic Python programming language and Flask framework.

Learning Objectives

- Understand the basics of REST architecture and how to create web services using it.
- Learn to use decorators in REST APIs.
- Learn to secure the APIs using authentication
- Learn to enable tracking and writing unit tests for the REST API

This article was published as a part of the Data Science Blogathon.

Table of contents

- What Is REST API?
  - Why REST?
- What is Flask Framework?
  - Why Flask Framework?
- Create Your First Rest API
- What Is Flask-Restful?
  - Why Flask Flask-Restful?
- Understanding HTTP Requests Through Flask REST API
- How to Use Decorators in Flask REST API?
- How to Make Flask API More Secure With Basic Authentication
- How to Enable Tracking on Flask API?
- How to Write a Unit Test Code for Your REST API
- Frequently Asked Questions

What Is REST API?

REST API stands for Restful API, which allows the integration of applications or interaction with RESTful web services. It is now growing as the most common method for connecting components in a microservice architecture. APIs will enable you to get or send data to a website and perform some actions like crud operations to get your task done over a web service. Each website uses different types of API, like stock market trading websites integrating with Sensex or Nifty to get a current price and ups-down. Ticket booking apps use a desired single portal API to keep updated data at a familiar interface.

Why REST?

REST (Representational State Transfer) architecture is preferred for building web services and APIs because of the following reasons-

- It allows the integration of applications or interaction with RESTful web services, which has become the most common method for connecting components in a microservice architecture.
- APIs built with REST principles enable you to get or send data to a website and perform CRUD (Create, Read, Update, Delete) operations over a web service.
- REST APIs are stateless, meaning they don't maintain any client context on the server-side, making them more scalable and easier to manage.
- REST APIs use standard HTTP methods (GET, POST, PUT, DELETE) to perform operations on resources, which makes them simple and easy to understand.

What is Flask Framework?

Why Flask Framework?

The following advantages of using the Flask framework for building web applications and APIs:

- Flask is a micro web framework, which means it is lightweight and has a small codebase, making it easy to learn and use.
- Flask is flexible and highly extensible, allowing developers to add or remove functionalities as per their requirements.
- Flask has a simple and clean codebase, which makes it easier to understand and maintain, especially for beginners.
- Flask provides a built-in development server and a debugger, which makes development and testing easier.
- Flask has a thriving community and a vast ecosystem of extensions and libraries, which can be used to add functionality to your applications.

Create Your First Rest API

Let's get over to the code editor. You can use any Python IDE that you are comfortable with. We are creating the Hello world API, which says that if you hit a get request on the API endpoint, you will get a JSON(JavaScript Object Notation) response because it's universal that API always gives a JSON-type

response. I hope you have the Flask installed; otherwise, use the PIP command and install it using the below code. Create a new py file and type the below code:

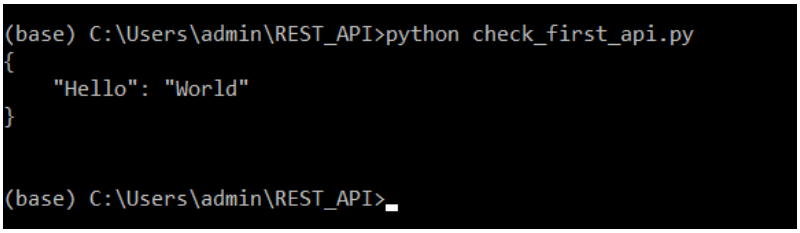
```
pip install flask pip install flask-restful from flask import Flask from flask_restful import Resource, Api app = Flask(__name__) api = Api(app) class Helloworld(Resource): def __init__(self): pass def get(self): return { "Hello": "World" } api.add_resource(Helloworld, '/') if __name__ == '__main__': app.run(debug=True)
```

What Is Flask-Restful?

Flask restful defines the resource class, which contains methods for each HTTP method. The Flask-Restful method name should be the same as its corresponding HTTP method and written in lowercase. You can observe this in the above code. However, Flask-Restful methods do not have a route decorator, so they are based on resource routes. Whatever class we define, we define the route to it using add resource method and on which route we have to call it.

Explanation ~ In the above code, we first load required parent classes, then initialize our app and API. After that, we create a course, and we make a GET request that states if anyone hits on this class, then he will get Hello world as the response in JSON format. To switch on a particular URL, we use the add resource method and route it to the default slash. To run this file, you can use the POSTMAN tool, an API maintenance tool, to create, test, and manage APIs. You can also make use request module to try this API using the below code. First, run the above file, which will give you the localhost URL, and in another command prompt, run the below code file.

```
import requests url = "http://127.0.0.1:5000/" response = requests.get(url=url) print(response.text)
```



Why Flask Flask-Restful?

Flask-Restful, an extension for Flask that simplifies the process of building RESTful APIs. The advantages of using Flask-Restful are:

- Flask-Restful provides a Resource class that allows you to define HTTP methods (GET, POST, PUT, DELETE) as class methods, making it easier to organize and manage your API endpoints.
- Flask-Restful separates the routes and resource methods, making it easier to maintain and scale your API.
- Flask-Restful handles automatic request parsing and response formatting, making it easier to work with various data formats, such as JSON, XML, and others.
- Flask-Restful provides built-in support for input validation, error handling, and rate limiting, which are essential features for building robust and secure APIs.
- Flask-Restful integrates well with other Flask extensions, allowing you to create powerful and feature-rich APIs.

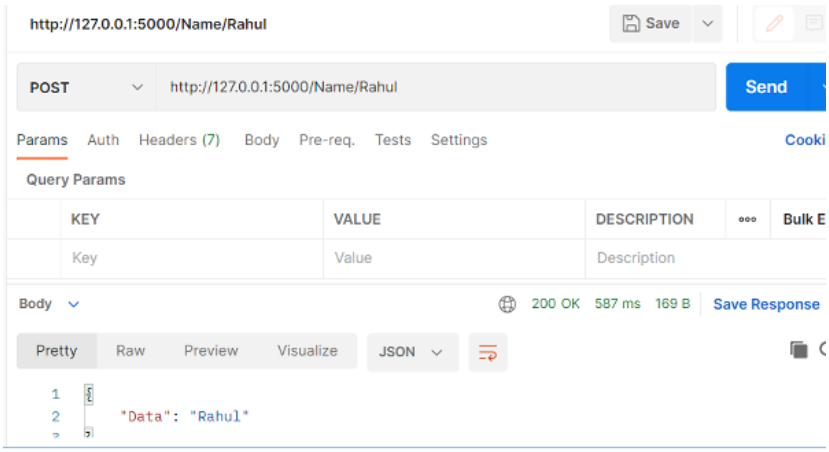
Understanding HTTP Requests Through Flask REST API

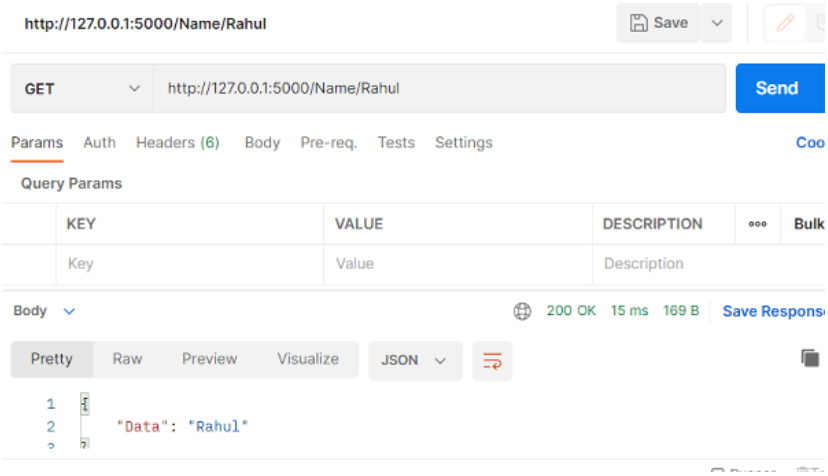
Now we are familiar with REST API. Under this heading, we will explore different HTTP methods using REST API, where we will define one list which will store all the data obtained from the server in the form of a dictionary(JSON object). This is important because we have different APIs in projects to get data post data but the data somewhere else.

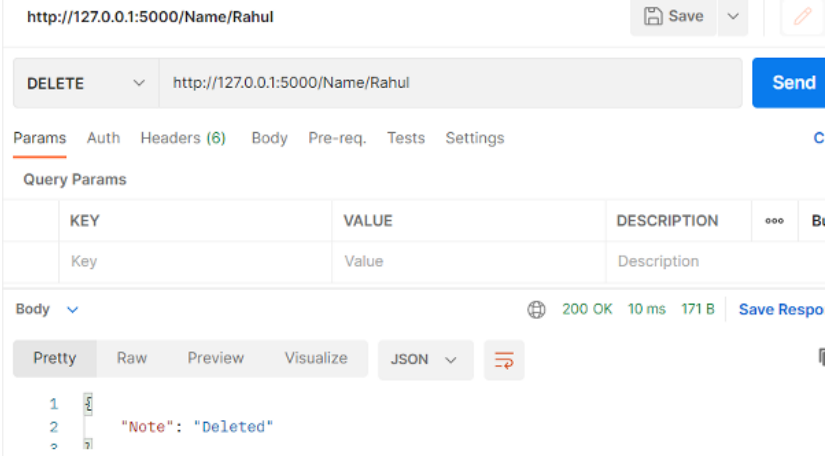
Here we are creating an API where we will create 3 HTTP methods named GET, POST, and DELETE; where we will create a customized URL and say that when you request the POST method, then it will take Name as input, and on hitting GET method, will give that Name back and In delete, we will delete that Name if it is present and again accessing that it will give us NULL. Create a file and write the below code.

```
from flask import Flask from flask_restful import Resource, Api app = Flask(__name__) api = Api(app) data = [] class People(Resource): def get(self): for x in data: if x['Data'] == name: return x return {'Data': None} def post(self, name): temp = {'Data': name} data.append(temp) return temp def delete(self): for ind, x in enumerate(data): if x['Data'] == name: temp = data.pop(ind) return {'Note': 'Deleted'} api.add_resource(People, '/Name/') if __name__ == '__main__': app.run(debug=True)
```

Open the POSTMAN API tool and hit on each HTTP method request. First, when we use post request using Name, it gives us a name. In getting the request, we get the name back. It is deleted on deleting, and when you try to get it back again, it will give you NULL. Observe the results below.





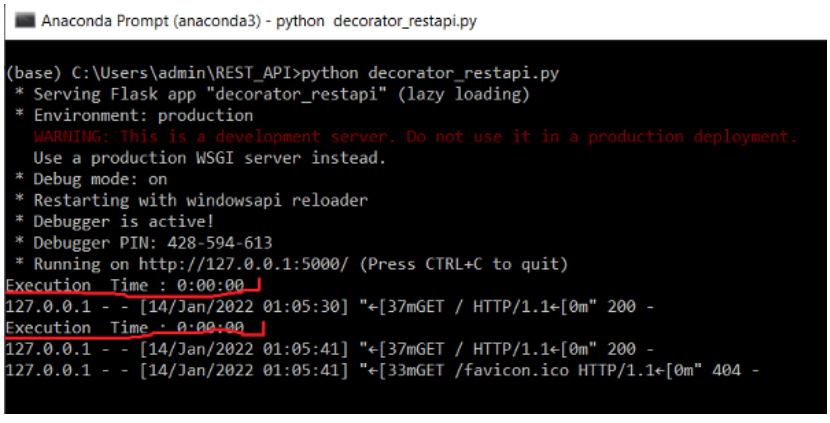


How to Use Decorators in Flask REST API?

We use decorators with APIs to monitor IP addresses, cookies, etc. So under this heading, we will learn how to leverage the flask API with decorators. A decorator is a function that takes another function as an argument and returns another function. You can also understand it as a function that provides some additional functionalities to the existing function without changing or modifying the current function.

Here we create a new file, and I will show you by creating two decorators. So in the first one, we are making the external time function which returns the code execution time. We import the wrap decorator applied to the wrapper function from the **functools** module(standard module for higher-order python functions). It updates the wrapped function by copying all its arguments.

```
from flask import Flask from flask_restful import Resource, Api import datetime from flask import request from functools import wraps app = Flask(__name__) api = Api(app) def time(function=None): @wraps(function) def wrapper(*args, **kwargs): s = datetime.datetime.now() _ = function(*args, **kwargs) e = datetime.datetime.now() print("Execution Time : {} ".format(e-s)) return _ return wrapper class HelloWorld(Resource): @monitor def get(self): return {'hello': 'world'} api.add_resource(HelloWorld, '/') if __name__ == '__main__': app.run(debug=True)
```



We create the second decorator for monitoring cookies and IP addresses, so make the below function. Instead of adding a time decorator to the hello world function, add a monitor decorator and run the code.

```
def monitor(function=None): @wraps(function) def wrapper(*args, **kwargs): _ = function(*args, **kwargs) print("Ip Address : {} ".format(request.remote_user)) print("Cookies : {} ".format(request.cookies)) print(request.user_agent) return _ return wrapper
```

How to Make Flask API More Secure With Basic Authentication

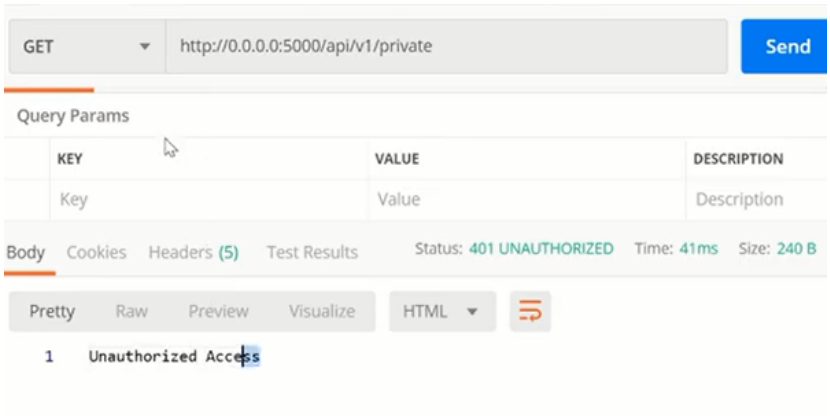
When we design the API, we should also take care of security because many people will access it. What if you want only authorized people to access the API because it may contain some confidential data between some parties so that we can do that? Using Flask basic authentication. You need to install this flask module using the following command.

```
pip install flask-httpauth
```

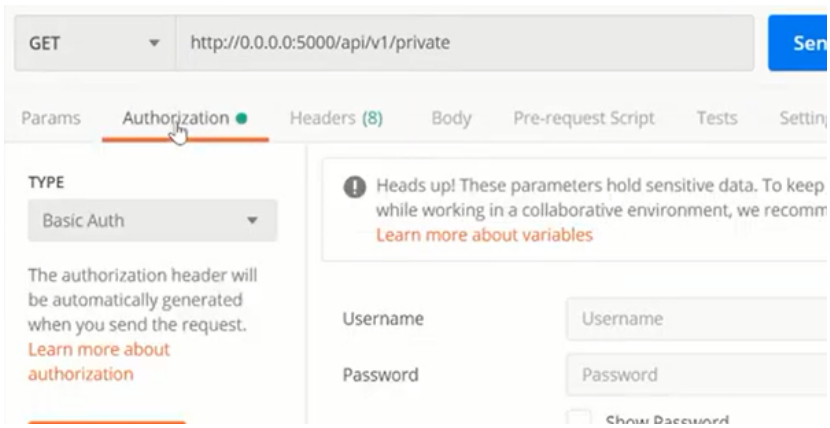
We are building an API and defining the User data dictionary, which contains a username and password. When you work in real-time use cases, you accept the username and password through a configuration file or from a database. First, we create a primary function to match the username and password and a GET method that says that anyone who hits on this API, so without login, we cannot access the data.

```
from flask import Flask from flask_restful import Resource, Api from flask_httpauth import HTTPBasicAuth app = Flask(__name__) api = Api(app, prefix="/api/v1") auth = HTTPBasicAuth() USER_DATA = { "admin": "SuperSecretPwd" } #route to verify the password @auth.verify_password def verify(username, password): if not(username and password): return False return USER_DATA.get(username) == password class PrivateResource(Resource): @auth.login_required def get(self): return {"How's the Josh": "High"} api.add_resource(PrivateResource, '/private') if __name__ == '__main__': app.run(debug=True)
```

When we run the above file using POSTMAN, we try to get the data without login in to give you unauthorized access.



Now go to authorization and click on Basic Authorization. Enter the username and password you have used, and then hit GET request to get the desired result.



This is how you can secure your Flask API. To learn more about Flask Basic Authorization, you can visit this blog.

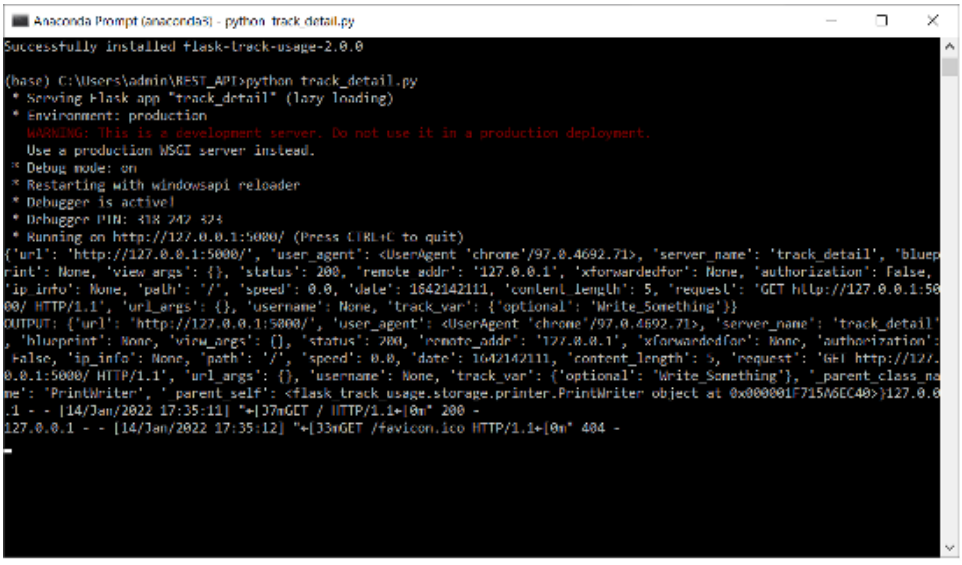
How to Enable Tracking on Flask API?

We have learned how we can secure our API from unauthorized login, but what if we also want to know the location(latitude and longitude points), IP address, and server name, like details of a person who is accessing our API, so we can configure the basic flask Tracking application with our REST API. First, install the flask tracking package using PIP Command.

```
pip install flask-track-usage
```

```
from flask import Flask, g app = Flask(__name__) app.config['TRACK_USAGE_USE_FREEGEOIP'] = False app.config['TRACK_USAGE_INCLUDE_OR_EXCLUDE_VIEWS'] = 'include' from flask_track_usage import TrackUsage from flask_track_usage.storage.printer import PrintWriter from flask_track_usage.storage.output import OutputWriter t = TrackUsage(app, [ PrintWriter(), OutputWriter(transform=lambda s: "OUTPUT: " + str(s)) ]) @t.include @app.route('/') def index(): g.track_var["optional"] = "Write_Something" return "Hello" #Run the application if __name__ == "__main__": app.run(debug=True)
```

Explanation: We will create a tracking application by importing Track Usage, Input writer, and output writer from the package installed. We pass our flask app to the Track package and use Output writer, and we use the lambda function, a single line function, to write the output in string format. After this, we create a basic routing on slash and include our track application as a decorator. G stands for global, which says data is global within the context. Hence we have created a basic API which, on the browser return, hello, but on the backend, you will get all the person's information. On the command prompt, you can observe the following output.



You can use JSON Formatter to see the output in a well-formatted way.

How to Write a Unit Test Code for Your REST API

Now you have created an excellent REST API for your case. Still, your manager will ask you to write a Unit-test code for REST API because it is essential to identify the common bugs in your API and help secure production. So I hope that you have a simple API created for you. If not, make a new file named run and develop the below simple API.

```
from flask import Flask from flask_restful import Resource, Api import json app = Flask(__name__) api = Api(app) class Helloworld(Resource): def __init__(self): pass def get(self): return json.dumps({"Message": "Fine"}) api.add_resource(Helloworld, '/') if __name__ == '__main__': app.run(debug=True)
```

Now create another file name test where we will be writing the code for unit testing our API. So most commonly, you are always supposed to perform the below three basic unit tests.

- Check that the response code is 200
- Check the content written from API is application JSON Formatted
- Check that all the keys we are accessing are present in API data processing.

```
from run import app import unittest class FlaskTest(unittest.TestCase): #Check for response 200 def test_inde(self): tester = app.test_client(self) #tester object response = tester.get("/") statuscode = response.status_code self.assertEqual(statuscode, 200) #check if the content return is application JSON def test_index_content(self): tester = app.test_client(self) response = tester.get("/") self.assertEqual(response.content_type, "application/json") #check the Data returned def test_index_data(self): tester = app.test_client(self) response = tester.get("/") self.assertTrue(b'Message' in response.data) if __name__ == '__main__': unittest.main()
```

So to elaborate, if you have done web scraping, then a 200 response code means your request to a particular URL is successfully made, which returns a response. The second is a type of data that the admin is provided that should be scrapped appropriately when someone requests it, and the third is all the data that is present that should go without any modification.

Going further, you can learn about fastAPI too, which is a fast, modern, and high-speed framework for building APIs. Also, you can develop the skills to create a fully functional REST API with Flask with authentication and authorization, and testing in a python 3 virtual environment, including database integration like SQL, and upload your project to GitHub.

Conclusion

We have learned to create Flask REST API from scratch and its maintenance easily and securely and learned Flask-Restful. REST APIs are of a different kind and are used for other purposes. You can also use routing with APIs, like creating a function in a separate file and using it as a decorator with a main Application use case. So this was all for this article. I hope it was easy to catch up with each heading, and if you have any queries, feel free to post them in the comment section below or connect with me.

Key Takeaways

- Flask is a micro web framework written in Python that is well-suited for building REST APIs due to its flexibility and simplicity.
- REST APIs are a way to access web services using a set of operations.
- We can make Flask API more secure with basic authentication using Flask basic authentication.

Frequently Asked Questions

Q1. What is REST API?

A. REST stands for Representational State Transfer. It is an architectural style for building web services that use HTTP protocol to create web APIs. RESTful APIs are stateless and allow clients to access and manipulate web resources using a standard set of operations.

Q2. How to install Flask?

A. Flask can be installed using pip, the Python package manager. Run the following command in your terminal: pip install flask. This will download and install the latest version of Flask.

Q3. How to handle incoming data in Flask?

A. Incoming data can be accessed through the request object in Flask. To access data in the request body, we can use the request.get_json() method, which parses the request body as JSON and returns a Python dictionary.

Article Url - https://www.analyticsvidhya.com/blog/2022/01/rest-api-with-python-and-flask/

Raghav Agrawal

I am a final year undergraduate who loves to learn and write about technology. I am a passionate learner, and a data science enthusiast. I am learning and working in data science field from past 2 years, and aspire to grow as Big data architect.