Your First Progressive Web App

Summary	In this codelab, we will learn how to create Flask based Web apps in Python.
URL	https://flask.palletsprojects.com/en/1.1.x/
Category	Web
Status	Published
Author	Nikhil Kohli

Flask Introduction

What is Flask?

"Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions"

It is a lightweight Web Server Gateway Interface WSGI web application framework that was created to make getting started easy and making it easy for new beginners.
Flask is based on the Werkzeug WSGI toolkit and Jinja2 template engine.

WSGI

Web Server Gateway Interface (WSGI) has been adopted as a standard for Python web application development. WSGI is a specification for a universal interface between the web server and the web applications.

Werkzeug

It is a WSGI toolkit, which implements requests, response objects, and other utility functions. This enables building a web framework on top of it. The Flask framework uses Werkzeug as one of its bases.

Jinja2

Jinja2 is a popular templating engine for Python. A web templating system combines a template with a certain data source to render dynamic web pages.

Virtual Environment & Installation

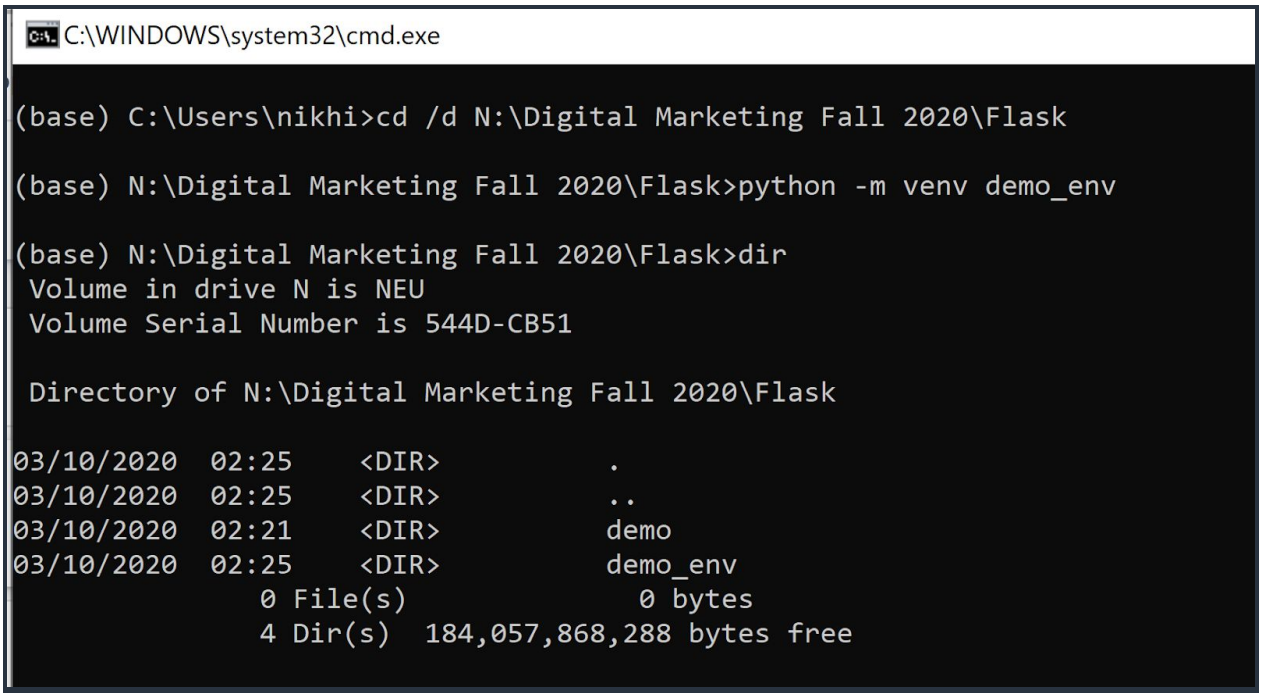
- Creating a Virtual Environment

  Mac: [Installation — Flask Documentation (0.12.x)]

  Windows:

```
python -m venv demo_env
```

```
dir
```



- Activate Virtual Environment

Windows:

```
demo_env\Scripts\activate.bat
```

Mac:

```
Source demo_env/bin/activate
```

```
(base) N:\Digital Marketing Fall 2020\Flask>demo_env\Scripts\activate.bat

(demo_env) (base) N:\Digital Marketing Fall 2020\Flask>
```

- Install Flask

```
pip install Flask
```

```
(demo_env) (base) N:\Digital Marketing Fall 2020\Flask>pip install Flask
Collecting Flask
  Using cached https://files.pythonhosted.org/packages/f2/28/2a03252dfb9ebf377f40fba6a7841b47083260bf8bd8e737b0c6952df83
f/Flask-1.1.2-py2.py3-none-any.whl
Collecting Jinja2>=2.10.1 (from Flask)
  Using cached https://files.pythonhosted.org/packages/30/9e/f663a2aa66a09d838042ae1a2c5659828bb9b41ea3a6efa20a20fd92b12
1/Jinja2-2.11.2-py2.py3-none-any.whl
Collecting click>=5.1 (from Flask)
  Using cached https://files.pythonhosted.org/packages/d2/3d/fa76db83bf75c4f8d338c2fd15c8d33fdd7ad23a9b5e57eb6c5de26b430
e/click-7.1.2-py2.py3-none-any.whl
Collecting Werkzeug>=0.15 (from Flask)
  Using cached https://files.pythonhosted.org/packages/cc/94/5f7079a0e00bd6863ef8f1da638721e9da21e5bacee597595b318f71d62
e/Werkzeug-1.0.1-py2.py3-none-any.whl
Collecting itsdangerous>=0.24 (from Flask)
  Using cached https://files.pythonhosted.org/packages/76/ae/44b03b253d6fade317f32c24d100b3b35c2239807046a4c953c7b89fa49
e/itsdangerous-1.1.0-py2.py3-none-any.whl
Collecting MarkupSafe>=0.23 (from Jinja2>=2.10.1->Flask)
  Using cached https://files.pythonhosted.org/packages/b9/82/833c7714951bff8f502ed054e6fbd8bd00e083d1fd96de6a46905cf2337
8/MarkupSafe-1.1.1-cp36-cp36m-win_amd64.whl
Installing collected packages: MarkupSafe, Jinja2, click, Werkzeug, itsdangerous, Flask
Successfully installed Flask-1.1.2 Jinja2-2.11.2 MarkupSafe-1.1.1 Werkzeug-1.0.1 click-7.1.2 itsdangerous-1.1.0
```

- Install Postman

[Download Postman | Try Postman for Free]

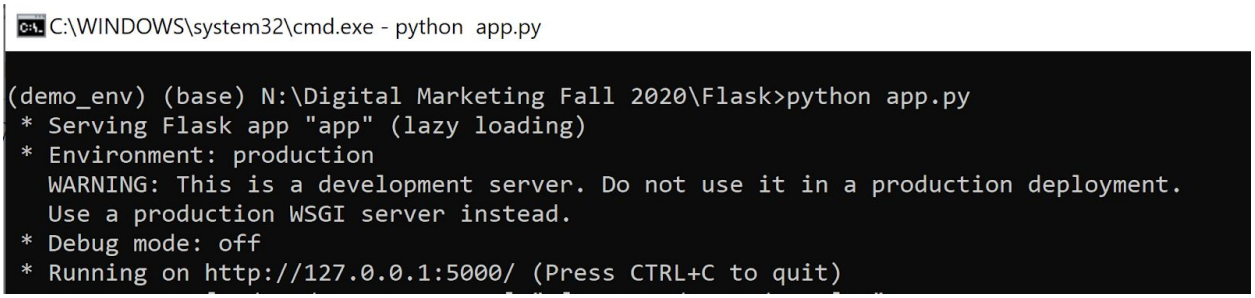
Running a Basic Example Application

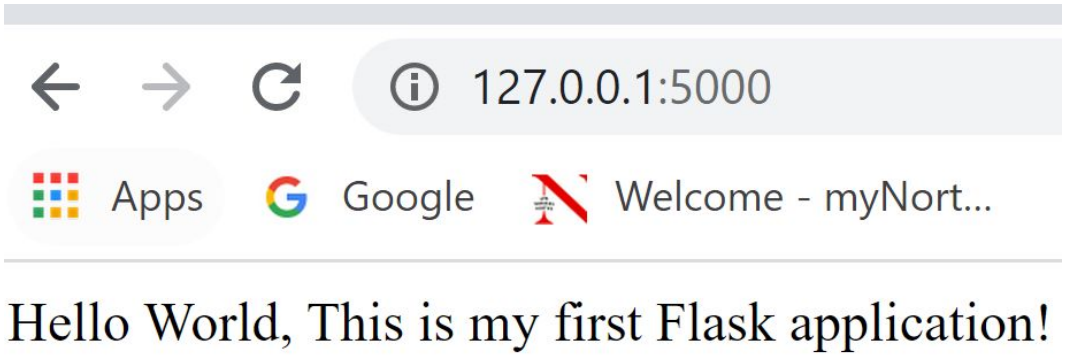
- Create a python file app.py

```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def welcome():
    return "Hello World, This is my first Flask application!"

app.run()
```

Run the file using below command

```
Python app.py
```



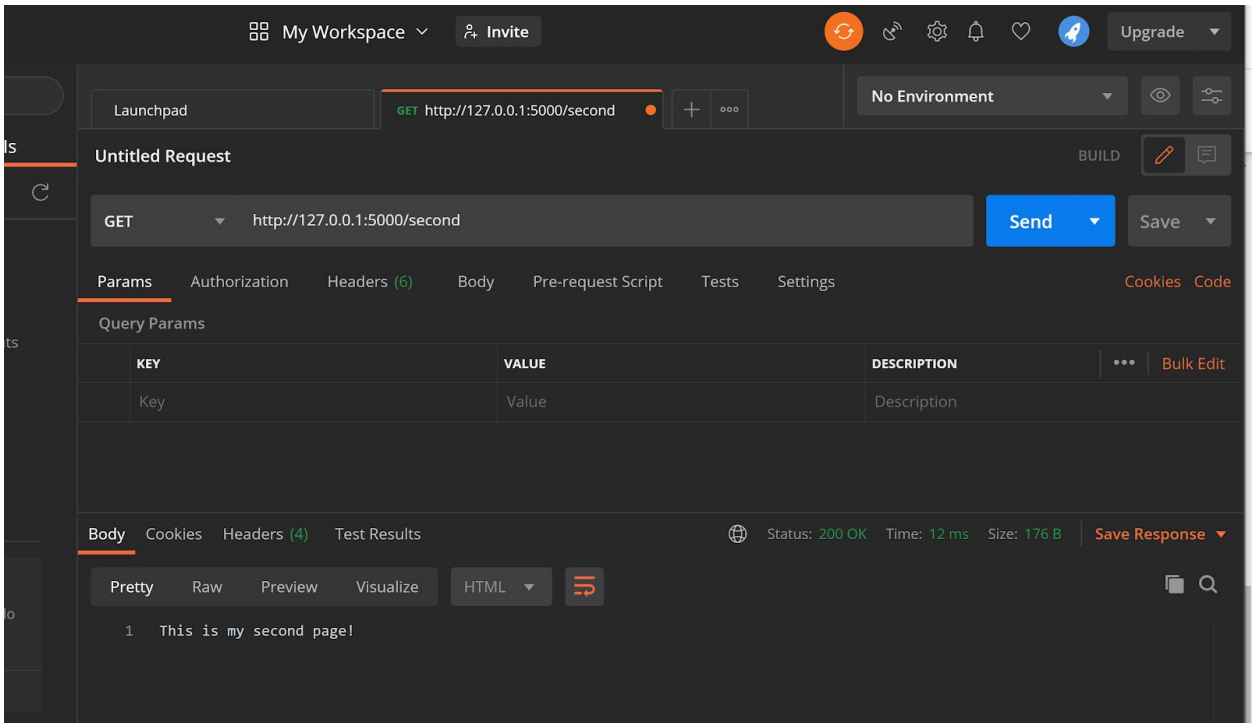



```
from flask import Flask,request

app = Flask(__name__)

@app.route('/')
def welcome():
    return "This is my first Flask app!"

@app.route('/second')
def secondpage():
    return "This is my second page!"

app.run()
```

Run it again

Go to postman -> API -> Create a Request

Paste under GET - http://127.0.0.1:5000/second



GET & POST request methods

HTTP Methods

- GET
- POST
- PUT
- HEAD
- DELETE
- PATCH
- OPTIONS

The most commonly used are GET and POST

GET requests

- GET is used to request data from a specified resource.
- GET is one of the most common HTTP methods.

Note that the query string (name/value pairs) is sent in the URL of a GET request:

```
/test/demo_form.php?name1=value1&name2=value2
```

Some other notes on GET requests:

- GET requests can be cached
- GET requests remain in the browser history

- GET requests can be bookmarked
- GET requests should never be used when dealing with sensitive data
- GET requests have length restrictions
- GET requests are only used to request data (not modify)

POST requests

POST is used to send data to a server to create/update a resource.
The data sent to the server with POST is stored in the request body of the HTTP request:

```
POST /test/demo_form.php HTTP/1.1
Host: somedomain.com
name1=value1&name2=value2
```

Some other notes on POST requests:

- POST requests are never cached
- POST requests do not remain in the browser history
- POST requests cannot be bookmarked
- POST requests have no restrictions on data length

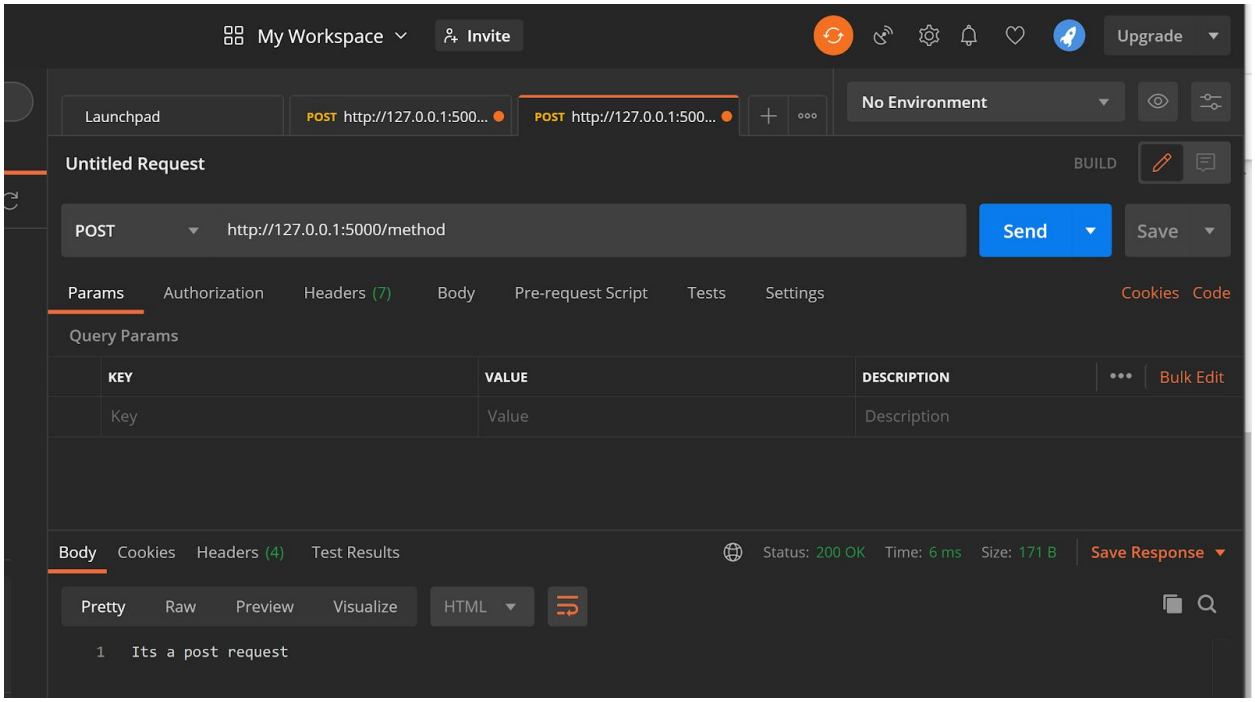
Difference Between GET & POST

GET is used for viewing something, without changing it, while POST is used for changing something. For example, a search page should use GET to get data while a form that changes your password should use POST

In general terms GET is used when server returns some data to the client and have no impact on server whereas POST is used to create some resource on server

Flask Application with GET and POST

```
from flask import Flask,request

app = Flask(__name__)

@app.route('/')
def welcome():
    return "Hello World, This is my first Flask application!"

@app.route('/second')
def secondpage():
    return "This is my second page!"


@app.route('/method',methods =['GET','POST'])
def method():
    if request.method == 'POST':
        return "It is a post request"
    else:
        return "It is a get request"

app.run()
```



Form Based Example

Create the below hierarchy -

```
Webapp
- app2.py
- templates
      - student.html
      - result.html
```

app.py

```
from flask import Flask, render_template, request
app = Flask(__name__)

@app.route('/')
def student():
   return render_template('student.html')

@app.route('/result',methods = ['POST', 'GET'])
def result():
   if request.method == 'POST':
      result = request.form
      return render_template("result.html",result = result)

app.run()
```

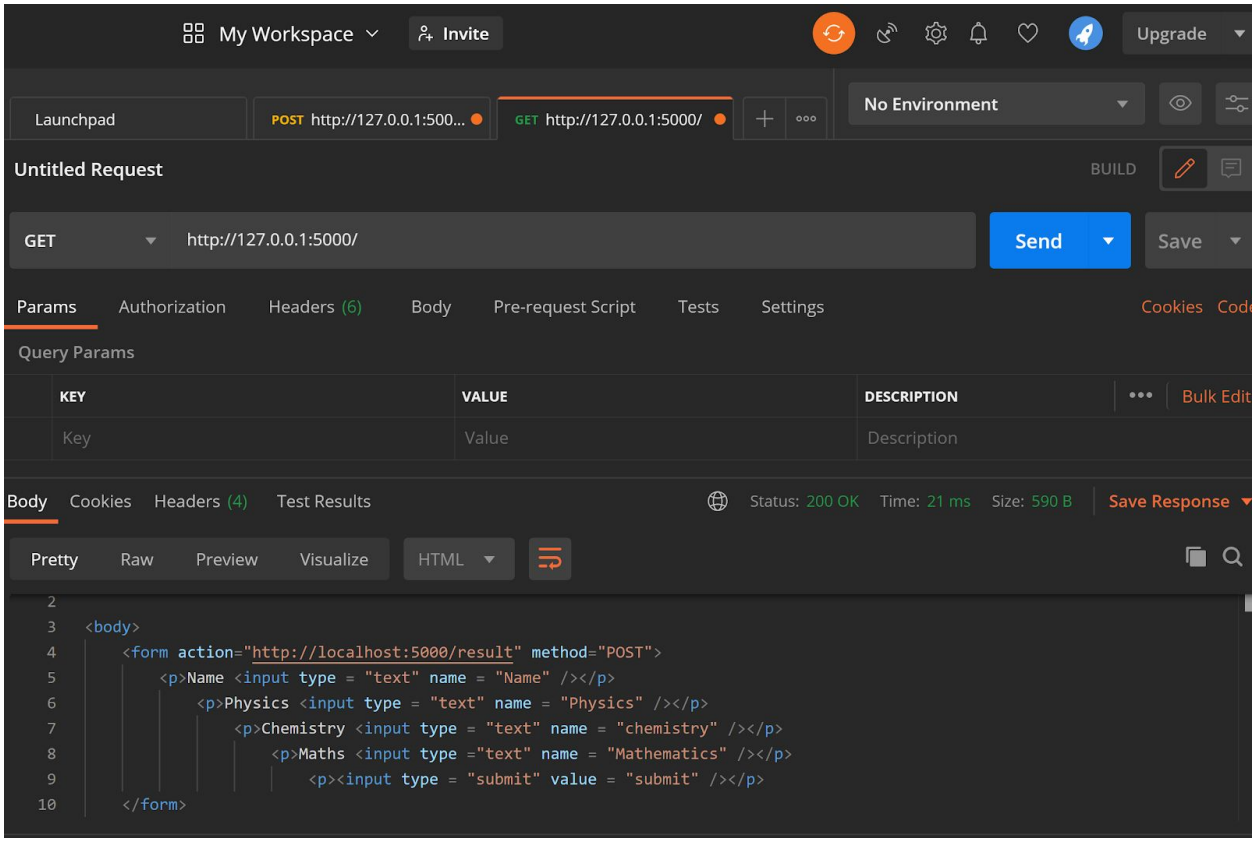
In the following example, '/' URL renders a web page (student.html) which has a form. The data filled in it is posted to the '/result' URL which triggers the result() function.
The results() function collects form data present in request.form in a dictionary object and sends it for rendering to result.html.
The template dynamically renders an HTML table of form data.

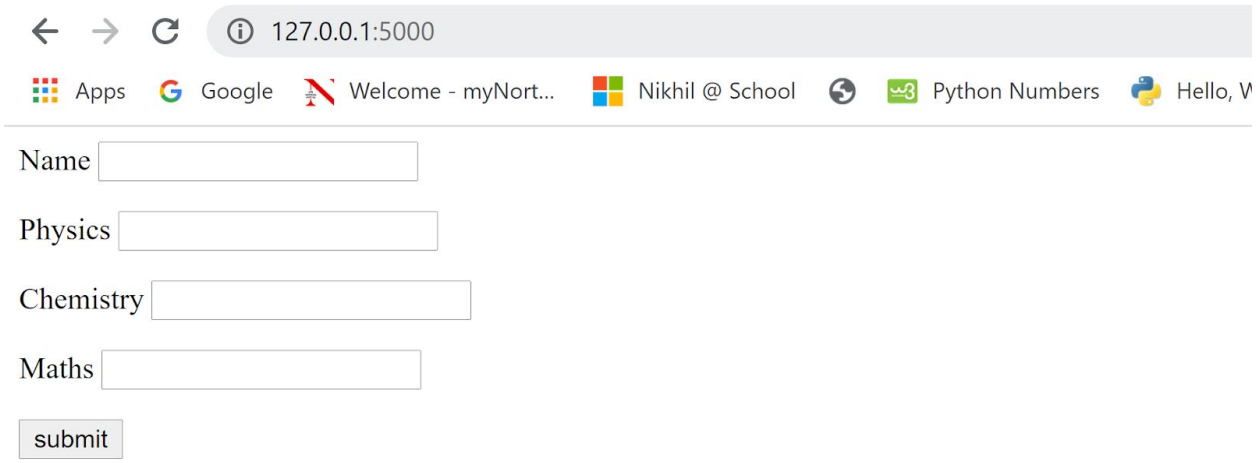
Student.html

```
<html>
   <body>
      <form action = "http://localhost:5000/result" method = "POST">
         <p>Name <input type = "text" name = "Name" /></p>
         <p>Physics <input type = "text" name = "Physics" /></p>
         <p>Chemistry <input type = "text" name = "chemistry" /></p>
         <p>Maths <input type ="text" name = "Mathematics" /></p>
         <p><input type = "submit" value = "submit" /></p>
      </form>
   </body>
</html>
```

Result.html

```
<!doctype html>
<html>
   <body>
      <table border = 1>
         {% for key, value in result.items() %}
            <tr>
               <th> {{ key }} </th>
               <td> {{ value }} </td>
            </tr>
         {% endfor %}
      </table>
   </body>
</html>
```




127.0.0.1:5000
Apps
Google
Welcome - myNort...
Nikhil @ School
Python Numbers
Hello, W
Name
Physics
Chemistry
Maths
submit