Flask Tutorial

The complete Flask beginner tutorial

#python #webdev #beginners

Gajesh Jul 17, 2019 · 10 min read

What is Flask?

According to flask's website Flask is a microframework for Python based on Werkzeug, Jinja 2 and good intentions. To put it in simple words, it's a python-based framework to build web applications.

But what is this Werkzeug (How to Pronounce?) and Jinja?

Werkzeug: It is one of the most advanced WSGI (Web Server Gateway Interface) modules that contain various tools and utilities that facilitate web application development. Flask is based on WSGI interface.

Jinja 2: It's a template rendering engine. It renders the web pages for the server with any specified custom content given to it by the webserver. We will talk about Jinja in detail in later parts of the series.

Why flask and what can you do with it?

The microframework in case of flask does not mean it can only do little or provides less functionality. It means it is more extensible and gives you the freedom to choose and decide the custom extensions and plugins to use. This can be helpful, as we shall see in the next part of the tutorial when we work on user management. (Log in, Sign up, Password change, etc.). Flask is more flexible compared to other Python web frameworks like Django, but the onus is on you to get things working correctly as there are not many rules or "Flask way" of doing things.

A Basic implementation

The best guide for flask is the flask documentation itself. It has answers to most of the questions, and I have to admit, it is one of the best-documented open source projects when it comes to details and clarity of writing. You can read the documentation for in-depth coverage.

You can create a flask application in a single file as described below. It's effortless to code even for the beginners. I assume that you have installed Python 3 and have it working. If not, you can Google and find some good tutorials for doing so with your concerned operating system.

Setting up Virtual Environments

To start working with the project, we have to do some basic set up so that we can proceed smoothly. Please do not skip these steps unless you know how to set up virtual environments and to run Flask.

Create a virtual environment so that you do not mess up your existing python libraries and functions on your system. To do that,

1. We will use `virtualenv` to manage a separate virtual environment for this project. That will help us avoid conflicts in dependencies across projects.
2. Install `virtualenv` in your system using the command `pip3 install virtualenv`.
3. Move to the project folder and type the command `virtualenv venv` to create a virtual environment by the name of `venv`, this will create a folder named `venv` which will have all the

11
5
21

python executables we will need to start working on a separate environment for the project w/o disturbing the existing system setup and dependencies of other libraries.

4. Remember to add `venv` to your .gitignore file so that it doesn't get written to your source control. (to put is simple words, it does not appear on your Github)

5. You can also change the python interpreter you are using by running the command `virtualenv -p /usr/bin/python3.6 venv`. Please do this, as we will be using Python 3 for the rest of the tutorial.

6. To start the virtual environment, you can type `source venv/bin/activate` Please remember to run this every time you want to work on the project. When the virtual environment is activated your shell prompt will look something like the prompt as follows
`(venv) yourusername@yourcomputername:~/Flask_project$`

7. When done working with the project, don't forget to deactivate by typing, `deactivate`

8. You can now install Flask inside the virtual environment using the command `pip3 install Flask`. It will also introduce some dependencies that it needs to work correctly. You can keep all the pip packages in a file called `requirements.txt`

Now that we have created a virtual environment, and installed flask, we can begin to write code. Below is the system to run a simple web application that says Hello world. This is straight from Flask's Docs.

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
   return 'Hello, World!'
```

Save it in a file named application.py (file name can be anything) and go to the command line and type (hit enter after typing each line of command)

```
export FLASK_APP=application.py
flask run
```

If you are using Windows, you can do the following,

1. If Powershell : `PS C:\path\to\app> $env:FLASK_APP = "application.py"`
2. If Command Prompt : `C:\path\to\app>set FLASK_APP=application.py`

Then type `flask run`

It will display a console output similar to

```
* Serving Flask app "application.py"
 * Environment: production
   WARNING: Do not use the development server in a production environment.
```

```
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [18/Jan/2019 23:56:53] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [18/Jan/2019 23:56:54] "GET /favicon.ico HTTP/1.1" 404
```

You can now go to `http://127.0.0.1:5000/` to view your first web application.

Some basic terminologies

The very first line imports the Flask from its library, and the next line creates a variable name and instantiates the app using Flask class.
The next line `@app.route('/')` (`/` means the home page) is called a route which is used to redirect the user to a specific page or perform a particular function when the user visit's a particular URL.

Example: `@app.route('/login')` redirects to login page. There are many other routes and labels we will go into later.

The rest of the code display the Hello world. But the method can do anything you would like it to do.

```
def hello_world():
    return 'Hello, World!'
```

You can create your routes and specify the things functions should do when particular routes are called.

Hat tip:

It is quite irritating to run `export FLASK_APP=application.py` every time you run your flask project. To solve this, you can add this line to your `~/.bashrc` (Linux) or `~/.bash_profile` (Mac) and save it so that the next time you don't have to rerun the export command. But remember, you will have to change the name of the file in the `bashrc` file if you change the name of the file `application.py` in your project. To enable all the developmental features, you can also add to `export FLASK_ENV=development` the bash file.

Working with routes, templates, and sessions in the Flask

Routes in Flask

Routes are places on a website that the users visit. Its also commonly known as URL (Uniform Resource Locator). It is important to have meaningful URL's to make it possible for users to remember and revisit the website easily. It also helps in SEO.

Flask uses the method `route()` to bind functions to a URL. Whenever a user visits a URL, the method attached to it executes.

```
@app.route("/")
def home_page():
    # '/' takes user to home page
@app.route("/login")
def login_page():
    # Process login
    return result
```

Flask supports different types of variable structures. You can also include custom strings, integers, and other kinds of characters in the URL's and display relevant results for them.

The value types are in the Flask documentation. I have reproduced it here.

You can create a custom URL with angle brackets (<>) and colon (:) as illustrated below.

```
@app.route("/user/<name>")
def display_user(name):
    # A string of any length(without slashes) can be assigned to the variable name.
    print(name)

@app.route("/total/<int:amount>")
def display_total_amount(amount):
    # Amount holds the value in int(Only Positive Integers). No other charcter accepted.
    print(amount)

@app.route("/path/<path:sub_path>")
def take_to_subpath(sub_path):
```

```
    # Accepts any string with slashes.
    print(sub_path)

@app.route("/key/<uuid:api_key>")
def display_key(api_key):
    # Unique 16 digit UUID. Helpful in API key or token generation/authentication.
    print(api_key)
```

Note on URL Behavior

If you add a leading slash to the route in Flask (Ex: `app.route("/user/ram/")`) and the URL is accessed without the slash then flask automatically redirects the user to slashed URL. But it's important to note that routes without leading slashes (`Ex: app.route("/user/ram")`) will not be redirected when the user accesses the URL with a leading slash. 404 error will be generated.

Accessing URL using method names

You can use `url_for()` to easily access the route or a method attached to the route. The first argument is function name, and you can pass other arguments of any length which comply with variable rules of the route. A simple example below illustrates the method `url_for`

```
@app.route("/login/<user>")
def login_user(user):
 # User login flow.
```

```
def redirect_to_login():
 url_for('login_user',user='some_user_name')
```

Handling Requests

When a user visits a website or makes a request, the website has to respond appropriately. Flask supports both GET and POST request. By default Flask only accepts GET request. In order to accept post requests, it has to be specified in parameters of the function `route()`

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        return do_the_login()
    else:
        return show_the_login_form()
```

Static Files and templates.

Every modern web page needs CSS and JavaScript. To use the CSS and JS in Flask, you need to create a `static` folder in the working directory of your flask application. You can place all your CSS and JS files there and access them in your project.

Example Project directory structure

```
|----- Project/
  |--- flask_app.py
  |--- static/
     |-- style.css
     |-- script.js
  |--- templates/
     |-- base.html
     |-- home.html
     |-- login.html
```

Templates folder in Flask is used to store HTML files and other web pages you want to display with the application. When you refer to the web page in your code, it will look into the folder templates in the project directory.

Rendering templates

You can render templates in the flask using the methodrender_template. Below code illustrates a simple example.

```
from Flask import render_template

@app.route("/")
def display_home():
    return render_template('home.html',thing_to_say='hello')
```

`home.html` looks like

```
<html>
    <body>
    <h1>
     Flask says {{ thing_to_say }}
     </h1>
    </body>
</html>
```

Note that in the above code, the variable you pass from the back end of your program is referenced inside the tag `<h1>` within `{{ }}` braces. You can read more about those details Jinja Docs and Rending templates in Flask Docs.

Reading data from Forms and working with Sessions

When you build web applications, you often fetch data from users using forms on the website. Flask's object request can handle that data.

Below an example from the Docs explains it clearly.

```
from flask import request

@app.route('/login', methods=['POST', 'GET'])
def login():
```

```
    error = None
    if request.method == 'POST':
        if valid_login(request.form['username'],
                       request.form['password']):
            return log_the_user_in(request.form['username'])
        else:
            error = 'Invalid username/password'
    # the code below is executed if the request method
    # was GET or the credentials were invalid
    return render_template('login.html', error=error)
```

`request.form` can be used to access fields of the form.

You might want to store some custom information about a user in his browser. Cookies will help you do that. But to use cookies plainly can be a security risk. Hence it is recommended to use `Sessions`.

`Sessions` allow you to store information specific to a user from one request to the next. The advantage of sessions is that it signs the cookie cryptographically so that hey cannot be copied or stolen.

To encrypt a session, you need to assign an Application secret. This must be a random string stored on your computer and should be stored securely as it can be used to decrypt all the sessions. Below code illustrates an example from the Docs.

```
from flask import Flask, session, redirect, url_for, escape, request
from os import environ
```

```
app = Flask(__name__)
# Set the secret key to some random bytes. DO NOT put it in source code. Instead use something l
app.secret_key = environ('FLASK_SECRET')
@app.route('/')
def index():
    if 'username' in session:
        return 'Logged in as %s' % escape(session['username'])
    return 'You are not logged in'
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        session['username'] = request.form['username']
        return redirect(url_for('index'))
    return '''
        <form method="post">
            <p><input type=text name=username>
            <p><input type=submit value=Login>
        </form>
    '''
@app.route('/logout')
def logout():
    # remove the username from the session if it's there
    session.pop('username', None)
    return redirect(url_for('index'))
```

You can generate a good Pseudo-random number using the python shell.
`$ python3 -c 'import os; print(os.urandom(30))'`

You can add the output of that command to `~/.bash_profile` or `~/.bashrc as FLASK_SECRET`.

Managing user accounts in Flask using Flask-User

Installation and setup

You can add `Flask-User==0.6.21` to your `requirements.txt` or just type pip install `Flask-User==0.6.21` to install flask. That installs flask user with all the necessary dependencies needed.

One more thing you need to do is to download the templates available in the Flask -user Github repository, to add customizations to your login page. To do that you need to create a template folder in your project folder which to store all the HTML files that you use for your web application.

Then download the source form the Github Repository link above and extract the zip file and move to the `flask_user` folder then to templates and copy all its contents to your project folder.

Don't reinvent the wheel

While I was researching for the project and writing this article, I found a series of simple tutorial videos of Anthony from Pretty Printed. So Instead of reinventing the wheel. I link you to tutorials on Flask user.

Part 1: Introduction to Flask-User

Part 2: Flask-User: Enabling Emails

Part 3: Flask-User: Configuration And Templates

Note: The Part numbering is done by me for you to follow the flow quickly.

If you get stuck somewhere, you can always refer to the documentation and feel free to send hugs or bugs my way.

Many thanks to Anthony of Pretty Printed for detailed and well-explained videos.

Reference and further reading

1. Flask Documentation
2. Virtual environment setup
3. Jinja Documentation
4. Flask-User Github Repo
5. Subscribe to Pretty Printed on Youtube

Discussion (0) Subscribe

Add to the discussion

Code of Conduct • Report abuse

Gajesh

Reading, Coding, and writing. In that order.

Follow

LOCATION
London, Ontario. Canada

EDUCATION
MSc. Computer Science Candidate at Western University

JOINED
Apr 13, 2019

More from Gajesh

All you need to know about python virtual environments

#python #webdev #beginners #productivity

The complete django-allauth guide

#python #django #webdev #beginners

Compress images in django

#python #django #webdev #beginners