Creating Flask Application

Introduction

Flask is a lightweight Python web framework that provides useful tools and features for creating web applications in the Python Language. It gives developers flexibility and is an accessible framework for new developers because you can build a web application quickly using only a single Python file. Flask is also extensible and doesn't force a particular directory structure or require complicated boilerplate code before getting started.

Learning Flask will allow you to quickly create web applications in Python. You can take advantage of Python libraries to add advanced features to your web application, like storing your data in a database, or validating web forms.

In this tutorial, you'll build a small web application that renders HTML text on the browser. You'll install Flask, write and run a Flask application, and run the application in development mode. You'll use routing to display various web pages that serve different purposes in your web application. You'll also use view functions to allow users to interact with the application through dynamic routes. Finally, you'll use the debugger to troubleshoot errors.

Prerequisites

- A local Python 3 programming environment.
- An understanding of basic Python 3 concepts, such as data types, lists, functions, and other such concepts.
- An understanding of basic HTML concepts.

Step 1 — Installing Flask

In this step, you'll activate your Python environment and install Flask using the pip package installer.

First, activate your programming environment if you haven't already:

```
source env/bin/activate
```

Once you have activated your programming environment, install Flask using the `pip install` command:

```
pip install flask
```

Once the installation is complete, you will see a list of installed packages in the last parts of the output, similar to the following:

```
Output

Installing collected packages: Werkzeug, MarkupSafe, Jinja2,
itsdangerous, click, flask

Successfully installed Jinja2-3.0.1 MarkupSafe-2.0.1 Werkzeug-2.0.1
click-8.0.1 flask-2.0.1 itsdangerous-2.0.1
```

This means that installing Flask also installed several other packages. These packages are dependencies Flask needs to perform different functions.

You’ve created the project folder, a virtual environment, and installed Flask. You can now move on to setting up a simple application.

Step 2 — Creating a Simple Application

Now that you have your programming environment set up, you’ll start using Flask. In this step, you’ll make a small Flask web application inside a Python file, in which you’ll write HTML code to display on the browser.

In your `flask_app` directory, open a file named `app.py` for editing, use `nano` or your favorite text editor:

```
1. nano app.py
```

Write the following code inside the `app.py` file:

flask_app/app.py

```
from flask import Flask

app = Flask(__name__)


@app.route('/')
def hello():
    return '<h1>Hello, World!</h1>'
```

Save and close the file.

In the above code block, you first import the `Flask` object from the `flask` package. You then use it to create your Flask application instance, giving it the name `app`. You pass the special variable `__name__`, which holds the name of the current Python module. This name tells the instance where it’s located; you need this because Flask sets up some paths behind the scenes.

Once you create the `app` instance, you can use it to handle incoming web requests and send responses to the user. `@app.route` is a decorator that turns a regular Python function into a Flask *view function*, which converts the function’s return value into an HTTP response to be displayed by an HTTP client, such as a web browser. You pass the value `'/'` to `@app.route()` to signify that this function will respond to web requests for the URL `/`, which is the main URL.

The `hello()` view function returns the string `'<h1>Hello, World!</h1>'` as an HTTP response.

You now have a simple Flask application in a Python file called `app.py`, in the next step, you will run the application to see the result of the `hello()` view function rendered in a web browser.

Step 3 — Running the Application

After creating the file that contains the Flask application, you'll run it using the Flask command line interface to start the development server and render on the browser the HTML code you wrote as a return value for the `hello()` view function in the previous step.

First, while in your `flask_app` directory with your virtual environment activated, tell Flask where to find the application (`app.py` in your case) using the `FLASK_APP` environment variable with the following command (on Windows, use `set` instead of `export`):

```
export FLASK_APP=app
```

Then specify that you want to run the application in development mode (so you can use the debugger to catch errors) with the `FLASK_ENV` environment variable:

```
export FLASK_ENV=development
```

Lastly, run the application using the `flask run` command:

```
flask run
```

Once the application is running, the output will be something like this:

```
Output

 * Serving Flask app "app" (lazy loading)

 * Environment: development

 * Debug mode: on

 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

 * Restarting with stat

 * Debugger is active!

 * Debugger PIN: 296-353-699
```

The preceding output has several pieces of information, such as:

- The name of the application you're running (`"app"`).
- The environment in which the application is being run (`development`).
- `Debug mode: on` signifies that the Flask debugger is running. This is useful when developing because it provides detailed error messages when things go wrong, which makes troubleshooting easier.
- The application is running locally on the URL `http://127.0.0.1:5000/`. `127.0.0.1` is the IP that represents your machine's `localhost` and `:5000` is the port number.

Open a browser and type in the URL `http://127.0.0.1:5000/`. You will see the text `Hello, World!` in an `<h1>` heading as a response. This confirms that your application is successfully running.