Chapter 16: Python Web Development with Flask

Contents

Time required: 120 minutes

HTML

If you are not familiar with HTML, this is a good time to start. Here are some tutorials to get you started on the basics.

- https://www.w3schools.com/html/html_basic.asp
- https://www.w3schools.com/html/html_elements.asp
- https://www.w3schools.com/html/html_attributes.asp
- https://www.w3schools.com/html/html_headings.asp
- https://www.w3schools.com/html/html_paragraphs.asp

What is Flask?

Flask is a full stack development framework using MVC (Model, View, Controller)

- The three components of the MVC pattern are decoupled and they are responsible for different things:

- **Model**: manages the data and defines rules and behaviors. It represents the business logic of the application. The data can be stored in the Model itself or in a database (only the Model has access to the database).
- **View:** presents the data to the user. A View can be any kind of output representation: a HTML page, a chart, a table, or even a simple text output. A View should never call its own methods; only a Controller should do it.
- **Controller** accepts user's inputs and delegates data representation to a View and data handling to a Model.

Since Model, View and Controller are decoupled, each one of the three can be extended, modified, and replaced without having to rewrite the other two.

The Flask MVC model

Model: SQLite

View: HTML CSS pages

Controller: Flask Py files

https://www.fullstackpython.com/table-of-contents.html

Flask is a web application framework written in Python. Armin Ronacher, who leads an international group of Python enthusiasts named Pocco, develops it.

Flask is a micro-framework developed in Python that provides only the essential components - things like routing, request handling, and sessions. Flask is considered "beginner friendly" for someone wanting to get started with active web site development.

It provides you with libraries, tools, and modules to develop web applications like a blog, wiki, or even a commercial website.

Micro-frameworks are the opposite of full-stack frameworks, which also offer additional modules for features such as authentication, database ORM, input validation and sanitization, etc.

Flask is known as a micro-framework because it is lightweight and only provides components that are essential, such as routing, request handling, sessions, and so on. For the other functionalities such as data handling, the developer can write a custom module or use an extension. This approach avoids unnecessary boilerplate code, which is not even being used.

Tutorial 1: Getting Started with Flask - Hello World

We will build the traditional Hello World project with Flask.

1. Install Flask: **pip install flask**
2. Create a folder for your project.

Add the following code to your project.

```
"""
    Name: hello_flask_1.py
    Author:
    Created:
    Purpose: My first Python Flask web application
"""

# pip install flask
# From the flask library, import the Flask class
from flask import Flask

# Create an instance of the flask class
app = Flask(__name__)


# Route decorator tells Flask which URL
# will trigger our function
# Go to the home or root page
# http://127.0.0.1:5000/
@app.route("/")
def hello_world():
    # This html is returned to our browser
    return "<p>Hello, World!</p>"


# Start the flask application
if __name__ == "__main__":
    app.run()
```

'/' URL is bound with **hello_world()** function. When the home page of the webserver is opened in the browser, the output of this function will be rendered accordingly.

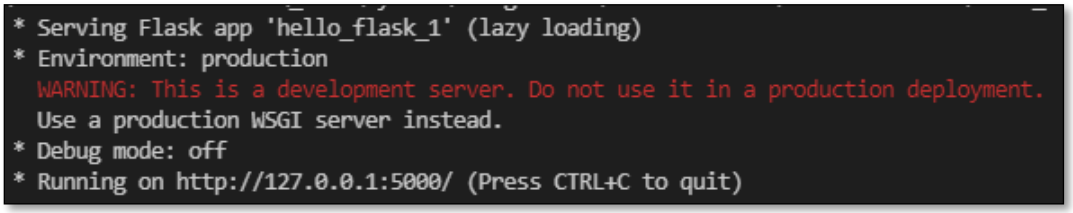
Web frameworks provide routing technique so that user can remember the URLs. It is useful to access the web page directly without navigating from the Home page. It is done through the following **route()** decorator, to bind the **URL** to a **function**.

Functions like **hello_world()** that handle URL's are called view functions. The return value of this function is the **response** the client receives, typically through a web browser

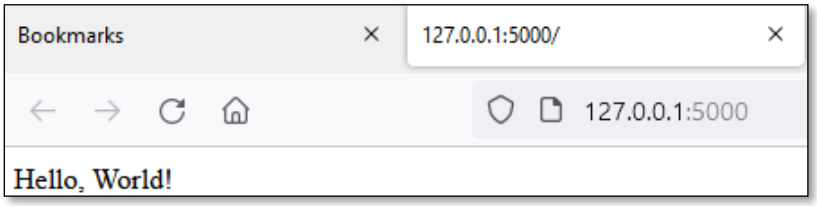
```
# Decorator to route URL
@app.route("/")
# Binding to the function of the route
def hello_world():
    return "<p>hello world</p>"
```

The Flask application is started by calling the run() function.

In VSCode, you will see the following.



CTRL Click on the URL to view the web page.



Tutorial 2: Another Route

In this one we add debugging to the app.run command.

```
app.run(debug=True)
```

We can edit and save code while the web server is running. When the code is saved, the web server automatically reloads the page.

In **VSCode → Run → Run Without Debugging**

We added another route. **/bye** will give us the URL of **http://127.0.0.1:5000/bye**

```python
"""
    Name: hello_flask_2.py
    Author:
    Created:
    Purpose: My first Python Flask web application
    We added debugging, this automatically reloads the web server
    whenever we modify and save the code.
    Run the program from the command line, Idle,
    or from VSCode without debugging
"""
# pip install flask
# From the flask library, import the Flask class
from flask import Flask
# Create a Flask object
app = Flask(__name__)


# Route decorator tells Flask which URL
# will trigger our function
# Go to the home or root page
# http://127.0.0.1:5000/
@app.route("/")
def hello_world():
    # This html is returned to our browser
    return "<p>Hello, World!</p>"


# http://127.0.0.1:5000/bye
@app.route("/bye")
def say_bye():
    # This html is returned to our browser
    return "<p>Bye</p>"


# Start the flask application
# debug=True will reload the application
# as changes are made and saved
if __name__ == "__main__":
    app.run(debug=True)
```

Tutorial 3: Variables

Variables in Flask are used to build a URL dynamically by adding the variable parts to the rule parameter. This is called a dynamic route. This variable part is marked as <variable>. It is passed as keyword argument to the function.

The parameter of **route()** decorator contains the variable part attached to the URL "/hello" as an argument. If a URL like http://localhost:5000/hello/Bill is entered then "Bill" will be passed to the **hello()** function as an argument.

Add the following code to the project.

```
# pip install flask
# From the flask library, import the Flask class
from flask import Flask, request
# Create a Flask object
app = Flask(__name__)
```

The above code add the request import.

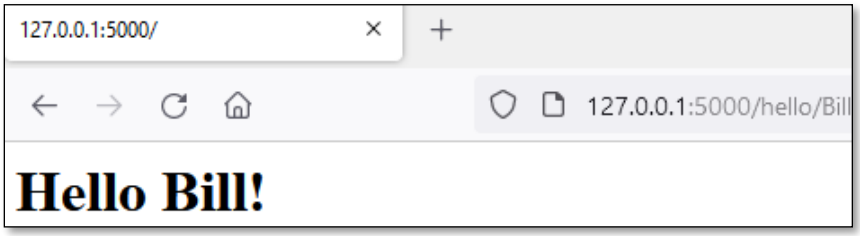
```
# Routing the decorator function hello_name
# http://127.0.0.1:5000/hello/YourName
@app.route('/hello/<name>')
def hello_name(name):
    return f"<h1>Hello {name}!<h1/>"
```

This new route display the name typed in at the end of the URL

```
@app.route("/")
def hello_world():
    # This html is returned to our browser
    user_agent = request.headers.get("User-Agent")
    return f"""<h2>Hello, World!</h2>
    <p>Your browser is {user_agent}
    """
```

This change to the default route returns the browser type.

Example run:



Tutorial 4: Render HTML with Jinja2 Templates

Flask uses the Jinja2 template engine for rendering templates.

The function **render_template()** is used to render a Jinja2 template to an HTML page for display on web browser. In this example, the template is simply an HTML file (which does not require rendering). A Jinja2 template may contain expression, statement, and other features.

Make the following changes to our project.

1. Create a folder named: **templates**
2. In that folder, create an html file named: **hello.html**
3. Add the following to the **hello.html** file.

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="utf-8">
    <title>Say hello</title>
</head>

<body>
    <h1>Hello, from templates</h1>
</body>

</html>
```

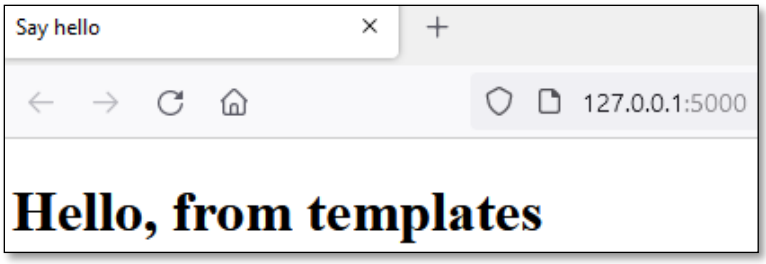
Make the following changes to the project file. Add **render_template** to the imports as shown below.

```
"""
    Name: hello_flask_4.py
    Author:
    Created:
    Purpose: My first Python Flask web application

"""
# pip install flask
# Import the Flask class and render_template
from flask import Flask, render_template
```

Modify the default route.

```
@app.route("/")
def hello_world():
    # Render an HTML template and return
    return render_template("hello.html")
```

Example run:



Tutorial 5: Calculator Project

GET and POST are two different types of HTTP requests. GET is used for viewing something, without changing it, while POST is used for changing something.

https://www.w3schools.com/tags/ref_httpmethods.asp

It is time for a project that actually does something.

1. Create a project folder: **Calculator**
    a. In the project folder create: **templates**
2. In the Calculator folder: **calculator.py**

Add the following code:

```python
"""
    Name: calculator.py
    Author:
    Created:
    Purpose: Add 2 numbers together

"""
# pip install flask
# Import the Flask class, render_template, and request
from flask import Flask, render_template, request
# Create Flask object
app = Flask(__name__)

# Go to the home or root page
# http://127.0.0.1:5000/


@app.route("/", methods=["GET", "POST"])
def home():
    # If the request is POST
    if request.method == "POST":
        # Get input from user into variables
        num1 = request.form.get("num1")
        num2 = request.form.get("num2")

        # Calculate the sum of the two numbers
        sum = float(num1) + float(num2)

        # Return the results using the variable sum
        # in a different web page.
        return render_template("result.html", sum=sum)

    # Otherwise we return index.html with GET
    return render_template("index.html")


if __name__ == "__main__":
    app.run(debug=True)
```

The following web page uses an html form to get user input using the POST method.

https://www.w3schools.com/html/html_forms.asp

Add **index.html** to the templates folder.

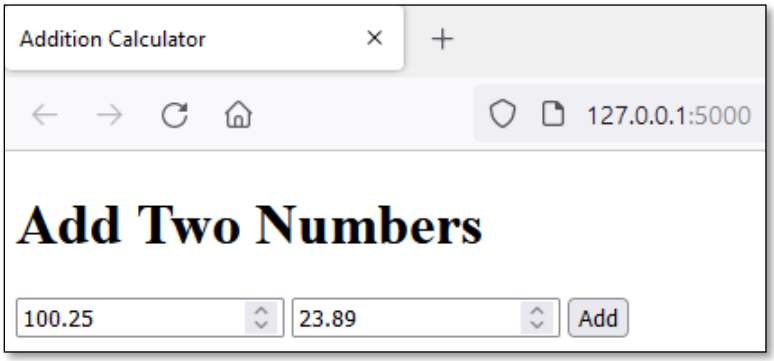
```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>Addition Calculator</title>
</head>

<body>
    <h1>Add Two Numbers</h1>
    <form action="/" method="POST">
        <input type="number" step="any" name="num1" placeholder="Enter num1" id="">
        <input type="number" step="any" name="num2" placeholder="Enter num2" id="">
        <button type="submit">Add</button>
    </form>
</body>

</html>
```

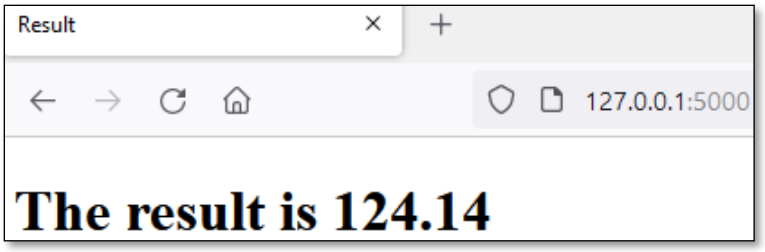
The following web page displays the variable **sum** from the Flask code.

Add **result.html** to the templates folder.

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>Result</title>
</head>

<body>
    <h1>The result is {{sum}}</h1>
</body>

</html>
```

Example run:





Assignment Submission

1. Attach all tutorials and assignments.
2. Attach screenshots showing the successful operation of each tutorial program.
3. Submit in Blackboard.