Chapter 8 Web applications Part 2 (Flask)

Contents

1 What is Flask?
- 1.1 Installing Flask
- 1.2 Creating a Simple Flask App
- 1.3 Accessing the App
- 1.4 Routing in Flask

2 File Hierarchy of a Flask project

3 Using a HTML Template

4 Dynamic webpage
- 4.1 Using the GET method
- 4.2 Using the POST method
- 4.3 Difference between GET and POST methods

Annex 1 – Layout.html

Syllabus Learning Outcomes

4.2 Web Applications

Understand the concepts and techniques for developing web applications.

4.2.3 Use HTML, CSS (for clients) and Python (for the server) to create a web application that is able to:
– accept user input (text and image file uploads)
– process the input on the local server
– store and retrieve data using an SQL database
– display the output (as formatted text/images/table).

4.2.4 Test a web application on a local server.

1 What is Flask?

Flask is a flexible and powerful web application framework that allows developers to build web applications and APIs using Python quickly and easily. Flask provides tools and libraries for handling HTTP requests and responses, managing routing, and handling errors. It also integrates with a variety of popular Python libraries, making it easy to build complex applications with Flask.

1.1 Installing Flask

Before we can start building our web application, we need to install Flask. Flask can be installed using pip, which is a package installer for Python. Open up your terminal or command prompt and type:

```
pip install flask
```

This will install Flask and its dependencies.

1.2 Creating a Simple Flask App

Now that we have Flask installed, we can write a simple web application by creating a Python file app.py and add the following code:

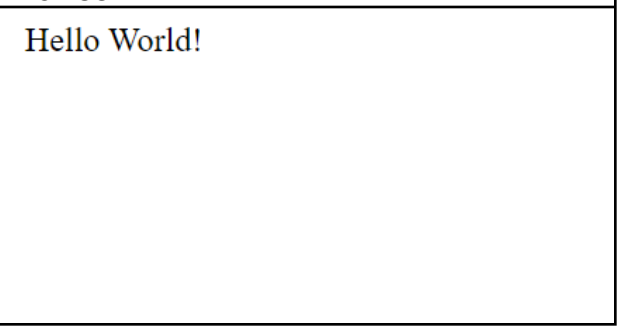
app.py

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

@app.route('/')
def index():
    return 'Hello World!'

if __name__ == '__main__':
    app.run()
```

Browser



This code imports the Flask module, creates a new Flask app, and defines a route that responds with "Hello, world!" when the root URL is requested.

In Python, the special variable `__name__` is a built-in variable that represents the name of the current module. When a Python script is executed, its `__name__` variable is automatically set to `'__main__'` if it is the entry point to the program.

In the context of a Flask web application, `__name__` is used to determine the name of the application package. When creating a Flask app, we typically pass `__name__` as an argument to the `Flask()` constructor to let Flask know the name of our application package.

The `app.run()` method is used to start the Flask development server. By wrapping it in the `if __name__ == '__main__':` block, we ensure that the development server is only started when the script is run as the main program. If the script is imported as a module, the `app.run()` method will not be executed.

Save the file and proceed to run the module. This can be done either through an Integrated Development Environment (IDE) such as IDLE, or by running the command 'python app.py' in either Powershell or Command Prompt.

1.3 Accessing the App

Now that the app is running, open up a web browser and navigate to http://localhost:5000 or http://127.0.0.1:5000/. You should see the message "Hello, world!" displayed on the page.

That's it! You've just created a simple Flask app.

1.4 Routing in Flask

In Flask, routing is defined using the @app.route() decorator. The decorator is used to specify the URL path that the route will respond to and the HTTP methods that are allowed for that path. Here's an example of a basic Flask route:

```
@app.route('/')
def index():
    return 'This is Home Page'

@app.route('/hello')
def hello():
    return 'This is Hello, World Page'
```

Now let's understand the code:

1. `app.route('/')` will route to the home page URL, trailing slash `'/'` is generally used as a convention for the home page.
2. `app.route('/hello')` will route to the hello page URL.

In this way, we can render as many distinct web page URLs we want.

2 File Hierarchy of a Flask project

A typical file hierarchy of a Flask project includes the following directories and files:

1. The **main** directory: This directory contains the main application file and other directories, including app.py, the main Flask application file that defines the routes, views, models, and other components of the application.
2. The **templates** directory: This directory contains the HTML templates used for rendering the views of the application.
3. The **static** directory: This directory contains static files, including CSS, JavaScript, and image files, used for styling and functionality in the application.

3 Using a HTML Template

App.py imports the Flask and render_template modules, creates a new Flask app, and defines a route that responds with an HTML template when the root URL is requested. The `render_template()` function is used to render an HTML template.

Create a new folder called ‘templates’ and create a new file called ‘index.html’ inside that folder. The templates folder has to be in the same directory as the app.py file. Add the following HTML code in index.html:

app.py

```
from flask import Flask, render_template

app = Flask(__name__)

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

if __name__ == '__main__':
    app.run()
```

Index.html

```
<!DOCTYPE html>

<html>
  <head>
    <title>hello</title>
  </head>
  <body>
    <h1>Hello, world!</h1>
  </body>
</html>
```

Run app.py and you should see the "Hello, world!" message displayed on your browser.

That's it! You've just created a simple Flask app that renders an HTML template.

Thus far, we have only been rendering static webpages. It is now time to explore the process of creating dynamic webpages.

4 Dynamic webpage

Instead of returning short HTML snippets or redirecting users, the following program illustrates how we can return full HTML documents. Also, we can serve content depending on the path.

4.1 Using the GET method

This simple Flask web application defines two routes for rendering HTML templates.

The first route is for the root URL ("/"). When a GET request is sent to this route, the index function is called, which renders an HTML template called "index.html". This template will be returned as the response to the request.

The second route is for the URL "/greet". When a GET request is sent to this route, the greet function is called. This function reads the 'name' parameter from the request's query string using the request.args.get method. It then passes this 'name' parameter as an argument to another HTML template called "greet.html". This template will be rendered with the 'name' parameter included, and the resulting HTML will be returned as the response to the request.

app.py

```
from flask import Flask, render_template, request

app = Flask(__name__)

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

@app.route('/greet')
def greet():
      name = request.args.get("name")
      return render_template("greet.html", name=name)

if __name__ == '__main__':
    app.run()
```

index.html

```
<!DOCTYPE html>

<html>
  <head>
    <title>hello</title>
  </head>
  <body>
    <form action="/greet" method="get">
      <input name="name" placeholder="Name" type="text">
      <button type="submit">Greet</button>
    </form>
  </body>
</html>
```

greet.html

```
<!DOCTYPE html>

<html>
  <head>
    <title>hello</title>
  </head>
  <body>
    <h1>Hello, {{ name }}</h1>
  </body>
</html>
```

Browser

The URL tells us that the route '/greet' is responsible for processing the data that was submitted via the form, and that 'David' is the actual data that was entered into the form. This could be problematic if sensitive or confidential information is being submitted.

A simple solution to this problem is to modify the form method from 'get' to 'post'. The Flask application, by default, uses the 'get' method, but we can switch to using the 'post' method instead.

By making this modification in the HTML, the browser is instructed to submit the data using the 'post' method, while the Flask application is configured to receive the data using the 'post' method as well. Consequently, when the form data is submitted, it will not be appended to the URL.

4.2 Using the POST method

app.py

```python
from flask import Flask, render_template, request

app = Flask(__name__)

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

@app.route('/greet', methods=['POST'])
def greet():
    name = request.form["name"]
    return render_template("greet.html", name=name)

if __name__ == '__main__':
    app.run()
```

index.html

```html
<!DOCTYPE html>

<html>
  <head>
    <title>hello</title>
  </head>
  <body>
    <form action="/greet" method="post">
      <input name="name" placeholder="Name" type="text">
      <button type="submit">Greet</button>
    </form>
  </body>
</html>
```

greet.html

```html
<!DOCTYPE html>

<html>
  <head>
    <title>hello</title>
  </head>
  <body>
    <h1>Hello, {{ name }}</h1>
  </body>
</html>
```

Browser

To make the program more streamlined, we could potentially consolidate the two routes into a single one, in an effort to reduce the number of routes being used.

app.py

```
from flask import Flask, render_template, request

app = Flask(__name__)

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

if __name__ == '__main__':
    app.run()
```

index.html

```
<!DOCTYPE html>

<html>
  <head>
    <title>hello</title>
  </head>
  <body>
    <form action="/" method="post">
      <input name="name" placeholder="Name" type="text">
      <button type="submit">Greet</button>
    </form>
  </body>
</html>
```

The route is set for the root URL ('/') and supports both GET and POST methods. When a GET request is sent to this route, the index function is called and it renders an HTML template called "index.html", which will be returned as the response to the request.

When a POST request is sent to this route, the index function checks if the request method is "GET" or "POST". If it's "POST", then the function reads the 'name' field from the form data submitted in the request using the request.form["name"] method. It then passes this 'name' parameter as an argument to another HTML template called "greet.html". This template will be rendered with the 'name' parameter included, and the resulting HTML will be returned as the response to the request.

4.3 Difference between GET and POST methods

The main difference between the GET and POST methods in HTTP is the way in which the data is transmitted between the client (e.g. browser) and the server.

GET requests are primarily used for fetching data from the server. The data is transmitted in the URL itself as query parameters, visible in the address bar of the browser. These parameters can be bookmarked or shared with others, as the URL itself contains all the necessary data to retrieve the requested resource. Because the data is part of the URL, GET requests have a limit on the amount of data that can be transmitted.

On the other hand, POST requests are used to submit data to the server to perform some action. The data is transmitted in the request body, rather than the URL, making it more secure than GET requests for sensitive information. POST requests have no limit on the amount of data that can be transmitted, making them suitable for large data sets. However, POST requests cannot be bookmarked or shared, as the data is not part of the URL.

In summary, GET requests are primarily used to retrieve data, while POST requests are used to submit data for processing on the server. GET requests transmit data in the URL, while POST requests transmit data in the request body.

Annex 1 Layout.html

Layout.html is a template file used in web development to define the structure and layout of a webpage. It typically contains the common elements that are shared across multiple pages of a website, such as the header, navigation menu, and footer.

In web development frameworks such as Flask, layout.html is often referred to as a base or parent template, as it serves as the foundation for all other templates in the application. Other templates can extend or inherit from layout.html, and then fill in the specific content unique to each page.

The structure and contents of layout.html vary depending on the needs of the website, but it usually includes HTML, CSS, and JavaScript code. It may also include placeholders or tags for dynamic content such as user authentication status, page title, or page-specific content.

To illustrate, suppose we use index.html and greet.html as our examples. By extending from layout.html, the updated versions of index.html and greet.html might appear as follows:

layout.html

```
<!DOCTYPE html>

<html>
  <head>
    <title>hello</title>
  </head>
  <body>
    {% block body %}{% endblock %}
  </body>
</html>
```

index.html

```
{% extends "layout.html" %}

{% block body %}

  <form action="/" method="post">
    <input name="name" placeholder="Name" type="text">
    <button type="submit">Greet</button>
  </form>

{% endblock %}
```

greet.html

```
{% extends "layout.html" %}

{% block body %}

  <h1>Hello, {{ name }}</h1>

{% endblock %}
```

Let's break down the code:

- The {% extends 'layout.html' %} tag tells the template to inherit from layout.html.
- The {% block body %} tag sets the page-specific body to be inserted into the layout.html file.

When a user visits the index.html, the template engine combines layout.html and index.html to create the final HTML document. The header and footer are taken from layout.html, while the content is taken from index.html.

You would create similar templates with their own page-specific content inserted into the {% block body %} tag. This allows you to reuse the common elements across all pages and makes it easy to maintain a consistent look and feel for the website.