Flask

An introduction to the world of python's web micro frameworks

Flask Introduction

- 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 based on Werkzeug WSGI toolkit and Jinja2 template engine.
- WSGI is a specification for a universal interface between the web server and the web applications.
- 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
- Flask does not have built-in abstraction layer for database handling, nor does it have form a validation support. Instead, Flask supports the extensions to add such functionality to the application.

Install virtualenv for development environment

- virtualenv is a virtual Python environment builder. It helps a user to create multiple Python environments side-by-side.
- The following command install virtualenv

```
pip install virtualenv
```

- Once installed, new virtual environment is created in a folder.

```
mkdir newproj
cd newproj
virtualenv venv
```

- To activate corresponding environment, use the following

```
venv\scripts\activate
```

- We are now ready to install Flask in this environment.

```
pip install Flask
```

Flask Applications

- After completing the installation of the package, let's get our hands on the code.

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

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

- Save it as **hello.py** or something similar.

```
$ flask --app hello run
 * Serving Flask app 'hello'
 * Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
```

Flask App Routing

- One of the key features of Flask is its HTTP routing system, which allows developers to map URLs to specific functions or views within their application.
- HTTP routing is the process of determining which function or view should handle a particular URL request. In Flask, this is done using the `@app.route()` decorator, which is applied to a Python function to specify the URL route that it should handle.

Defining Routes in Flask

- To define a route in Flask, you use the `@app.route()` decorator followed by the URL pattern that you want to match. For example, if you want to handle requests to the root URL of your application, you would use `@app.route('/')`.
- You can also include variables in your URL patterns by enclosing them in angle brackets (`<>`). These variables can then be passed as arguments to your view functions. For example, `@app.route('/user/<username>')` would match any URL that starts with `/user/` followed by a username variable.

```python
@app.route('/')
def index():
    return 'Index Page'

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

Flask Url Routing

- Flask also provides a way to generate URLs dynamically based on the names of your view functions and the arguments that they expect. This is useful for creating links between different pages of your application without hard-coding the URLs.
- To generate a URL in Flask, you use the `url_for()` function, passing in the name of the view function and any arguments that it expects. For example, `url_for('user_profile', username='john')` would generate a URL for the `user_profile` view with the `username` argument set to `'john'`.

HTTP Methods

- Flask has 5 main HTTP Methods
  - POST : Used to send HTML form data to the server. The data received by the POST method is not cached by the server.
  - GET : The most common method. A GET message is send, and the server returns data
  - HEAD : Same as GET method, but no response body.
  - PUT : Replace all current representations of the target resource with uploaded content.
  - DELETE : Deletes all current representations of the target resource given by the URL.

Static Files

- Static files are an essential part of any web application. They include images, stylesheets, JavaScript files, and other assets that are used to enhance the user experience. In Flask, serving static files is a breeze thanks to its built-in support for static file routing.
- By default, Flask will look for static files in a folder named **'static'** located in the root directory of your application. You can change the location of this folder by passing the '**static_folder**' parameter when creating your Flask application instance. Additionally, you can use the '**url_for**' function to generate URLs for your static files, making it easy to reference them in your HTML templates.

Templates

Templates are files that contain static data as well as placeholders for dynamic data.

A template is rendered with specific data to produce a final document.

Rendering Template

Flask provides a **render_template()** function that makes it easy to render templates. This function takes the name of the template file as its first argument, followed by any necessary data to be passed to the template. The function returns a complete HTML page that can be sent back to the client.

```
from flask import render_template

@app.route('/hello/')
@app.route('/hello/<name>')
def hello(name=None):
    return render_template('hello.html', name=name)
```

Let see a basic example

```
<!DOCTYPE html>
<html>
  <head>
    <title>CBIT App</title>
  </head>
  <body>
    {% if name == 'CBIT' %}
      <h1>{{ 'CBIT ' }}</h1>
    {% else %}
      <p>{{ name }}</p>
    {% endif %}
  </body>
</html>
```

Request Object

- In web development, a request object is typically an object that contains information about an incoming HTTP request made to a web server.
- This object includes details such as the request method (e.g. GET or POST), URL, headers, query parameters, and form data.
- Developers use the request object to retrieve data from the client and perform the appropriate actions based on the data received.

```
from flask import request
```

Let see a basic example

```
@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)
```

Thank you