B2 - Introduction to Web Development

B-WEB-200

Bootstrap

EPyTodo

1.4

Bootstrap

group size: 1
repository name: epytodo_bootstrap_$YEAR
repository rights: ramassage-tek
language: python3

- Your repository must contain the totality of your source files, but no useless files (binary, temp files, obj files,...).

The main purpose of this bootstrap is to give you all the tools and basics for your project.
You really should ask an assistant or check on Epitech official communication channels if you do not understand this bootstrap.

- Table of contents
  - How to code into Python language
  - Installation directory
  - Creating views
  - Rendering with HTML - CSS - JavaScript
    - What is HTML
    - What is CSS
    - What is JavaScript
  - How to use all of those into the project
    - Example 1 - dynamic title and content
    - Example 2 - navigation list with statement
  - Using MySQL server
  - Using with Python

+ How to code into Python language

Python is easy ! Really ! You should look up these links (especially the first one):

- Norm (you can look at pycheckcode)
- Indentation is really important!
- Standard types (int / str / others)
- Structures (lists / dictionaries)
- Import statement

Some simple examples:

```
~/B-WEB-200> cat main.py
// basic types
a = 42
str = "hello"
print(a)
print(str)

// mixing both
mul_str = 3 * str
print(mul_str)

// structure type
tab = []
tab.append(3)
tab.append(6)
tab.append(9)
print(tab)

// function
def my_function(nb):
    if nb >= 0 :
        print("Positive number")
    else
        print("Negative number")
    return 0

my_function(42)
my_function(-21)
```

```
~/B-WEB-200> python main.py
42
hello
hellohellohello
[3, 6, 9]
Positive number
Negative number
~/B-WEB-200>
```

You can also use Python into interactive mode:

```
~/B-WEB-200> python
>>>a=42
>>>print(a)
42
>>>
~/B-WEB-200>
```

+ INSTALLATION DIRECTORY

To begin with, we'll install our project environment with:

- virtualenv (tool to create isolated Python environment / external link)
- pip (Python package manager / external link)
- Flask (micro web framework / external link)
- pymysql (Python package to connect to MySQL server)

Your project's architecture will look like this :

```
~/B-WEB-200> mkdir bootstrap_epytodo
~/B-WEB-200> cd bootstrap_epytodo
~/B-WEB-200> virtualenv . -p /usr/bin/python3
~/B-WEB-200> source bin/activate
~/B-WEB-200> pip install --upgrade pip
~/B-WEB-200> pip install flask
~/B-WEB-200> pip install pymysql
~/B-WEB-200> tree

.
|-- app
|   |-- __init__.py
|   |-- controller.py
|   |-- models.py
|   |-- views.py
|   |-- templates
|       |-- index.html
|   |-- static
|       |-- css
|           |-- index.css
|       |-- img
|           |-- favicon.ico
|           |-- banner.png
|       |-- js
|           |-- index.js
|-- config.py
|-- run.py

~/B-WEB-200>
```

The previous output does not include virtualenv directories nor the creation steps of your different files / directories

```
~/B-WEB-200> cat app/__init__.py
from flask import Flask


app = Flask(__name__)
app.config.from_object('config')
~/B-WEB-200> cat run.py
#!/usr/bin/env python3
from app import app


app.run()
~/B-WEB-200>
```

Alternative runs :

```
~/B-WEB-200> python run.py &
~/B-WEB-200>
```

```
~/B-WEB-200> chmod +x run.py
~/B-WEB-200> ./run.py &
~/B-WEB-200>
```

In another terminal, you can use **curl** to reach your URL. You'll get this message and that's normal!

```
~/B-WEB-200> curl 127.0.0.1:5000
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>404 Not Found</title>
<h1>Not Found</h1>
<p>The requested URL was not found on the server.  If you entered the URL manually
   please check your spelling and try again.</p>
~/B-WEB-200>
```

You can also use your favourite browser

Indeed, we didn't specify the rules to access our pages : let's continue to the next point!

+ CREATING VIEWS

Into **app** folder, create a **views.py** file.
In this file, we're defining all the routes which will be used and also the methods which are allowed (GET, POST, etc.).

- A route is a path which determines the location of your web application.
- A method is a way of sending or receiving data via HTTP (all methods are defined into RFC 2616, CHAPTER 9.1.2)

For example, see below :

```
~/B-WEB-200> cat views.py
from app import app


@app.route('/', methods=['GET'])
@app.route('/index', methods=['GET'])
def     route_home():
    return "Hello world!\n"


@app.route('/user/<username>', methods=['POST'])
def     route_add_user(username):
    return "User added!\n"
```

The route '**/**' is the same as **http://127.0.0.1:5000/**.
The route '**/index**' is the same as **http://127.0.0.1:5000/index**.
The route '**/user/marvin**' is the same as **http://127.0.0.1:5000/user/marvin**.
And so on.
When we'll reach one of these routes into our browser, some data may appear like plain text, html content or other content.
Once you're done with your views, you need to tell your app which routes you're using.
Here is the solution :

```
~/B-WEB-200> tail -n1 __init__.py
from app import views
```

Then, restart your server and reload your page into your browser. You'll get this time an "**Hello world!**" message.

+ Rendering with HTML - CSS - JavaScript

What is HTML

HTML is a markup language used by your browsers to display content.
Its structure is easy to understand as there are open and close markups in most cases.
Look at the documentation for full understanding.

What is CSS

CSS is a language to make your content lok beautiful. In the ".css" files, describe the behavior of your HTML elements (size, color, shape, etc.).
Look at the documentation for full understanding.

What is JavaScript

JavaScript is a language which allows you to create dynamic content.
The syntax is close to the one in C language but their usages are kind of different.
Look at the documentation for full understanding.

How to use all of those into the project

If you look closer at our project architecture, you can see unexplained folders like **templates** and **static**.
Those folders are used by the template engine : Jinja2.
To make it simple, this is an engine that helps you render your HTML contents (Documentation).

EXAMPLE 1 - DYNAMIC TITLE AND CONTENT

```
~/B-WEB-200> cat index.html
<!DOCTYPE html>
<html lang="en">
<head>
    <title>{{ title }}</title>
</head>
<body>
    <div>
        {{ myContent }}
    </div>
</body>
</html>

~/B-WEB-200>
```

```
~/B-WEB-200> cat views.py
from flask import render_template


@app.route('/', methods=['GET'])
def     route_index():
    return render_template("index.html",
                           title="Hello World",
                           myContent="My SUPER content!!")


@app.route('/user/<username>', methods=['GET'])
def     route_user(username):
    return render_template("index.html",
                           title="Hello " + username,
                           myContent="My SUPER content for " + username + "!!!")

~/B-WEB-200>
```

EXAMPLE 2 - NAVIGATION LIST WITH STATEMENT

```
~/B-WEB-200> cat index.html
<!DOCTYPE html>
<html lang="en">
<head>
    <title>{{ title }}</title>
</head>
<body>
    {% set navlist = [
        ('/', 'index', 'Index'),
        ('/products', 'products', 'Products'),
        ('/account', 'account', 'Account')
    ] -%}
    <nav>
        <ul>
        {% for href, id, caption in navlist %}
            <li><a id="{{ id|e }}" href="{{ href|e }}">{{ caption|e }}</a></li>
        {% endfor %}
        </ul>
    </nav>
    <p>
        Some content
    </p>
;
</body>
</html>

~/B-WEB-200>
```

Now that we can run our server, display some content, we're going to use a database to handle data.

+ USING MYSQL SERVER

To install your MySQL server, refer to the official documentation
To configure your MySQL server, refer to the official documentation
Queries examples can be found here

+ USING IT WITH PYTHON

Earlier, we talked about **pymysql**. Now we're going to use it.
To do so, let's take a look at the following example (simplified version with comments):

```
~/B-WEB-200> cat views.py
from flask import jsonify
import pymysql as sql


@app.route('/user')
def     route_all_users():
    result = ""
    try:
        ## We're creating connection between our mysql server and our app
        connect = sql.connect(host='localhost',
                              unix_socket='path_to_our_mysql_socket',
                              user='_user',
                              passwd='_password',
                              db='name_of_your_database'
                              )
        ## We're retrieving a "pointer" aka "cursor" to our database
        cursor = connect.cursor()
        ## We're executing a SQL command,
        ## assuming that all tables are already created
        cursor.execute("SELECT * from user")
        ## We're retrieving all results
        result = cursor.fetchall()
        ## We're closing our cursor and our connection
        cursor.close()
        connect.close()
    except Exception as e :
        print("Caught an exception : ", e)
    ## We're sending the data
    return jsonify(result)

~/B-WEB-200>
```

NEVER DO THAT for your project (there's no MVC architecture, neither configuration variables in this example!)