Python Web Framework

Carlos Matheus Barros da Silva[1]

Igor Bragaia[1]

Prof. Lourenço Alves Pereira Jr

I. Objective

This project aimed to manipulate HTTP requests on the Application Layer using Python. The project has been made for the Second Bimester Trail of CES35. To manipulate the HTTP requests, the original objective was to create a Python Web Framework, It was conducted following the Rahmonov's tutorial[1][2][3]. The Python Web Framework was made to support GET, POST, PUT, DELETE requests, and also supported routing.

After the initial development, we found out that the core implementation that interacts directly with the HTTP request is the Web Server Gateway Interface (WSGI), therefore the project core migrated to the WSGI developed.

The initial activity schedule and planning was:

There won't be a production deploy of the webserver, just localhost.

Activity schedules:

- Week 4: Activity planning
- Week 5: Study the implementation of the Python Web Framework.
- Week 6: Development of basic framework, supporting, at least, HTTP GET requests.
- Week 7: Development continuation and finish the presentation content.
- Exam Week 1: Creation of final presentation material, including simulation server/client on an application example that uses the developed framework.

[1]Computer Engineering Bachelor Student of ITA

II. Project development

What was developed is a Web Framework written in Python running with WSGI also developed by us. The Web Server Gateway Interface (WSGI) is an stadart interface to connect python web frameworks to a server connection socket and interchange data between the socket and the server framework.

The main idea is that the WSGI creates the HTTP server and provides it an HTTP handler. The HTTP server initiates a socket on that port and starts to listen to an HTTP request. Once it gets a request it passes it to the HTTP handler. The HTTP handler implements the package treatment according to the HTTP protocol, it sets up and read the message headers and handle errors. After that, the handler passes all that treated headers and the package to the server framework itself, from that point it is the server web framework responsibility what will be done regarding that message. After the framework execute what it is to execute to that message, the framework calls the WSGI back with the response message and the WSGI delivers that message back to the client.

This interaction between client, WSGI server and web framework application can be seen in Figure 1, it represents a sequence diagram of that interaction.

The whole interaction between the classes can be seen in Figure 2, it shows the project classes, interactions, and hierarch.

III. Project validation

In order to make everything work, both the Python Web framework server and the WSGI were successfully developed and they can be accessed by the *Github Repository*[2]. The usage process can be followed by the repository *Readme* file.

In order to test the code follow the steps:

- Clone the repository[2].
- Make sure to have python3 installed.
- Go to the project folder.
- Install the project dependencies: “pip3 install -r requirements.txt”
- Run the server python file: “python3 server.py”

[2]https://github.com/CarlosMatheus/Python-Web-Framework

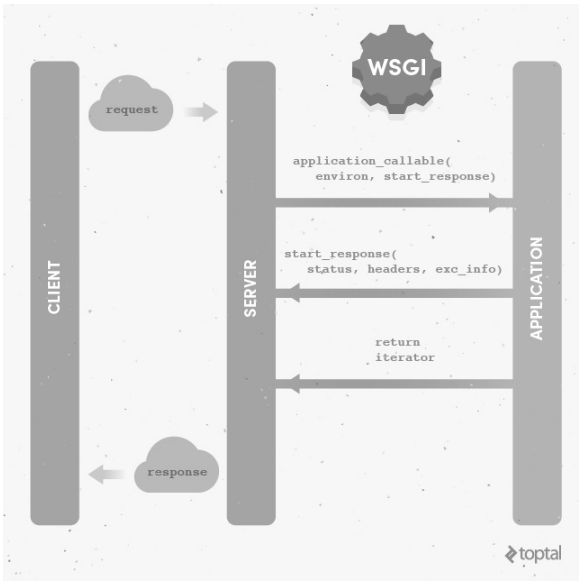

Fig. 1. WSGI Sequence diagram. The Client sends a message that when it arrives on the host computer it goes to the WSGI HTTP server socket. The WSGI process the message and passes it to the Web Framework application. The web framework processes the message and sends the response message to the WSGI server. The WSGI sends the message back to the client through its HTTP socket.

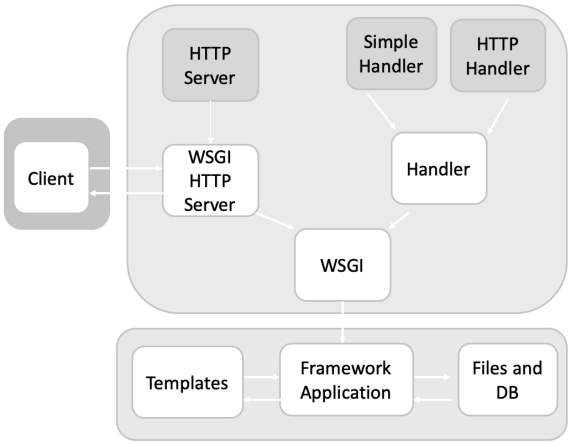

Fig. 2. Project Class architecture. The Classes HTTP Server, Simple Handler, HTTP Handler are from standard Python libraries and they perform basic HTTP protocol operations, therefore they were not implemented for the scope of this project and some classes used them or inherited from them.

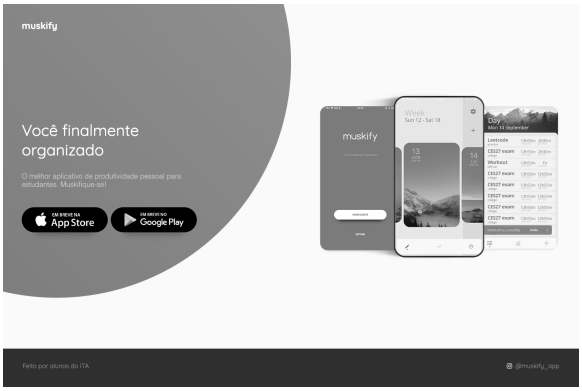

Fig. 3. Example HTML page with embedded CSS and with images files referencied.

- Now you can access the server by “http://127.0.0.1:8080”
- On the example there are the /, /about, and /put_test_route routes.
- The / route returns a simple text answer with any HTTP method you send to it.
- The /about route returns an HTTP web page as an answer to any HTTP method to that route.
- The /put_test_route route tests the PUT HTTP method and give a different response to the PUT method and it accepts a query. An example would be as shown on Code 3.

The project also supports HTML files. You just have to put them on the “templates” folder as done on the download repository example. As an example, by going to the /about the route it is got an HTML website as shown in Figure 3.

To summarize all that information it is possible to follow the usage of the Web Framework on the example Code 2. The usage is simple, just import the Framework and define the routers function using the route decorator as seen on Code 1, and then call the start server method.

```
@server.route("/about")
```

Code 1. Example of routing decorator.

```
from web_framework import WebFramework

server = WebFramework()


@server.route("/")
def home(request, method, query):
    # Simple response test
    return "Hello from the HOME page"
```

```
@server.route("/put_test_route")
def put_test_route(request, method, query):
    # Test different methods
    if method == server.methods.put:
        print('See that you are in fact using a PUT method')
        print(method)
        print('Received query:')
        print(query)
        return "Hello from the put_test_route page"
    else:
        return "Error: not put method"


@server.route("/about")
def about(request, method, query):
    # Test usage of HTTP templates
    return server.open_template('index')


server.start_server()
```

Code 2. Example server file that uses our Python Web Framework

```
http://127.0.0.1:8080/put_test_route?test_query
```

Code 3. Test PUT query

IV. Conclusion

A. Discussion

The project was developed with a good level of deepness both on WSGI implementation and on Web framework implementation. It surely could be improved in a lot of ways, mostly on the web framework side, since it could have better support to CSS files, integration to databases with Models scheme and a lot of other features expected from a web Framework. From the side of the WSGI, it would be possible to implement it deeper and depend less on Python libraries, it would fall on developing many aspects of the HTTP protocol. As future improvements, it could be developed better support for HTML templates in general from the Web Framework side, mostly regarding a better integration with different import files like CSS. Today only images are imported correctly.

B. Conclusion

The work was quite enlightening when it comes to understanding HTTP servers, HTTP handling, WSGI implementation, and the overall workflow of a Web Framework. The results were expresive since the whole project was developed from scratch and in the end, there is a fully functional WSGI connected with a functional Web Framework.

REFERENCES

[1] Jahongir Rahmonov. *How to write a Python web framework. Part I.* Feb. 2019. URL: https://rahmonov.me/posts/write-python-framework-part-one/.

[2] Jahongir Rahmonov. *How to write a Python web framework. Part II.* Feb. 2019. URL: https://rahmonov.me/posts/write-python-framework-part-two/.

[3] Jahongir Rahmonov. *How to write a Python web framework. Part III.* Mar. 2019. URL: https://rahmonov.me/posts/write-python-framework-part-three/.