Assignment 2

Web Architectures 2018-2019

Nicolò Pomini

September 26, 2018

1 Introduction

The goal of this assignment is to discover the potentialities of the cgi-bin functionalities, setting up a specific URL that makes as alias to a local folder, in order to execute the scripts contained in the latter as they were web scripts. In particular, two main scripts are requested:

- one that lists all the files in the *aliased* directory;
- the other that handles the requests of a form in a web page and launches an external java program that has to print the parameters received in the request.

2 Explanation

The problem was solved in several steps.

First of all, the Apache Web server was configured to enable the cgi-bin functionality, in order to let it listen for a specific request (`localhost/active`) for launching a script in a particular directory. In fact this functionality allows to associate an URL with a local folder, and all the requested files are mapped in the associated folder. For instance, if the URL `/active` is associated with the folder `/usr/local/cgi-bin`, the request `/active/script.sh` launches the script `/usr/local/cgi-bin/script.sh`.

Secondly, the script `lista.sh` was created, and its role is to print all the files in its directory (a sort of `ls` bash command).

Thirdly, a java program that produces an HTML page was written. It takes GET and POST parameters as arguments (first and second argument respectively) and prints them as a list.

Finally, a static web page with two forms and its handler – a bash script – were created. The script simply forwards the GET and POST parameters launching the java program and giving to the latter those parameters as arguments.

3 Implementation

As regards the server configuration, several edits in the `httpd.conf` file were needed. First of all, a `ScriptAlias` directive was added, to let the server to make an URL as alias of a directory. In particular, the following directive was added inside the `IfModule alias_module` tag:

```
ScriptAlias /active /ass2/scripts
```

where `/active` is the URL we want to listen to and the rest is the path of the folder containing the scripts. After that, the permissions for the directory were given, adding the following lines:

```
<Directory "/ass2/scripts">
    AllowOverride None
    Options None
    Require all granted
</Directory>
```

Finally, the handlers for the scripts were provided, adding the string

```
AddHandler cgi-script .cgi .sh
```

inside the `IfModule mime_module` tag.

As regards the `lista.sh` script, it is a simple loop that prints every entry found in the directory, separated with a new line character. A very important thing for the bash scripts is that they have to start with the string `"#!binbash"`, to be executed by the bash interpreter. Furthermore, they have to print as a first thing a HTTP header, to let the server to treat them as web scripts. The header is the following:

```
Content-type: text/html

/*Rest of the output of the script*/
```

It is very important that an empty line divides the header to the content of the output.

The java program that produces an HTML page to show the received parameters expects the GET query as the first argument, or the POST query as the second argument. Precisely, in case of a GET request it expects only one argument, the whole GET query, while in case of a POST request two arguments are expected: the first one is ignored, and the second contains the whole POST query. This program prints an HTML page containing a list, with one item for each received parameter. Each query string has the form

```
par1=val1&par2=val2&...&parN=valN
```

The following function analyzes these kinds of strings, splitting them when the '&' character is met, and for each substring it prints the parameter name and its value.

```
public static void parseArgs(String p) {
    System.out.println("<ul>");
    String[] vars = p.split("&");
    for(int i = 0; i < vars.length; i++) {
        System.out.println("<li>");
        String[] param = vars[i].split("=");
        if(param.length == 1)   //empty param
```

```
            System.out.println(param[0] + ": No value passed for this attribute");
        else
            System.out.println(param[0] + ": " + param[1]);
        System.out.println("</li>");
    }
    System.out.println("</ul>");
}
```

This function can handle a query of any length, and in case some parameter is empty it prints "*No value passed for this attribute*", as can be seen in Figure 4.

The static web page is very simple, and it contains two forms: one that uses the GET method, which sends two strings; the other one that uses the POST method, sending one string.

Finally, the handler script receives the request of the static web page, prints the HTTP header and launches the java program, passing the GET or POST query as argument.

4 Deployment

To deploy the project, every bash script has to be made executable, using the command

```
chmod +x script_name
```

After that, the java program has to be compiled, and a jar file is created using the following commands:

```
javac HTMLPrinter.java
jar cfe htmlprinter.jar HTMLPrinter HTMLPrinter.class
```

All the bash scripts are contained into a folder called `scripts`, because the `ScriptAlias` directive expects that all the files contained into the folder are executable files for which the handler was provided in the `httpd.conf` file.

Furthermore, the path of the folder has to be copied into the `ScriptAlias` directive and in the `Directory` tag of the `httpd.conf` file. Precisely, let `PATH` be the path of the project folder, the directives of the `httpd.conf` file should look like:

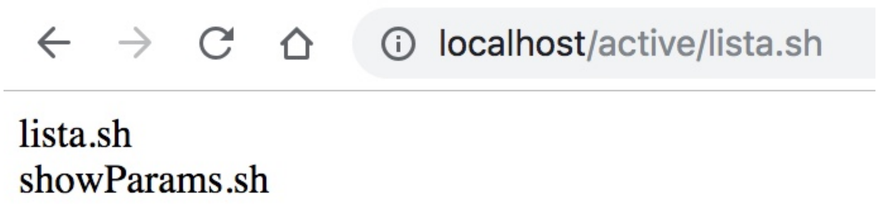
```
ScriptAlias /active PATH

<Directory "PATH">
    /*List of permissions*/
</Directory>
```

Finally, the Apache Web server is started with the command:

```
sudo apachectl start
```

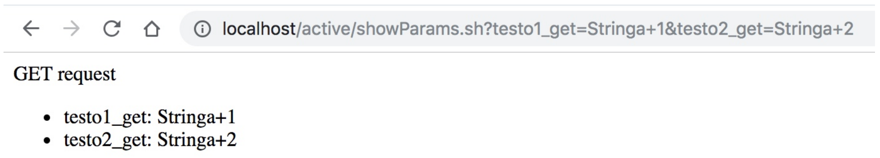
The file `home.html` placed inside the project folder is the static HTML page that contains the two forms to send data in GET or POST method. It's enough to open it and to insert some data and click the send button.

As regards the results of the app running, the output of the `lista` script can be seen in Figure 1. This output is generated requesting the `localhost/active/lista.sh`.



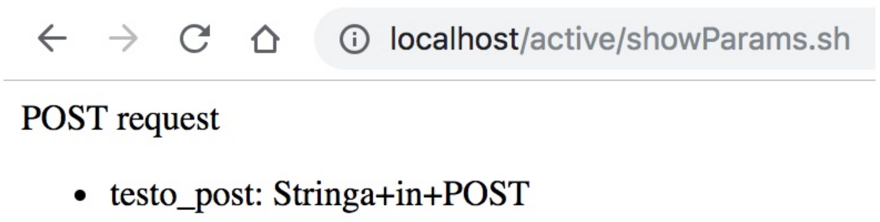
```
localhost/active/lista.sh

lista.sh
showParams.sh
```

Figure 1: The output of the lista.sh script.



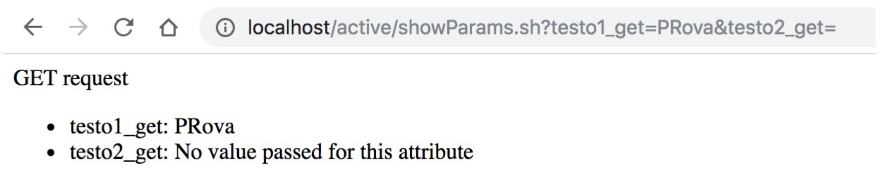
```
localhost/active/showParams.sh?testo1_get=Stringa+1&testo2_get=Stringa+2

GET request

  • testo1_get: Stringa+1
  • testo2_get: Stringa+2
```

Figure 2: The output with a GET request.



```
localhost/active/showParams.sh

POST request

  • testo_post: Stringa+in+POST
```

Figure 3: The output with a POST request.



```
localhost/active/showParams.sh?testo1_get=PRova&testo2_get=

GET request

  • testo1_get: PRova
  • testo2_get: No value passed for this attribute
```

Figure 4: The output with at least an empty parameter.

As regards the GET and POST requests generated by the static HTML page, three possible outputs are visible in Figure 2 (GET request), Figure 3 (POST request) and Figure 4 (GET request with an empty parameter).

5 Comments and notes

I faced several problems with the setup of the `cgi-bin` function of the server. In fact, originally the folder was placed in my home directory, and there was an issue with permissions. The result was a `403 Forbidden` error displayed in the browser, with the following message on the error log: *access denied because search permissions are missing on a component of the path.* For this reason I placed the folder in the root directory, and everything worked fine.