Department of Information Engineering and Computer Science
Fintech
University of Trento

Assignment 1
17 October 2022

Index

The case

A little sum-up of the assignment.

1. The first part consists in a web service that accepts a string as a parameter and returns the reversed version of it.
2. The business code needs to be written and executed by an external program.
3. The second part consists in a shell script that is called by an Apache Web server. The script should after call the business code written in the first part.

My solution

1. Since no framework has been used the assignment guarantees the maximum flexibility. I've been so able to define a custom structure to implement routes.

Listing 1: User.java

```
public class Route {
    public String path;
    public String resource;
    public ProcessRunnable task;

    public Route(String path, String resource, ProcessRunnable task) {
        this.path = path;
        this.resource = resource;
        this.task = task;
    }
}
```

This class allows a simple a straightforward definition of all the route available to clients. For each one can be defined a *path*, a *resource* file (if needed), and a callback that follows a *Runnable* pattern. This allows the execution of custom code for each route.

Listing 2: Route registration

```
routes.add(new Route("", "index.html", (Map<String, String> params, OutputCallback callback) -> {
    callback.onSuccess(null);
}));
routes.add(new Route("favicon.ico", "favicon.ico", (Map<String, String> params, OutputCallback callba
    callback.onSuccess(null);
}));
routes.add(new Route("process", null, (Map<String, String> params, OutputCallback callback) -> {
    try {
        if (params.get("par1") == null) throw new Exception("Invalid␣parameters.");
        String output = CmdProcessBuilder.execute(new String[]{params.get("par1")});
        callback.onSuccess(output);
    } catch (Exception e) {
        System.out.println("Error");
        e.printStackTrace();
        callback.onError(e.toString());
    }
}));
```

In the last defined route in Listing 2 we can see how the process endpoints check for the correct path and parameters naming.

2. Looking at the sort of middleware that acts before redirecting to the matching route, we can see how the parameters are individuated aside of the URI tokens. They are afer available in a simple key-value map.

Listing 3: Catching the URI parameters

```
Map<String, String> params = new HashMap<>();
if (req.contains("?")) {
    String rawArgs = req.split("\\?")[1];
    req = req.split("\\?")[0];
    String[] rawParams = rawArgs.split("&");
    for (String param : rawParams) {
        params.put(param.split("=")[0], param.split("=")[1]);
    }
}
```

3. In the listing we can see how the matching route is find, if exists, and how the custom code is executed.

Listing 4: SharedData.java

```
Route route = this.getRoute(req);
if (route == null) {
    this.onErrorResponse();
    return;
}

route.task.run(params, new OutputCallback() {
    @Override
    public void onSuccess(String output) {
        onSuccessResponse(route, output);
    }

    @Override
    public void onError(String err) {
```

```
            onErrorResponse(route);
        }
    });
```

4. The HTML response is build as simple as it follows:

Listing 5: HTML building

```
private byte[] buildHtml(String output) {
        String res = "";

        res += "<!DOCTYPE␣html>" +
                "<html␣lang=\"en\">" +
                "<head␣style=\"text-align:␣center\">" +
                "␣␣␣␣<meta␣charset=\"UTF-8\">" +
                "␣␣␣␣<title>Assignment␣1</title>" +
                "</head>" +
                "<body␣style=\"text-align:␣center\">";

        res += "<h3>Output:␣" + output + "</h3>";

        res += "<br><br><br><br>" +
                "</body>" +
                "</html>";

        return res.getBytes();
    }
```

Each wrong request, with missing or wrong parameters, are handled correctly handled by *onErrorResponse()* method.

The second program, written always in Java, that revers the string is simple as it follows:

Listing 6: HTML building

```
public class ReverseString {
    public static void main(String[] args) {
        if (args[0] == null) return;
        System.out.println(new StringBuilder(args[0]).reverse());
    }
}
```

And showing how the second part of the assignment is implements it follows the bat code:

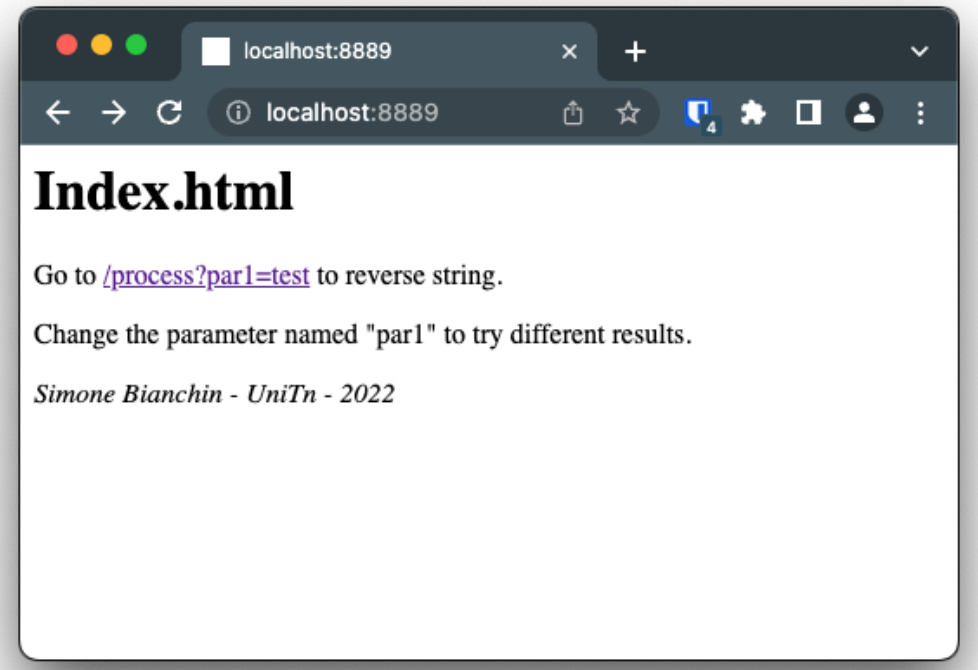
Listing 7: service.bat

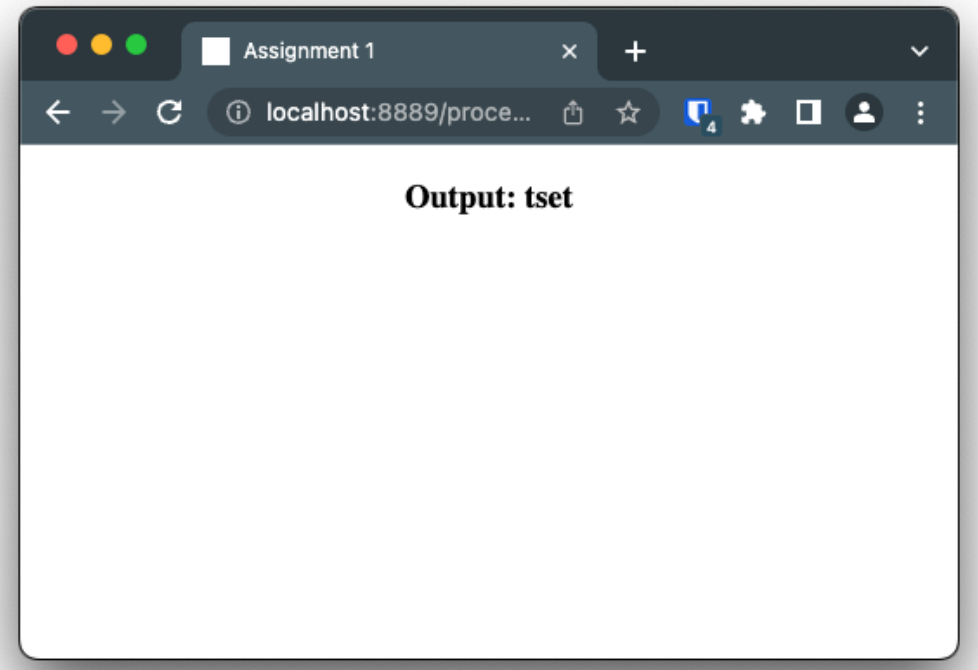
```
@echo off
echo Content-type: text/plain
echo.

for  %%k in (%QUERY_STRING%) do set ARGS=%%k
for /f %%k in ('java␣-jar␣\path\to\jar\processes.jar␣%ARGS%') do set res=%%k
echo %res%
```

The bat file is correctly put in *cgi-bin* folder provided by Apache. And with the correct tuning of the Apache configuration file binary files like this can be requested by clients and executed.

App running

Here some screenshots of the app running. There hasn't been the slightest effort to implement a better UI since the bare business code was taking too much time but the result is at least functional.





Comments

1. The assignment gave me the opportunity to see and test how, under the hood, the HTTP requests are handled. It feels like a more structure way to define a dynamic HTTP response is required. I think that the next assignment will gives this opportunity.