Assignment 4
Web Architectures 2018-2019

Nicolò Pomini

October 11, 2018

1 Introduction

The goal of this assignment is building a Java web application that has a simple login procedure, and that shows some information (in this case the current date and the current time) only to logged-in users. The session handling has to be done using a cookie, and in case a not logged-in user tries to access the date and time pages, the system has to redirect her to the login page.

In particular, the application has:

- a home page, which displays a link for the date page and for the time page;
- a date page and a time page, which show the current date and the current time respectively;
- a login page, to let the user log into the system;
- a logout function, to let the user log out of the system;
- a filter that intercepts all pages except the home page, checking if a valid cookie is available, and if it is not redirects to a login page.

Furthermore, in case a user is logged-in, every page has to display the username.

2 Explanation

The web application is designed and developed following the MVC pattern. In particular:

- the *model* part is composed only by a XML file that stores usernames and passwords;
- the *view* part is a set of `jsp` files, which have several roles – do the home page, output the information (date and time), and greet the user if the latter is logged-in, otherwise to link the login page;
- the *controller* part is done by a couple of `servlets`, which handle the login and logout procedure.

In addition to that, a `filter` acts as a sort of *proxy*, intercepting all the requests to the date and time pages and checking whether a user is logged into the system. If it is not the case, the user is redirect to the login page.

3 Implementation

First of all, the web application can have mainly two statuses:

- no user is logged-in;
- a user is logged-in.

To understand which is the current status, a cookie is used: when a *valid* cookie is set, a user is logged-in, otherwise no. A cookie can be seen as a pair $< key, value >$ of strings: in this application the key is the arbitrary string `"ass4_cookie"`, while the value is the username. In this scenario, the cookie is *valid* if and only if its key is equal to `"ass4_cookie"`. A cookie has also an expire time, after which it is not valid anymore: in this application it has the default duration, so they are valid as long as the browser exits, or as long as the user logs out.

As regards the *model* and the *view* parts there are very few things to say. The XML files that stores all the username and passwords is called `users.xml`, and it is placed in the `WEB-INF` directory, in order not to be directly accessible with an HTTP request, but being anyway accessible by the *controller* part. The structure of the file is very simple: on the top there is a root tag called `users`, made by many nodes (one for every existing user) called `user`. The latter are so structured:

```
<user>
    <name>username</name>
    <password>user password</password>
</user>
```

The `jsp` files that compose the *view* part are easy to understand, and they simple show what they have to show: the current date for the date page, the current time for the time page, a login form for the login page, etc. The only thing worthy of notice is an auxiliary `jsp` file that prepares the portion of HTML to greet the user: this file checks whether a *valid* cookie is set, and in this case prints the username and a link to the logout; otherwise it prints a link to the login. The latter page is included by any other `jsp` file, using the `jsp:include` directive.

As regards the *controller* part, two servlets are implemented: one that takes care of the login and one for the logout.

The login servlet, called `LoginServlet.java`, handles POST requests, expecting an username and a password to log the user into the system. Then it reads the `users.xml` file, scanning every entry looking for a corresponding username and password. In case a user is found, a cookie $<$ "*ass4_cookie*", *username* $>$ is created and added to the response. Finally, a redirect to the home page is sent. The following code analyzes the XML file and sets the cookie:

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
try {
    DocumentBuilder builder = factory.newDocumentBuilder();
    InputStream input = getServletContext().getResourceAsStream("/WEB-INF/users.xml");
    Document doc = builder.parse(input);
    NodeList nodes = doc.getElementsByTagName("user"); //list of users
```

```
    Cookie c = null;
    for (int i = 0; i < nodes.getLength(); i++) {   //analyze every user entry tag
        String tagName, tagPassword;
        String[] userDetails = nodes.item(i).getTextContent().split("\\s+");
        tagName = userDetails[1];
        tagPassword = userDetails[2];
        if(tagName.equals(username) && tagPassword.equals(password)) {
            //login
            c = new Cookie("ass4_cookie", tagName);
            response.addCookie(c);
            break;
        }
    }
    //in any case, redirect to the home page
    response.sendRedirect("index.jsp");
} catch (ParserConfigurationException | SAXException ex) {
    System.err.println("Something went wrong during XML parsing");
}
```

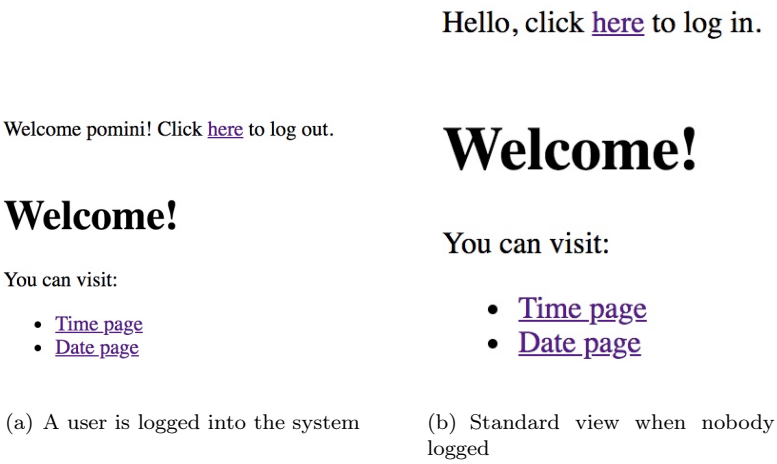
The logout servlet, called `Logout.java`, is much simpler, it just scan every cookie looking for the one with the key equals to `"ass4_cookie"`, and deletes it, setting its `maxAge` to 0.

Both servlets are Java classes that extend `javax.servlet.http.HttpServlet` class.

Finally, the filter, called `LoginFilter.java`, is a Java class that implements `javax.servlet.Filter`. Likely the logout servlet, it scan every cookie looking for the one with the key equals to `"ass4_cookie"`, but this time not to delete it, but to ensure that a user is logged-in. In case it is, the request is forwarded to the original request of the user, otherwise the user is redirect to the login page. To make the filter works properly, the following entries have to be added to the `web.xml` file:

```
<filter>
    <filter-name>LoginFilter</filter-name>
    <filter-class>it.unitn.disi.webarch18.ass4.LoginFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>LoginFilter</filter-name>
    <url-pattern>/time.jsp</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>LoginFilter</filter-name>
    <url-pattern>/date.jsp</url-pattern>
</filter-mapping>
```

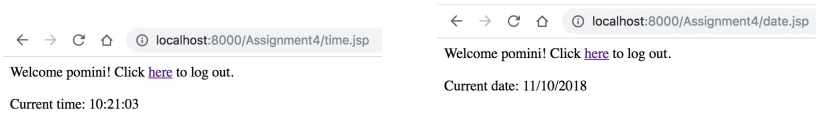
In this way all the requests for the time and date page are intercepted by the filter. It is not needed to intercept also the login or the home page, because the latter are public accessible.



(a) A user is logged into the system

(b) Standard view when nobody is logged

Figure 1: The two possible appearances of the home page.



(a) Time page

(b) Date page

Figure 2: Time and date page appearance.

4 Deployment

To deploy the project is necessary to import it into an IDE like NetBeans and then launch it from the IDE itself, or manually deploy the WAR file into Tomcat.

The home page can appear in two different ways, as shown in Figure 1, one in case of a non logged user, and the other when logged into the system. The difference is at the top of the page, on the *greetings* section: in case a user is logged her username is printed and the logout page is linked, otherwise there is a link to the login page.

The time and date pages have a very similar layout, and they are only visible to a logged-in user. They can be seen in Figure 2.

5 Comments and notes

This web application has big security problems. In fact, user data is stored into a file which has no access restrictions, and furthermore it is made of plain text, thus readable by every human or machine. In addition to that, all the requests are done with HTTP, so data is not encrypted. Finally, save the username into the cookie is not secure. Every web application running on the browser can read the cookies and know which is the user name.