Web Architectures
Assignment 2

Antonio Pio Padalino

October 16, 2022

Contents

1 Introduction

1.1 The project goal

The assignment concerns the development of a simple web game in which users can test their geography knowledge matching country flags to the respective capital. In order to start playing, authentication is necessary, and the webapp will keep track of the user's score throughout the session. If the admin logs in, they will be greeted with a control page displaying currently active users and their scores. The whole webapp has been developed following an MVC pattern based on Servlets, Beans and JSPs.

1.2 The project structure

The project makes use of a number of Java classes. In particular:

- `LoginServlet`, `RegistrationServlet`, `WelcomeServlet`, `GameServlet` and `AdminServlet` manage the different pages the user can browse;
- `LoginFilter` and `AdminFilter` make sure that an unauthenticated user can only access the login and registration pages, and that only the admin can visit the control page;
- `UserBean` define a JavaBean used to store user login information, score and status;
- `GameLogic` exposes a method `pickFlag()` used to randomly pick 3 flags among those placed in the `/flags` directory;
- `StringHasher` is simply used to calculate SHA256 on user's password to avoid storing it in plain text;
- `ServletContextListener` is used to perform actions at the startup and at the shutdown of the server;
- `ReaderWriter` is used to read and write serializable `UserBean` instances;
- and finally `ValidateInput` is used to double check server-side that the user's input in the game session is valid.

Aside from the business logic part, `.jsp` files[10] for the `login`, `register`, `welcome`, `game` and `control` pages deal with the presentation of the webapp, alongside `simple.css` which gives a nicer flavour to the whole application.

Finally, the usual `web.xml` file is used to:

- set a welcome page, namely the `/login` page;
- set the listener `ServletContextListener`;
- set the session timeout T, expressed in minutes;
- set the `LoginFilter` and the `AdminFilter`.

More on each component will be discussed later, in the Methods section.

2 Methods

2.1 Authentication

When starting the webapp the user is always greeted by a login page[5]. If it's their first time, they can easily go to the registration page and sign up by clicking on the link below the form. The access to any other page of the webapp, at this time, is restricted by `LoginFilter`[4], which checks for the

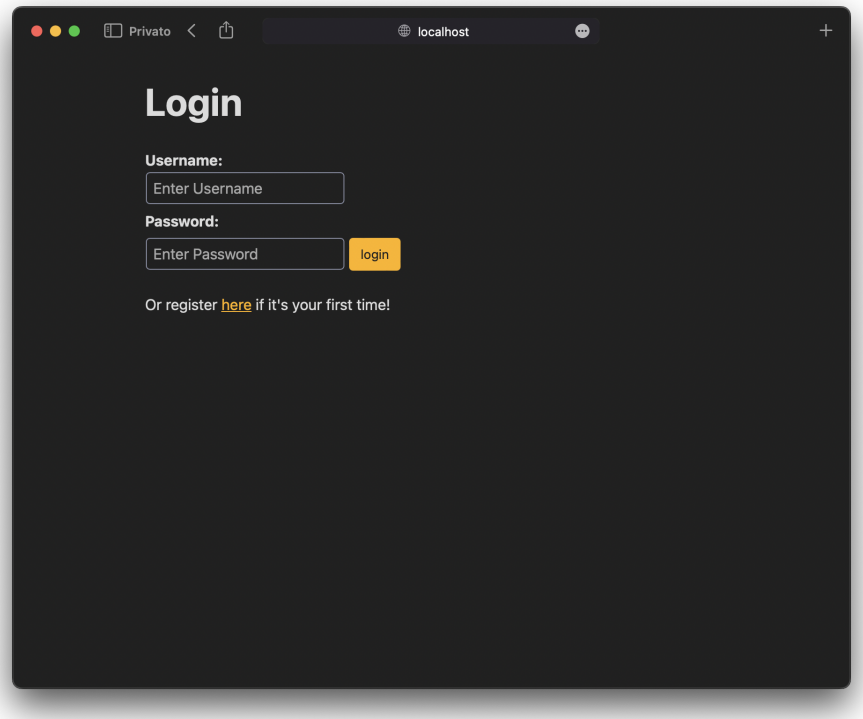

Figure 1: The login page the user is greeted with at the webapp startup.

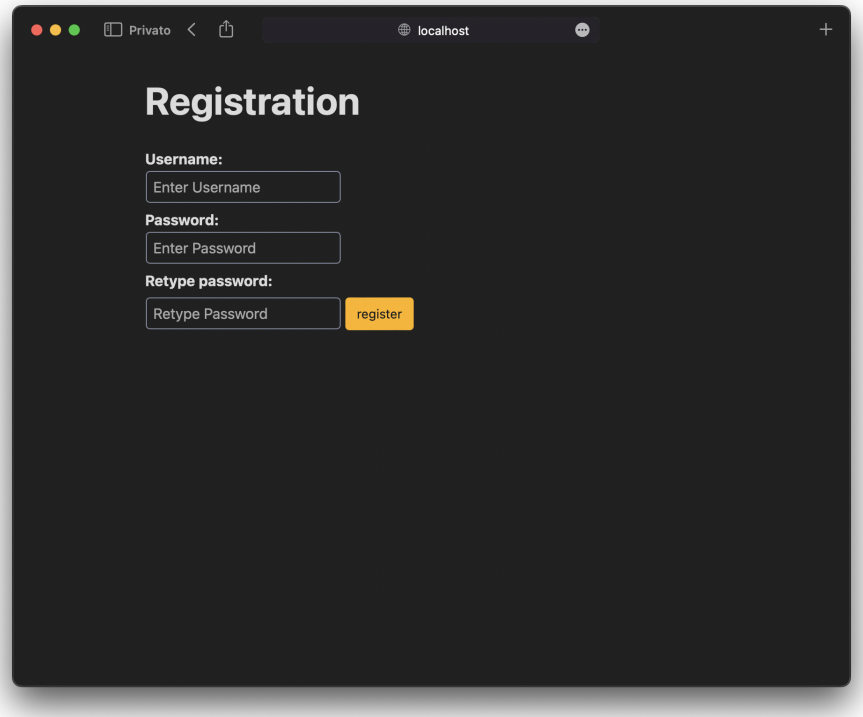

Figure 2: The registration page, used to create a new account.

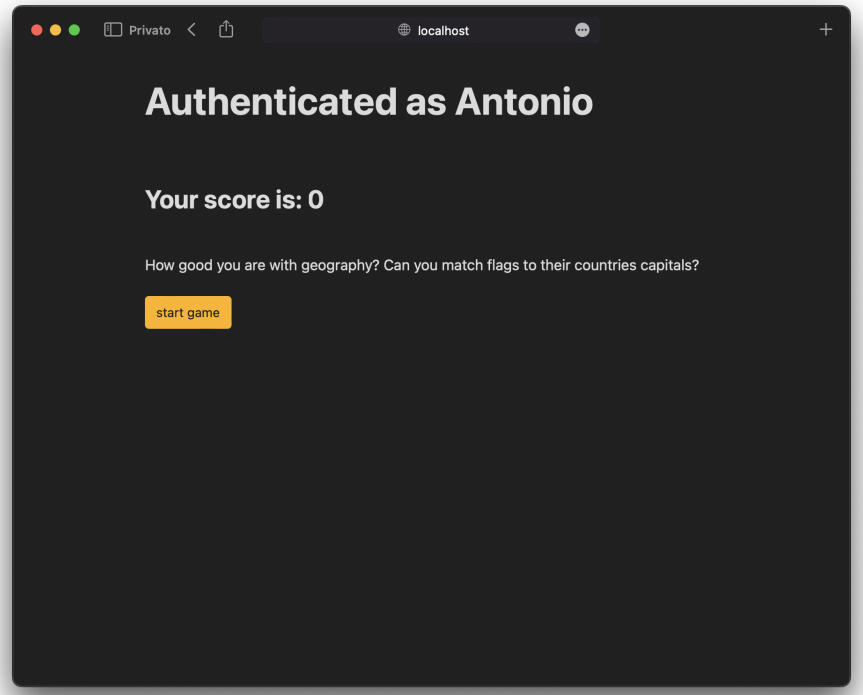

Figure 3: The welcome page, displaying a user's username and score.

user to be authenticated before directing them to the requested page. Below you can see the login and registration pages as they are presented to the user.

If the user authentication is successful, `LoginServlet` directs[2] them to a `/welcome` page, showing the name of the user and their current score (reset to 0 at login).

2.2 Handling User Beans

One of the crucial points of the project was to make an effective use of JavaBeans to store users' information and make it available in later usages of the webapp, for instance to log in existing users. This was made possible with the `UserBean` bean, which implements `Serializable` interface. As every JavaBean[6], `UserBean` also implements a no-argument constructor and provide methods to set and get the values of the properties, known as getter and setter methods. As for its properties, `UserBean` handles the following:

- `username`
- `password`
- `active`
- `sessionPoints`

While attributes `username` and `password` are pretty obvious, `active` is a boolean flag that is only true for users active in the current session, and `sessionPoints` keeps the score of the user (again in the current session). The `active` and `sessionPoints` attributes are, in fact, reset to `false` and 0 at each redeploy of the webapp. `UserBean` also implements a `.toString()` method, which can be used to print a user's name and current score. Notice how neither the `active` status nor the hashed password are considered by the method, being attributes that we don't ever want to display.

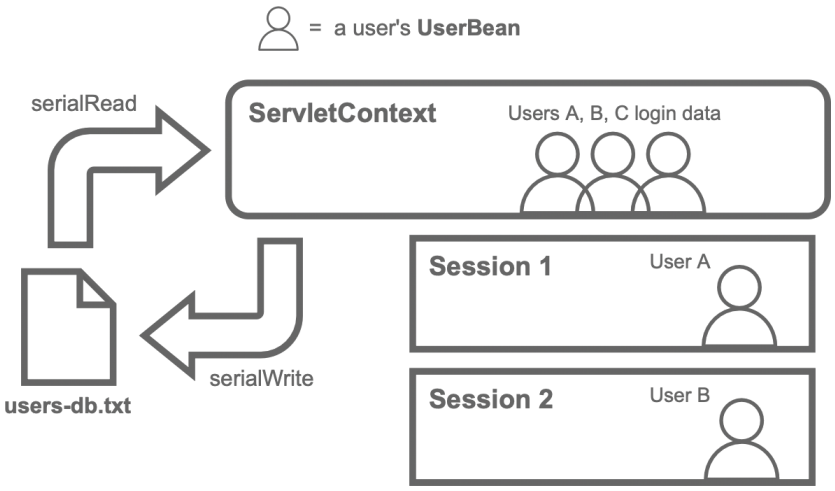

Figure 4: Representation of the pattern used to store users info among `ServletContext` and `Sessions`.

About the actual handling of user's data, the webapp works in the following way:

1. When a user registers (hence their inserted username & password are valid and not already registered) a new `UserBean` is created, also initializing the `Points` to 0 and `active` to `false` (the user registered, but still hasn't logged in!);
2. Such `UserBean` is then moved to the `ServletContext`, setting it to an attribute with the same name as the bean `username` for easier future usage;
3. Afterwards, at login time, if the username inserted in the login form is found as an attribute inside the `ServletContext` and the password matches, the `UserBean` is moved to the session;
4. Also, at this point, the status of the same bean inside the `ServletContext` is switched to active (so that the admin can see the user among the list of active ones) and its points are reset to 0;
5. At the webapp shutdown, the `ServletContextListener` comes into play, using `ReaderWriter.serialWriter()` to write each `UserBean` object inside the Context to a file;
6. When the webapp is started again, `ServletContextListener` will read the same file and put every `UserBean` object from the file to the `ServletContext` again, naming each bean after the corresponding username (as before);
7. In this way, the webapp will always "remember" the registered users' identities, even if data are actually mantained in the `ServletContext` only at runtime.

2.3 The game logic

The game basically consists of matching each country to the corresponding capital, after showing the user three randomly picked flags among the ten available. To this account, the `GameLogic` class implements a `randomPick()` method which provides three random picks among the items of a supplied list. This is used, inside the `doGet()` of `GameServlet`, to choose three random flags and passing their filenames as attributes of the session. In turn, the `game.jsp` page provides for the page presentation by connecting the flags filenames to their path and extension and presenting them to the user. Notice how the filename is first parsed with `.replace(" ","%20")` to account for spaces and encode them correctly[9].

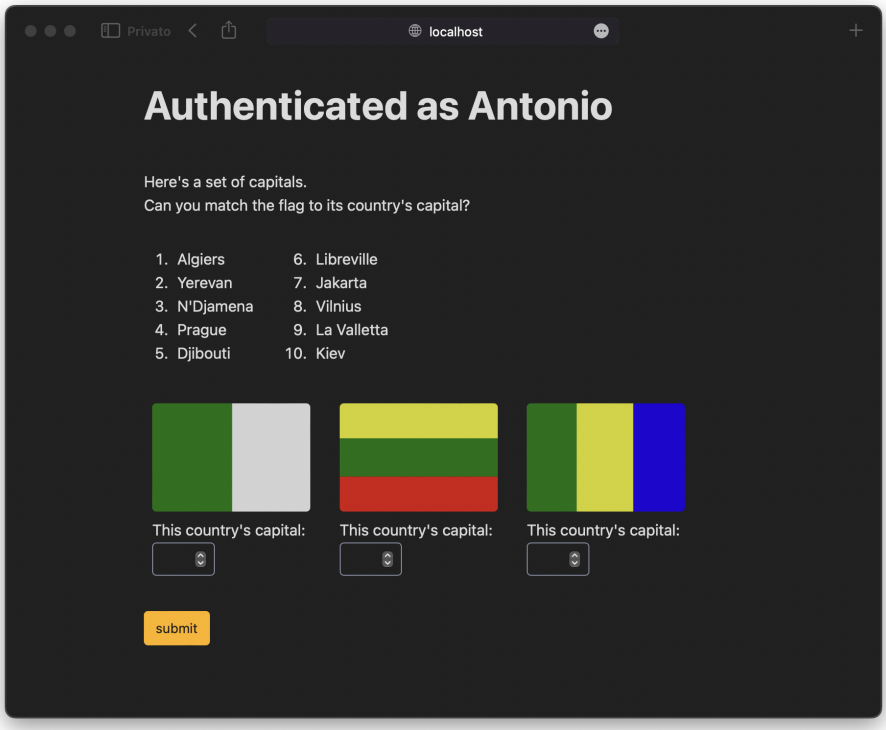

Figure 5: The game page, where the user is presented with three random flags.

Example.
If `Czech Republic` is chosen as first country, then with `("/flags/" + session.getAttribute("flag1") + ".jpg").replace(" ","%20")` the final filename will be `/flags/Czech%20Republic.jpg`, correctly showing the image in the final html.

Presented with the game page, the user can choose the correct numbers from the list of countries above and link them to the flags. The forms can only accept numerical inputs between 1 and 10 (our pool of choices). This check is carried out twice:

- *client-side*, `game.jsp` contains `input type="number" required min="1" max="10"`[7][8] in the declaration of all forms to prevent the user from inserting invalid input such as characters, null, or numbers outside of the range;
- *server-side*, the `ValidateInput` servlet takes the inserted parameters and applies the same checks as before, putting corresponding error messages inside a HashMap. Such messages are to be displayed next to the submit button, in the game page, as written in `game.jsp` (but that will hardly never happens, since the check is already performed earlier).

If the input is valid, `GameServlet` checks the user's answers, and assign them 3 points (if they are all correct) or -1 points (if at least one is wrong). In both cases, the user is sent back to the start page, showing the updated score. The check for the correctness of the answers is carried out quite easily: the available countries are defined in an (ordered, of course) array. The capitals the user can associate are also displayed in a fixed order, which matches the flags' one. So, if for the current game session country #2 Armenia is chosen and the user replies with #2 Yerevan it is easy to determine that they got that one match correctly. Of course, the user is blind to the order of flags.

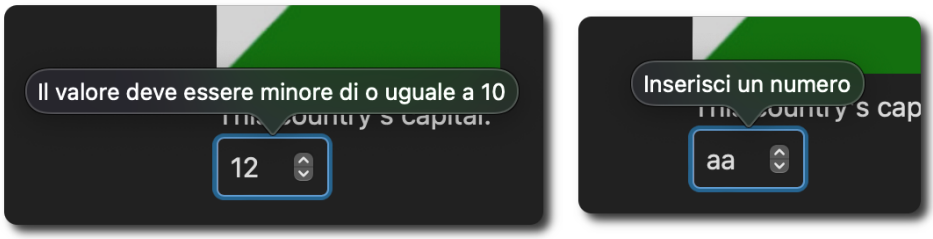

Figure 6: An example of input validation, done client-side inside the `game.jsp` page.

2.4 Admin page

Another functionality the webapp implements is a `/control` page, used to glance at all connected users and their current scores. This is realized by `AdminServlet`, which looks into the `ServletContext` for `UserBeans` objects with `active` attribute equals to `true`. Then such beans are represented with `.toString()` and presented right after `control.jsp`.
The `active` flag is used because admin's only way of accessing all users data is by looking inside `ServletContext`, which also contains data about user who registered in previous deployment of the webapp (to eventually log them in if they come back). So basically, `AdminServlet` needs a way to access Context data while distinguishing between active users (who logged in since startup) and inactive ones. To this account, each time users' data is read at startup, their `active` status is set to `false`.

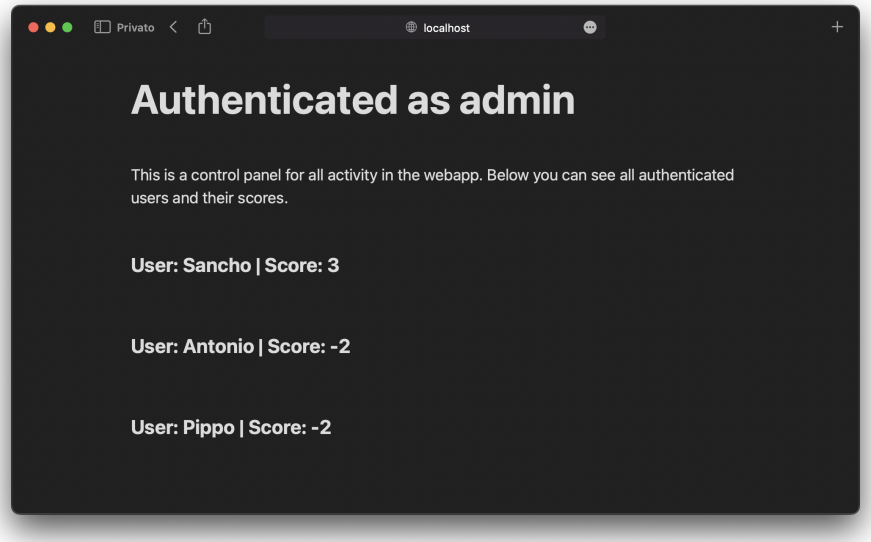

Figure 7: The admin control page, displaying all logged users and their scores.

Also notice that:

- the control page is also by filtered by `AdminFilter` preventing anyone who is not authenticated as admin to access the page, resulting in an `Error 401:Unathorized`;
- the `admin` user identity is unique and predefined: no other user can register using `admin` as username, and at login a check is issued right away to assert if the user put username: "`admin`" and password: "`nimda`"

3 Comments

3.1 Hashing passwords

Storing users' passwords in plain text would of course represent a serious security issue. For such reason, hashed[3] password are store instead of plain passwords, using a `StringHasher` class using a SHA256 algorithm. However, hashed passwords are not totally unique to themselves due to the deterministic nature of hash function: given the same input, the same output is always produced. This can give rise to a number issues, one of them being duplicate passwords by different users. To address the problem, in a further implementation of the webapp, a good practice that can be taken into account is *salting* the passwords.

3.2 Adding a .css

Although not being part of the evaluation, at the end of the assignment, a simple .css file has been applied to all `.jsp` pages to enhance the presentation side of the webapp. The original `simple.css` file used was found online[1] and edited to better fit the pages of the webapp and some parts of the presentation (such as the margins between usernames in the control page, or the alignment of the flags), also removing rules simply unnecessary for the project.

References

[1] github.com/kevquirk. *simple.css Github page by @kevquirk.* https://github.com/kevquirk/simple.css.

[2] initialcommit.com. *Forward() vs. SendRedirect() vs. Include().* https://initialcommit.com/blog/forward-vs-sendredirect-vs-include.

[3] stackoverflow.com. *How to hash some string with sha256 in java.* https://stackoverflow.com/questions/5531455/how-to-hash-some-string-with-sha256-in-java.

[4] www.javatpoint.com. *Servlet filter.* https://www.javatpoint.com/servlet-filter.

[5] www.studytonight.com. *Login system example in servlet.* https://www.studytonight.com/servlet/login-system-example-in-servlet.php.

[6] www.tutorialspoint.com. *JSP and JavaBeans.* http://www.tutorialspoint.com/jsp/jsp_java_beans.htm.

[7] www.w3schools.com. *HTML Input Attributes.* https://www.w3schools.com/html/html_form_attributes.asp.

[8] www.w3schools.com. *HTML input max attribute.* https://www.w3schools.com/tags/att_input_max.asp.

[9] www.w3schools.com. *HTML URL Encoding Reference.* https://www.w3schools.com/tags/ref_urlencode.ASP.

[10] www3.ntu.edu.sg. *Java Server-side Programming: Getting started with JSP by Examples.* https://www3.ntu.edu.sg/home/ehchua/programming/java/JSPByExample.html.