WEB ARCHITECTURES
Assignment 2

Riccardo Gennaro

16 October 2022

1 Introduction

1.1 Problem statement

This project aims at developing a web application capable of displaying dynamic pages satisfying the MVC framework. More specifically, the cited pattern has to be implemented using different Java API for servlets, JSPs pages and Java beans (see 1.2).

Regarding the application itself, it has to satisfy the following, shown synthetically, requirements:

- Role Base Access Control (RBAC). The application is accessible only by authenticated users. Users can be of three types:

  1. Admin. It can access the login page, registration page, and control page, this showing the active users and theirs score.
  2. Authenticated user. It can access the game, the login page and the registration page.
  3. Unauthenticated user. It can access only the login and the registration page.

  The Admin role is hardcoded, i.e. is specified in the source code and , therefor it is static. Registrated users information must be stored in file.

- Game. The application main logic must show a game in which the user must recognise the capital cities of three different nations chosen randomically between ten different options stored in a given folder containing ten .JPEG files of national flags. If the user rightly guesses all the capital cities, three points are added to it's score, otherwise, one point is subtracted. The user's score has session scope, meaning that it will not be persistance between different login sessions.

1.2 Model-View-Controller

This section wants to convey a basic idea of what the architecture of the application looks like.

The MVC pattern is implemented as follows:

- Model. It represents the application state. In our case, it consists of Java beans containing the data recovered at run-time from the persistent memory (in this case, a .txt file). The Java Bean model is accessed both by the View component (read), and by the Controller component (read/write).
- View. It defines the User Interface. The requirements asked the developer to use JSPs (Java Server Pages) to implement the View component. Using JSP technology enable the application to show dynamic content by establishing an interaction between the View and the Controller components.
- Controller. It is the application core, i.e. it contains the business logic. In this implementation it was required to use Java servlets, Java web filters and Java web listeners to satisfy the requirements.

2 Proposed solution - Workflow

2.1 Connection to the web application

When a user connects to the web application, it's request is redirected to the HomeServlet, as specified in the web.xml. At this point the request is filtered by the AuthServlet that checks whether if the user is an unauthenticated user, an authenticated user, or the admin.
In the first case the request is redirected to the LoginServlet; in the second, it is redirected to the HomeServlet; in the third, to the ControlServlet. This is decided by checking the session attribute `userBean`.

In order to initialize the application data, i.e. admin user definition, loading users info from .txt file, declaring needed data structures, the LoginServlet overrides the init method to call the constructor of the Initializer class.

2.2 Login

Once the request come to the LoginServlet, page login.jsp is sent to the client. Once the login form is filled and submitted, a POST request is sent to the AuthServlet that checks in the previously initialized HashMap if the user credentials are correct. If so, the request is redirected to the HomeServlet or the ControlServlet depending on the user's role. Otherwise, the request is redirected to the LoginServlet.
If the authentication is successful, session attribute `userBean` containing the user info is added, and the ActiveUserListener handle the event by adding the user data to an HashMap containing the active users.

2.3 Registration

A GET request to the RegistrationServelt can be sent by clicking on the registration link in the login.jsp page on client-side.
This servlet send to the client the registration.jsp page, prompting the user to fill the form with the desired username and password (the latter, two times). Submitting the form will send a POST request to the AddUserServlet.
At this point, the servlet checks if the two input of the password are not equal or if the submitted username already exists; if so, the request is redirected to the RegistrationServlet, otherwise, a new UserBean is declared with the user information and is stored both in the HashMap residing at context level and in the .txt file. Lastly, the request is redirected to the LoginServlet.

2.4 Game - Presentation

After having successfully completed the login, and if the authenticated user is not an admin, the HomeServlet will send a response to the client containing home.jsp. This page is dynamically generated since it contains both the username and the score of the authenticated user.
Home.jsp contains a submit button that sends a GET request to the GameServlet. This servlet randomly selects three flags from the flags directory contained in the deployed application directory (using method `javax.servlet.ServletContext.getRealPath()`). Having done this, the three flags are saved in an ArrayList and put in the session table as `chosenFlagIndex` attribute. At this point, a response containing game.jsp is sent. The page is dynamically built to show a form with the three images of the flags. Once the form has been submitted, a POST request containing the index of the chosen answer is sent to the CheckAnswerServlet.

2.5 Game - Answer checking

The indexes contained in the POST request are compared with the chosen flags. If one of them do not match, one point is subtracted from the score attribute of the user UserBean, otherwise, three points are added.
Both in CheckAnswerServlet and in game.jsp, the answers are checked in order to assure that none of them are null or NaN. In game.jsp this check is ran setting the required and type field of every input in the form; while in the servlet, this checked is performed by parsing the request parameter to Integer and handling `NullPointerExeption` and `NumberFormatException`.

2.6 Control page - Presentation

If the user authenticate as admin, then, after validation, the request is redirected to ControlServlet. Here, a chek is performed to make sure that the request was made by a user with admin role; if not, the response status is set to 401, and the relative error page is sent. If the user is in fact an admin, a response containing controlPage.jsp is sent to the client. The page contains the list of active users.

2.7 Control page - Active users

In order to keep track of all the active users, a web listener has been implemented. The listener waits for a state change in the session (i.e. attribute added, attribute removed, attribute replaced), and depending from the type of event, it adds or remove a given user from the HashMap of the active users. This events can be triggered by authenticating as another user, or by letting expire the session (as specified in the web.xml, the session timeout is 3 minutes).

2.8 Filters

This web application uses two filters:

- AuthFilter. This filter is used for every request to every paths, with exceptions being /LoginServlet, /RegistrationServlet, /AddUserServlet, /AuthServlet, /login.jsp, and /registration.jsp. This filter is used to enforce the above cited RBAC.
- JspFilter. This filter is used to prevent direct access to the .jsp pages. Since multiple pages are dynamically generated, trying to access it without sending the request to a servlet can lead to unexpected behaviour both on client and server side.

Mapping and definition of the filters can be found in the web.xml.

3 Running application

3.1 Login

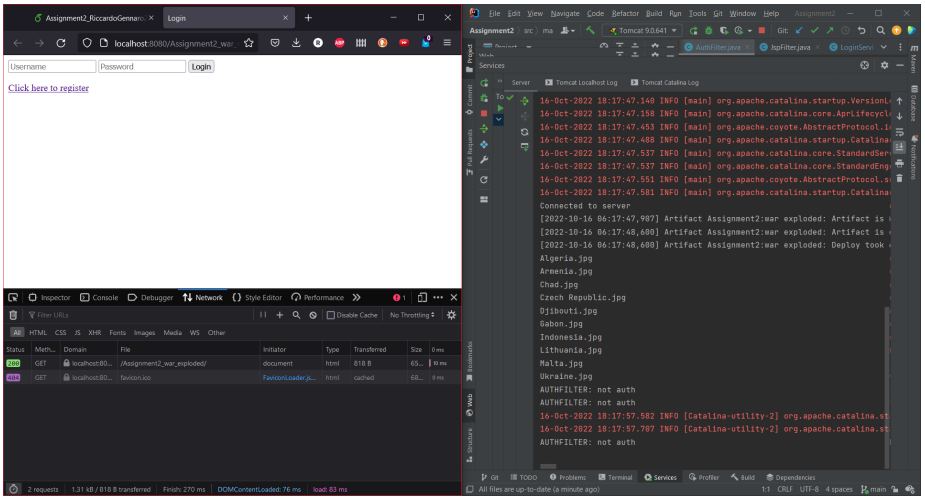

Figure 1: Login page

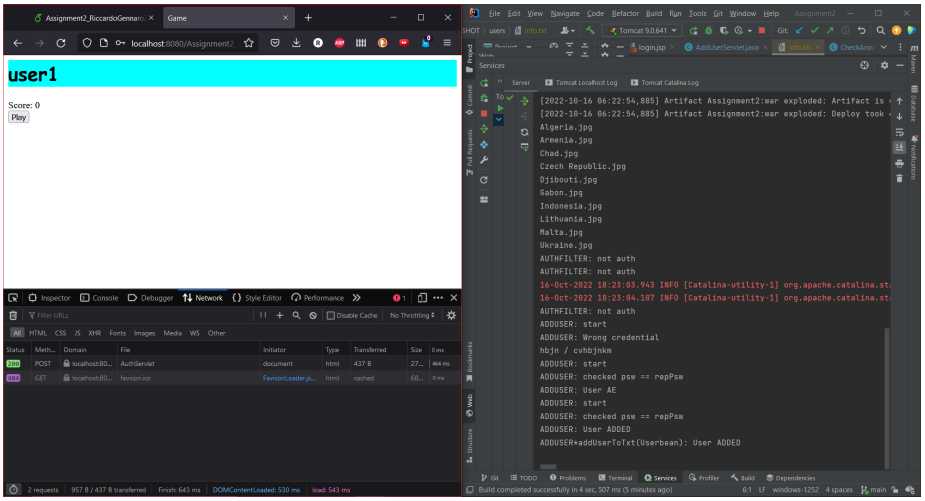

Figure 2: Successful login as user

3.2 Registration

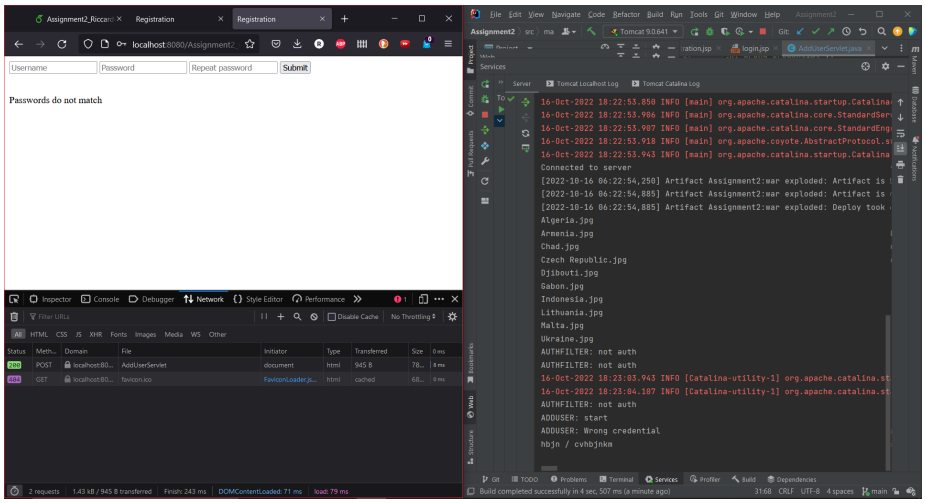

Figure 3: Passwords do not match during registration

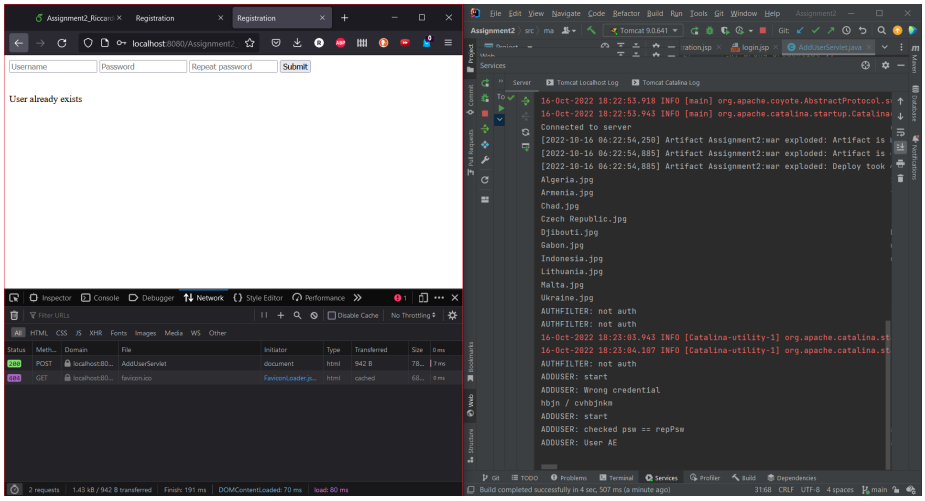

Figure 4: Trying to register already existing user

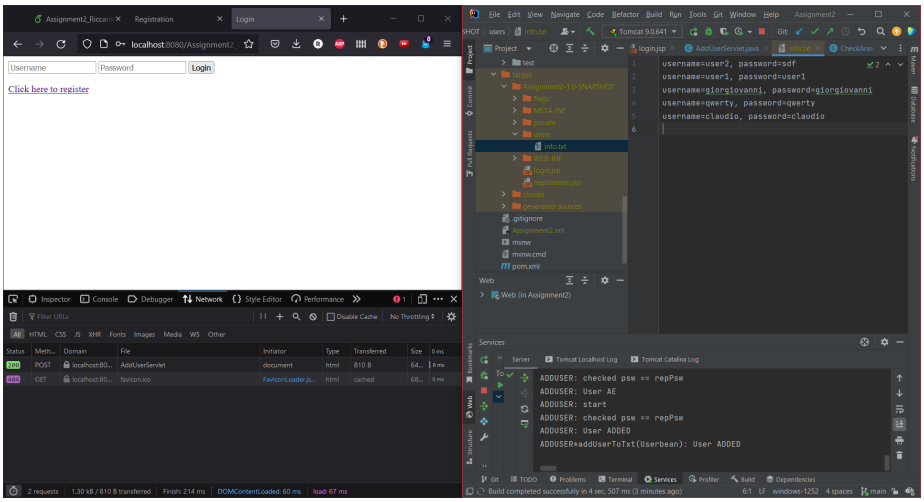

Figure 5: Successful user registration

3.3 Game

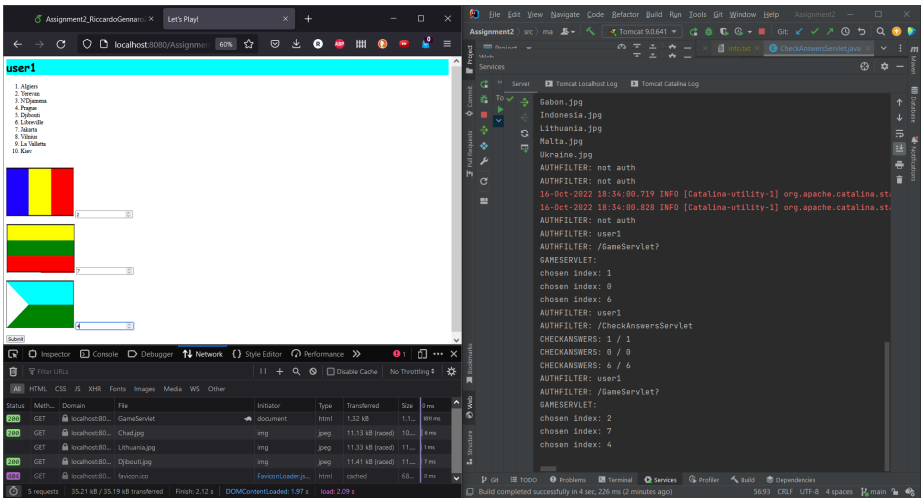

Figure 6: Game page with correctly selected answers

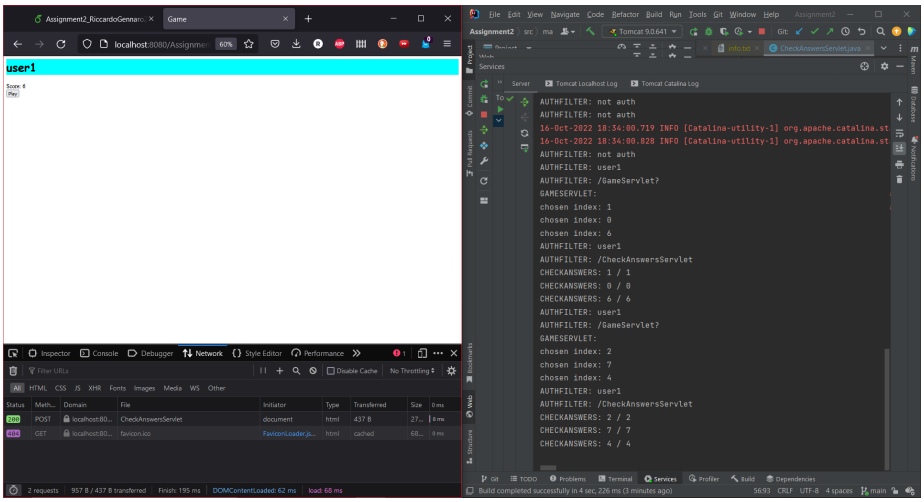

Figure 7: Home page after two all correct answers

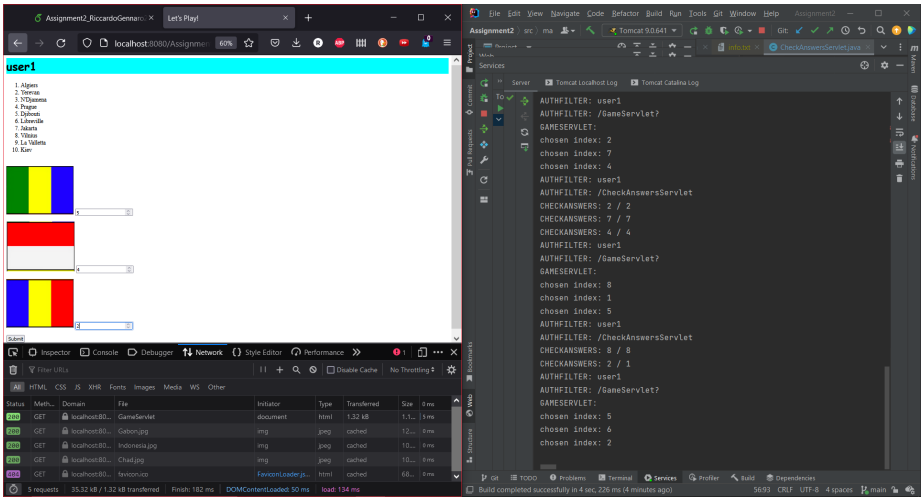

Figure 8: Game page with wrongly selected answers

3.4 Control page

Note that reloading the control page with Ctrl+R (and relative MacOS/Linux key), will resend the request to the AuthServlet, thus not needing to authenticate again after the session has expired.

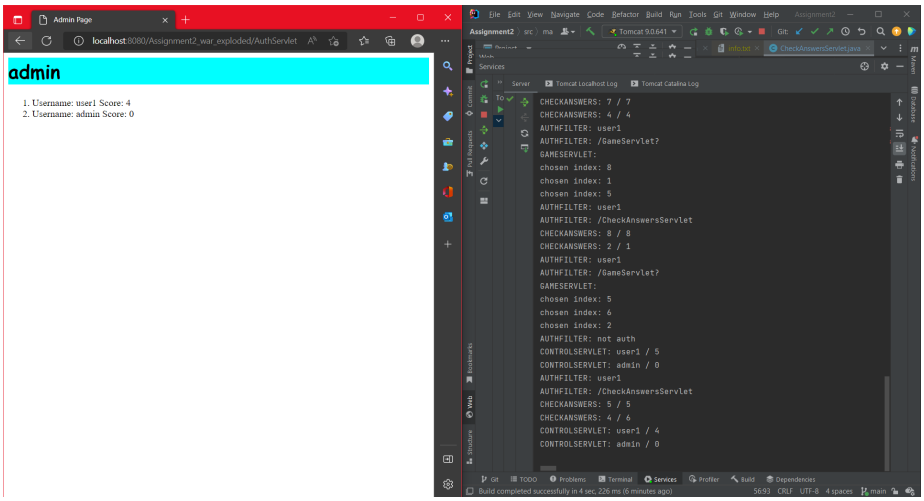

Figure 9: Successful login as admin. Control page

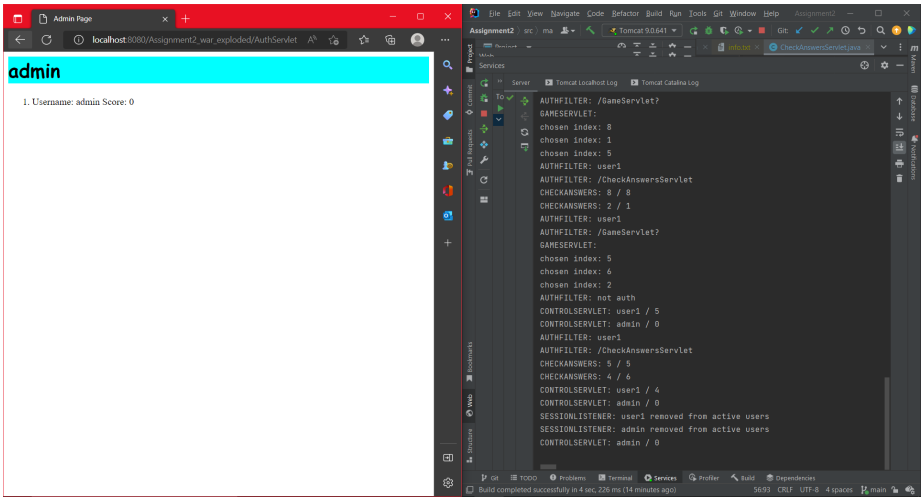

Figure 10: Control page after user1 session has been destroyed

3.5 Comments and notes

The only problem that I've encountered was during the read/write operations on the user info text file. It appeared that the solution proposed in one of the link listed at the end of the project requirements was not working as intended (problems with the charset of the output stream). In order to quickly solve the problem I used a simple FileWriter and FileReader to implement the file I/O, I then manually parsed the text document using String.split().