Assignment 2

Web Architectures - 16/10/2022

Sabin Andone

Introduction

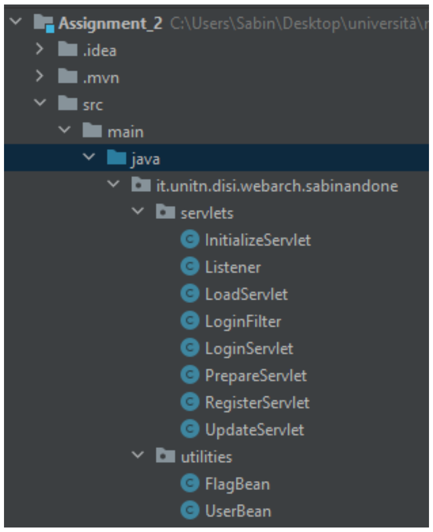
The objective of this assignment is to develop a website that makes use of an MVC pattern based on Servlet+Beans+JSPs. The website in this case is a game where the user, after the authentication, plays a game where s/he has to guess the correct capital for each flag of states and based on the result, his/her points increase or decrease. This website must satisfy some points that are listed below:

1. The webapp requires the user to authenticate him/herself in the login page, and if s/he does not have an account, then s/he has to create a new one in the registration page.
2. The webapp must have a start page where there are 3 elements: a header which contains the user's name, the number of points the user has in that moment (initially zero) and a button named "Play" that starts the game.
3. The header must be contained in all pages, except in authentication and registration pages
4. Each game session must consist in associating the respective capital with three flags that are chosen randomly from a pool. The game page contains in this case: the header with the user's name, the list of all capitals from which the user can choose from and a form with three flags and a field where the user choose the correct capital by choosing its number in the list.
5. The user must indicate the number of the capital in the fields next to each flag. Once s/he chooses the three capitals, s/he submits. In this case the fields must be filled and they have to contain a number between 1 and 10.
6. The system checks the answers: if they are all correct it increases the score by three points, otherwise it decreases by one point. After the computation of the new point value, it returns to the start page.
7. Attempts to access the home and game pages by unaunthenticated users redirect to the authentication page.
8. The manager (which is a predefined user, authenticated with username admin and password nimda) goes to a control page where s/he can see the list of all active users and for each of them what the current score is. In order to see an updated version of the list a page reload is needed.
9. All data must be maintained as long as the webapp is running. When the webapp shuts down, user identities are saved into a file. The system reads that file when it is redeployed.

Solution

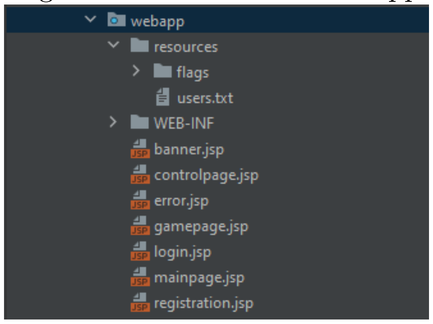
The solution for this assignment is a webapp that is structured as follows:

Figure 2.1: Structure of servlets and utilities.



```
Assignment_2 C:\Users\Sabin\Desktop\università\
  .idea
  .mvn
  src
    main
      java
        it.unitn.disi.webarch.sabinandone
          servlets
            InitializeServlet
            Listener
            LoadServlet
            LoginFilter
            LoginServlet
            PrepareServlet
            RegisterServlet
            UpdateServlet
          utilities
            FlagBean
            UserBean
```

Figure 2.2: Structure of webapp



```
webapp
  resources
    flags
    users.txt
  WEB-INF
  banner.jsp
  controlpage.jsp
  error.jsp
  gamepage.jsp
  login.jsp
  mainpage.jsp
  registration.jsp
```

Regarding the servlets and utilities, in summary these are:

- InitializeServlet: the servlet that is invoked at the deployment of the webapp. It redirects either to the mainpage.jsp or to login.jsp
- Listener: the ServletContextListener which reads the users from the user.txt file at the deployment of the webapp.
- LoadServlet: the servlet that loads the game page
- LoginFilter: the filter used to not let unauthorized users to access the mainpage.jsp, gamepage.jsp and error.jsp
- LoginServlet: the servlet that is invoked when the user does the login

- PrepareServlet: the servlet that is invoked when the user starts the game. It prepares the game session by choosing three random flags.
- registerServlet: the servlet that is invoked when the user creates a new account. It adds the new user to the list of all users and writes such updated list in the users.txt file.
- UpdateServlet: the servlet that updates the users's point value and that sends him/her back to the start page.
- Flag Bean: the bean class for the flags.
- User Bean: the bean class for the users.

Regarding the jsp pages that have been developed, in summary these are:

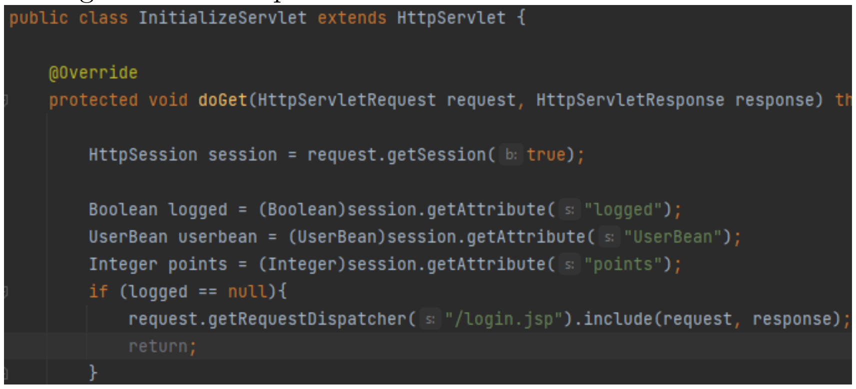
- banner.jsp: the header which is used for the start page and the game page
- controlpage.jsp: the page in which the admin can see the list of all connected users
- error.jsp: the 401 error page that is shown when an unauthorized user tries to access the control page
- gamepage.jsp: the game page where the user plays the game
- login.jsp: the login page in which the user identifies him/herself
- mainpage.jsp: the start page where the user starts the game session
- registration.jsp: the registration page in which the user creates a new account

Considering the structure and the requested features listed in the Introduction chapter, the work was organized as follows:

The first step was creating InitializeServlet, login.jsp, register.jsp, LoginServlet, registerServlet, LoginFilter, UserBean, header.jsp and mainpage.jsp. Initially, to retrieve the users from the list of users known to the system, an ArrayList of UserBean defined inside the code has been used, but at the last step a ServletContextListener has been developed and thus it is possible to retrieve the users from the users.txt file.

Initialize Servlet is a servlet that is invoked when the webapp is deployed. This because it has been set as the welcome page in the web.xml configuration file of the project. This servlet takes some attributes from the session, one of them being "logged" attribute which indicates if the user is logged or not. In practice, when the webapp is deployed, that attribute is null and thus the user is redirected to login.jsp.

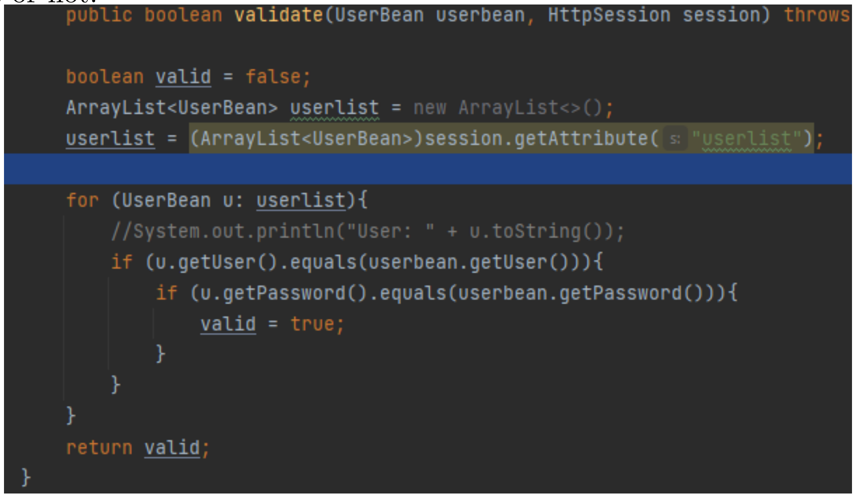
Figure 2.3: First part of doGet method in InitializeServlet.



```
public class InitializeServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) th

        HttpSession session = request.getSession( b: true);

        Boolean logged = (Boolean)session.getAttribute( s: "logged");
        UserBean userbean = (UserBean)session.getAttribute( s: "UserBean");
        Integer points = (Integer)session.getAttribute( s: "points");
        if (logged == null){
            request.getRequestDispatcher( s: "/login.jsp").include(request, response);
            return;
        }
```

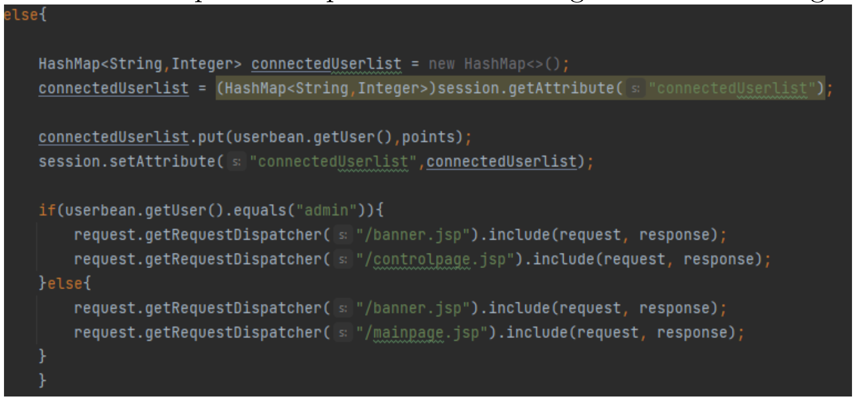
The login.jsp page is basically a form which contains two text fields, one for the username and the other one for the password. When the user submits his/her credentials it is redirected to the LoginServlet, which checks from the user list if the username and password are correct or not [Figure 2.4]. If they are correct, then the user is redirected to the InitialServlet whcih now will send the user to the start page[Figure 2.5].

Figure 2.4: validate method is the method used inside the doPost method of LoginServlet to check if the credentials are correct or not.



```
    public boolean validate(UserBean userbean, HttpSession session) throws

        boolean valid = false;
        ArrayList<UserBean> userlist = new ArrayList<>();
        userlist = (ArrayList<UserBean>)session.getAttribute( s: "userlist");

        for (UserBean u: userlist){
            //System.out.println("User: " + u.toString());
            if (u.getUser().equals(userbean.getUser())){
                if (u.getPassword().equals(userbean.getPassword())){
                    valid = true;
                }
            }
        }
        return valid;
    }
```

Figure 2.5: Second part of doGet method in InitializeServlet. Notice that there is also a connectedUserlist HasMap. This HashMap has been developed later for the control page, since the admin has to see the list of all active users and this is possible by putting all the users in this HashMap. The .put method is called in InitializeServlet because it can also update the point value after a game session for a given user.

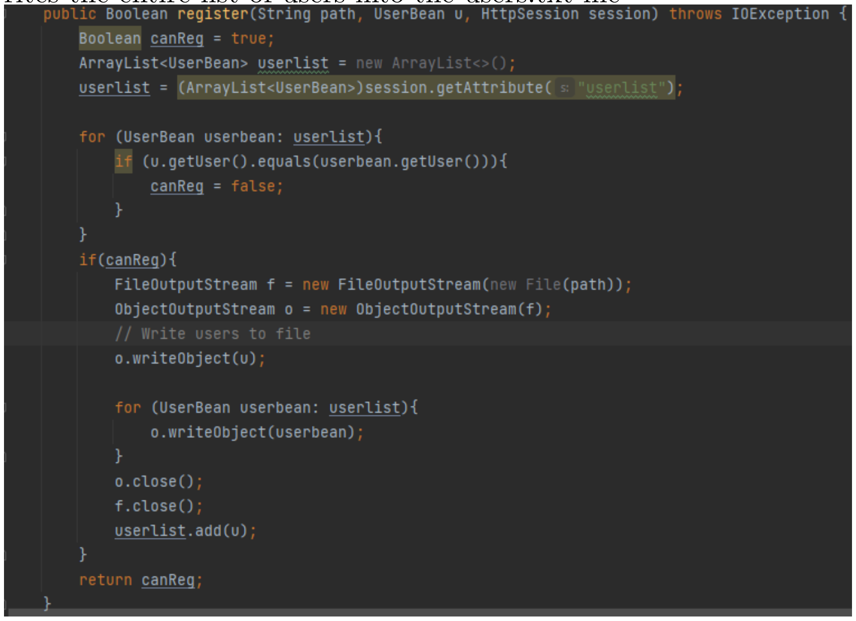


```
else{

    HashMap<String,Integer> connectedUserlist = new HashMap<>();
    connectedUserlist = (HashMap<String,Integer>)session.getAttribute( s: "connectedUserlist");

    connectedUserlist.put(userbean.getUser(),points);
    session.setAttribute( s: "connectedUserlist",connectedUserlist);

    if(userbean.getUser().equals("admin")){
        request.getRequestDispatcher( s: "/banner.jsp").include(request, response);
        request.getRequestDispatcher( s: "/controlpage.jsp").include(request, response);
    }else{
        request.getRequestDispatcher( s: "/banner.jsp").include(request, response);
        request.getRequestDispatcher( s: "/mainpage.jsp").include(request, response);
    }
    }
```

If the user instead does not have an account, then s/he has to go to register.jsp by clicking on the link that appears under the form in the login.jsp. In this case another form appears, where this time the user has to put his/her username and password (and confirm it). The submition of the register form will invoke RegisterServlet which will add the new user to the user list [Figure 2.6]. Once registered, the user is redirected back to the login where s/he can this time to access with the new account.

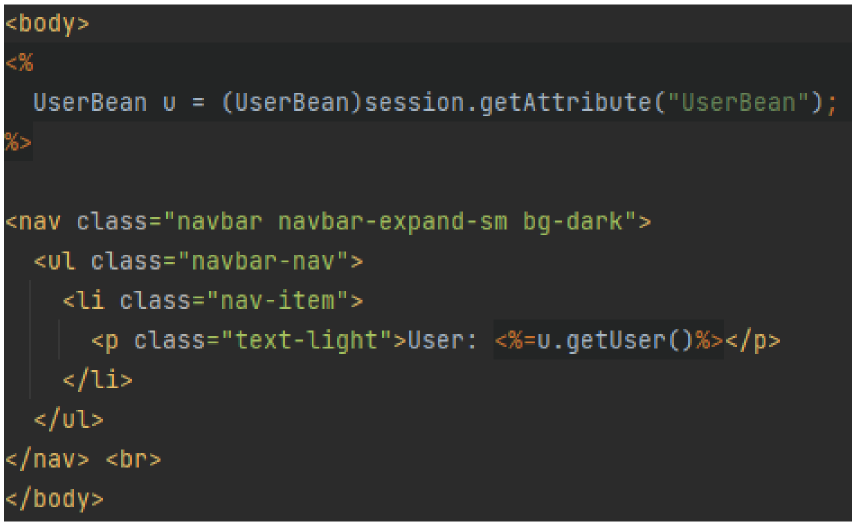
In any case, after the login, the user is redirected to the start page, which contains the header and the main page (the greetings and the play button). [Figure 2.7 and 2.8]

Figure 2.6: register method is the method used inside the doPost method of RegisterServlet to add the user to the user list. It also writes the entire list of users into the users.txt file



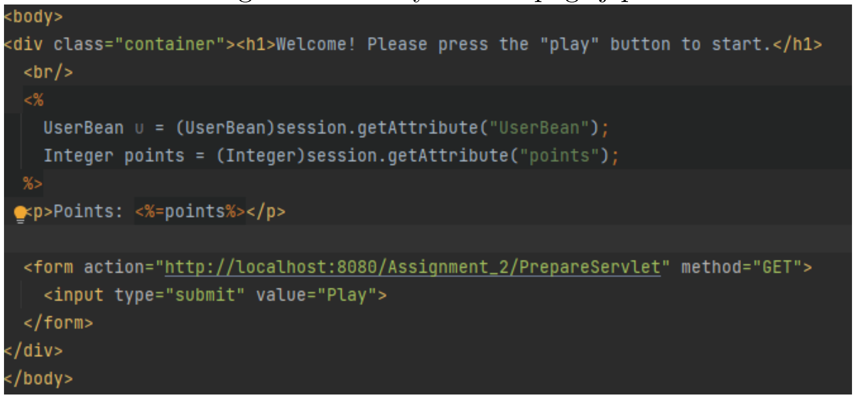
```
public Boolean register(String path, UserBean u, HttpSession session) throws IOException {
    Boolean canReg = true;
    ArrayList<UserBean> userlist = new ArrayList<>();
    userlist = (ArrayList<UserBean>)session.getAttribute( s: "userlist");

    for (UserBean userbean: userlist){
        if (u.getUser().equals(userbean.getUser())){
            canReg = false;
        }
    }
    if(canReg){
        FileOutputStream f = new FileOutputStream(new File(path));
        ObjectOutputStream o = new ObjectOutputStream(f);
        // Write users to file
        o.writeObject(u);

        for (UserBean userbean: userlist){
            o.writeObject(userbean);
        }
        o.close();
        f.close();
        userlist.add(u);
    }
    return canReg;
}
```

Figure 2.7: Body of banner.jsp. In the header of the banner there is a link to bootstrap whcih makes the UI more pleasant to see.



```
<body>
<%
  UserBean u = (UserBean)session.getAttribute("UserBean");
%>

<nav class="navbar navbar-expand-sm bg-dark">
  <ul class="navbar-nav">
    <li class="nav-item">
      <p class="text-light">User: <%=u.getUser()%></p>
    </li>
  </ul>
</nav> <br>
</body>
```

Figure 2.8: Body of mainpage.jsp.



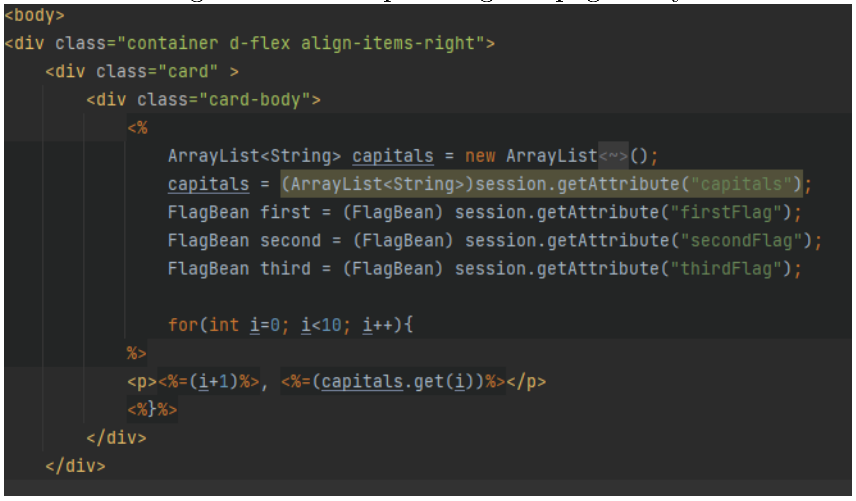
```
<body>
<div class="container"><h1>Welcome! Please press the "play" button to start.</h1>
  <br/>
  <%
    UserBean u = (UserBean)session.getAttribute("UserBean");
    Integer points = (Integer)session.getAttribute("points");
  %>
  <p>Points: <%=points%></p>

  <form action="http://localhost:8080/Assignment_2/PrepareServlet" method="GET">
    <input type="submit" value="Play">
  </form>
</div>
</body>
```

Once the login part has been developed, the next step was creating the pages and the servlets for the actual game session. In this case FlagBean, PrepareServlet, LoadServlet, UpdateServlet and gampage.jsp have been developed.

When the user presses the play button in the mainpage.jsp, the PrepareServlet is invoked. It prepares the list of nations and capitals, and then it chooses three nations at random so it can create three FlagBean instances. Finally, it saves the three flags and the capital list in the session and then it redirects to the LoadServlet, which is a servlet that just loads the game page.

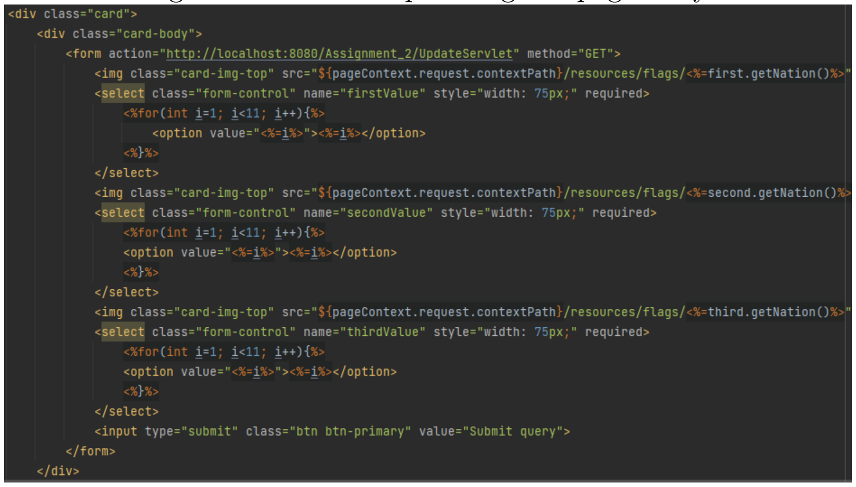
The game page consists of the header and the principal game page which has two parts: the list of capitals[Figure 2.9], and a form which contains the three flags and the three fields to compile[Figure 2.10].

Figure 2.9: First part of game page body.



```
<body>
<div class="container d-flex align-items-right">
    <div class="card" >
        <div class="card-body">
            <%
                ArrayList<String> capitals = new ArrayList<~>();
                capitals = (ArrayList<String>)session.getAttribute("capitals");
                FlagBean first = (FlagBean) session.getAttribute("firstFlag");
                FlagBean second = (FlagBean) session.getAttribute("secondFlag");
                FlagBean third = (FlagBean) session.getAttribute("thirdFlag");

                for(int i=0; i<10; i++){
            %>
            <p><%=(i+1)%>, <%=(capitals.get(i))%></p>
            <%}%>
        </div>
    </div>
```

Figure 2.10: Second part of game page body.

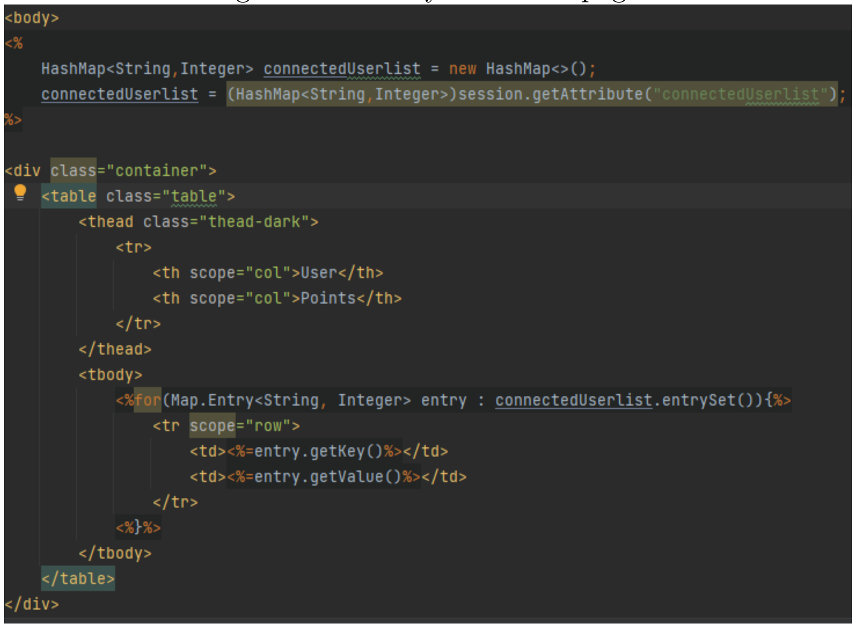


```
<div class="card">
    <div class="card-body">
        <form action="http://localhost:8080/Assignment_2/UpdateServlet" method="GET">
            <img class="card-img-top" src="${pageContext.request.contextPath}/resources/flags/<%=first.getNation()%>"
            <select class="form-control" name="firstValue" style="width: 75px;" required>
                <%for(int i=1; i<11; i++){%>
                    <option value="<%=i%>"><%=i%></option>
                <%}%>
            </select>
            <img class="card-img-top" src="${pageContext.request.contextPath}/resources/flags/<%=second.getNation()%>
            <select class="form-control" name="secondValue" style="width: 75px;" required>
                <%for(int i=1; i<11; i++){%>
                <option value="<%=i%>"><%=i%></option>
                <%}%>
            </select>
            <img class="card-img-top" src="${pageContext.request.contextPath}/resources/flags/<%=third.getNation()%>"
            <select class="form-control" name="thirdValue" style="width: 75px;" required>
                <%for(int i=1; i<11; i++){%>
                <option value="<%=i%>"><%=i%></option>
                <%}%>
            </select>
            <input type="submit" class="btn btn-primary" value="Submit query">
        </form>
    </div>
```

Once the user submits his/her guesses such parameters are passed to UpdateServlet, which checks the such guesses. At this point, if the answers are all correct the system increases the user's points by three, otherwise it decreases them by one. In other word, this servlet gets parameters from game page, checks them and then update the points value accordingly. After that it redirects to the InitializeServlet, and from there the users can start a new game session.

The next feature required to satisfy for this assignment is the control page part, which states that when the admin logs in instead he enters into a control page where he can see all the active users that are connected into the webapp in that moment. This page consists in a table that contains the list of connected users and their points in that moment. For this assignment, since it was necessary to use just servlets, beans and jsp code, it is mandatory to reload the page to update the values on the table.

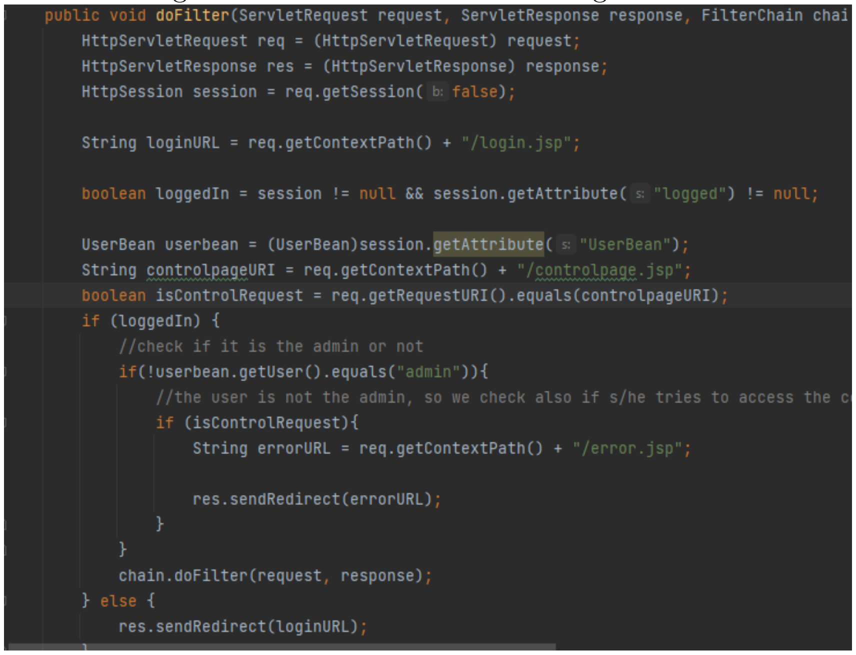
Figure 2.11: Body of control page.



```
<body>
<%
    HashMap<String,Integer> connectedUserlist = new HashMap<>();
    connectedUserlist = (HashMap<String,Integer>)session.getAttribute("connectedUserlist");
%>

<div class="container">
    <table class="table">
        <thead class="thead-dark">
            <tr>
                <th scope="col">User</th>
                <th scope="col">Points</th>
            </tr>
        </thead>
        <tbody>
            <%for(Map.Entry<String, Integer> entry : connectedUserlist.entrySet()){%>
                <tr scope="row">
                    <td><%=entry.getKey()%></td>
                    <td><%=entry.getValue()%></td>
                </tr>
            <%}%>
        </tbody>
    </table>
</div>
```

When the user attempts to enter a page and it is unauthorized to enter in that page, the LoginFilter[Figure 2.12] either redirects him into the login page or ,in case s/he wants to access the control page and s/he is not the admin, to the into the error.jsp page which is a 401 error page with a link to go back to the hompage (it redirects to InitializeServlet).

Figure 2.12: doFilter method of LoginFilter.

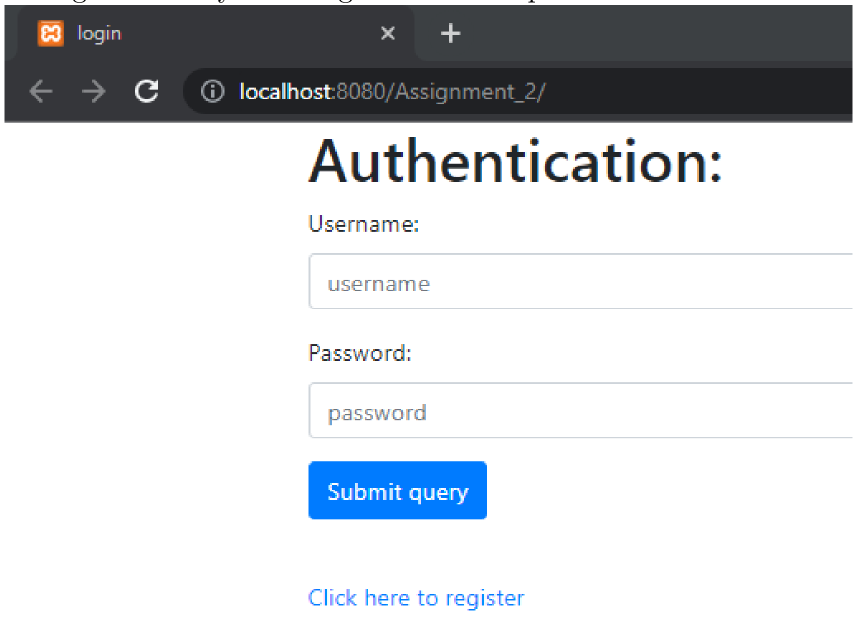


```
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chai
    HttpServletRequest req = (HttpServletRequest) request;
    HttpServletResponse res = (HttpServletResponse) response;
    HttpSession session = req.getSession( b: false);

    String loginURL = req.getContextPath() + "/login.jsp";

    boolean loggedIn = session != null && session.getAttribute( s: "logged") != null;

    UserBean userbean = (UserBean)session.getAttribute( s: "UserBean");
    String controlpageURI = req.getContextPath() + "/controlpage.jsp";
    boolean isControlRequest = req.getRequestURI().equals(controlpageURI);
    if (loggedIn) {
        //check if it is the admin or not
        if(!userbean.getUser().equals("admin")){
            //the user is not the admin, so we check also if s/he tries to access the c
            if (isControlRequest){
                String errorURL = req.getContextPath() + "/error.jsp";

                res.sendRedirect(errorURL);
            }
        }
        chain.doFilter(request, response);
    } else {
        res.sendRedirect(loginURL);
```

Finally, the last step was to create the ServletContextListener which reads the users.txt file and puts all the users into the userlist in the system. Then it puts such userlist into the HttpSession in order to be accessible by the other servlets. The file reading is executed by using ObjectInputStream, which reads the bytes of deserialized primitive data and objects previously written using an ObjectOutputStream.

The webapp running

Figure 3.1: The login page. In this case the user is not authenticated and every attempt to access the main page or the game page sends to user directly to the login page. Also, if the user puts wrong credentials, the system will warn him with a message which says "Wrong username or password!".



login

localhost:8080/Assignment_2/

Authentication:

Username:

username

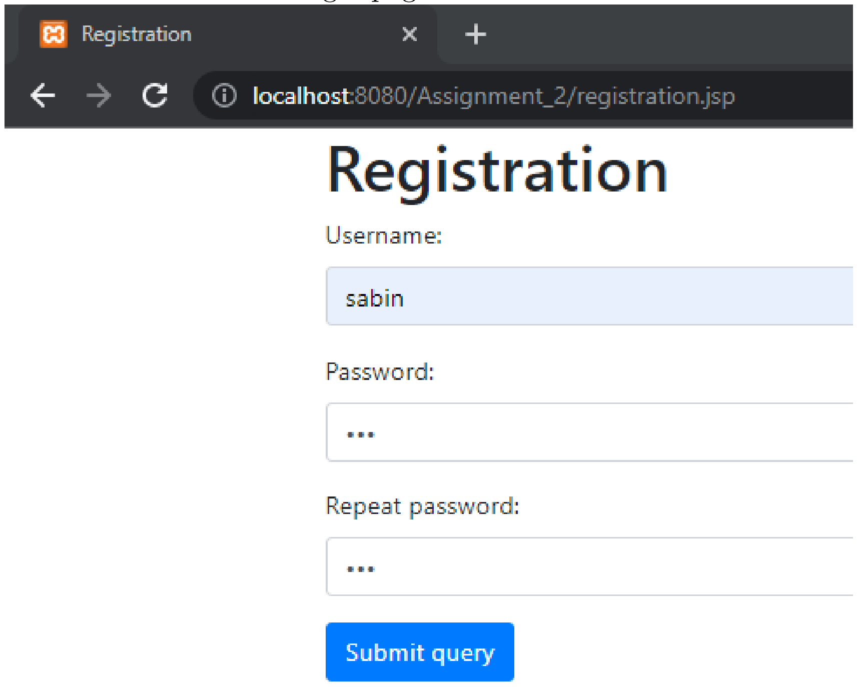
Password:

password

Submit query

Click here to register

Figure 3.2: The registration page. In this case the user does not have an account, so he must register. After the registration he will be redirected back to the login page.



Registration

localhost:8080/Assignment_2/registration.jsp

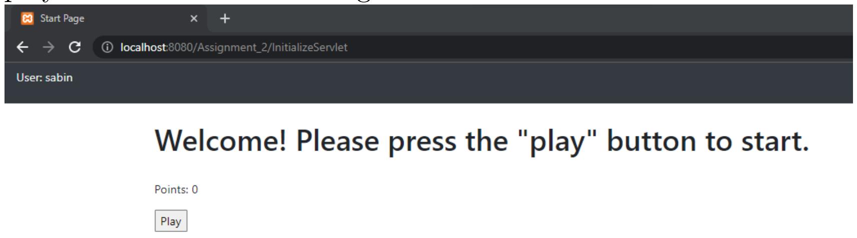
Registration

Username: sabin

Password: •••

Repeat password: •••

Figure 3.3: After the user authenticates himself, he goes to the start page. This page presents itself with a header which contains the user's name and a message which greets the user and explains to press the play button to start the game. At the bottom part of the page there are the number of points that the user has at that moment and the play button that starts the game session



Start Page

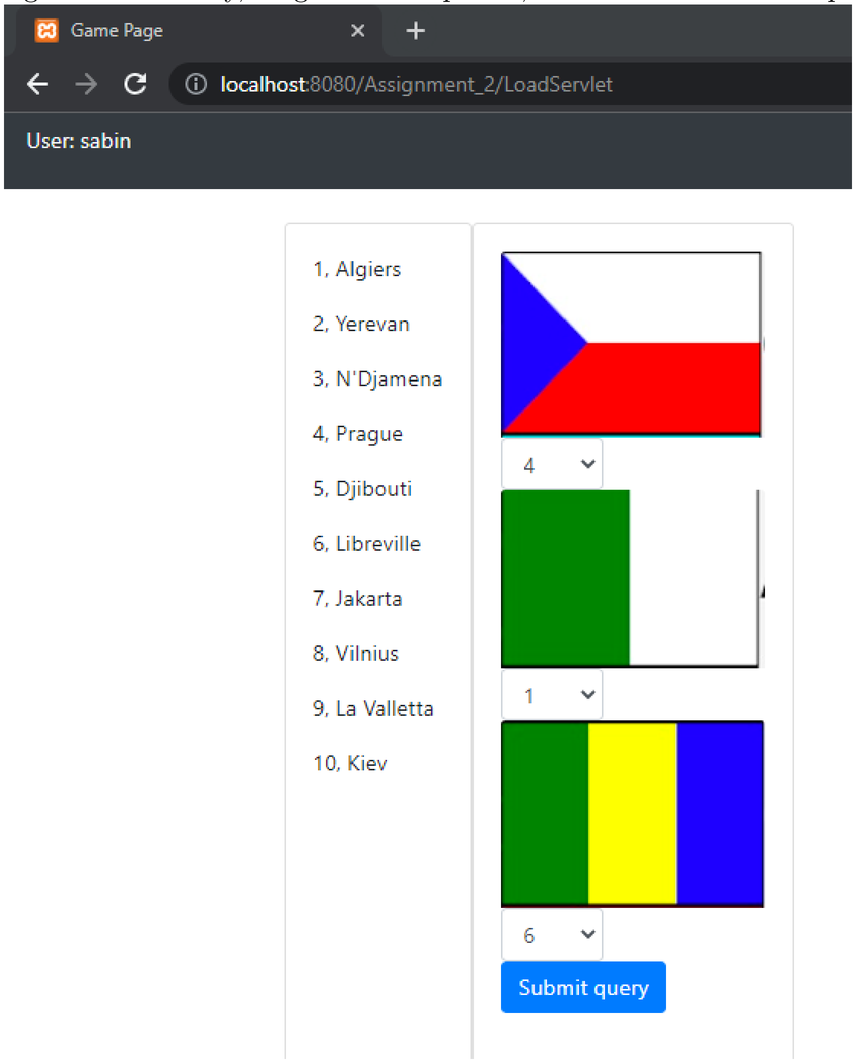
localhost:8080/Assignment_2/InitializeServlet

User: sabin

Welcome! Please press the "play" button to start.

Points: 0

Figure 3.4: The game session starts when the user presses the play button in the start page. The game session is prepared and the game page is loaded. This page has the same header as the start page and then it contains the list of capitals and the three flags for which the user has to guess the correct capital. For each flag there is a field that allows only numbers from 1 to 10. After the user submits his guesses, the system checks if the user guesses right. If he has guessed correctly, he gains three points, otherwise he loses one point.



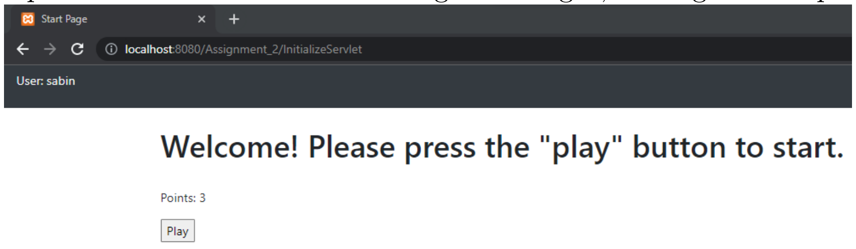
Game Page

localhost:8080/Assignment_2/LoadServlet

User: sabin

1, Algiers
2, Yerevan
3, N'Djamena
4, Prague
5, Djibouti
6, Libreville
7, Jakarta
8, Vilnius
9, La Valletta
10, Kiev

Figure 3.5: After the system assigns the new point value to the user, it redirects back to the start page and shows the user the new point value. In this case he has guessed right, so he got three points.



Start Page

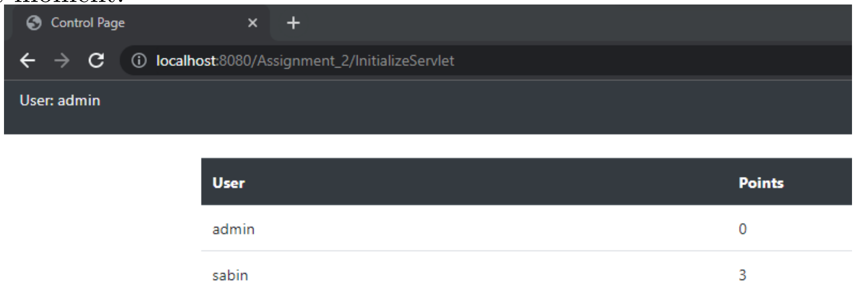
localhost:8080/Assignment_2/InitializeServlet

User: sabin

Welcome! Please press the "play" button to start.

Points: 3

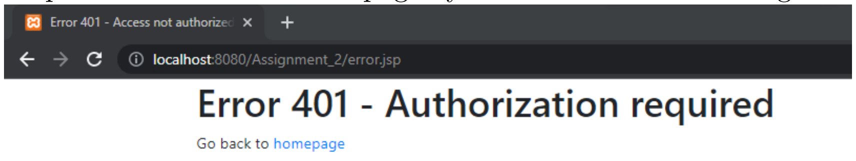
Figure 3.6: If the admin logs in instead, he would be redirected to a control page where s/he can see the users that are active in that moment.



User: admin

User	Points
admin	0
sabin	3

Figure 3.7: Attempts to access the control page by users other than admin generate a 401 error.


Error 401 - Access not authorized
localhost:8080/Assignment_2/error.jsp

Error 401 - Authorization required

Go back to homepage

Comments

The main issue I have had to deal during the development of this assignment was the short amount of time at disposal since i started the assignment a few days before the deadline due to further academic commitments. Due to this fact i had to focus on satisfying the main points listed in the first chapter and i managed to provide only some elements for a better user experience, e.g. by adding bootstrap in my project for a better graphic design. Probably with more time at my disposal the project could have been more complex and more efficient but then it would have contained features not requested for this assignment.

Other than that, the development of this assignement has not been problematic at all. An issue i have encountered was reading and writing the user.txt file because initially i manually add the users into the file and then at running time i got a StreamCorruptedException error specifying that stream headers were invalid. I resolved this issue by writing one time the users into the file before shutdown and then at future redeployments read them again from the same file. It was necessary to add the users one by one (admin included, which has "nimda" as password). Regarding the writing of the file, I had to put the writing method inside the register method of RegisterServlet, which updates the file and at the shutdown the user list is correct anyway.