Homework 4: Database Backend

For this homework, you will practice working with databases by adding a database backend to the web app that you created in the last homework.

You will do this homework as a team; however, each member of your team will be responsible for the completion of a particular task.

Step 0. Copy the Connector/J library into Tomcat's lib folder

The Connector/J library contains a JDBC (Java Database Connectivity) driver for MySQL. You can download a copy of the library from me:

http://www.cs.memphis.edu/~sdf/comp4081/resources/mysql-connector-java-5.1.25.jar

A copy of this jar file must be placed in the **lib/** folder of your Tomcat installation; otherwise, the user login functionality below will not work.

Step 1. Add database backend to movieclub

You will continue working on the **movieclub** web app by adding MySQL-based persistence to the dynamic web pages. For an example of a Java EE web app with a database backend, checkout this example:

https://utopia.cs.memphis.edu/course/comp4081-2013fall/examples/booksrus-db/trunk/

Each team member must complete one of the tasks defined below. Each of the tasks will involve working with a MySQL database that I've created for your team. You can find the database connection information in **src/main/webapp/META-INF/context.xml**.

The tasks will also refer to some new features that I've added to your team's project—namely, user login capabilities. The login functionality follows the form used in this example app, which you can checkout:

https://utopia.cs.memphis.edu/course/comp4081-2013fall/examples/dudeman-security/trunk/

Similar to the example, I have created three tables in your database: one for user IDs and passwords, one for security roles, and one that maps user IDs to roles. You will find login form and login error JSPs in **src/main/webapp/WEB-INF/**. I have also added a **LogoutServlet** Java class.

To activate login protection for your servlets add a **@ServletSecurity** tag as follows:

```
@WebServlet("/foo.do")
@ServletSecurity(value=@HttpConstraint(rolesAllowed = {"member"}))
public class FooServlet extends HttpServlet {
    …
```

Note that there is only one security role at the moment: "member." All users have this role.

I have included two SQL scripts in the top-level directory of your project: one for creating the tables, and one for populating them with test data. As you create tables and test data, you must add it to these scripts. That way, if your database becomes corrupted, the tables can be deleted and regenerated.

Task 1: Register User

RegisterUserServlet and **ListUsersServlet** must function as in **hw3**, except they must store and retrieve user data from the database.

Task 2: Add Movie

AddMovieServlet and **ListMoviesServlet** must function as in **hw3**, except they must store and retrieve movie data from the database.

Task 3: Top-10 List

The **EnterTop10List** and **ShowAllTop10Lists** servlets must function as in **hw3**, except:

- they must store and retrieve movie data from the database.
- the form produced by the servlets must no longer have a user field. Instead, the servlets must figure out the ID of the current user via the method call:
    - `request.getRemoteUser()`

Task 4: Most Popular Movies

ShowMostPopularServlet must work as in **hw3**, except it must retrieve data from the database.

Task 5: Movie Playlist

CreatePlayListServlet must function as in **hw3**, except:

- it must store and retrieve movie data from the database.
- the form produced by the servlet must no longer have a user field. Instead, the servlet must figure out the ID of the current user via the method call:
    - `request.getRemoteUser()`

Task 6: Movie Review

PostReviewServlet must function as in **hw3**, except:

- it must store and retrieve movie data from the database.
- the form produced by the servlet must no longer have a user field. Instead, the servlet must figure out the ID of the current user via the method call:
    - `request.getRemoteUser()`

Step 2. Submit (by tagging) your team's submission

The following instructions are essentially the same as last time; only the tag name has changed.

Attention! Before performing this step, you must make sure that all team members have committed their edits to the **trunk** in the repository.

Only one team member (the leader) performs the following.

First, you must fill out the **README.txt** file in your project's **trunk**. The file should list which team member performed each task (one team member per task).

To submit work in this course, you must tag it. Then, I will checkout the revision that you tagged and grade it. By tagging, you tell me that you are done, and this is the version you want me to grade.

The tag you must use for this homework is **hw4** (case sensitive, no spaces).

To tag the current revision of your trunk as **hw4**, do as follows:

1. Go to the **SVN Repository Exploring** perspective in Eclipse.
2. In the **SVN Repositories** view, find the **trunk** folder that you want to tag.
3. Right-click on the **trunk** folder, and click **Show History**. This should open the **History** view with a table listing the past commits to the **trunk**.
4. In the History table, right-click the newest revision (i.e., the one with the greatest revision number), and click **Tag from…** This should open a **Create Tag** dialog.
5. Enter **hw4** into the **Tag** field and optionally enter a log comment, then click **OK**. This should create the tag!

To verify that tagging was successful, open the following URL in a web browser (replacing *YOUR_TEAM* with the appropriate name):

https://utopia.cs.memphis.edu/course/comp4081-2013fall/teams/YOUR_TEAM/movieclub/tags/

You should see an **hw4** folder, and within that folder should be **src**, **README.txt**, etc.