Università di Trento

Web Architectures

Assignment 3

Author:
Marcel Pascal Stolin
marcelpascal.stolin@studenti.unitn.it

October 27, 2021

1 Introduction

The task of the third assignment is to create a web version of the Memory game. In this game, different user can play a game and see the highscore of all users who have played before.

The applications is composed of two systems, the backend and frontend. The backend is responsible to authenticate the user with a username, save the highscore of each user, and respond the number of points for each guess made by the user. Additionally, it should be possible to set a *development mode*, where the grid of the memory is deterministic. Otherwise, in *production mode*, the grid is generated randomly.

The frontend side consists of the game (User Interface),v where the user can play the actual game. This frontend has to be implemented using JavaScript and it should communicate with the backend using Ajax.

2 Conceptual Design

The conceptual design is based on the problem statement introduced in Section 1.

2.1 Backend

The backend is responsible to provide authentication to the user. Additionally, after the user is authenticated, the user should be able to see a ranking of all users. From there, the user can start a new game of Memory. Therefore, the following pages have to be created:

- Authentication-Page `/authentication`
- Ranking-Page `/ranking`
- Memory-Play-Page `/play`

Furthermore, the grid of the memory game is supposed to be generated on the server side. The frontend has to fetch the value of a card, everytime a user has made a guess. Given this, an API has to be implemented, were the frontend can ask for the value of a selection. This is called the Grid-API and it is available via `/memory/grid`.

2.1.1 Authentication

Whenever a user visits the application for the first time, the user has to provide a username. If not, the user should not be able to access the game, and will be redirected to the authentication page until the user enters a name. After the user has entered a name, the user will be redirected to the ranking page.

2.1.2 Ranking

On the ranking page, the user can see a Top 5 list of all user who played a game before. Additionally to the username, the list shows the number of points the user has achieved. Furthermore, on the ranking page, the user can click the button *Play game* to start the memory game.

2.1.3 Memory Game

The *Memory-Play-Page* consists if a 4x4 grid. Each cell in this grid shows a card. By default, only the back of the card is shown. The user is able to click on each to flip a card and then to search for the equal value in the grid. If the user finds the equal card, the value is multiplied by two and added to the score. The cards will then stay with the visible value. Otherwise one point is removed from the current score and both are flipped back. In addition to the grid, the *Memory-Play-Page* also shows the current score, and the number of tries. After each click on a card,

the tries are being updated. In total, the user has 8 tries to find pairs. After 8 tries a lable with the text *"Game Over!"* is shown and the user is being redirected to the *Ranking-Page*.

2.1.4 Grid

When the user starts the game, the game needs to know how which value is behind a card. However, the frontend is not supposed to know the arrangement of the cards. Therefore, the backend has to generate a 2D array of the grid. The grid consists of 16 cards in a 4x4 grid. Each card exists 2 times, therefore there are 8 different cards on the grid in total. Then, when a user clicks on a card in the grid on the *Memory-Play-Page*, the frontend has to make a request to the *Grid-API* to get the value of the selected card.

In addition, it should be possible to decide if the backend operates in a *development* or *production* mode. If development mode is activated, the grid is generated in a deterministic way. Otherwise, in production mode, the grid is generated in a random way. Figure 1 shows an example of the different grid versions for each mode.

1	1	2	2
3	3	4	4
5	5	6	6
7	7	8	8

Development Mode

5	2	3	8
2	7	7	4
6	4	6	1
1	5	8	3

Production Mode

Figure 1: Example of the grid for *development* and *production* mode

2.2 Frontend

The frontend part of the application consists of the Memory game. It show 16 cards in a grid (introduced in Section 2.1.4). The user can click on a card a the card will flip. After that, the user can click on a second card and the card will flip as well. After a card has been clicked, it will become unclickable as long as the user makes finishes the guess. If both cards match the guess was successful and the points will be added to the score of the user. The added points are two times the guessed card value (e.g. if both cards have the value 4, 8 points will be added to the user score). Otherwise, if the guess was incorrect, one point will be subtracted from the points of the user. To get the value of a card, the frontend has to send a request to the backend given the index of the card in the grid. Then, the backend returns the card value. The user can make 8 guesses (4 attempts to find pairs) in total. After that, the game finishes. Then, a *Game Over* label appears, and after 1 second, the user will be redirected to the ranking page.

3 Implementation

This section explains the implementation based on the conceptual design introduced in Section 2. Therefore, this applications is composed of two different parts: A backend, and a frontend.

3.1 Backend

As introduced in SEC DESIGN BACKEND, the backend part of the application is responsible for authentication, keeping a ranking list of games played by different users, and generating the grid for the Memory game. To achieve this functionality, the following servlets are created:

- Welcome-Servlet
- Ranking-Servlet
- Memory-Play-Servlet
- Memory-Grid-Servlet

In addition to the servlets, a Welcome-Filter needs to be implemented to check whenever a user is authenticated or not.

3.1.1 Models

A user is allowed to authenticate itself with a username. Therefore, a model called User is needed in the system. This model will be used to authenticate the user on all pages, introduced in SEC DESIGN, and to save a ranking with all users who played a game before.

Additionally, the backend keeps track of a scoreboard to safe the points achieved by a user. This is achieved by the Scoreboard model. The Scoreboard model saves User models and their points in a Map object. Additionally, the Scoreboard model provides a method called getTop5 that returns the Top 5 scores. This used for the Ranking-Page.

FIG XY describes both models. The User model will be saved in the HTTP session of the client. However, the same scoreboard instance needs to be available for all users. Therefore, the Scoreboard model is saved in the servlet-context. Additionally, to use these models in JSP views, both models follow the Java Bean specification.

3.1.2 Welcome

As introduced before, the user needs to authenticate itself with a username, and a password is not required. If the user is not authenticated, the user can not access the game or the ranking page. A user can authenticate at the *Welcome-Page*.

There, the user user can enter a name in an HTML form. After submitting the form, the username is being send as a `POST` request to the *Welcome-Servlet*. After that, the *Welcome-Servlet* validates the `POST` request. If the request is valid, a new *User* model will be added to the client HTTP session. Then, the user will be forwarded to the *Ranking-Page*. To ensure, that only authenticated users can access the *Ranking-Page*, and the *Memory-Play-Page*, a filter called Welcome-Filter is implemented, that checks if the a *User* objects is available in the HTTP session of the client. If yes, the user is can access all pages, otherwise the user is being forwarded to the welcome page.

3.1.3 Ranking

The Ranking-Servlet is responsible to keep track of all games played.

If a GET request is send to the Ranking-Servlet, it will response an HTML page with the ranking of the Top 5 users. The ranking displays the name and the points of the user. The view is implemented using a JSP view. It includes the current User and the Scoreboard via jsp:useBean to get its properties. Additionaly, it uses JSP - Standard Tag Library (JSTL) to show the Top 5 users, which is shown in LST XY. Furthermore, the ranking shows the name of the current user at the top.

To save a new score, the Ranking-Servlet accepts POST request as well. It requires a number of points with the content-type of `bla bla`. The points will be added to the current score of the user of the client session using the addScore method of the Scoreboard model. The overall ranking needs to be available to all users. Therefore, it is saved in the servlet-context instead of the the HTTP session. It is important to mention, that it is not required to save the overall ranking persistently. Therefore, if the webserver exits, the servlet-context is removed from the host memory and wont be available after a restart. However, the ranking is available as long as the webserver is running.

3.1.4 Grid

The Memory-Grid-Servlet is reponsible to generate the grid for the memory game. The concept of the grid has been introduced in SEC DESIGN. To recieve the value of a index in the grid, the frontend has to send a GET request and attach the index value to the query string. For example, to get the value of the second card in the grid, the URL has to look like the `http://localhost/memory/grid?index=2`. The Memory-Grid-Servlet will response the card value of the given index in text/plain.

As being mentioned, the grid is a 2D list, that contains multiple lists of integers. Each Integer represents the card value at the given index. Therefore, to get the value for the given index in the GET request, it has to be translated into a 2D index (index in column and index in row). LST AB shows the method translateIndexToValue that is used to translate a single index into a 2D index which can be used to get the card value from the grid.

In addition, it is possible to set the operation mode, which decides if the grid is generated randomly or deterministic. This property is set as a init-param in

the `web.xml`, as illustrated by FIG BC. Then, the Memory-Grid-Servlet can read this value using `getInitParameter("mode")`.

3.2 Frontend