REI ASHIMI, KLEA PREGJA

23.02.2023

Assignment 2

THEATER WEB APPLICATION

**Table of Contents**

[Problem Statement 2](#_Toc128077977)

[Our Approach 2](#_Toc128077978)

[Database Schema 3](#_Toc128077979)

[Data Access Objects 6](#_Toc128077980)

[Controllers and Beans 8](#_Toc128077981)

[AJAX 8](#_Toc128077982)

[API 9](#_Toc128077983)

[User Interface 9](#_Toc128077984)

## **Problem Statement**

In this assignment, we were required to create a user interface for a theater business. Essentially, we were asked to create an interface through which anonymous users can see the various plays of the theater, the shows that are being put on display, and other important information such as the genre of the play, the list of actors and the number of halls of the theater.

Furthermore, a user can log in and reserve seats for any future shows, look at past reservations and update the information on their profile page.

## **Our Approach**

In this project we have made use of the IntelliJ IDEA, and we have configured it accordingly to the requirements of the project. We have implemented Tomcat Version 9 as the server for our web application. JavaServlet Faces (JSF) is used for the web tier, and the connection to the database is done through Jakarata EE’s JPA. For the database itself we have used Oracle SQL.

The APIs are developed through REST web services, and therefore they are all RESTful APIs. Through them users can retrieve all the information about a play or the information about a show. Furthermore, users can register on the website, and after the registration they can do all the functionalities described above – such as checking play and show information, reserving seats or changing their profile information. Finally, a user registered as an authorized user is the only one who can invoke our two APIs.

## **Database Schema**

A screenshot of a computer

Description automatically generated with low confidence

From the schema we can see the database that we have built for the purpose of this assignment, using Oracle SQL. There are several tables in our database, the explanation for which can be found below:

* **Customers:** This is where the information for the users is saved. The customers can enter their full name, the username they want to use, a password, their birthday, gender, and email address. They will be provided with a CustomerID which will be the primary key for this table.
* **Plays:** This table has the information for all the plays, including a PlayID, the title of the play, the writer, director and genre of the play. As for the actors we have decided to separate them into another table called **Actors.** This table simply has the actors name, and their connection with the **Plays** table is done through a third table called **Actor\_Plays** where we have taken the PlayID and the ActorID as foreign keys.
* **Shows:** As mentioned in the requirements, each play differs from the shows, because shows are actually being put in the stage. This is why shows have the PlayID as foreign key, but furthermore they have the date that shows the date and time when they are being played, and the number of seats occupied for that particular show. Lastly, the HallID is another foreign key used to identify the hall where the show is being shown.
* **Halls:** The halls have as primary key their HallID, and furthermore we have provided for the halls a number of rows and columns that will create the seat plan. The Halls table will also show the total number of seats available – and those occupied, as mentioned, are kept track of in the Shows table.
* **Reservations:** This is the table that keeps track of the reservations that registered users are able to do. It’s a simple table that only keeps its ReservationID as primary key, the number of seats reserved, and two foreign keys which are the ShowID for the show that is being reserved, and CustomerID for the customer that is booking the reservation.
* **Users:** This table is for the authorized B2B users who can log in with their username and password and are the only ones who can invoke the APIs – “Retrieve All Plays” and “Retrieve Show Details” APIs.

## **Data Access Objects**

Graphical user interface, text, application

Description automatically generated

In order to access the above database, we have Data Access Objects for each of the tables mentioned above – except for **Actor\_Plays** which as mentioned is just composed of the two foreign keys from **Actors** and **Plays**.

DAOFactory is the class that contains the getters for calling all DAO implementations; for example, *getHallDAO* that returns the Hall Data Access Object. All of the DAO Implementations implement the DAO<T> Interface. For example, **HallDAOImpl** implements DAO<Hall> and so on for all other tables. The DAO interface has five main methods that perform the basic functions of indexing, showing, storing, updating and deleting.

There is one extra method found in CustomerDAO Interface – which extendes DAO<Customer> - and that method is *authenticate,* which like the name implies, authenticates the users with their email and password. This method is also used for the UserDAO Interface, to authenticate the authorized users with their username and password.

The five basic functions all work as expected; *index* shows all of the entities of the database; *show* takes the primary key of an entity and then shows the appropriate Object that belongs to that primary key; *storing* creates a new object into the database; *updating* updates an already existing object of the database and *deleting* completely removes one of the objects of the database.

## **Controllers and Beans**

Graphical user interface, text, application, website

Description automatically generated

The managed beans could directly access the DAO Objects, however we have implemented the controllers as the ones to be accessed by the beans, instead of the DAO, therefore adding a layer in between the DAO and the beans. The five basic operations mentioned earlier – indexing, showing, storing, updating, deleting – are found in all of the Controllers and each one of them calls the respective DAO Model to perform them.

## **AJAX**

As an extra point in our project, we could add AJAX in our web application. We have added AJAX in our *.xhtml* forms, since the JSF tag, <f:ajax> can handle AJAX calls. An example of this in our code is for example when we are submitting the login form, with the components that are included in the AJAX request – username, password – which are also the ones that need to be updated after the AJAX request.

Text

Description automatically generated

## **API**

The APIs in this application are developed through RESTful web services. Essentially, we had to develop two APIs – “Retrieve All Plays” and “Retrieve Show Details”. Only an authorized B2B user, after checking the credentials, can invoke the two APIs. The authorized user therefore, for the first API, can authenticate with their username and password, which should be passed as header params in the request, and then will be able to get as output all the details of all of the shows that are available in our application.

As for the second API, the user does not have to just provide their log in information in the header params of the request, but also the show ID, as path param, in order to get the information about the particular show that the user is looking for. The output will of course be the information about the show – such as date and time, play title and the number of seats available for the show.

## **User Interface**

The first view that the user sees when they open the web application is the home page, which contains information about the theater. This information is available for all user, regardless, if they are logged in or not.

Firstly, we show the halls of the theater, the seat plan of the halls – therefore, the number of rows and columns – and finally the total number of seats in each hall. For each hall we show the show that is currently playing on that particular hall. On top of this navigation bar, the user has the possibility to click the buttons of “View Plays” and “View Shows”. If the user clicks on one of them, they will be linked to their respective views.

Table

Description automatically generated with medium confidence

In the *plays.xhtml* view, the user can see all the information about the plays; the title, writer, director, genre and a list of actors. In the *shows.xhtml* view, we once again get the play title, and the information about the show – the hall where it is displayed, the date and time when it will be shown as well as the total seats of the hall.

Graphical user interface, application, table

Description automatically generated

In the *shows.xhtml* view, the user is also able to filter all of the shows, by the date and time they are being played, so that we only show the shows on a particular date.

Graphical user interface, application

Description automatically generated

The user can register themselves on the website, by clicking the Register button and being sent to the *register.xhtml* view. Here the user has to complete a form before registering, entering their username, full name, birthday, password, email, gender.

Graphical user interface

Description automatically generated

All of this information is required before the user can submit the form, and then this information is stored in the database, and the new user has the possibility of logging in. If the information is not entered, error messages will appear as seen below:

Graphical user interface

Description automatically generated

By clicking on the Log In button the user is transported to the *login.xhtml* view, where the user has to provide the correct username and password in order to be logged in and continue with the functionalities of the logged in user.

Graphical user interface, website

Description automatically generated

If the user does not input the correct username and password, error messages are displayed, as seen below:

Graphical user interface

Description automatically generated

After being authenticated the user is sent back to the *home.xhtml* view, however this time on the top right we can find the Profile displayed, as well as the Log Out button.

Graphical user interface, application, table

Description automatically generated

If they click on Profile, the user is sent to the *profile.xhtml* view, where they can update the information that they input in the registration form, except for the username. By saving the changes – clicking the Update button – they are sent back to the *home* view. Alternatively, they can click on the Cancel button and not save any of the changes.

Graphical user interface

Description automatically generated

Furthermore, the user also has the possibility to make a reservation in the *make-reservation.xhtml* view. They can enter this view from the *shows* view, where they can press on the Reserve Seat button. In this view the users are able to see the play title, hall name, show date and time, the number of total seats and they are able to reserve up to five seats. After making the reservation they are sent back to *shows* view and the number of free seats in the database will be changed. If the user is not logged in but presses the Reserve Seat button, they are redirected to the *login.xhtml* view.

Graphical user interface, table

Description automatically generated

Moreover, a user can browse past reservations in *reservations.xhtml* view, by clicking on the View Reservations button. The registered user can also filter their past reservations by date and time, just like they can filter the shows.

Graphical user interface, application

Description automatically generated

Finally, the registered user can log out by clicking on the Log Out button on the top right of the page. After logging out, the user goes back to the *home.xhtml* and sees the first page as it was described in the beginning of this section.