Advanced Java

Assignment 2
Project Report

Kristi Lamaj
Paris Xhetani

February, 2023

Table of Contents

Technologies

Our Advanced Java Course Project has been a challenging but rewarding experience for us. Throughout the project, we utilized several technologies including JSF, JakartaEE 9.1.0, GlassFish 6.1.0, and MySQL. JSF proved to be an excellent choice for our web application framework, as it allowed us to easily create dynamic and interactive web pages. JakartaEE 9.1.0 provided a robust and reliable platform for our backend processing. GlassFish 6.1.0 served as our application server and provided seamless integration with our chosen technologies. Finally, MySQL proved to be an excellent choice for our database management needs, allowing us to store and retrieve data quickly and efficiently. Overall, we are proud of our project and the technologies we utilized to bring it to life.

Code Explanation

Models:

We have two classes in our models package: play_model and show_model. The play_model class has five private attributes: id, title, writer, director, and actors. It also has getter and setter methods for each attribute, which allow us to retrieve and modify their values. Similarly, the show_model class has five private attributes: id, showDate, showTime, playTitle, and hallName. It also has getter and setter methods for each attribute, which allow us to retrieve and modify their values. These classes provide a basic structure for creating objects that represent plays and shows, with attributes such as ID, title, writer, director, actors, show date, show time, play title, and hall name that can be accessed and modified as needed. Both classes can be used in a theater management system to keep track of information related to plays and shows, such as scheduling, cast and crew, and venue details.

Beans:

We are defining several Java classes in a package called "bean". These classes are used in a web application that allows users to view and search for plays and shows. The first class is called "show_bean" and is annotated with "@Named" and "@ViewScoped". This class implements the "Serializable" interface and has several private variables, including a list of "show_model" objects and a string called "searchTerm". There are several getter and setter methods defined to manipulate these variables. In the "filterData()" method of the "show_bean" class, we are filtering a list of shows based on a search term. We loop through each show and check if its attributes (such as date, hall name, show time, and play title) contain the search term. If a show matches the search criteria, it is added to a new list called "filteredShows". The "get_show_by_id()" method of the "show_bean" class retrieves a list of shows from a RESTful API based on a provided play ID. This method uses the "Client" class from the "jakarta.ws.rs.client" package to make HTTP requests to the API and retrieve the data in JSON format. The "redirect_play_bean" class is used to redirect the user to a specific play page or to the login or signup page, based on the user's selection. This class has several methods that construct the appropriate URL and redirect the user to that URL using the "FacesContext" class. The "play_bean" class is used to display a list of plays and search

for plays based on a search term. This class has a private variable called "searchTerm" and several getter and setter methods to manipulate this variable. In the "filterData()" method of this class, we are filtering a list of plays based on a search term, similar to the "filterData()" method in the "show_bean" class. The "getPlays()" method retrieves a list of plays from a RESTful API and uses the "Client" class to make HTTP requests and retrieve the data in JSON format.

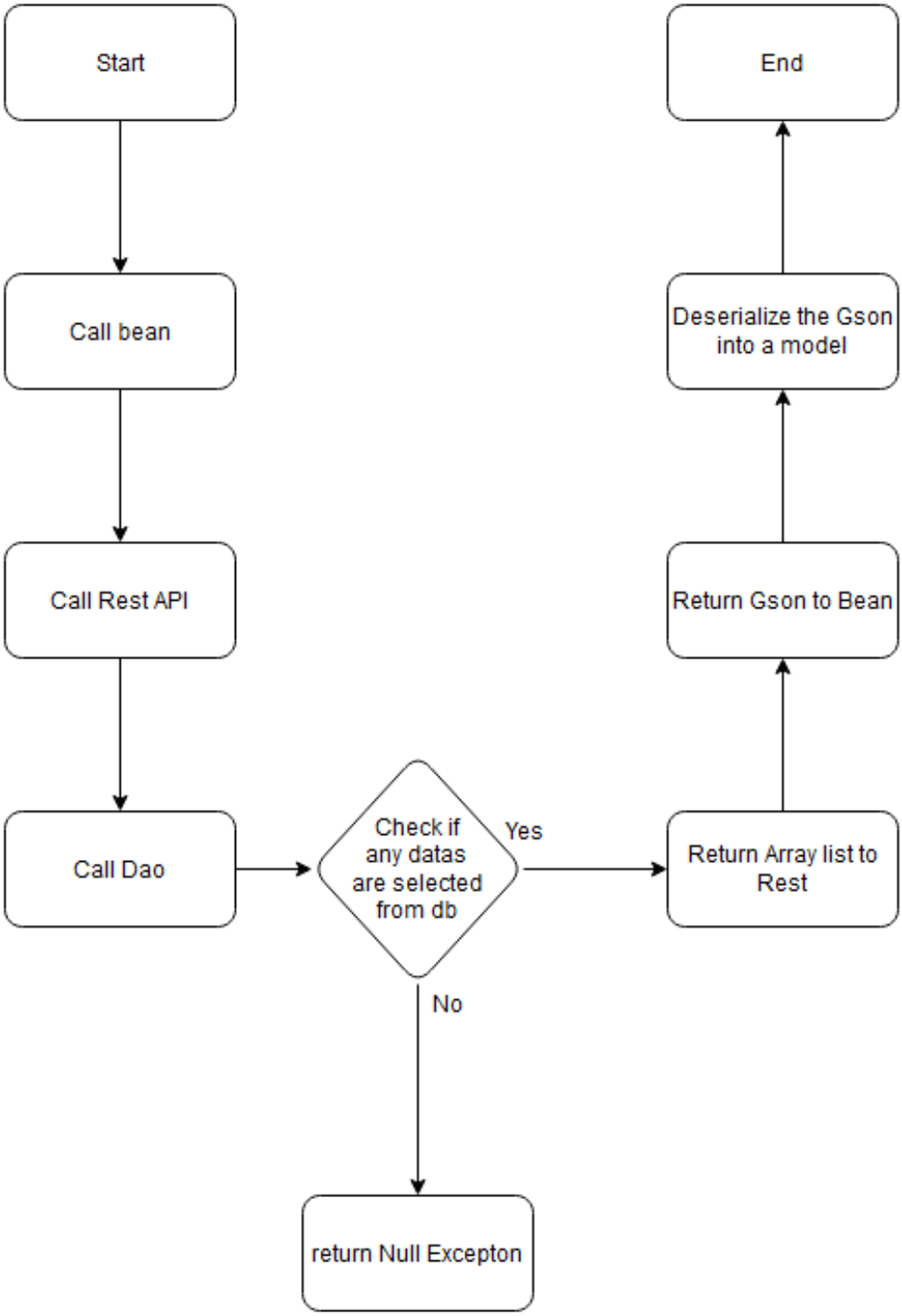
DAO:

We have three classes in the dao package: login_dao, play_dao, and show_dao. The login_dao class has a method called login that takes in a username and password as parameters. Inside the method, it establishes a connection to a MySQL database and retrieves a result set of users matching the inputted username and password. If the result set contains a matching user, it sets a session attribute for the username and returns true. If there is no matching user, it simply returns false. The play_dao class has a method called getAllPlays that retrieves all the play entries from a MySQL database and returns them as a list of play_model objects. Each play_model object contains fields for the play's id, title, writer, director, and actors. The show_dao class has a method called get_shows_by_play_id that takes in a play_id parameter and retrieves all the show entries that match the given play id from a MySQL database. It returns the show entries as a list of show_model objects. Each show_model object contains fields for the show's id, play title, hall name, show date, and show time. All three classes establish a connection to the same MySQL database using the same credentials. They also use try-with-resources to automatically close the connection and statement objects.

REST API:

We have four Java classes in a package called rest_apis, each representing a REST API endpoint for different functionality. The first class, login_rest_api, handles user authentication. When a GET request is made to this API, it expects two parameters: username and pass (password). It then passes these parameters to an instance of login_dao (from another package), which checks if the username and password match any user in the database. The API returns a JSON response indicating whether the login was successful or not. The second class, play_rest_api, retrieves all plays from the database using an instance of play_dao (from another package) when a GET request is made to this API. It then returns the list of plays as a JSON response. The third class, show_rest_api, retrieves all shows for a given play ID when a GET request is made to this API. It expects a parameter play_id in the request, which it passes to an instance of show_dao (from another package) to retrieve all shows for that play ID. It then returns the list of shows as a JSON response. The fourth and final class, test_api, is a simple endpoint that just returns the string "Hello, world!" when a GET request is made to it. This is useful for testing purposes. In summary, these classes implement different REST API endpoints for user authentication, retrieving all plays, retrieving all shows for a given play ID, and a simple test endpoint.

(Bellow there is an image that shows how all these components are connected and flow with each other.)



JSF Files:

This code contains three files that are part of a web directory. In the first file, we define an XHTML page that includes various JSF components such as forms, input text, and buttons. It also uses several custom tags from the JSF library. The purpose of this file is to display a table of data and provide a search function that filters the data based on user input. Additionally, the file includes two buttons that link to login and signup pages. The second file defines an XHTML page that contains a login form. The form has two input fields for the username and password, and a submit button. When the user clicks on the login button, the form triggers the login method in the corresponding bean. The third file defines an XHTML page that contains a search bar and two buttons. The search bar lets the user search for content, and the two buttons redirect the user to the login or signup pages. This file uses PrimeFaces, a popular JSF component library, to style the search bar and buttons.

Lib/Libraries:

The libraries that we used for our project include common-primeface-utilities-1.0.0.jar, gson-2.8.9.jar, jackson-core-2.13.4.jar, jackson-databind-2.13.4.2.jar, jakarta.jakartaee-api-9.1.0.jar, and mysql-connector-java-8.0.30.jar. These libraries provide us with essential utilities for our project, such as JSON parsing, database connectivity, and server communication. We also have a lib folder that contains various javax jars, including javax.annotation.jar, javax.ejb.jar, javax.jms.jar, javax.persistence.jar, javax.resource.jar,javax.servlet.jar, javax.servlet.jsp.jar, javax.servlet.jsp.jstl.jar, and javax.transaction.jar. These jars are important for implementing Java EE features in our project, such as Enterprise JavaBeans, Java Message Service, and Java Persistence API. Overall, these libraries and jars play a crucial role in our project's functionality, and we rely on them heavily to ensure that our application runs smoothly and efficiently.

MySQL:

The Bookings table has a composite primary key consisting of three foreign keys: IDShow, IDUser, and IDSeat, along with two additional columns, RowSeat and Column. These foreign keys establish a relationship between the Bookings table and the Shows, Users, and Seats tables.

The Halls table has a primary key column, ID, and additional columns including Name, Raws-nr, and Columns-nr. The Shows table has a foreign key column, Hall_ID, that references the ID column in the Halls table. This establishes a one-to-many relationship between the Halls table and the Shows table, where a single Hall can be associated with multiple Shows.

The Plays table has a primary key column, ID, and additional columns including Title, Writer, Director, and Actors. The Shows table has a foreign key column, Play_ID, that references the ID column in the Plays table. This establishes a one-to-many relationship between the Plays table and the Shows table, where a single Play can be associated with multiple Shows.

The Shows table has a primary key column, ID, and additional columns including Show_Date, Shot_Time, Play_ID, and Hall_ID. The Bookings table has a foreign key column, IDShow, that references the ID column in the Shows table. This establishes a one-to-many relationship between the Shows table and the Bookings table, where a single Show can have multiple Bookings.

The Users table has a primary key column, ID, and additional columns including Name, Username, and Pass. The Bookings table has a foreign key column, IDUser, that references the ID column in the Users table. This establishes a one-to-many relationship between the Users table and the Bookings table, where a single User can have multiple Bookings.