Paul Pezold

Advanced Applications

Design Document

Brief Description:

Within this design document, I will only talk about the diary component of this project. As I developed it, I refined my design to be a Trader's Diary. It is a regular diary with added features specifically for stock traders. Several components of the project, including those found in the Market Resource section as well as their repo files and view classes, were added on to this project for a project in a different class. For the purposes of this assignment, only the diary and user files are relevant, though all of the files included are functional.

Architecture:

This web application's architecture follows the MVC pattern—Model, View, and Controller. My model classes are all found within the Model package. The Servlets package contains all my web servlets, which act as the controller classes, handling routing, business logic, and database calls. The view classes are all found in the Webapp package. The majority are in View-JSP, along with some CSS files in the Styles package. The three technologies covered in class that I implemented in this project are JSP, which I used for the View, Servlets, and JDBC which I used to connect to a MySQL database. My project also makes use of some of the latest web technologies including Tomcat 10, Bootstrap 5, and Maven. I refer to any files that interact directly with the database as repos; they are located in the Repo package.

The general data flow:

1. User navigates to index.jsp view page (home)
2. They click "Start Journey" to create a new user, or "Continue Journey" to log in
3. From the login form, they submit a username and password to the login servlet

4. In general, the servlet will instantiate the appropriate repo file, and use that class's methods to get any necessary data needed to perform business logic or CRUD operations, in this case validation
5. If valid, the servlet forwards the browser to ListEntry.jsp view file
6. From there, the user may use the header links to navigate to various jsp pages, utilize the forms and their respective servlets/repos, or log out.

Database:

This project makes use of MySQL. The appropriate connector is found as a Maven dependency within the pom.xml file. Using MySQL workbench, a file called BuildAA_Final can be executed to create the project schema along with some sample data. It is entitled db_final_project. To configure this connection, a JDBC_Connection class exists within the Utils package to simplify configuration for a new user. That class also contains some helper methods for parsing date objects from MySQL and Java. If you want to use the Market Resource section, see the video for additional configuration details.

Model Overview:

There are three models in the package. In general, all models have a no-args constructor in addition to any other constructor required. All fields are private to follow best practice; getters and setters are implemented so that the classes are encapsulated. All models also contain a toString() override to aid in testing. The models DiaryEntry and User are both represented verbatim in MySQL and have the same attributes as the models have. An Integer value serves as the id and primary key in each class.

Diary Model:

The DiaryEntry class implements Comparable so that diary entries can be sorted. It therefore also contains an override compareTo() function that compares two the entries' date fields. It also contains a string attribute for title, description, user_name, and contents, along with a few other relevant fields. It uses the DiaryServlet.

Login Model:

Login is a simple utility class that implements Serializable so that the database can maintain the ID for you. It has only two private fields, the string username and the string password, which are assigned whenever you construct a login object. It makes use of the LoginServlet and is used to validate users at login time.

User Model:

A standard user with standard fields.

Repos:

The Repo package contains what I call repo classes, which I think traditionally may be referred to as DAO classes. They interact directly with MySQL database to perform CRUD operations. I included one interface to practice the design technique of loose coupling, implemented in Entry_Repo, which retrieves or otherwise modifies entries from the MySQL database pertaining to DiaryEntry objects. Login_Repo retrieves username and password information from the database for validation. Finally, User_Repo retrieves/manipulates a user object from the database.

Servlet/Controller Overview:

DiaryServlet:

The servlet classes are where all of the business logic is contained. The DiaryServlet contains the logic that applies to the diary entries. It works by instantiating an Entry_Repo object in the inti() method so that the servlet can access the database. It contains the other standard servlet override methods as well, most prominently the doGet() method. It extracts the last portion of the URL, runs it through an if-else block, and performs an action accordingly. For example, if the URL contains the keyword delete, it will execute the delete command from the repository using a prepared statement and the session 'id' parameter. Within doPost(), logic for updating the database is contained. Helper methods are used to process the data and then forward that data either back to the original view page or to the new, appropriate view page.

Login & Logout Servlet

These servlets make use of the User_Repo and simply validate the user and log them in or out, creating the session or ending it, respectively.

UserServlet:

Modifies the current user object attributes.

SearchServlet:

The only unique feature of this servlet is that it uses the LIKE operator in a prepared statement to search for the submitted string within the content of a user's diary entries. The search algorithm is not very sophisticated and would need to be optimized if the user had thousands of entries.

View Overview:

All View files live in the webapp package. This folder contains all the files that are displayed in the web browser. Technically all files not in the web info package are visible to users in the browser; however, for the purposes of this app, I decided not to put them all in the web info folder for simplicity. The Components folder contains reusable components that are included using scriplets in other JSP files. From within the View-JSP package, the Error.jsp file is displayed if any URL other than the correct one is entered into the browser search bar. The footer contains a simple copyright and is anchored at the bottom of all pages. The verified-header.jsp is applied only to pages after a successful login attempt. It contains links that allow the user to navigate to the various pages like SearchEntries.jsp or ListEntries.jsp.

All files within the DatabaseSearch package apply to the other project. The files in View-JSP are essentially the forms for updating, changing, or viewing a user's information. The diary form is the JSP file that allows the user to create new entries. The ListEntry file is the file that displays all the entries; it is the default home page upon successful log in. The true home page is index.jsp. From it, whenever you click on the “Continue Journey” button, you are redirected to the Login view. The Search view feature allows you to search from all of the entries that a user has in the database. Styles-header is simply an import of the link to the Bootstrap CDN, which this project was built with. The UpdateUser form is what a user uses to change his or her personal information. The web.xml file was simply used to dress up the URLs that are displayed in the browser, mapping requests from the view page to the appropriate servlet.

Styles package:

Contains a little custom styling, found within css files. It is used to create the button expanding effects as well as to toggle the display on the side menu for the “Market Resources” section.