Please unzip (Tickets\_SP20) and test run the program zip file. It is highly recommended to test the code after reading this doc file entirely first to help think through logical steps of actions from start to finish. Check suggested comments in various areas in the code to help finish your code work. Test the program any way you like. A helpful flow of things will be explained shortly.

Remember: it is your code and the database tables have been setup for testing. Feel free to modify the code and any tables you will create. Existing code can be entirely used to build your project upon or you can merely use some of the code provided as a useful snippets for your own build if you desire.

|  |  |
| --- | --- |
| File structure shown on the right is from the zip file which includes start up files that will be explained below. |  |

Each file in the project plays some significant role and will be explained below. Check over suggestions in **red, bold and in borders**, that suggest needed accomplishments or investigation.

**DAO**.java file

The Dao.java can do a lot of the CRUD work for you with methods you define inside the file.

So far functionality is described as follows:

A **getConnection()** method is defined and useful to call anytime, anywhere in your program to have a connection to your underlying database.

Call it anytime to work some query statements. Examples follow..

Connection connect = dao.getConnection(); or …

Statement statement = dao.getConnection().createStatement();

A **createTables()** method that creates a users table and tickets table. Supplied for you are queries that create two tables, **jpapa\_tickets** and **jpapa\_users**. These tables can be used as “shareable” tables for practice. Feel free to use/adjust the supplied query statements to include not only your own UNIQUE table names but any desired *additional* fields if you desire so your own tables are created appropriately.

\*\*\***For turning in finalized work, of course please adjust any table name in any of the files to your own names to reflect your own works**. Otherwise points will be diminished as a result if not.

Finally incorporated in code, is an **addUsers()** method that is called at the end of createTables() method that takes the file you see in the snapshot above called **userlist.csv**, and inserts each record into the users table. You can use this feature if you like or something else to get some users into your **users** table. Modify the user list if you like that consists of a username (Col. 1) and password (Col. 2) to names your familiar with. **Make sure also no matter what, to add at least your name in and a password to your users table.**

**Finish up with Dao.java file with some CRUD methods of your own, like a delete function and an update function. Call these functions where deemed appropriate in your file(s).**

**LOGIN**.java file

This is your starter file which *authenticates* users both admin and regular users.

For the first time you run your app, please enter in the hard coded admin credentials in the file namely **admin** and **admin1** respectively for the user name and password. After the admin is authenticated the Tickets window appears and any of your tables are created if necessary.

Further check out the conditional logic for the loginButton eventHandler which nicely checks over credentials and *passes* the user name of a regular user (verified against the user table using a prepared statement) or *denotes* a user as “Admin” in the Tickets.java window that triggers on a successful login. Any unsuccessful login fires up a label message. Try it out!

Of course once you know your users table has been created or your testing the sample users table out, feel free to login with the credentials listed in the userlist file or just

view created users by connecting to the papaserver via phpMyAdmin or MySQLWorkbench to gain an all access pass to table record data!

**TICKETS**.java file

Run thru some of the menus that have event handling put in place already for various sub menus. For example Open a ticket and View a ticket. Then exit. These should all work. But wait.

Should a regular user for example see everybody’s tickets or just there own? **Adjust for that.**

Also should a regular user see the Admin menu or its sub menu features? Not at all. **Adjust for that as well.**

A quick and simple suggestion that would differentiate admin menu options versus a regular user’s menu options would be to block off or set menu option visibilities to off if the user is not an admin.

**Finish coding the sub menus in the actionPerformed event where you see comments towards the end of the event handler namely where it says …**

 /\*

\* continue implementing any other desired sub menu items (like

\* for update and delete sub menus for example) with similar

\* syntax & logic as shown above\*

\*/

Note coding sub menu items in the event handler can follow any order. Order in other words does not matter, what matters how you handle events fired by a menu selection.

The main thing when coding any menu items is to just use the name of your sub menu object you create or that has been created already in code

ex. for the following

else if (e.getSource() == someMenuItem)

you would include for someMenuItem, your desired sub menu object name and code in your action for when the user chooses that particular menu item.

Finally a nice little goodie or “easter egg” has been provided in the file in your listener event handler. The code shown below retrieves the id of any id that was just “auto incremented” as a result of an insert!! Very cool and useful indeed. You can show the auto generated id to the console or via a dialog message box as hard coded in and/or use it for some query.

Example use case: take the ticket id number generated and add that id on some insert to some other table along with any data for later referencing. Example code logic follows….

// retrieve ticket id number newly auto generated upon record insertion

ResultSet resultSet = null;

resultSet = statement.getGeneratedKeys();

int id = 0;

if (resultSet.next()) {

id = resultSet.getInt(1); // retrieve first field in table

}

// display results if successful or not to console / dialog box

if (result != 0) {

System.out.println("Ticket ID : " + id + " created successfully!!!");

JOptionPane.showMessageDialog(null, "Ticket id: " + id + " created");

} else {

System.out.println("Ticket cannot be created!!!");

}

**TICKETSJTABLE**.java file

Nothing to do here – everything works automatically pulling your metadata and placing that into the JTable to set up a column structure and the grabbing of row data is accomplished automatically as well. Unless you want to “tweak” the table that is up to you. Check this site for awesome tweaks for sure!

<https://docs.oracle.com/javase/tutorial/uiswing/components/table.html>

**After your code work is pretty much complete, please include a 3rd table of choice and what interaction you would like that table to contain. Perhaps run some report to show verification of table data from your new table. Example tables can be a location table, ticket history table, some table to hold analytics (current vs. past?) for quick queries, etc.**

Finally if your using the starter source, please check over all the files and make sure table names are correct for all queries, selects, creates, deletes, updates, etc. Table uniqueness is very important for your grade so make sure ample fields are included, good naming conventions and data types are adhered too, etc. Point deduction will be given for a lack of table uniqueness and quality.

Other “great” references that can be useful for your UI:

<https://docs.oracle.com/javase/tutorial/uiswing/components/list.html>

<https://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html>

Best of luck!