Assignment 03: Database Communication

Overview:

During the previous assignments you set up the View (jsp pages) and Controller (servlets), this assignment is intended to further expand on the MVC approach by focusing on the Model. To this end, a simple database will be incorporated for which classes that communicate with the database will be developed. As the overall goal of this project is to develop a fully functional application, it is imperative that previous assignment requirements were met. Therefore, the requirements described in this document will overlap with previous assignments.

Assignment Description:

Create a web application that uses servlets as the controller component, jsp pages as the view component, and classes as the model component to create and interactive card trading experience for the user. The following is an outline of the pages and servlets that will make up the web application as a whole.

1. A java class for the session bean
2. **A java class to provide connection to the database (model in MVC)**
3. **A java class to check login credentials and register individuals (model)**
4. **A java class to enter and obtain card collection info to and from the database (model)**
5. **A java class to obtain available cards from the database (model)**
6. A jsp page for user login
7. A servlet that handles login (e.g., LoginControllerServlet.java)
8. A jsp page for new user registration
9. A servlet that handles registration (e.g., RegistrationControllerServlet.java)
10. A servlet to serve as the controller for the cardTrader.jsp (e.g., CardTradingControllerServlet.java)
11. A jsp page for card collection and available cards (e.g., cardTrader.jsp)
12. A jsp page for the transaction (e.g., TransactionDetails.jsp)
13. A servlet to handle the transaction (e.g., TransactionControllerServlet.java)
14. A servlet to handle logout
15. **Make sure you have the mysql-connector in your WEB-INF lib folder (see example img)**

A few items to note:

1. When your program runs, login.jsp must be the page that loads by default (i.e., don't have to manually put login.jsp in the url).
2. Java class for the session bean. Make sure you use encapsulation and at a minimum include the following with the appropriate getters and setters:
    - userName
    - message
    - cardCollection
    - cardsForSale
    - **userPrimaryKey**

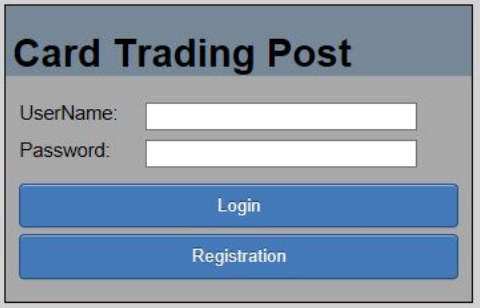
3. **model.Database_Connect.java**
    - Create two methods in this class. The first should try to connect to a local database (XAMPP is recommended for a local phpMyAdmin) and the second should try to connect to your remote database (as deployed on Google). If the local connection is not available, call the remote method.
4. **model.User_Login_Register_Info.java**
    - Obtain connection
        - (e.g., Connection con = model.Database_Connect.Connect2LocalDB();)
    - At a minimum, provide methods to accomplish the following (you must use prepared statements):
        - Check if userName is in database (can be used to check for new registration or if a user tries to log in with a non-existent userName)
        - Check if a userName password combo is in the database.
            - Option 1 - Pass in the provided userName and userPassword to a method that returns a boolean
            - Option 2 - Create a method that returns the password based on the username. In the controller, check if the returned password matches the user provided password.
        - Obtain userPrimaryKey
        - Enter new userName / userPassword into database
        - The database table should have the following at a minimum

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	**userPrimaryKey**	int(11)			No	*None*	AUTO_INCREMENT
2	**userName**	varchar(20)	latin1_swedish_ci		No	*None*	
3	**userPassword**	varchar(20)	latin1_swedish_ci		No	*None*	

5. **model.User_Collection_Info.java**
    - Obtain connection
        - (e.g., Connection con = model.Database_Connect.Connect2LocalDB();)
    - Enter, update, obtain and return cards as stored in the database. Queries should be based off the user's primary key which is stored in the session. All communication with database must be done with prepared statements.
        - One option is to place the all of an individual's cards directly in the bean as they are obtained from the database.
        - The database table should have a minimum of the following (note some of the types are set to varchar(xx) - you are not required to match types):

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	**PrimaryKey**	int(5)			No	*None*	AUTO_INCREMENT
2	**userPrimaryKey**	int(11)			No	*None*	
3	**cardName**	varchar(100)	latin1_swedish_ci		No	*None*	
4	**cardNumber**	varchar(10)	latin1_swedish_ci		No	*None*	
5	**cardValue**	varchar(10)	latin1_swedish_ci		No	*None*	
6	**amountOwned**	varchar(10)	latin1_swedish_ci		No	*None*	

6. **model.Available_Cards_Info.java**
    - Obtain connection
        - (e.g., Connection con = model.Database_Connect.Connect2LocalDB();)
    - At a minimum, provide methods to accomplish the following (you must use prepared statements):
        - Return available cards as stored in the database
7. **login.jsp**
    - Create / obtain session bean object with the scope of session
    - User enters a username/password, clicking the submit button on the form should lead to the LoginControllerServlet.java.
    - Clicking the registration button or link that looks like the login button should lead to the registration page
    - Below the title / above the username, create a placeholder with something like bean.getMessage().



8. **LoginControllerServlet.java**
    - Obtain the session bean object
        - HttpSession session = request.getSession(false);
        - SessionBean sessionInfo = (SessionBean) session.getAttribute("bean");
    - Obtain the username and password
    - **Call the appropriate method[s] in model.User_Login_Register_Info.java (see #3) to check the validity of the userName / Password combo.**
    - If the combos are equal, put the pertinent information into the session bean (e.g., userName, **userPrimaryKey**) and forward on to the CardTradingControllerServlet.java
    - If they do not equal, put an unsuccessful message into the session bean, send the user back to the login page, and have the mssg show up under the title.
        - Once login.jsp loads the mssg, set the mssg to ""
            - This can be tested by reloading the page - the mssg should go away

9. **registration.jsp**
    - Create / obtain the session bean object
    - Create a simple registration form (username, password, confirm).
    - Below the title, create a placeholder with something like bean.getMessage(). This will be used in #10
    - Clicking the Register button form should lead to RegistrationControllerServlet.java.
        - Accomplish this as a form action or with javascript.

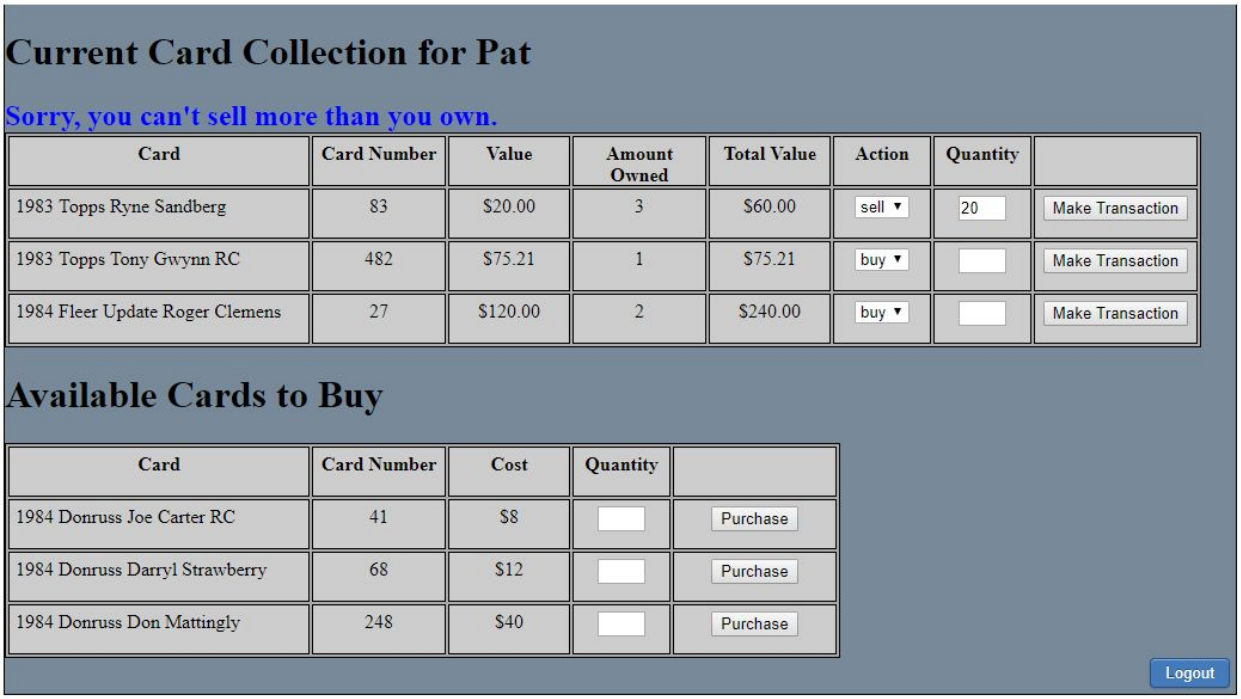
10. **RegistrationControllerServlet.java**
    - Create / obtain session bean object
        - HttpSession session = request.getSession(false);
        - SessionBean sessionInfo = (SessionBean) session.getAttribute("bean");
        - The bean should have been created at the login.jsp but this handles the issue if someone navigated directly to registration
    - **Check that their provided userName or Password/confirmPassword was not null or empty and that the Password/confirmPassword match.**
        - If null or empty or no match - send them back to registration with mssg
            - The mssg should go away when the page reloads
    - **Check that the username is not already in the database by using a method in the model.User_Login_Register_Info.java See #4**
        - If the userName is already taken
            - Put their username into the mssg stating the username is taken
                - Send them back to registration.jsp and have the mssg show up under the title.
            - Once registration.jsp loads the mssg, set the mssg to ""
                - This can be tested by reloading the page - the mssg should go away
        - **If the userName is available**
            - Use a method in **model.User_Login_Register_Info.java** to enter the userName and userPassword into the database
            - Use a method in **model.User_Collection_Info.java** to enter the following as sent from the RegistrationControllerServlet
              userPrimaryKey, "1983 Topps Tony Gwynn RC", "482", "75.21", "1"
              userPrimaryKey, "1984 Fleer Update Roger Clemens", "27", "120.00", "2"
              userPrimaryKey, "1983 Topps Ryne Sandberg", "83", "20.00", "3"
              (each of these should be a separate entry into the database card_collections table that has 5 columns - PrimaryKey, userPrimaryKey, cardName, cardNumber, cardValue, amountOwned)
            - put the username and userPrimaryKey into the session and forward them on to CardTradingControllerServlet.java

11. CardTradingControllerServlet.java

- Obtain the session bean object
- If the userName is not in the session, send them do login.jsp. If it was:
  - Call a method in **model.User_Collection_Info.java (#5)** to obtain the current card collection of a single user as stored in the db.
    - Note - one approach is to put the database results directly into the bean as they are acquired in the User_Collection_Info.java method. Another is to have them all returned by the method to the CardTradingControllerServlet.java. I prefer doing it in the User_Collection_Info.java
  - Populate the session bean (either in the User_Collection_Info.java or in User_Collection_Info.java) with the obtained card collection.
    - setCardCollection(.....) *** this is setting what is returned from the db into the cardCollection HashMap
  - Call the method in **model.Available_Cards_Info.java (#6)** to obtain the available cards as stored in the db.
  - Populate the session bean with obtained available cards (e.g., put them in the cardsForSale HashMap)

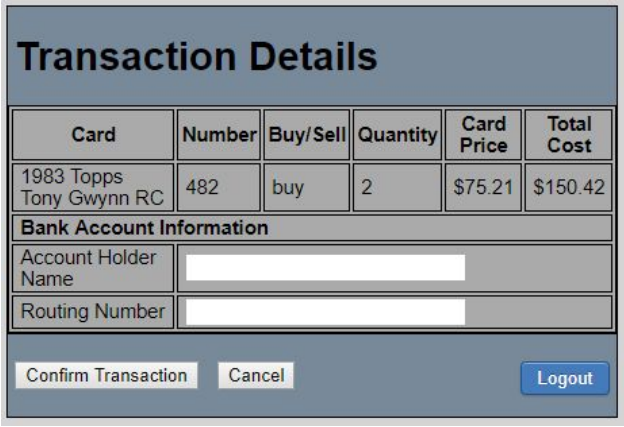
12. cardTrader.jsp

- Obtain the session bean object
- When the jsp loads, include the username with the title
- The cardTrader.jsp must list the current card collection (as represented in the db) and the available cards to buy (as represented in the db).
  - Should/could have been built in CardTradingControllerServlet.java or a separate class that specifically handles the creation of forms
  - Note - the forms / form data cannot be made statically.
- The action drop down box on the page should include "Buy" and "Sell".
- Clicking the "Make Transaction" button or "Purchase" button should lead to the TransactionControllerServlet.java and send the following:
  - Card name
  - Card number
  - Value
  - Amount owned
  - Action (buy or sell)
  - Quantity
  - Hint the use of hidden fields really helps
    - <input type="hidden" name="cardName" value="<%= cardName %>">
- All fields in the image should be included (order/location replication not required)
- Under the title, put messages about incorrect transactions as handled by the TransactionControllerServlet.java - e.g., Sorry, you can't sell more than you own.



13. TransactionDetails.jsp

- Provide a summary of the purchase which includes all fields shown in the image.
- Clicking the "Confirm Transaction" submit button should lead to the TransactionControllerServlet (#14) and pass the necessary data.



14. TransactionControllerServlet.java

- Obtain the: cardName, cardNumber, cardPrice, ownedCards, buySell, quantity
- Check if any of the previous was null or empty - if so, send back with a message (message goes away on refresh of page).
- Check if they are trying to sell a negative quantity or sell more than they own. If so, send back with a message (message goes away on refresh of page).
- Check if IfReferredBy came from the cardTrader.jsp or the transactionDetails.jsp (see code example at end of this document).
    - If from cardTrader.jsp
        - Make appropriate actions/calculations and then redirect to transactionDetails.jsp.
    - If from transactionDetails.jsp

- Make appropriate actions/calculations/updates (e.g., update the session bean HashMap)
- Use a method in the model.User_Collection_Info.java to update the purchase or sale of cards in the database.
    - Note - you may need to update the amount of cards to an existing entry rather than a new entry (e.g., already own the card - update amount. New card - new entry)
- Rebuild the form based on what is in the db (you might have done this in the CardTraderController.java or you might have created a separate form class).
- Redirect to cardTrader.jsp
    - From this point, cardTrader.jsp should show the updated purchase or sale.

15. LogOutServlet.java
    - Used whenever someone clicks the Logout link / button.
        - **Clear the session with invalidate()**
        - Redirect user to login page.

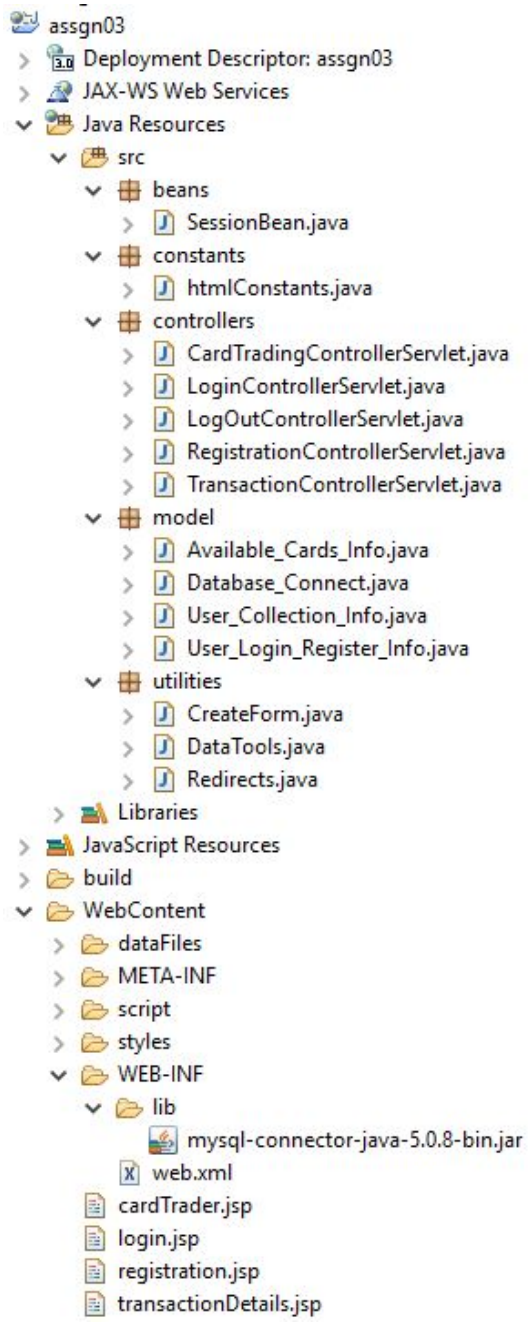
Assignment Submissions:

Your completed application must be loaded and available for public viewing on your Google server.

What to submit:

1. **HW3.war** - An archive of the entire web application (project) stored in a standard war file. When creating the war file **you must include your java source files**. This war file will be imported into Eclipse for grading.
2. **info.doc** - Document with the following assignment information:
    - Link to your Login page on your Google server
    - Explanation of status and stopping point, if incomplete.
    - Explanation of additional features, if any.
3. **Database** - download your database and submit with assignment. Note - make sure it is your entire database and not just a single table.

Workspace Example:



IfReferredBy(request, "registration.jsp")

```
public static boolean IfReferredBy(HttpServletRequest
request, String cameFrom)
   {
      if(request.getHeader("Referer")==null)
      {
         return false;
      }
      else
      {
         String Referer = (String)request.getHeader("Referer");
         if(Referer.contains(cameFrom))
         {
            return true;
         }
         else
         {
            return false;
         }
      }
   }
```

```
public static boolean NullOrEmpty(String input)
   {
      if(input==null||input.trim().equals(""))
      {
         return true;
      }
      else
      {
         return false;
      }
   }
```