Assignment 4
JSP, Servlet, and MVC
Database Integration
Points: 100

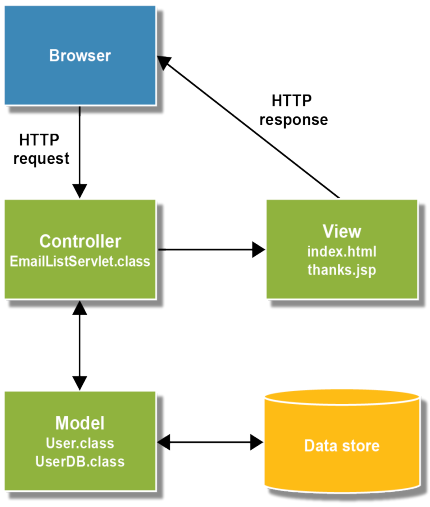
This assignment is to be done by the previously defined teams of two students. If you have not done so, you should establish a sensible way to share your project development work. You **MUST NOT** make your assignment code available publicly, so do not use a public repository service such as GitHub – any such repository must be private.

Assignment Description:

This assignment is intended to familiarize you with using databases to persist the data in your web application.

In this assignment you will replace the hard-coded "database" with an actual database, which entails some modifications to your previous JSP/Servlet MVC web application, according to the following specifications:

1. Correct any errors identified or partial/missing functionality from the previous assignment.
2. All structure, design, and content requirements from previous assignments are mandatory, unless explicitly updated in this assignment description.
3. Use JavaBeans to implement the business layer of the application (**model**).
4. Use JSP pages to present the **view** to the browser.
5. Use Servlet pages to **control** the flow of the application.
6. Functionality that does not follow the assignment specifications will not receive credit.



Database Creation

In addition to the standard WAR file for this assignment, you will submit a text file (plain text, NOT Word or RTF or PDF or any other fancy document format). This file will contain all of the SQL statements that you use to (1) create, and (2) populate your database. Your script will look very similar to the first part of the `create_databases.sql` script used to create the murach database used in the textbook exercises. (See the example table creation and data insert statements there for the murach database User and Download tables). Your database script file must be called:

hw4_create_db.sql

Create a database in MySQL to hold your application data. The following is a **template** database creation script that you can use to start out. All of the columns listed here should be in your table definitions, but you may need to modify the to reflect the naming scheme and data types you used in your User and Product JavaBeans from the previous assignments. Note that the User table will include additional fields for user ID and password.

```
DROP DATABASE IF EXISTS <db name here>;

CREATE DATABASE <db name here>;

use <db name here>;

CREATE TABLE PRODUCT(
        productCode VARCHAR(10),
        productName VARCHAR(50),
        catalogCategory VARCHAR(50),
        description VARCHAR(200),
        price decimal,
        PRIMARY KEY (productCode)
);

CREATE TABLE USER(
        userID int NOT NULL AUTO_INCREMENT,
        firstName VARCHAR(50),
        lastName VARCHAR(50),
        emailAddr VARCHAR(100),
        address1Field VARCHAR(80),
        address2Field VARCHAR(80),
        city VARCHAR(50),
        state VARCHAR(50),
        zipPostalCode VARCHAR(50),
        country VARCHAR(50),
        password VARCHAR(50),
        PRIMARY KEY (userID)
);
```

Database Population

Your database should be populated with data according to the same requirements from the previous assignments (at least one user, at least 2 categories of product, with at least 3 products per category).

Within the same script file for database and table creation (hw4_create_db.sql), add statements at the end of the script to insert your data items into each of the two tables (Product, User).

Running the hw4_create_db.sql script should re-create / reset your database to its initial state with initial data.

Code Changes/Additions

Update your User JavaBean to accommodate the additional data fields for user ID and password.

In the following descriptions, data types representing User or Product fields (code, name, etc) are representative. You may need to adjust some of the methods signatures, depending on the data types you have previously chosen for those fields in your JavaBeans. The bean types of Product and User should remain the same.

For the following updates, you may use any of the approaches discussed in class to implement the database connectivity (single connection, connection pooling with tomcat-dbcp, or JPA). Implementing your database connectivity with Connection Pooling or JPA will receive extra credit (providing it is functional). As detailed later in the assignment text, example support files are provided for single connection connectivity.

ProductDB - Refactor class “ProductDB” to get products from the database:

- addProduct(string productCode, String productName, int categoryCode, String catalogCategory, float price, String description, String imageUrl) - creates a product with the provided values and calls addProduct(Product product)
- addProduct(Product product) - method adds a product with the attribute values from the provided Product Bean to the database
- getAllProducts() - this method returns an ArrayList<Product> of all the products in the PRODUCT table from the database
- getProduct(int productCode) - this method returns a Product Bean for the provided product code
- Prepared Statements are preferred.

UserDB - Refactor the class “UserDB” to get users from the database:

- addUser(String firstName, String lastName, String email, String address1, String address2, String city, String state, String zipcode, String country - this method adds a user with the provided values as the user attributes. Note: userId is not provided as the ID auto increments.
- addUser (User user) - this method adds a user with the attribute values from the provided User Bean
- getAllUsers() - this method returns an ArrayList<User> of all the users in the User table
- getUser(String userID) - this method returns a User Bean for the provided user ID
- You can add update and delete methods now if you like. They will be needed later.
- Prepared Statements are preferred.

As needed, you may also include support classes or configuration file definitions such as the examples from the textbook, particularly if you are using connection pooling:

- DBUtil – helper class managing prepared statements, etc. (if used)
- ConnectionPool – helper class managing the connection pool.
- The data source resource should be set up in the web.xml, the connection pool in the context.xml.

Support Files

The following support files are provided as an example starting point for database connectivity. Note that these classes use the basic single connection (not connection pool) model for database access.

Note that if you have chosen different data types for some of the instance variables in your Java Beans than are used in these example files, you will need to adjust either your Bean or the support file accordingly.

DbAdmin.java

- This is a utility class to create a connection to the database.
- You need to insert a value for the schema name variable, "schemaName". This is the name of the database that your project is using.
- You need to insert a value for the user name variable, "dbUsername". This is the username used to login to your project's database.
- You need to insert a value for the password variable, "dbPassword". This is the password for the database user used to login to your project's database.

ProductDB.java

- This class provides the following methods to create and access the PRODUCT table.
    - addProduct(int productCode, String productName, int categoryCode, String catalogCategory, float price, String description, String imageUrl)
        - this method adds a product with the provided values as the product attributes
    - addProduct(Product product)
        - this method adds a product with the attribute values from the provided Product Bean
    - getAllProducts()
        - this method returns an ArrayList<Product> of all the products in the PRODUCT table
    - getProduct(int productCode)
        - this method returns a Product Bean for the provided product code

UserDB.java

- This class provides the following methods to create and access the USER table.
    - addUser(String userID, String firstName, String lastName, String email, String address1, String address2, String city, String state, String zipcode, String country)
        - this method adds a user with the provided values as the user attributes
    - addUser (User user)

        - this method adds a user with the attribute values from the provided User Bean
    - getAllUsers()
        - this method returns an ArrayList<User> of all the users in the User table
    - getUser(String userID)
        - this method returns a User Bean for the provided user ID

Changes from Assignment 2

- Use the "ProductDB" java class provided to retrieve products form the database.
- Make the necessary changes throughout the project to use this class to retrieve products from the database.

Changes from Assignment 3

- Use the "UsersDB" java class provided to retrieve users form the database.
- Make the necessary changes throughout the project to use this class to retrieve the user information from the database

Extra Credit (10 Points for Connection Pooling, 20 Points for JPA)

1. Update connecting to the database to use connection pooling.
2. Update retrieving data from the database to use JPA.

Moodle Assignment Submissions:

What to submit to Moodle (Email submissions will NOT be accepted):

1. **hw4.war -** An archive of the entire web application (project) stored in a standard WAR file. You must ensure that the java source files are included as part of the archive. The WAR file will be imported into Netbeans for grading and may be required to be deployed as part of the submission.
2. **hw4_create_db.sql** – SQL script file to create and populate the Product and User tables.
3. **info.pdf** – PDF document with the following assignment information :
    a) Explanation of status and stopping point, if incomplete.
    b) Explanation of additional features, if any.
    c) Did you use Connection Pooling or JPA for extra credit?
    d) Discuss the easy and challenging parts of the assignment. How did you overcome all or some of the challenges?
    e) Discuss division of labor specifying who did what and why this is a fair and equal split (team)

Finally, set up a time to demo your assignment to your grader using the process described in the course Moodle site. All team members should be present at the demo. Students should be prepared to answer questions posed by the grader about their work. **Failing to demonstrate the assignment to the grader will result in no credit for all team members**.