Assignment 3
JSP, Servlet, and MVC
Managing Cart using Sessions
Points: 100

This assignment is to be done by a team of two students. This must be the same team previously documented to course staff in the last assignment. 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.

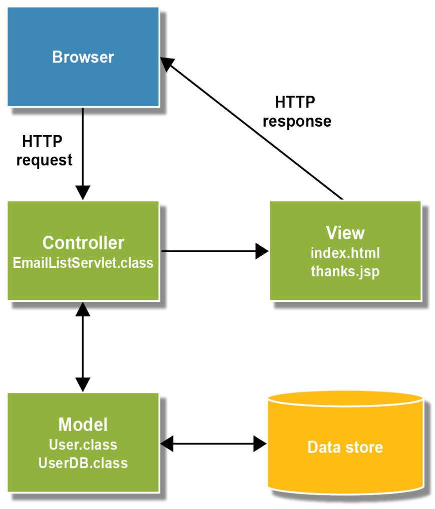
This assignment is intended to familiarize you with using sessions to maintain state in your web application.

For this assignment, you will continue working on your servlet/jsp-based application from assignment 2. The assignment builds upon and completes the previous assignment's MVC architecture.

Assignment Description:

In this assignment you will add new features 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.



Create New JavaBeans for Business Objects (Model)

Create JavaBeans for the following Model elements, with the specified instance variables and additional methods. Bean class names and methods MUST use the specified naming.

User

- First Name
- Last Name
- Email Address
- Address 1 Field
- Address 2 Field
- City
- State / Region
- Post Code
- Country

OrderItem – represents a line from the cart / invoice, associates a product and quantity

- Product
- Quantity
- getTotal() – total cost based on product price and specified quantity

Cart

- List / Collection (your choice) of OrderItem
- addItem(Product, Quantity) – adds an OrderItem for the product / quantity to the cart. The cart should not allow multiple OrderItems for the same product, but should update appropriately if one already exists.
- removeItem(Product) – removes any OrderItem associated with the given Product.
- List/Collection<OrderItem> getItems() – returns the set of current OrderItem-s
- getItems() – returns a List / Collection of OrderItems from the cart
- emptyCart() –clears the entire cart contents

Order

- Order Number
- Date
- User
- List / Collection (your choice) of OrderItem
- Tax Rate – sales tax rate
- Total Cost
- Paid? – flag to indicate whether the order been paid in full or not

Create New Utility Class for Persistent Data

Use a hard-coded "database" to represent your sets of users similar to what you did with the products in assignment 2. You should hard-code a fixed set of users within the following class. You must have at least one user.

UserDB

- Hard-Coded set of user details (your choice on how to represent)
- List/Collection<User> getUsers() – returns a set of all the users in the hardcoded "database"

Create Servlets for Business Logic (Controller)

For this assignment, you will implement dynamic shopping cart functionality. You must use sessions to pass data. This means that the shopping cart will be able to hold multiple kinds of product at a time, and will maintain its content even if you navigate away from the cart.

Implement the following Controller servlet to operationalize the business logic. You will need to send parameters as part of the GET or POST http requests from button / form submissions as the context information that tells the controllers how to proceed.

OrderController.java

- Loads the database of products.
- Checks the session for a current shopping cart, using the attribute "theShoppingCart"
    - If there is no shopping cart parameter, or it is not a valid shopping cart, create a new empty shopping cart and add it to the session under this attribute name.
- Checks the http request for a parameter called "action"
    - If there is an action parameter, validate that its value is either "updateCart" or "checkout"
    - If there is no action parameter, or if it has an unknown value, dispatch directly to the **shopping cart** JSP view.
    - If action is "updateCart"
        - Checks the http request for parameter(s) called "productList"
            - The values returned by *request.getParameterValues("productList")* should return a list of all product codes that appear in the view. If request is coming from the product JSP view this list should have one item.
              Tip: For updating the **shopping cart** JSP view update later on, use a hidden field for each product, named "productList" with the product code as value, and a text field where the name attribute of

        the field is the product code value (which should be unique), and the value is the current quantity for that product.

- Loop through the product list to get the product codes
    - For each product code, validate that its value matches your product code format and is a valid current product code. This tells you which product is to be added to (or updated in) the cart / order.
    - For each valid product code
        - The *quantity* associated with each product code for potential update should be stored in the request as a parameter named exactly as the product code itself. For a product with code "XYZ", request.getParameter("XYZ") should be the quantity.
        - Check the quantity value
            - If there is no quantity value specified at all, set the quantity to be one greater than the current cart quantity (see next main step).
            - If there is a quantity value, validate that is a sensible quantity value (including zero).
            - If the quantity value is present but invalid, disregard the update for this product and move on to the next product code.
        - Update the shopping cart quantity
            - If the request parameter quantity is zero, remove the product from the cart (if present) and move on to the next product code.
            - Compare the request parameter quantity to the value of the quantity for that product currently in the cart.
                - If the product doesn't already exist in the cart, the current cart quantity should be considered zero to begin.
                - "add 1" – If there was no quantity specified in the request parameter, add 1 to the quantity of the product in the cart.
                - If the request parameter quantity and current cart quantity are equal, do not update the cart.
                - If the two quantities are different, update the cart with the new value for quantity for that product from the request parameter.
- Having updated the cart, dispatch to the **shopping cart** JSP view.

- If the action is “checkout,”
    - Create a User bean, by selecting the first (or random) user from the UserDB. This is currently a placeholder for having the user go through all the steps of entering their details or logging in to their account.
    - Add the User bean to the current session as “theUser”
    - Create a new Order bean, using the User bean, the cart contents, and filling in the other details as appropriate.
    - Add the order bean to the session object as “currentOrder” and redirect to the **order** JSP view.

Update JSP Views to Include Dynamic Content From Bean Data

Each of the principal views for placing an order (cart, order) will now receive some kind of dynamic data. Update the JSP views to replace all of the placeholder information with the dynamic data.

Also, if the user has been added to the session as part of a checkout, we consider them to have logged in to their account, and so all the page headers should now display the user name.

Update Form / Button Actions and Links in the Site to Dispatch Correctly

Each of the places in the site where a user can take an action that uses the dynamic data from this assignment should be updated to make the appropriate link or GET or POST request with the necessary parameters or form data.

Assignment Submissions

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

1. **hw3.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. **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) Discuss the easy and challenging parts of the assignment. How did you overcome all or some of the challenges?
   d) 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**.