DAW

M07 Desenvolupament d'aplicacions entorn servidor

Prova Pràctica

Implementar la següent especificació. S'inclou al final codi d'ajuda per a la part d'autenticació d'usuaris.

E-commerce Cart Prototype Specification

1. Overview

The E-commerce Cart Prototype is a part of an online shopping system that allows users to add, view, modify, and remove products they intend to purchase. It should provide a seamless user experience, integrating with the overall store, managing product selections, calculating totals, and providing a pathway to checkout.

2. Features and Functionalities

2.1 Cart Management

- **Add to Cart:** Users can add products to the cart from product listing or product detail pages.
    - Input: Product ID, Quantity.
    - Validation: Check stock availability before adding.
- **Remove from Cart:** Users can remove one or more items from the cart.
    - Input: Product ID.
    - Action: Remove the specified item and update the cart.
- **View Cart:** Users can view the contents of their cart at any time.
    - Display: List of products with thumbnails, names, quantities, individual prices, and total cost.
    - Display subtotals and total costs (with and without tax).
- **Update Cart:** Users can modify the quantity of items in the cart.
    - Input: New quantity for the product.
    - Action: Update the cart with the new quantity and adjust the total cost.

2.2 Pricing & Discounts

- **Price Calculation:** The system calculates the total price based on the quantity of items, individual product prices, and any discounts applied.
    - Taxes and shipping fees should be excluded from this initial total.
- **Discounts and Promotions:**

    - Promo codes: Users can enter promo codes to apply discounts.
    - Auto-applied discounts: The system can apply discounts based on predefined rules (e.g., bulk purchases or seasonal promotions).

2.3 Shipping & Tax Calculation

- **Shipping Calculation:** Based on user's location and shipping method, the system should estimate shipping costs.
- **Tax Calculation:** Calculate applicable taxes based on the user's shipping address and local regulations.

2.4 Checkout Integration

- **Proceed to Checkout:** Users can proceed from the cart to the checkout page.
    - Input: Clicking a "Proceed to Checkout" button redirects to the checkout process.
    - Pre-check: Ensure the cart is valid (e.g., check if all items are in stock, no discrepancies in quantities).

2.5 Save and Load Cart

- **Save Cart for Later:** Users can save the contents of their cart for future visits if logged in.
    - This functionality can be achieved using cookies or a server-side session for guest users.
- **Load Saved Cart:** Returning users (logged in) should be able to retrieve previously saved cart items.

2.6 Notifications

- **Stock Availability:** Notify users if the item is no longer in stock when they attempt to add it to the cart.
- **Price Changes:** If a price changes while items are in the cart, the user should be notified before proceeding to checkout.

2.7 Responsive Design

The cart prototype should be fully responsive across devices:

- Mobile, Tablet, and Desktop views should be supported with intuitive layout changes.

2.8 Error Handling

- **Product Unavailability:** Handle cases where products are no longer available by showing an error message.
- **Invalid Promo Code:** Display error messages when a user applies an invalid or expired promo code.

3. User Interface (UI) Requirements

- **Cart Page Layout:**
    - A clean, easy-to-navigate layout that lists all items in the cart.

    - Product image, name, quantity selection, price, and remove button for each item.
- **Cart Summary:**
    - Display subtotal, discount (if applicable), tax estimate, shipping estimate, and final total.
    - Buttons: "Proceed to Checkout," "Update Cart," "Continue Shopping."

4. Technical Requirements

4.1 Frontend

- **Frameworks:** Basic html and JS minimal front-end
- **UI Components:** Should be modular and reusable (cart item, product summary, total display).
- **AJAX/Fetch:** For dynamic cart updates without full page reloads.

4.2 Backend

- **Frameworks:** PHP basic
- **Data Management:** Use session storage for guests and database or XML files for logged-in users to persist cart data.

4.3 Database

- **Cart Data:** Store cart contents, including product IDs, quantities, applied discounts, and prices at the time of cart creation. The database will be implemented in XML based files.

4.4 API Integration

- **Product Service:** API to fetch product details, prices, and stock availability.
- **Checkout Service:** API to hand over cart data during checkout initiation.
- **Tax and Shipping APIs:** Optionally integrate with external services for tax and shipping rate calculations.

5. Security Requirements

- **Authentication:** Only authenticated users should be able to save carts.
- **Data Protection:** Secure the cart data using HTTPS encryption and protect it from unauthorized access.

6. Performance & Scalability

- **Efficient Load:** Optimize the performance to handle a large number of items in the cart without slowing down the user interface.
- **Caching:** Use caching strategies to minimize server requests when possible (e.g., caching product details for faster loading).

7. Testing

- **Unit Testing:** Individual components of the cart system (e.g., price calculation, promo code validation).

- **Integration Testing:** Ensure cart integrates correctly with other parts of the system (e.g., product service, checkout).
- **UI Testing:** Ensure cart functionality works across all supported devices.

8. Future Enhancements

- **Wishlist Integration:** Allow users to move items from the cart to a wishlist.
- **Multi-currency Support:** Enable pricing and checkout in different currencies based on the user's location.

9. Conclusion

This specification outlines the core features and technical requirements of an e-commerce cart prototype. It should be user-friendly, efficient, and integrated with other systems to enhance the overall shopping experience.

Anexo

```php
<?php

// Function to check if a user is already connected
////////////////////////////////////////////////////////
function isUserConnected($username, $connections)
{
   foreach ($connections->connection as $connection) {
       if ($connection->user == $username) {
           // Check if the connection is still valid (within 5 minutes)
           $currentTime = time();
           $connectionTime = strtotime($connection->date);
           $expirationTime = $connectionTime + (5 * 60);

           if ($currentTime < $expirationTime) {
               return true; // User is already connected
           }
       }
   }
   return false; // User is not connected or the connection has expired
}

// Function to write a connection to the connection.xml file
////////////////////////////////////////////////////////
function writeConnection($username)
{
   // Load existing connections or create a new XML document
   if (file_exists('connection.xml')) {
       $connections = simplexml_load_file('connection.xml');
   } else {
```

```
        $connections = new SimpleXMLElement('<connections></connections>');
    }

    // Create a new connection entry
    $connection = $connections->addChild('connection');
    $connection->addChild('user', $username);
    $connection->addChild('date', date('Y-m-d H:i:s'));

    // Save the updated connections to connection.xml
    $connections->asXML('connection.xml');
}
////////////////////////////////////////////////////////////////
// Check if username and password are provided in the URL
if (isset($_GET['username']) && isset($_GET['password'])) {
    $username = $_GET['username'];
    $password = $_GET['password'];

    // Load user.xml file
    $users = simplexml_load_file('user.xml');

    // Check if the user exists and the password matches
    foreach ($users->user as $user) {
        if ($user->username == $username && $user->password == $password) {
            // Check if the user is already connected
            $connections = simplexml_load_file('connection.xml');
            if (!isUserConnected($username, $connections)) {
                // Write the new connection to connection.xml
                writeConnection($username);
                echo "Connection successful for user: $username";
            } else {
                echo "User $username is already connected.";
            }

            exit(); // Stop execution after successful connection
        }
    }

    echo "Invalid username or password";
} else {
    echo "Username and password are required in the URL";
}

?>
```