Comsats University, Islamabad, Attock Campus

Department of Computer Science

Assignment No. 01
Subject: Mobile Application Development
Program: BSSE-4

Assignment Title:
JavaScript Mobile Shopping Cart Implementation

Submitted To:
Mr. Muhammad Kamran

Submitted By:
Laiba Noor

Registration Number:
SP22-BSE-001

Dated: 26th September, 2024

Introduction:

The objective of this assignment is to implement a comprehensive mobile shopping cart feature utilizing JavaScript. This project aims to demonstrate the application of modern scripting techniques in mobile application development, focusing on enhancing user experience and functionality. By employing ES6 arrow functions, array methods such as map, filter, and reduce, along with robust object manipulation, this implementation provides an effective way to manage items in a shopping cart.

The shopping cart feature encompasses several key operations:

1. **Adding Items to the Cart**: Users can add products to the cart by providing essential details such as product ID, name, quantity, and price. Each product is stored as an object, ensuring easy access and management.
2. **Removing and Updating Items**: The application includes functionality to remove items from the cart based on their unique product ID, utilizing the `splice` method for efficient deletion. Additionally, users can update the quantity of items already in the cart, using array methods to ensure the cart reflects accurate stock levels.
3. **Calculating Total Cost**: A crucial feature of the shopping cart is the ability to calculate the total price of all items present. This is achieved through the `reduce` method, which considers both the price and quantity of each item, providing users with an accurate overview of their expenses.
4. **Displaying Cart Summary**: To enhance user engagement, the application generates a detailed summary of the cart contents. This includes a list of each product's name, quantity, and individual total price, allowing users to review their selections easily.
5. **Filtering Zero Quantity Items**: The implementation also incorporates a filtering mechanism to remove items with zero quantity from the cart, ensuring that the cart remains clean and manageable.
6. **Bonus Feature - Discount Codes**: As an optional enhancement, the application allows users to apply discount codes, which adjusts the total cost based on predefined rates. This feature not only encourages user interaction but also simulates real-world shopping experiences.

Code Explanation with screenshots:

1. **Adding Items to the Cart: addItemToCart(productId, productName, quantity, price)**
    - **Logic**: This function adds a new product to the cart array. It takes four parameters: productId, productName, quantity, and price. The function creates a product object with these properties and uses the push method to add it to the cart. A confirmation message is logged to inform the user that the item has been added successfully.
    - **Example Usage**: When a user adds a laptop with ID 1357, the function creates an object { productId: 1357, productName: 'Laptop', quantity: 2, price: 1000 } and adds it to the cart.

```
C:\Program Files\nodejs\node.exe .\1.js
Laptop has been added to the cart.
Phone has been added to the cart.
```

2. **Removing Items from the Cart: removeItemFromCart(productId)**
   - **Logic**: This function removes an item from the cart based on its productId. It utilizes the findIndex method to locate the index of the item in the cart. If found, the splice method is used to remove the item from the array. A message is logged to confirm the removal. If the item is not found, an appropriate message is displayed to inform the user.
   - **Example Usage**: If a user attempts to remove an item with productId 1357, the function locates it in the cart and removes it.
   - ```
     Cart Summary:
     Product: Laptop, Quantity: 2, Total Price: $2000
     Product: Phone, Quantity: 10, Total Price: $8000
     Total Cost after removal: $10000
     Items with zero quantity have been removed.
     ```
3. **Updating Item Quantity: updateItemQuantity(productId, newQuantity)**
   - **Logic**: This function updates the quantity of an item identified by its productId. It maps through the cart array, checking for a matching productId. If found, it creates a new object with the updated quantity while keeping the other properties unchanged. A confirmation message is logged once the update is made.
   - **Example Usage**: If a user wants to change the quantity of the item with productId 1357 to 3, the function updates it accordingly.

```
Cart Summary:
Product: Laptop, Quantity: 2, Total Price: $2000
Product: Phone, Quantity: 10, Total Price: $8000
Total Cost: $10000
Quantity updated for productId: 2.
```

4. **Calculating Total Cost: calculateTotalCost()**
   - **Logic**: This function calculates the total cost of all items in the cart. It uses the reduce method to iterate through the cart, accumulating the total price by multiplying each item's price by its quantity. The final total is returned to the caller.
   - **Example Usage**: If the cart contains two items—a laptop at $1000 each and a phone at $800 each—the function computes the total as (1000 * 2) + (800 * 10).

```
Cart Summary:
Product: Laptop, Quantity: 2, Total Price: $2000
Product: Phone, Quantity: 10, Total Price: $8000
Total Cost after update: $10000
```

5. **Displaying Cart Summary: displayCartSummary()**
```

- **Logic**: This function generates a summary of the cart's contents. It logs a header and then maps through the cart array to calculate and display each product's name, quantity, and total price. This provides the user with a clear overview of their selections.
- **Example Usage**: When invoked, it displays something like:

```
Quantity updated for productId: 2.
Cart Summary:
Product: Laptop, Quantity: 2, Total Price: $2000
Product: Phone, Quantity: 10, Total Price: $8000
Total Cost after update: $10000
```

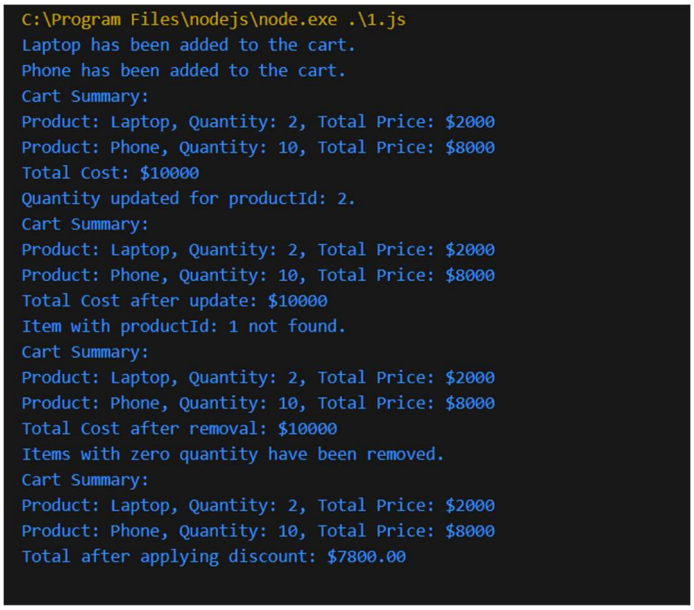
6. **Filtering Zero Quantity Items: filterZeroQuantityItems()**
   - **Logic**: This function removes items with a quantity of zero from the cart. It uses the filter method to create a new array containing only items with a quantity greater than zero. A message is logged to indicate that the filtering has been completed.
   - **Example Usage**: If the cart has an item with a quantity of zero, it will be removed upon calling this function.

```
Cart Summary:
Product: Laptop, Quantity: 2, Total Price: $2000
Product: Phone, Quantity: 10, Total Price: $8000
Total Cost after removal: $10000
Items with zero quantity have been removed.
```

7. **Applying Discount Codes: applyDiscount(discountCode)**
   - **Logic**: This optional function allows users to apply a discount code to the total cost. It defines a set of discount rates based on specific codes. The function checks if the provided code is valid, retrieves the corresponding discount rate, and applies it to the total cost calculated by calculateTotalCost(). The new total after the discount is logged.
   - **Example Usage**: If a user applies the discount code DISCOUNT20, the total cost will be adjusted according to the specified discount rate.

```
Cart Summary:
Product: Laptop, Quantity: 2, Total Price: $2000
Product: Phone, Quantity: 10, Total Price: $8000
Total after applying discount: $7800.00
```

Complete screenshot of the whole code:



Conclusion:

Through this assignment, I gained valuable insights into the implementation of a shopping cart feature using JavaScript, particularly focusing on modern ES6 syntax and array manipulation methods. I became proficient in defining and utilizing functions, especially arrow functions, which enhance code readability and conciseness. Leveraging methods like map, filter, and reduce allowed me to manipulate arrays efficiently, optimizing my code's performance and improving maintainability. Additionally, working with objects to represent cart items deepened my understanding of encapsulating related data and functions, a foundational aspect of JavaScript programming.

However, I encountered several challenges during this process. Ensuring the logical flow of operations, particularly when updating and removing items, required careful planning to avoid unintended consequences. Initially, I faced issues with incorrect calculations of total costs and quantities due to errors in logic, teaching me the importance of systematically testing each function in isolation before integrating them into the larger application. Creating a flexible discount code system was also challenging, as I had to consider various scenarios, such as invalid codes and multiple discounts, which required additional logic to ensure the application behaved as expected.