

**Name: Afifa Javed**  
 **Reg. No: sp22-bse-021**  
 **Subject: Mobile Application & Development**

**Submitted to: Sir Kamran**   
 **Date:26/09/24**

**Assignment no 01**

**COMSATS UNIVERSITY OF ISLAMABAD ATTOCK CAMPUS**

### **Overview:**

This code simulates a simple shopping cart system, allowing a user to:

* **Add items** to the cart.
* **Remove items** from the cart.
* **Update item quantities**.
* **Calculate the total cost** of the cart.
* **Display a summary** of the cart’s contents.
* **Apply discount codes** for potential savings.

The program also ensures that products with zero quantity are removed from the cart.

### **Key Components:**

1. **addItemToCart**:
   * Adds items to the cart or updates the quantity if the item already exists.
   * Uses find() to check if the product is already in the cart.
2. **removeItemFromCart**:
   * Removes items from the cart by their productId using splice() after finding the index via findIndex().
3. **updateItemQuantity**:
   * Updates the quantity of a specific product using map() to return a new cart array with updated quantities.
4. **calculateTotalCost**:
   * Uses reduce() to calculate the total cost of the cart by multiplying the price and quantity of each item.
5. **displayCartSummary**:
   * Uses map() to display each product's name, quantity, and total price in the cart.
6. **filterZeroQuantityItems**:
   * Uses filter() to remove items that have a quantity of zero.
7. **applyDiscount**:
   * Applies discount codes like 'DISCOUNT10' or 'DISCOUNT20' by reducing the total cost.

**Example usage**

At the end of the code snippet, there is an example usage section where an instance of `ShoppingCart` is created (`myCart`). Various methods are called on this instance to demonstrate adding items, updating quantities, removing items, displaying summaries, filtering zero quantities, and applying discount codes. This showcases how all functionalities can be utilized seamlessly in practice.

### **How It Works:**

* **Adding Items**: If the product exists, its quantity is increased. If not, a new product object is added to the cart.
* **Removing Items**: findIndex() locates the product by its productId, and splice() removes it from the cart.
* **Updating Quantities**: map() returns a new array, updating the product with the correct quantity without mutating the original array.
* **Calculating Costs**: reduce() calculates the sum of all product prices based on their quantity.
* **Discount Application**: A discount is applied to the total cost by multiplying it with the discount rate based on the code.
* **Filtering**: filter() keeps only products with a quantity greater than zero.

**Code**

// Shopping cart array to hold product items

let cart = [];

// 1. Add Items to the Cart

const addItemToCart = (productId, productName, quantity, price) => {

// Check if item already exists in the cart

const existingProduct = cart.find(item => item.productId === productId);

if (existingProduct) {

// Update the quantity if product already exists

existingProduct.quantity += quantity;

} else {

// Push a new product object into the cart

cart.push({ productId, productName, quantity, price });

}

};

// 2. Remove Items from the Cart

const removeItemFromCart = (productId) => {

// Find index of the product to remove by productId

const productIndex = cart.findIndex(item => item.productId === productId);

if (productIndex !== -1) {

// Remove product using splice

cart.splice(productIndex, 1);

}

};

// 2. Update Item Quantity

const updateItemQuantity = (productId, newQuantity) => {

cart = cart.map(item =>

item.productId === productId ? { ...item, quantity: newQuantity } : item

);

};

// 3. Calculate Total Cost

const calculateTotalCost = () => {

// Use reduce to calculate total price based on quantity and price

return cart.reduce((total, item) => total + (item.price \* item.quantity), 0);

};

// 4. Display Cart Summary

const displayCartSummary = () => {

// Use map to generate summary for each product

const summary = cart.map(item => ({

productName: item.productName,

quantity: item.quantity,

totalPrice: item.price \* item.quantity

}));

console.log('Cart Summary:', summary);

return summary;

};

// 4. Filter out Items with Zero Quantity

const filterZeroQuantityItems = () => {

// Use filter to remove items with zero quantity

cart = cart.filter(item => item.quantity > 0);

};

// 5. Apply Discount Code (Optional)

const applyDiscount = (discountCode) => {

const discounts = {

'DISCOUNT10': 0.10, // 10% discount

'DISCOUNT20': 0.20, // 20% discount

};

// Check if discount code is valid

const discount = discounts[discountCode] || 0;

const totalCost = calculateTotalCost();

// Apply discount to total cost

return totalCost - (totalCost \* discount);

};

// Test the shopping cart functionality

// Adding items to the cart

addItemToCart(1, 'Android Mobile', 5, 120000);

addItemToCart(2, 'Tablets', 4, 8000);

addItemToCart(3, 'EarPhones', 6, 1500);

// Display cart summary

displayCartSummary();

// Calculate total cost

console.log('Total Cost:', calculateTotalCost());

// Apply a discount

console.log('Total Cost after DISCOUNT10:', applyDiscount('DISCOUNT10'));

// Update quantity of an item

updateItemQuantity(2, 3);

console.log('Cart after updating quantity:');

displayCartSummary();

// Remove an item from the cart

removeItemFromCart(3);

console.log('Cart after removing Headphones:');

displayCartSummary();

// Filter out items with zero quantity (if any)

filterZeroQuantityItems();

console.log('Cart after filtering zero-quantity items:');

displayCartSummary();

### **What We Learned:**

* **Array Methods**: This task demonstrates how powerful array methods like map(), filter(), reduce(), find(), and findIndex() are when manipulating data in JavaScript. They offer concise, readable solutions for iterating over arrays and modifying data.
* **Arrow Functions**: Using arrow functions simplifies function expressions and enhances readability in modern JavaScript.
* **Object Manipulation**: Copying and updating objects using the spread operator ({...item}) avoids direct mutation and ensures immutability, which is important for maintaining predictable code behavior.
* **Basic E-commerce Logic**: Building a cart system like this lays the foundation for more complex e-commerce systems involving stock management, product options, and more advanced pricing strategies.