COMSATS UNIVERSITY ISLAMABAD,

ATTOCK CAMPUS

DEPARTMENT OF COMPUTER SCIENCES

ASSIGNMENT NO: 1

SUBMITTED BY: EMAN CHAUDHARY

REGISTRATION NO: SP22 - BSE - 031

SUBMITTED TO: SIR. KAMRAN

SUBJECT: MOBILE APP DEVELOP

DATE: 26 SEPTEMBER, 2024

1. INTRODUCTION:

OBJECTIVE:

The objective of this assignment is to develop a **JavaScript-based shopping cart feature** for a mobile application, using modern ES6 syntax and array manipulation methods. The purpose is to manage a list of products in a virtual shopping cart, allowing users to:

OPERATIONS:

1. **Add items to the cart** with details such as product ID, name, quantity, and price.
2. **Update or remove items** from the cart, modifying the quantity or deleting a product.
3. **Calculate the total cost** of the cart based on the prices and quantities of the products.
4. **Display a summary** of the cart, showing each product's name, quantity, and total price.
5. **Filter items** with zero quantity to remove them from the cart automatically.
6. **Apply a discount code**, reducing the total price by a specific percentage.

The assignment focuses on applying **ES6 features** like **arrow functions**, **spread operators**, and **array methods** (`map`, `filter`, `reduce`), improving code readability, efficiency, and functionality. The goal is to build a user-friendly shopping cart system that can be easily integrated into a mobile shopping platform. The operations implemented include adding items to the cart, removing or updating them, calculating the total cost, and displaying a cart summary. Additional functionality allows filtering items with zero quantity and applying discount codes.

2. CODE EXPLANATION:

1. Shopping Cart Initialization:

```
let cart = [];
```

- The variable cart is initialized as an empty array. This array will store objects representing the items added to the shopping cart. Each object will contain details like the product's ID, name, quantity, and price.

2. Add Items to the Cart:

```
const addItemToCart = (productId, productName, quantity, price) => {
  const product = {
    productId,
    productName,
    quantity,
    price
  };
  cart.push(product);
};
```

- This function addItemToCart allows us to add new items to the cart.
- It takes four arguments: productId, productName, quantity, and price, which describe the product.
- Inside the function, these details are bundled into a new object product, which is then **pushed** into the cart array using the push method.

Example:

```
addItemToCart(1, 'Shoes', 2, 50);
```

This will add a product with:

- productId = 1
- productName = 'Shoes'
- quantity = 2
- price = 50

3. Remove Item by Product ID:

```
const removeItemFromCart = (productId) => {
  const index = cart.findIndex(item => item.productId === productId);
  if (index !== -1) cart.splice(index, 1);
};
```

- This function removeItemFromCart removes a product from the cart based on its productId.
- The findIndex method is used to search the cart array for the index of the product that matches the provided productId.
- If a matching product is found (i.e., index !== -1), the splice method removes that product from the cart at the found index.

Example:

```
removeItemFromCart(2);
```

This will remove the product with productId = 2 (e.g., the 'T-Shirt') from the cart.

4. Update Item Quantity by Product ID:

```
const updateItemQuantity = (productId, newQuantity) => {
  cart = cart.map(item =>
    item.productId === productId ? { ...item, quantity: newQuantity } : item
  );
};
```

- This function updateItemQuantity updates the quantity of a specific product in the cart.
- It uses the map method to create a new version of the cart array.
- For each item in the cart, the function checks if the productId matches the given productId. If it does, it creates a **new object** by copying the existing product's details using the **spread operator** ({ ...item }) and updating the quantity with newQuantity.
- If the productId does not match, the item remains unchanged.

Example:

```
updateItemQuantity(1, 3);
```

This will update the quantity of the product with `productId = 1` to `3`.

5. Calculate Total Cost:

```
const calculateTotalCost = () => {

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

- This function calculateTotalCost computes the total price of all the products in the cart.
- It uses the reduce method, which iterates through each product in the cart and accumulates the total cost.
- For each product (item), the total price is calculated by multiplying the price by the quantity, and this value is added to the accumulated total.
- The initial value of total is 0.

Example:

```
console.log(calculateTotalCost());
```

This will return the total cost of all the items in the cart, considering their prices and quantities.

6. Display Cart Summary:

```
const displayCartSummary = () => {
  return cart.map(item => ({
    productName: item.productName,
    quantity: item.quantity,
    totalPrice: item.price * item.quantity
  }));
};
```

- This function displayCartSummary generates a summary of the products in the cart.
- It uses the map method to transform each item in the cart into a new object with three properties:
  - productName: The name of the product.
  - quantity: The number of items of that product.
  - totalPrice: The total cost for that product (price * quantity).

Example:

```
console.log(displayCartSummary());
```

This will display an array of objects where each object contains the product name, quantity, and total price.

7. Filter Out Items with Zero Quantity:

```
const filterZeroQuantityItems = () => {
  cart = cart.filter(item => item.quantity > 0);
};
```

- This function filterZeroQuantityItems removes items with zero quantity from the cart.
- It uses the filter method to create a new version of the cart that only includes items with a quantity greater than 0.

Example:

```
filterZeroQuantityItems();
```

This will remove any products with a quantity of 0 from the cart.

8. Apply Discount Code:

```
const applyDiscountCode = (code) => {
  const discountCodes = {
    'SAVE10': 0.10,  // 10% discount
    'SAVE20': 0.20   // 20% discount
  };
  const discount = discountCodes[code] || 0;
  const totalCost = calculateTotalCost();
  return totalCost - (totalCost * discount);
};
```

- This function applyDiscountCode applies a discount to the total cost based on a given discount code.
- The discountCodes object stores valid discount codes and their respective discount percentages.
- The code parameter is checked against discountCodes. If the code is valid, the corresponding discount is applied to the total cost, calculated using calculateTotalCost(). If the code is invalid, no discount is applied (discount = 0).

Example:

```
console.log(applyDiscountCode('SAVE10'));
```

This will apply a 10% discount (as per the `SAVE10` code) and return the new total cost.

9. Example Usage and Output:

```
// Add items to the cart
addItemToCart(1, 'Shoes', 2, 50);
addItemToCart(2, 'T-Shirt', 1, 30);
addItemToCart(3, 'Hat', 3, 20);

console.log("Cart Summary:", displayCartSummary());  // Displays all items

updateItemQuantity(1, 3);  // Update quantity of 'Shoes' to 3
removeItemFromCart(2);     // Remove 'T-Shirt'
console.log("Total Cost:", calculateTotalCost());    // Calculate total cost
filterZeroQuantityItems();  // Filter items with zero quantity
console.log("Cart after filtering:", displayCartSummary());  // Display updated cart
console.log("Total after discount (SAVE10):", applyDiscountCode('SAVE10'));  // Apply
discount
```

This example:

1. Adds three items (`Shoes`, `T-Shirt`, `Hat`) to the cart.
2. Updates the quantity of `Shoes` and removes the `T-Shirt`.
3. Calculates the total cost and applies a discount code (`SAVE10`).

The operations demonstrate how the shopping cart is dynamically managed using modern JavaScript features like array methods, arrow functions, and object manipulation.

3. SCREENSHOTS CODE/OUTPUT:

CODE:

```
// Shopping Cart
let cart = [];

// 1. Add Items to the Cart
const addItemToCart = (productId, productName, quantity, price) => {
  const product = {
    productId,
    productName,
    quantity,
    price
  };
  cart.push(product);
};

// 2. Remove and Update Items
// Remove item by product ID
const removeItemFromCart = (productId) => {
  const index = cart.findIndex(item => item.productId === productId);
  if (index !== -1) cart.splice(index, 1);
};

// Update item quantity by product ID
const updateItemQuantity = (productId, newQuantity) => {
  cart = cart.map(item =>
    item.productId === productId ? { ...item, quantity: newQuantity } : item
);
};
```

```
// 3. Calculate Total Cost
const calculateTotalCost = () => {
  return cart.reduce((total, item) => total + (item.price * item.quantity), 0);
};

// 4. Display Cart Summary
const displayCartSummary = () => {
  return cart.map(item => ({
    productName: item.productName,
    quantity: item.quantity,
    totalPrice: item.price * item.quantity
  }));
};

// Filter out items with zero quantity
const filterZeroQuantityItems = () => {
  cart = cart.filter(item => item.quantity > 0);
```

```
};

// 5. Bonus: Apply Discount Code
const applyDiscountCode = (code) => {
  const discountCodes = {
    'SAVE10': 0.10,  // 10% discount
    'SAVE20': 0.20   // 20% discount
  };
  const discount = discountCodes[code] || 0;
  const totalCost = calculateTotalCost();
  return totalCost - (totalCost * discount);
};

// Example Usage
addItemToCart(1, 'Shoes', 2, 50);
addItemToCart(2, 'T-Shirt', 1, 30);
addItemToCart(3, 'Hat', 3, 20);

console.log("Cart Summary:", displayCartSummary());

updateItemQuantity(1, 3);  // Update shoes quantity to 3
removeItemFromCart(2);     // Remove T-shirt

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

filterZeroQuantityItems();  // Remove items with zero quantity
console.log("Cart after filtering:", displayCartSummary());

console.log("Total after discount (SAVE10):", applyDiscountCode('SAVE10'));
```

OUTPUT:

```
[Running] node "c:\Users\ic\Desktop\tempCodeRunnerFile.js"
Cart Summary: [
  { productName: 'Shoes', quantity: 2, totalPrice: 100 },
  { productName: 'T-Shirt', quantity: 1, totalPrice: 30 },
  { productName: 'Hat', quantity: 3, totalPrice: 60 }
]
Total Cost: 210
Cart after filtering: [
  { productName: 'Shoes', quantity: 3, totalPrice: 150 },
  { productName: 'Hat', quantity: 3, totalPrice: 60 }
]
Total after discount (SAVE10): 189

[Done] exited with code=0 in 0.222 seconds
```

3. CONCLUSION:

Through this assignment, I deepened my understanding of JavaScript's ES6 features, such as **arrow functions**, **spread operators**, and array methods like **map**, **filter**, and **reduce**. These tools proved invaluable in efficiently managing arrays and objects, especially for building a dynamic shopping cart system. I also learned the importance of functional programming techniques in improving code readability and maintainability.

Challenges faced included updating specific item quantities without mutating the cart, properly applying discounts, and handling edge cases like filtering out items with zero quantity. Overcoming these helped reinforce key programming concepts.