INTERNATIONAL UNIVERSITY

School of Computer Science & Engineering

WEB APPLICATION DEVELOPMENT

LABORATORY

Lab 03

JavaScript Fundamentals

Submitted by:

Nguyễn Hà An Thạnh – ITITWE22051

Date submits:	25/10/2025
Date performs:	25/10/2025
Lab section:	Lab 3
Course instructor:	Msc. N. T. Nghĩa

Task 1.1: Interactive Form Validator

Objective

Build a real-time form validator with multiple validation rules and visual feedback.

Requirements

- Fields: Username (4–20 characters, alphanumeric), Email, Password (≥8 chars, ≥1 uppercase, ≥1 number), Confirm Password.
- Submit button disabled until all fields valid.
- Real-time validation as the user types (instant feedback).
- Visual feedback: green border for valid, red border for invalid, error messages below fields.

Design Overview

1. HTML: Semantic form with four inputs and placeholders for error messages.
2. CSS: Utility classes `.valid`, `.invalid`, and `.error-message.show` for borders and messages.
3. JS Functions: Small, focused validators returning boolean values.
4. State: `state` object tracks validity; `touched` object improves UX so empty fields aren't flagged immediately.
5. Events: `input` and `blur` listeners trigger `validateForm()` to update UI and button state in real time.

Core Validation Functions

```
// Username: 4–20 chars, letters, numbers, or underscores
        const USERNAME_REGEX = /^[a-zA-Z0-9]{4,20}$/;
        // Email: simple, practical pattern
        const EMAIL_REGEX = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
        // Password: at least 8 chars, at least one uppercase and
one number
        const PASSWORD_REGEX = /^(?=.*[A-Z])(?=.*\d).{8,}$/;
```

- validateUsername(username): Checks `/^[a-zA-Z0-9_]{4,20}$/`.
- validateEmail(email): Checks `/^[^\s@]+@[^\s@]+\.[^\s@]+$/`.
- validatePassword(password): Checks `/^(?=.*[A-Z])(?=.*\d).{8,}$/`.
- validatePasswordMatch(pass1, pass2): Ensures equality and non-empty.

UI Helper Functions

```
function showError(fieldId, message) {
            const errorDiv = document.getElementById(fieldId +
'Error');
            const input = document.getElementById(fieldId);
            errorDiv.textContent = message;
```

```
            errorDiv.classList.add('show');
            input.classList.add('invalid');
            input.classList.remove('valid');
        }

        function clearError(fieldId) {
            const errorDiv = document.getElementById(fieldId +
'Error');
            const input = document.getElementById(fieldId);
            errorDiv.textContent = '';
            errorDiv.classList.remove('show');
            input.classList.add('valid');
            input.classList.remove('invalid');
        }

        function resetField(fieldId) {
            const errorDiv = document.getElementById(fieldId +
'Error');
            const input = document.getElementById(fieldId);
            errorDiv.textContent = '';
            errorDiv.classList.remove('show');
            input.classList.remove('valid', 'invalid');
        }
```

- showError(fieldId, message): Shows error text, toggles `.invalid`.
- clearError(fieldId): Clears error text, toggles `.valid`.
- resetField(fieldId): Removes both valid/invalid states before the field is interacted with.

Algorithm (validateForm)

6. Read current values from DOM.
7. Validate each field; update `state` and toggle UI classes/messages.
8. Set the submit button's `disabled` based on all `state` values being true.

Testing Guide

- Username: Try `john_doe123` (✓) and `ab` (✗).
- Email: Try `name@example.com` (✓) and `name@ex` (✗).
- Password: Try `Hello1234` (✓) and `hello` (✗).
- Confirm Password: Must exactly match the password.
- Submit button: Only enabled when all fields are valid.

Rubric Mapping (Task 1.1)

Requirement / Rubric Item	Where Implemented / Evidence

All validation functions work correctly (6 points)	Functions: validateUsername, validateEmail, validatePassword, validatePasswordMatch; exercised in validateForm().
Real-time validation on input (3 points)	Event listeners on each input for 'input' and 'blur' call validateForm().
Visual feedback (3 points)	Classes `.valid`/`.invalid` and `.error-message.show` + messages.
Submit button enable/disable logic (2 points)	Button `#submitBtn` set `disabled` based on aggregated `state`.
Clean, readable code (1 point)	Small functions, comments, clear naming, consistent formatting.

Full Source (Task 1.1)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <title>Form Validator</title>
    <style>
        .form-container {
            max-width: 400px;
            margin: 50px auto;
            padding: 30px;
            border: 1px solid #ddd;
            border-radius: 8px;
        }

        .form-group { margin-bottom: 20px; }

        label {
            display: block;
            margin-bottom: 5px;
            font-weight: bold;
        }

        input {
            width: 100%;
            padding: 10px;
            border: 2px solid #ddd;
            border-radius: 4px;
            font-size: 14px;
            outline: none;
```

```
        }

        input.valid { border-color: #28a745; }
        input.invalid { border-color: #dc3545; }

        .error-message {
            color: #dc3545;
            font-size: 12px;
            margin-top: 5px;
            display: none;
        }

        .error-message.show { display: block; }

        button[type="submit"] {
            width: 100%;
            padding: 12px;
            background: #007bff;
            color: white;
            border: none;
            border-radius: 4px;
            font-size: 16px;
            cursor: pointer;
        }

        button[type="submit"]:disabled {
            background: #ccc;
            cursor: not-allowed;
        }
    </style>
</head>
<body>
    <div class="form-container">
        <h2>Sign Up Form</h2>

        <!-- 'novalidate' prevents the browser's native messages
from interfering -->
        <form id="signupForm" novalidate>
            <!-- Username -->
            <div class="form-group">
                <label for="username">Username</label>
```

```
                <input type="text" id="username" placeholder="Enter
username" autocomplete="username" />
                <div class="error-message" id="usernameError" aria-
live="polite"></div>
            </div>

            <!-- Email -->
            <div class="form-group">
                <label for="email">Email</label>
                <input type="email" id="email" placeholder="Enter
email" autocomplete="email" />
                <div class="error-message" id="emailError" aria-
live="polite"></div>
            </div>

            <!-- Password -->
            <div class="form-group">
                <label for="password">Password</label>
                <input type="password" id="password"
placeholder="Enter password" autocomplete="new-password" />
                <div class="error-message" id="passwordError" aria-
live="polite"></div>
            </div>

            <!-- Confirm Password -->
            <div class="form-group">
                <label for="confirmPassword">Confirm
Password</label>
                <input type="password" id="confirmPassword"
placeholder="Confirm password" autocomplete="new-password" />
                <div class="error-message" id="confirmPasswordError"
aria-live="polite"></div>
            </div>

            <button type="submit" id="submitBtn" disabled>Sign
Up</button>
        </form>
    </div>

    <script>
        // ----------------------------
```

```
        // Validation patterns (rules)
        // ------------------------------
        // Username: 4–20 chars, letters, numbers, or underscores
(matches expected 'john_doe123')
        const USERNAME_REGEX = /^[a-zA-Z0-9]{4,20}$/;
        // Email: simple, practical pattern
        const EMAIL_REGEX = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
        // Password: at least 8 chars, at least one uppercase and
one number
        const PASSWORD_REGEX = /^(?=.*[A-Z])(?=.*\d).{8,}$/;

        // ------------------------------
        // Required functions
        // ------------------------------
        function validateUsername(username) {
            return USERNAME_REGEX.test(username);
        }

        function validateEmail(email) {
            return EMAIL_REGEX.test(email);
        }

        function validatePassword(password) {
            return PASSWORD_REGEX.test(password);
        }

        function validatePasswordMatch(pass1, pass2) {
            return pass1 === pass2 && pass1 !== '';
        }

        // ------------------------------
        // UI helpers
        // ------------------------------
        function showError(fieldId, message) {
            const errorDiv = document.getElementById(fieldId +
'Error');
            const input = document.getElementById(fieldId);
            errorDiv.textContent = message;
            errorDiv.classList.add('show');
            input.classList.add('invalid');
            input.classList.remove('valid');
```

```
        }

        function clearError(fieldId) {
            const errorDiv = document.getElementById(fieldId +
'Error');
            const input = document.getElementById(fieldId);
            errorDiv.textContent = '';
            errorDiv.classList.remove('show');
            input.classList.add('valid');
            input.classList.remove('invalid');
        }

        function resetField(fieldId) {
            const errorDiv = document.getElementById(fieldId +
'Error');
            const input = document.getElementById(fieldId);
            errorDiv.textContent = '';
            errorDiv.classList.remove('show');
            input.classList.remove('valid', 'invalid');
        }

        // ------------------------------
        // Validation orchestration
        // ------------------------------
        const state = { username: false, email: false, password:
false, confirmPassword: false };
        const touched = { username: false, email: false, password:
false, confirmPassword: false };

        function validateForm() {
            const usernameEl = document.getElementById('username');
            const emailEl = document.getElementById('email');
            const passwordEl = document.getElementById('password');
            const confirmEl =
document.getElementById('confirmPassword');
            const submitBtn = document.getElementById('submitBtn');

            const username = usernameEl.value.trim();
            const email = emailEl.value.trim();
            const password = passwordEl.value;
            const confirm = confirmEl.value;
```

```
            // Username
            if (username === '') {
                state.username = false;
                if (touched.username) showError('username',
'Username is required');
                else resetField('username');
            } else if (!validateUsername(username)) {
                state.username = false;
                showError('username', 'Username must be 4–20
characters; letters, numbers, underscores only');
            } else {
                state.username = true;
                clearError('username');
            }

            // Email
            if (email === '') {
                state.email = false;
                if (touched.email) showError('email', 'Email is
required');
                else resetField('email');
            } else if (!validateEmail(email)) {
                state.email = false;
                showError('email', 'Please enter a valid email
(e.g., name@example.com)');
            } else {
                state.email = true;
                clearError('email');
            }

            // Password
            if (password === '') {
                state.password = false;
                if (touched.password) showError('password',
'Password is required');
                else resetField('password');
            } else if (!validatePassword(password)) {
                state.password = false;
                showError('password', 'Min 8 chars, include 1
uppercase and 1 number');
```

```
            } else {
                state.password = true;
                clearError('password');
            }

            // Confirm Password
            if (confirm === '') {
                state.confirmPassword = false;
                if (touched.confirmPassword)
showError('confirmPassword', 'Please confirm your password');
                else resetField('confirmPassword');
            } else if (!validatePasswordMatch(password, confirm)) {
                state.confirmPassword = false;
                showError('confirmPassword', 'Passwords do not
match');
            } else {
                state.confirmPassword = true;
                clearError('confirmPassword');
            }

            // Enable submit when all valid
            submitBtn.disabled = !(state.username && state.email &&
state.password && state.confirmPassword);
        }

        // ------------------------------
        // Real-time validation
        // ------------------------------
        const fieldIds = ['username', 'email', 'password',
'confirmPassword'];
        fieldIds.forEach(id => {
            const el = document.getElementById(id);

            // Mark as touched and validate as user types
            el.addEventListener('input', () => {
                touched[id] = true;
                validateForm();
            });

            // Also validate on blur for users who tab through
fields
```

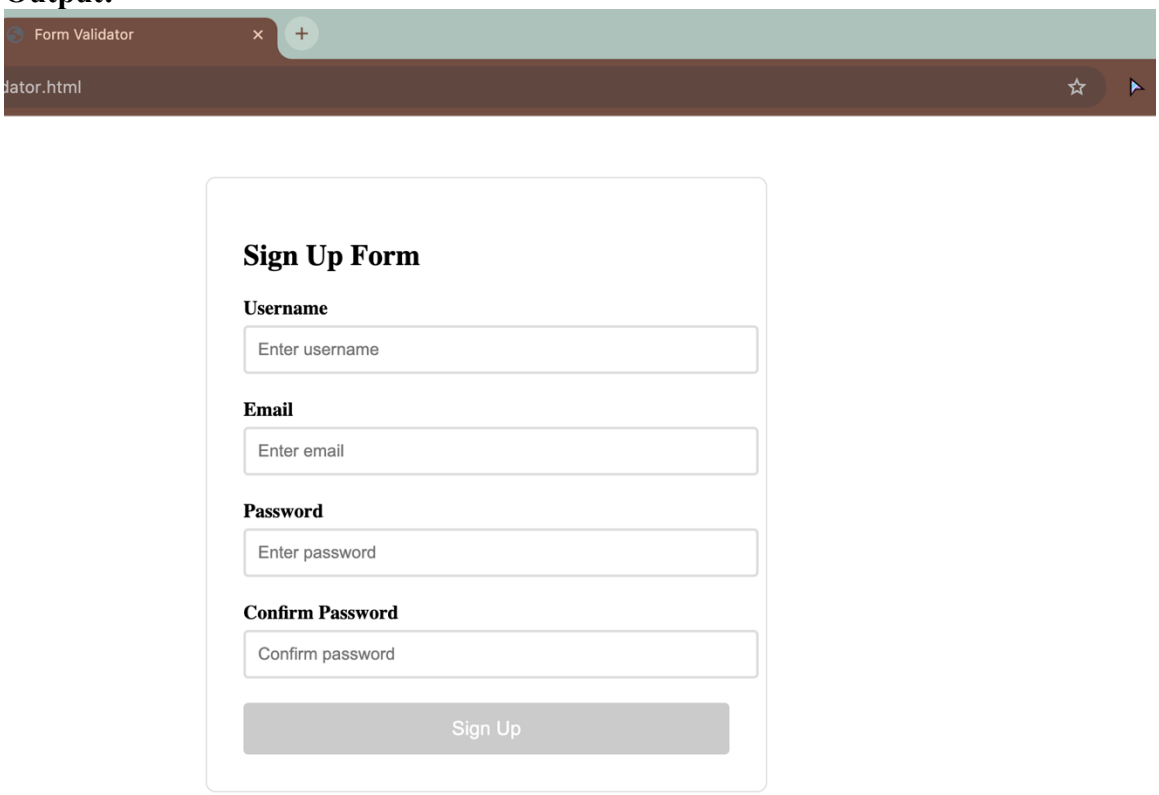
```
            el.addEventListener('blur', () => {
                touched[id] = true;
                validateForm();
            });
        });

        // ------------------------------
        // Submit handler
        // ------------------------------

document.getElementById('signupForm').addEventListener('submit',
function (e) {
            e.preventDefault();
            // Force all touched on submit, then validate
            fieldIds.forEach(id => (touched[id] = true));
            validateForm();

            const allValid = Object.values(state).every(Boolean);
            if (allValid) {
                alert('Form submitted successfully!');
                // Optional reset block:
                // this.reset();
                // fieldIds.forEach(id => { touched[id] = false;
resetField(id); });
                // document.getElementById('submitBtn').disabled =
true;
            }
        });
    </script>
</body>
</html>
```

Output:



Task 1.2: Dynamic Shopping Cart

Objective

Build a shopping cart with add/remove items, quantity updates, item subtotals, total calculation, and a cart badge.

Requirements

- Display at least 4 products with emoji image, name, price, and an "Add to Cart" button.
- Cart shows each line item with quantity controls (− / +), Remove button, and per-item subtotal.
- Cart total updates automatically as items/quantities change.
- Cart icon shows a count badge.

Design Overview

9. Data: `products` is a static array; `cart` holds items `{ id, name, price, image, qty }`.
10. Rendering: `renderProducts()` builds product cards; `renderCart()` builds line items and calls `calculateTotal()`.
11. Actions: `addToCart()`, `removeFromCart()`, `updateQuantity()` mutate `cart` and re-render UI.
12. Badge: `updateCartCount()` shows **distinct** item count by default (as per example).
13. UX: `toggleCart(true)` opens the cart when adding items; `−` is disabled at qty 1 (use Remove to delete).

Core Functions

- addToCart(productId): If item exists, increments `qty`; otherwise pushes a new item with `qty = 1`. Re-renders and updates badge.

```
function addToCart(productId) {
            const id = Number(productId);
            const existing = cart.find(item => item.id === id);
            if (existing) {
                existing.qty += 1;
            } else {
                const p = products.find(prod => prod.id === id);
                if (!p) return;
                cart.push({ id: p.id, name: p.name, price: p.price,
image: p.image, qty: 1 });
            }
            renderCart();
            updateCartCount();
            toggleCart(true); // open cart when adding
        }
```

- removeFromCart(itemId): Filters the item out of `cart`, then re-renders and updates badge.

```
function removeFromCart(itemId) {
            const id = Number(itemId);
            cart = cart.filter(item => item.id !== id);
            renderCart();
            updateCartCount();
        }
```

- updateQuantity(itemId, change): Applies `qty = max(1, qty + change)` and re-renders (prevents 0 via button; removal via Remove).

```
function updateQuantity(itemId, change) {
            const id = Number(itemId);
            const item = cart.find(i => i.id === id);
            if (!item) return;
            const newQty = item.qty + Number(change);
            // Keep at least 1; use "Remove" to delete
            item.qty = Math.max(1, newQty);
            renderCart();
            updateCartCount();
        }
```

- calculateTotal(): Sums `price * qty` over cart items and updates the total text.

```
function calculateTotal() {
            const total = cart.reduce((sum, item) => sum +
item.price * item.qty, 0);
            cartTotalEl.textContent = formatPrice(total);
            return total;
        }
```

- renderProducts(): Creates emoji cards and "Add to Cart" buttons.

```
function renderProducts() {
            productsGrid.innerHTML = products.map(p => `
                <div class="product-card">
                    <div class="product-image">${p.image}</div>
                    <div class="product-name">${p.name}</div>
                    <div class="product-
price">$${formatPrice(p.price)}</div>
                    <button class="add-to-cart-btn"
onclick="addToCart(${p.id})">Add to Cart</button>
                </div>
            `).join('');
        }
```

renderCart(): Builds each line item row with controls and per-item subtotal, then calls `calculateTotal()`.

```
        function renderCart() {
            cartItems.innerHTML = '';

            if (cart.length === 0) {
```

```
                cartItems.innerHTML = `<p class="empty">Your cart is
empty.</p>`;
                calculateTotal();
                return;
            }

            cart.forEach(item => {
                const subtotal = item.price * item.qty;

                const row = document.createElement('div');
                row.className = 'cart-item';

                row.innerHTML = `
                    <div class="item-left">
                        <span class="item-
emoji">${item.image}</span>
                        <div>
                            <div><strong>${item.name}</strong></div>
                            <div>$${formatPrice(item.price)}</div>
                        </div>
                    </div>

                    <div class="quantity-controls">
                        <button onclick="updateQuantity(${item.id},
-1)" ${item.qty === 1 ? 'disabled' : ''}>-</button>
                        <span>${item.qty}</span>
                        <button onclick="updateQuantity(${item.id},
1)">+</button>
                        <button class="remove-btn"
onclick="removeFromCart(${item.id})">Remove</button>
                    </div>

                    <div
class="subtotal">$${formatPrice(subtotal)}</div>
                `;

                cartItems.appendChild(row);
            });

            calculateTotal();
```

•

- toggleCart(forceOpen): Shows/hides the cart section. Called with `true` on add to make it visible.

```
function toggleCart(forceOpen = null) {
            const isHidden = cartSection.style.display === 'none' ||
cartSection.style.display === '';
            const shouldOpen  = forceOpen === true || (forceOpen ===
null && isHidden);
            const shouldClose = forceOpen === false || (forceOpen
=== null && !isHidden);

            if (shouldOpen) {
                cartSection.style.display = 'block';
                // Optional: scroll into view
                // cartSection.scrollIntoView({ behavior: 'smooth'
});
            } else if (shouldClose) {
                cartSection.style.display = 'none';
            }
        }
```

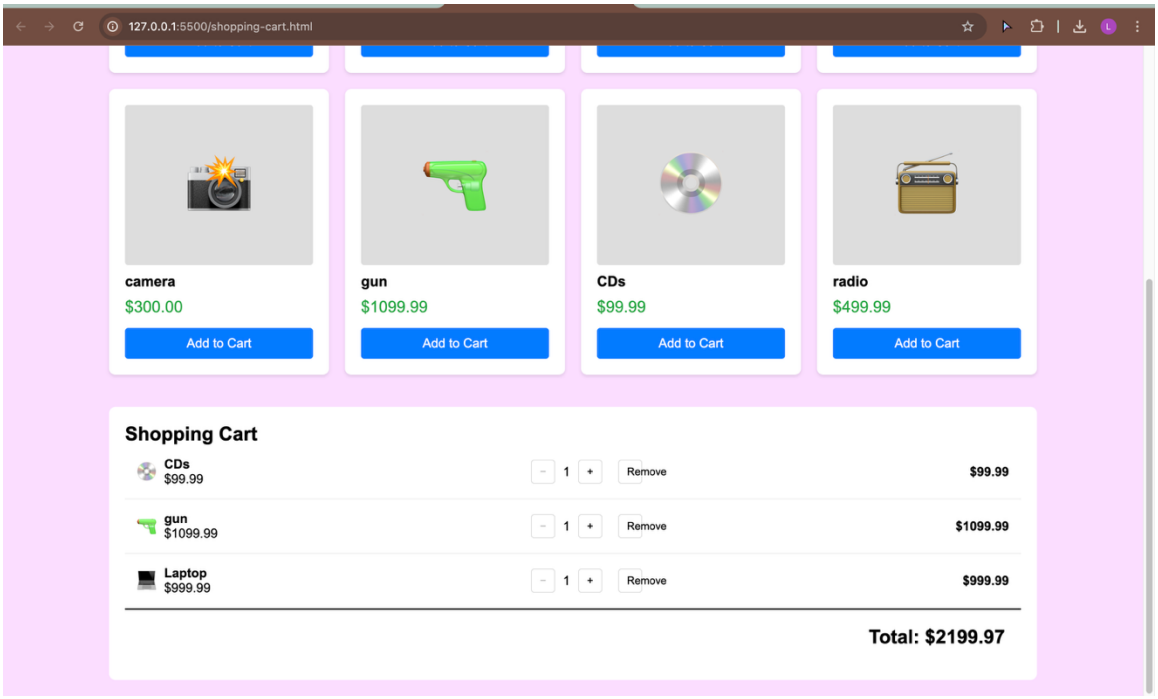
Badge Behavior

By default, the red badge shows the **number of distinct items** (e.g., Laptop + Smartphone = 2). To display total quantity instead, change `updateCartCount()` to sum `qty` across items.

Testing Guide

- Click "Add to Cart" for Laptop and Smartphone — both appear (2 distinct items).
- Use + to increase quantity and − to decrease (stops at 1).
- Click Remove to delete a line item from the cart.
- Verify that each line subtotal and the total at the bottom update immediately.

- Check the cart badge count after every change.



Rubric Mapping (Task 1.2)

Requirement / Rubric Item	Where Implemented / Evidence
Add to cart functionality (3 points)	addToCart(productId) increments qty or adds a new item; UI updates via renderCart().
Quantity increase/decrease (3 points)	updateQuantity(itemId, change) with buttons in renderCart().
Remove from cart (2 points)	removeFromCart(itemId) with 'Remove' button in renderCart().
Total calculation (3 points)	calculateTotal() computes sum(price * qty) and updates #cartTotal.
Cart count badge updates (2 points)	updateCartCount() sets the badge to cart.length (distinct items).
Clean UI and code (2 points)	Separation of concerns; helper formatPrice(); minimal DOM manipulation.

Full Source (Task 1.2)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <title>Shopping Cart</title>
    <style>
        * { margin: 0; padding: 0; box-sizing: border-box; }
```

```
        body { font-family: Arial, sans-serif; background: #fbddff;
}

        .header {
            background: #ea87f1; color: rgb(12, 12, 12); padding:
20px;
            display: flex; justify-content: space-between; align-
items: center;
        }

        .cart-icon { position: relative; cursor: pointer; }
        .cart-count {
            position: absolute; top: -10px; right: -10px;
            background: red; color: white; border-radius: 50%;
            width: 24px; height: 24px; display: flex; align-items:
center;
            justify-content: center; font-size: 12px;
        }

        .container { max-width: 1200px; margin: 0 auto; padding:
20px; }

        .products-grid {
            display: grid; grid-template-columns: repeat(auto-fill,
minmax(250px, 1fr));
            gap: 20px; margin-bottom: 40px;
        }

        .product-card {
            background: white; padding: 20px; border-radius: 8px;
            box-shadow: 0 2px 4px rgba(0,0,0,0.1);
        }

        .product-image {
            width: 100%; height: 200px; background: #ddd; border-
radius: 4px;
            margin-bottom: 10px; display: flex; align-items: center;
justify-content: center;
            font-size: 80px;
        }
```

```
        .product-name { font-size: 18px; font-weight: bold; margin-
bottom: 10px; }
        .product-price { color: #28a745; font-size: 20px; margin-
bottom: 15px; }

        .add-to-cart-btn {
            width: 100%; padding: 10px; background: #007bff; color:
white;
            border: none; border-radius: 4px; cursor: pointer; font-
size: 16px;
        }
        .add-to-cart-btn:hover { background: #0056b3; }

        .cart-section { background: white; padding: 20px; border-
radius: 8px; }
        .cart-item {
            display: flex; justify-content: space-between; align-
items: center;
            padding: 15px; border-bottom: 1px solid #eee;
        }

        .item-left { display: flex; align-items: center; gap: 10px;
}
        .item-emoji { font-size: 24px; }

        .quantity-controls { display: flex; gap: 10px; align-items:
center; }
        .quantity-controls button {
            width: 30px; height: 30px; border: 1px solid #ddd;
background: white;
            cursor: pointer; border-radius: 4px;
        }

        .cart-total {
            text-align: right; font-size: 24px; font-weight: bold;
            padding: 20px; border-top: 2px solid #333;
        }

        .empty { color: #777; padding: 10px 0; }
        .remove-btn {
```

```
            margin-left: 10px; border: 1px solid #ddd; background:
white;
            padding: 6px 10px; border-radius: 4px; cursor: pointer;
        }
        .remove-btn:hover { background: #f5f5f5; }
        .subtotal { font-weight: bold; }
    </style>
</head>
<body>
    <div class="header">
        <h1>An Thanh's Shop</h1>
        <div class="cart-icon" onclick="toggleCart()">
            🛒 Cart
            <span class="cart-count" id="cartCount">0</span>
        </div>
    </div>

    <div class="container">
        <h2>Products</h2>
        <div class="products-grid" id="productsGrid"></div>

        <div class="cart-section" id="cartSection" style="display:
none;">
            <h2>Shopping Cart</h2>
            <div id="cartItems"></div>
            <div class="cart-total">
                Total: $<span id="cartTotal">0.00</span>
            </div>
        </div>
    </div>

    <script>
        // --------------------------------
        // Product data
        // --------------------------------
        const products = [
            { id: 1, name: 'Laptop',      price: 999.99, image: '💻'
},
            { id: 2, name: 'Smartphone',  price: 699.99, image: '📱'
},
```

```
            { id: 3, name: 'Headphones',  price: 199.99, image: '🎧' },
            { id: 4, name: 'Smartwatch',  price: 299.99, image: '⌚' },
            { id: 5, name: 'camera',  price: 300.00,   image: '📸' },
            { id: 6, name: 'gun',  price: 1099.99, image: '🔫' },
            { id: 7, name: 'CDs',  price: 99.99, image: '💿' },
            { id: 8, name: 'radio',  price: 499.99, image: '📻' },
        ];

        // ---------------------------------
        // Cart state
        // Each item: { id, name, price, image, qty }
        // ---------------------------------
        let cart = [];

        // Cache DOM nodes
        const productsGrid =
document.getElementById('productsGrid');
        const cartSection  = document.getElementById('cartSection');
        const cartItems    = document.getElementById('cartItems');
        const cartTotalEl  = document.getElementById('cartTotal');
        const cartCountEl  = document.getElementById('cartCount');

        // ---------------------------------
        // Utilities
        // ---------------------------------
        const formatPrice = (n) => n.toFixed(2);

        function updateCartCount() {
            // Count DISTINCT items (matches sample "Cart (2)")
            cartCountEl.textContent = cart.length;
        }

        // ---------------------------------
        // Required functions (spec)
        // ---------------------------------
        function addToCart(productId) {
            const id = Number(productId);
            const existing = cart.find(item => item.id === id);
```

```
            if (existing) {
                existing.qty += 1;
            } else {
                const p = products.find(prod => prod.id === id);
                if (!p) return;
                cart.push({ id: p.id, name: p.name, price: p.price,
image: p.image, qty: 1 });
            }
            renderCart();
            updateCartCount();
            toggleCart(true); // open cart when adding
        }

        function removeFromCart(itemId) {
            const id = Number(itemId);
            cart = cart.filter(item => item.id !== id);
            renderCart();
            updateCartCount();
        }

        function updateQuantity(itemId, change) {
            const id = Number(itemId);
            const item = cart.find(i => i.id === id);
            if (!item) return;
            const newQty = item.qty + Number(change);
            // Keep at least 1; use "Remove" to delete
            item.qty = Math.max(1, newQty);
            renderCart();
            updateCartCount();
        }

        function calculateTotal() {
            const total = cart.reduce((sum, item) => sum +
item.price * item.qty, 0);
            cartTotalEl.textContent = formatPrice(total);
            return total;
        }

        function renderProducts() {
            productsGrid.innerHTML = products.map(p => `
                <div class="product-card">
```

```
                    <div class="product-image">${p.image}</div>
                    <div class="product-name">${p.name}</div>
                    <div class="product-
price">$${formatPrice(p.price)}</div>
                    <button class="add-to-cart-btn"
onclick="addToCart(${p.id})">Add to Cart</button>
                </div>
            `).join('');
        }

        function renderCart() {
            cartItems.innerHTML = '';

            if (cart.length === 0) {
                cartItems.innerHTML = `<p class="empty">Your cart is
empty.</p>`;
                calculateTotal();
                return;
            }

            cart.forEach(item => {
                const subtotal = item.price * item.qty;

                const row = document.createElement('div');
                row.className = 'cart-item';

                row.innerHTML = `
                    <div class="item-left">
                        <span class="item-
emoji">${item.image}</span>
                        <div>
                            <div><strong>${item.name}</strong></div>
                            <div>$${formatPrice(item.price)}</div>
                        </div>
                    </div>

                    <div class="quantity-controls">
                        <button onclick="updateQuantity(${item.id},
-1)" ${item.qty === 1 ? 'disabled' : ''}>-</button>
                        <span>${item.qty}</span>
```

```
                        <button onclick="updateQuantity(${item.id},
1)">+</button>
                        <button class="remove-btn"
onclick="removeFromCart(${item.id})">Remove</button>
                    </div>

                    <div
class="subtotal">$${formatPrice(subtotal)}</div>
                `;

                cartItems.appendChild(row);
            });

            calculateTotal();
        }

        function toggleCart(forceOpen = null) {
            const isHidden = cartSection.style.display === 'none' ||
cartSection.style.display === '';
            const shouldOpen  = forceOpen === true || (forceOpen ===
null && isHidden);
            const shouldClose = forceOpen === false || (forceOpen
=== null && !isHidden);

            if (shouldOpen) {
                cartSection.style.display = 'block';
                // Optional: scroll into view
                // cartSection.scrollIntoView({ behavior: 'smooth'
});
            } else if (shouldClose) {
                cartSection.style.display = 'none';
            }
        }

        // ---------------------------------
        // Initialize
        // ---------------------------------
        renderProducts();
        renderCart();
        updateCartCount();
    </script>
```

```
</body>
</html>
```

Result: