**International University**

School of Computer Science and Engineering

# Web Application Development

# Laboratory

# IT093IU

# Lab #3

**Submitted by**

**Nguyễn Hồng Ngọc Hân - ITCSIU22229**

## Task 1.1: Interactive Form Validator

```
<!DOCTYPE html>

<html lang="en">

 <head>

   <meta charset="UTF-8" />

   <title>Form Validator</title>

   <style>

     * {

       box-sizing: border-box;

     }



     .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;
  }

  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: #c6d1dd;
      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>
    <form id="signupForm">
      <div class="form-group">
```

```
    <label for="username">Username</label>
    <input type="text" id="username" placeholder="Enter username" />
    <div class="error-message" id="usernameError"></div>
  </div>

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

  <!-- TODO: Add password field -->
  <div class="form-group">
    <label for="password">Password</label>
    <input type="password" id="password" placeholder="Enter passwword" />
    <div class="error-message" id="passwordError"></div>
  </div>

  <!-- TODO: Add confirm password field -->
  <div class="form-group">
    <label for="confirmPassword">Confirm Password</label>
    <input
      type="password"
      id="confirmPassword"
      placeholder="Confirm password"
    />
```

```
        <div class="error-message" id="confirmPasswordError"></div>
      </div>

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

  <script>
    // TODO: Implement validation functions
    const fields = {
      username: false,
      email: false,
      password: false,
      confirmPassword: false,
    };

    const username = document.getElementById("username");
    const email = document.getElementById("email");
    const password = document.getElementById("password");
    const confirmPassword = document.getElementById("confirmPassword");
    const submitBtn = document.getElementById("submitBtn");

    function validateUsername(username) {
      // TODO: Check length (4-20) and alphanumeric
      const regex = /^[a-zA-Z0-9]{4,20}$/;
      return regex.test(username);
```

```
}


function validateEmail(email) {
  // TODO: Use regex for email validation
  const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  return emailRegex.test(email);
}


function validatePassword(password) {
  // TODO: Check length >= 8, has uppercase, has number
  const passwordRegex = /^(?=.*[A-Z])(?=.*\d).{8,}$/;
  return passwordRegex.test(password);
}


function validatePasswordMatch(pass1, pass2) {
  // TODO: Compare passwords
  return pass1 == pass2 && pass1 != "";
}


function showError(fieldId, message) {
  // TODO: Display error message
  const error = document.getElementById(`${fieldId}Error`);
  const input = document.getElementById(fieldId);
  input.classList.remove("valid");
  input.classList.add("invalid");
  error.textContent = message;
```

```
    error.classList.add("show");
  }

  function clearError(fieldId) {
    // TODO: Clear error message
    const error = document.getElementById(`${fieldId}Error`);
    const input = document.getElementById(fieldId);
    input.classList.remove("invalid");
    input.classList.add("valid");
    error.textContent = "";
    error.classList.remove("show");
  }

  function validateForm() {
    const usernameVal = username.value.trim();
    const emailVal = email.value.trim();
    const passwordVal = password.value.trim();
    const confirmPasswordVal = confirmPassword.value.trim();

    if (!validateUsername(usernameVal)) {
      showError("username", "Username must be 4-20 letters");
      fields.username = false;
    } else {
      clearError("username");
      fields.username = true;
    }
```

```
  if (!validateEmail(emailVal)) {
    showError("email", "Invalid email");
    fields.email = false;
  } else {
    clearError("email");
    fields.email = true;
  }

  if (!validatePassword(passwordVal)) {
    showError(
      "password",
      "Password must be 8+ chars, 1 uppercase, 1 number"
    );
    fields.password = false;
  } else {
    clearError("password");
    fields.password = true;
  }

  if (!validatePasswordMatch(confirmPasswordVal, passwordVal)) {
    showError("confirmPassword", "Passwords do not match");
    fields.confirmPassword = false;
  } else {
    clearError("confirmPassword");
    fields.confirmPassword = true;
```

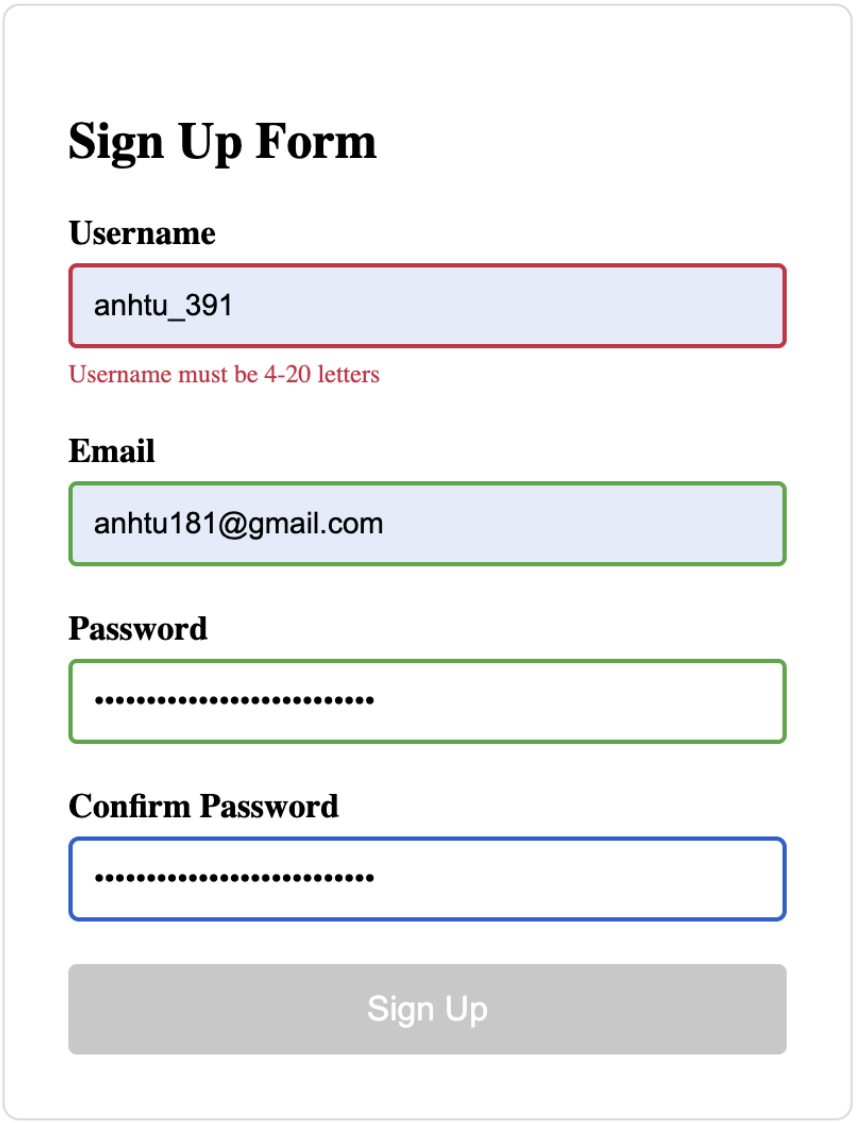
```
      }


      //enable/disable submit
      const allValid = Object.values(fields).every(Boolean);
      submitBtn.disabled = !allValid;
    }


    // TODO: Add event listeners for real-time validation
    username.addEventListener("input", validateForm);
    email.addEventListener("input", validateForm);
    password.addEventListener("input", validateForm);
    confirmPassword.addEventListener("input", validateForm);


    document
      .getElementById("signupForm")
      .addEventListener("submit", function (e) {
        e.preventDefault();
        // TODO: Handle form submission
        alert("Successfully !!!");
      });
  </script>
</body>
</html>
```

**Output:**




**Explain:**

// TODO: Implement validation functions

→ This section defines all the validation functions for each input field (username, email, password, etc.). Each function receives the user's input and returns true if it meets the criteria or false if it's invalid. These functions keep the code modular and easy to maintain.

// TODO: Check length (4-20) and alphanumeric

→ The validateUsername() function uses the regex /^[a-zA-Z0-9]{4,20}$/ to ensure the username contains only letters and numbers and has 4–20 characters. It returns true if valid and shows an error otherwise.
// TODO: Use regex for email validation
→ The validateEmail() function uses regex /^[^\s@]+@[^\s@]+\.[^\s@]+$/ to verify that the input follows a proper email format with “@” and a domain like .com or .vn. If the pattern doesn’t match, it’s considered invalid.

// TODO: Check length >= 8, has uppercase, has number
→ The validatePassword() function uses regex /^(?=.*[A-Z])(?=.*\d).{8,}$/ to check that the password has at least 8 characters, one uppercase letter, and one number. If it fails, an error message appears below the input.

// TODO: Compare passwords
→ The validatePasswordMatch() function compares the password and confirmPassword fields. If both match and are not empty, it returns true; otherwise, it displays “Passwords do not match.”

// TODO: Display error message
→ The showError() function displays the error by adding a "show" class and changing the input border to red (invalid). It also sets the error text in the  under the input field.

// TODO: Clear error message
→ The clearError() function removes the error message when the input becomes valid again. It switches the border to green (valid) and hides the message by removing the "show" class.

// TODO: Add event listeners for real-time validation
→ Event listeners are attached to each input field using the "input" event. Each time the user types or edits, the validateForm() function runs immediately to check validity and update feedback in real time.

// TODO: Handle form submission

→ When the “Sign Up” button is clicked, the "submit" event is triggered. The code uses e.preventDefault() to stop the default reload and shows an alert “Successfully !!!” — this is where you could later add server submission logic.

### Task 1.2: Dynamic Shopping Cart

```
<!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: #f5f5f5;

     }

     .header {

       background: #333;

       color: white;

       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;
    display: flex;
    align-items: center;
    justify-content: center;
    font-size: 80px;
    margin-bottom: 10px;
  }
  .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;
    }
    .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;
    }
  </style>
```

```
</head>

<body>

  <div class="header">

    <h1>My 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: "⌚" },
];

// Cart array
let cart = [];

function addToCart(productId) {
  // TODO: Add product to cart or increase quantity
  const product = products.find((p) => p.id === productId);
  const item = cart.find((i) => i.id === productId);
  if (item) {
    item.quantity++;
  } else {
    cart.push({ ...product, quantity: 1 });
  }
  renderCart();
}

function removeFromCart(productId) {
  // TODO: Remove item from cart
  cart = cart.filter((item) => item.id !== productId);
  renderCart();
}

function updateQuantity(productId, change) {
```

```
  // TODO: Update item quantity (change is +1 or -1)
  const item = cart.find((i) => i.id === productId);
  if (item) {
    item.quantity += change;
    if (item.quantity <= 0) removeFromCart(productId);
    renderCart();
  }
}

function calculateTotal() {
  // TODO: Calculate total price
  return cart.reduce((sum, item) => sum + item.price * item.quantity, 0);
}

function renderProducts() {
  // TODO: Display products
  const grid = document.getElementById("productsGrid");
  grid.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">$${p.price.toFixed(2)}</div>
    <button class="add-to-cart-btn" onclick="addToCart(${
      p.id
```

```
        })">Add to Cart</button>
      </div>
    `
      )
      .join("");
  }

  function renderCart() {
    // TODO: Display cart items
    const cartItems = document.getElementById("cartItems");
    const cartCount = document.getElementById("cartCount");
    const cartTotal = document.getElementById("cartTotal");

    cartItems.innerHTML = cart
      .map(
        (item) => `
    <div class="cart-item">
      <div>${item.image} ${item.name} - $${item.price.toFixed(2)}</div>
      <div class="quantity-controls">
        <button onclick="updateQuantity(${item.id}, -1)">-</button>
        <span>${item.quantity}</span>
        <button onclick="updateQuantity(${item.id}, 1)">+</button>
        <button onclick="removeFromCart(${item.id})">🗑️</button>
      </div>
    </div>
    `
```

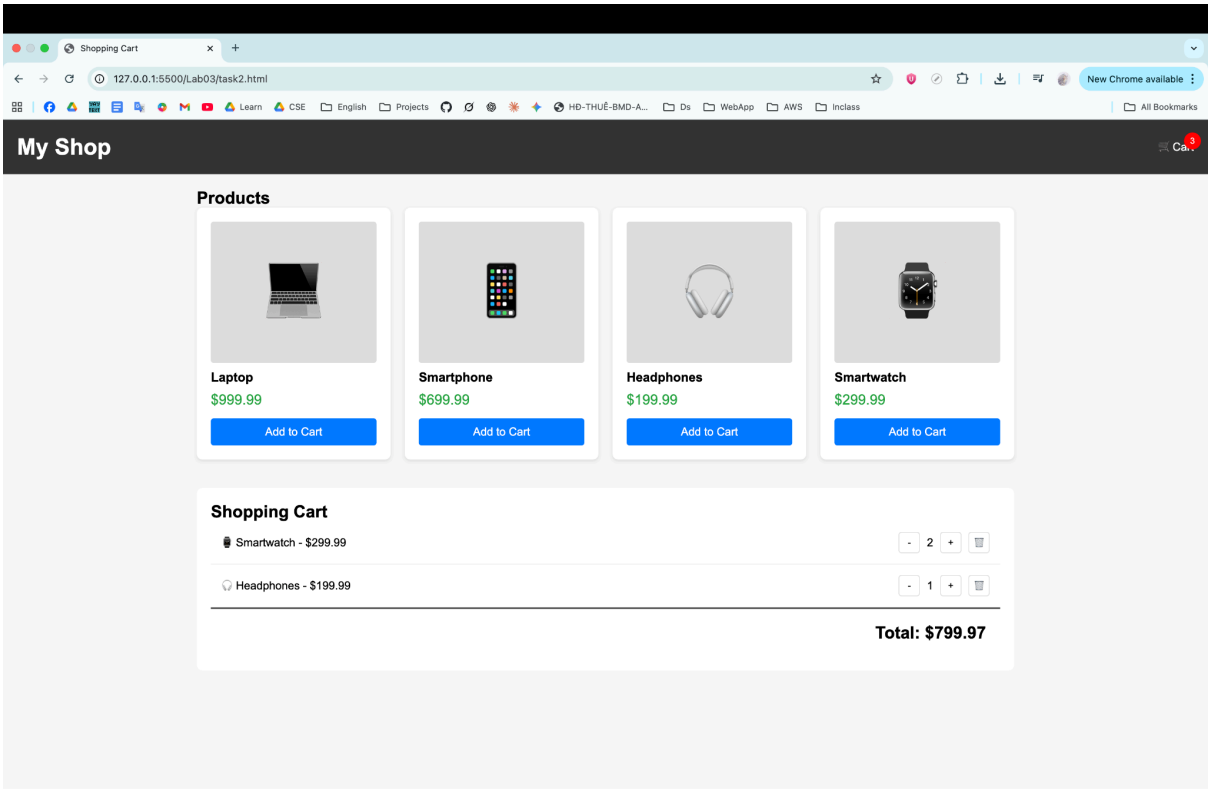
```
        )
        .join("");

      const total = calculateTotal();
      cartTotal.textContent = total.toFixed(2);
      cartCount.textContent = cart.reduce((sum, i) => sum + i.quantity, 0);
    }

    function toggleCart() {
      // TODO: Show/hide cart section
      const section = document.getElementById("cartSection");
      section.style.display =
        section.style.display === "none" ? "block" : "none";
    }

    // Initialize
    renderProducts();
  </script>
</body>
</html>
```

## Output:




**Explain:**

// TODO: Add product to cart or increase quantity

→ This function adds a selected product to the cart array. If the product already exists in the cart, its quantity increases by one; otherwise, a new item is added with quantity: 1. After updating, it calls renderCart() to refresh the cart display.

// TODO: Remove item from cart

→ This function removes an item from the cart using filter() to keep only products with a different ID. It then calls renderCart() to update the cart view immediately.

// TODO: Calculate total price

→ This function uses reduce() to sum up each item's total price (price * quantity). It returns the overall total value of all items in the cart.

// TODO: Display products

→ The renderProducts() function dynamically generates HTML cards for each product using .map() and template literals. Each product has an

image, name, price, and an “Add to Cart” button that triggers addToCart().

// TODO: Display cart items
→ The renderCart() function builds the shopping cart UI dynamically by looping through the cart array. It shows product names, prices, quantity controls (+/–), a remove button, total price, and item count. It also updates the red badge showing the total number of items in the cart.

// TODO: Show/hide cart section
→ The toggleCart() function toggles the visibility of the cart section. When the user clicks the cart icon, it switches between showing (display: block) and hiding (display: none) the cart.