task1-1-form-validator.html

1. Form Structure

The form includes four input fields:

- username — checks length and alphanumeric characters
- email — checks for valid email format
- password — checks strength (uppercase + number + minimum length)
- confirmPassword — checks that both passwords match

Each field has:

- A <label> for accessibility
- An <input> for user input
- A to display validation errors

2. Validation Functions

- validateUsername(username)
    - Regex: ^[a-zA-Z0-9]{4,20}$
    - Rules:
        - Only letters and numbers
        - Length between 4–20 characters
    - Result: Works correctly and meets requirements.
- validateEmail(email)
    - Regex: /^[^\\s@]+@[^\\s@]+\\.[^\\s@]+$/
    - Checks standard email format.
    - Result: Correct and reliable.
- validatePassword(password)
    - Regex: /^(?=.*[A-Z])(?=.*[0-9])[A-Za-z0-9]{8,}$/
    - Rules:
        - Minimum 8 characters
        - At least one uppercase letter
        - At least one number
    - Result: Works perfectly, easy to extend for special characters.
- validatePasswordMatch(pass1, pass2)
    - Compares two password strings.
    - Prevents validation if either is empty.
    - Result: Correct and simple implementation.

3. UI Feedback

- showError(fieldId, message)
    - Displays the error message under the field.
    - Adds red border (input.invalid).
    - Shows the message using .error-message.show.

- clearError(fieldId)
    - Removes error text.
    - Adds green border (input.valid).
    - Hides the error message.
- Result: Provides clear and immediate visual feedback.

4. Real-Time Validation

- Uses input event listeners on all four fields.
- Each keystroke triggers validateForm(), dynamically updating field states and error messages.
- Result: Real-time validation works smoothly.

5. Submit Button Logic

- “Sign Up” button is disabled by default.
- When all fields are valid (every(state === true)), the button is enabled.
- Result: Correct logic and behavior.

6. Form Submission

- Prevents default submit behavior (e.preventDefault()).
- Revalidates before final submission:
    - If valid → shows success alert.
    - If invalid → shows error alert.
- Result: Works correctly and matches assignment intent.

task1-2-shopping-cart.html

1. Overview

- Project Title: Shopping Cart
- Language: HTML, CSS, JavaScript
- Objective: Build a shopping cart system that allows users to browse products, add or remove items, adjust quantities, and view the total price dynamically.

2. Add to Cart Functionality

- Function: addToCart(productId)
- Behavior:
    - Finds the product by its ID.
    - If it already exists in the cart: increases quantity.
    - If it does not exist: adds a new object { id, product, quantity: 1 }.
    - Re-renders the cart after every addition.
- Result: Fully functional, handles repeated additions correctly, and updates the cart in real time.

3. Quantity Increase/Decrease

- Function: updateQuantity(productId, change)
- Behavior:
    - Updates the quantity of a given product.
    - Prevents negative quantities and removes the item if it reaches zero.
    - Re-renders the cart immediately after changes.
- Result: Smooth and accurate quantity updates. No negative or invalid states.

4. Remove from Cart

- Function: removeFromCart(itemId)
- Behavior:
    - Uses filter() to remove the selected item from the cart array.
    - Calls renderCart() to refresh the display.
- Result: Works correctly and removes the item instantly from both the UI and cart count.

5. Total Calculation

- Function: calculateTotal()
- Behavior:
    - Uses reduce() to calculate the sum of all items:
        - const sum = cart.reduce((total, item) => total + item.product.price * item.quantity, 0);
    - Formats total with toFixed(2) for currency display.
    - Called in renderCart() to keep it always updated.
- Result: Accurate, responsive, and neatly formatted total price display.

6. Cart Count Badge Updates

- Feature: Cart badge dynamically shows total item count.
- Behavior:
    - Updated every time renderCart() is called:
        - cartCount.textContent = totalItemsInCart;
    - Reflects live quantity adjustments and item removal.
- Result: Works perfectly and stays synchronized with cart contents.