1. **Cart.js**

// Cart data structure

let cart = {

    items: [],

    subtotal: 0,

    shipping: 0,

    total: 0

};

// Load cart from localStorage when page loads

document.addEventListener('DOMContentLoaded', () => {

    console.log("DOM loaded - initializing cart functionality");

    // Initialize cart

    loadCart();

    updateCartIcon();

    // Add event listener for "Add to Cart" buttons on product pages

    const addToCartButtons = document.querySelectorAll('.add-to-cart-btn');

    console.log(`Found ${addToCartButtons.length} 'Add to Cart' buttons`);

    addToCartButtons.forEach(button => {

        button.addEventListener('click', handleAddToCart);

        console.log("Added click event to button:", button.dataset.productName || "unnamed product");

    });

    // Handle checkout button if on cart page

    const checkoutBtn = document.querySelector('.checkout-btn');

    if (checkoutBtn) {

        checkoutBtn.addEventListener('click', () => {

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

                alert('Your cart is empty. Please add some products before checking out.');

                return;

            }

            // Check if user is logged in

            const currentUser = JSON.parse(localStorage.getItem('currentUser'));

            if (!currentUser) {

                alert('Please login to proceed with checkout.');

                // Store cart in session to retrieve after login

                sessionStorage.setItem('proceedToCheckout', 'true');

                // Open login modal

                const loginModal = document.getElementById('login-modal');

                if (loginModal) {

                    loginModal.style.display = 'flex';

                } else {

                    window.location.href = 'login.html';

                }

                return;

            }

            // If logged in, redirect to checkout page

            window.location.href = 'checkout.html';

        });

    }

    // Render cart if on cart page

    if (document.querySelector('.cart-items')) {

        renderCart();

    }

    // Render order summary if on checkout page

    if (document.querySelector('.order-items')) {

        renderOrderSummary();

    }

});

/\*\*

 \* Loads cart data from localStorage

 \*/

function loadCart() {

    const savedCart = localStorage.getItem('cart');

    if (savedCart) {

        try {

            cart = JSON.parse(savedCart);

            console.log("Cart loaded from localStorage:", cart);

        } catch (e) {

            console.error("Error parsing cart from localStorage:", e);

            localStorage.removeItem('cart'); // Clear invalid cart data

        }

    } else {

        console.log("No cart found in localStorage, using empty cart");

    }

}

/\*\*

 \* Saves cart data to localStorage

 \*/

function saveCart() {

    localStorage.setItem('cart', JSON.stringify(cart));

    console.log("Cart saved to localStorage:", cart);

}

/\*\*

 \* Updates cart count in the header

 \*/

function updateCartIcon() {

    const cartCountElement = document.querySelector('.cart-count');

    if (cartCountElement) {

        const itemCount = cart.items.reduce((total, item) => total + item.quantity, 0);

        cartCountElement.textContent = itemCount;

        // Make the cart count badge visible only if there are items

        cartCountElement.style.display = itemCount > 0 ? 'flex' : 'none';

        console.log("Cart icon updated, count:", itemCount);

    } else {

        console.warn("Cart count element not found in the DOM");

    }

}

/\*\*

 \* Handles adding a product to cart

 \* @param {Event} event - The click event

 \*/

function handleAddToCart(event) {

    console.log("Add to cart button clicked");

    const button = event.currentTarget;

    const productId = button.dataset.productId;

    const productName = button.dataset.productName;

    const productPrice = parseFloat(button.dataset.productPrice);

    console.log("Product details:", {

        id: productId,

        name: productName,

        price: productPrice

    });

    let productImage = '';

    // Safely get the image src

    try {

        const productCard = button.closest('.product-card');

        if (productCard) {

            const imgElement = productCard.querySelector('.product-image img');

            if (imgElement) {

                productImage = imgElement.src;

            } else {

                console.warn("Image element not found for product:", productName);

                productImage = './images/placeholder.jpg'; // Use a placeholder

            }

        } else {

            console.warn("Product card not found for button:", button);

            productImage = './images/placeholder.jpg'; // Use a placeholder

        }

    } catch (e) {

        console.error("Error getting product image:", e);

        productImage = './images/placeholder.jpg'; // Use a placeholder

    }

    // Check if the product is already in the cart

    const existingItem = cart.items.find(item => item.id === productId);

    if (existingItem) {

        existingItem.quantity += 1;

        console.log(`Increased quantity for "${productName}" to ${existingItem.quantity}`);

    } else {

        cart.items.push({

            id: productId,

            name: productName,

            price: productPrice,

            image: productImage,

            quantity: 1

        });

        console.log(`Added new product "${productName}" to cart`);

    }

    // Recalculate cart totals

    updateCartTotals();

    saveCart();

    updateCartIcon();

    // Show success message

    showNotification(button, `${productName} added to cart!`, 'success');

}

/\*\*

 \* Updates the cart totals

 \*/

function updateCartTotals() {

    // Calculate subtotal

    cart.subtotal = cart.items.reduce((total, item) => {

        return total + (item.price \* item.quantity);

    }, 0);

    // Calculate shipping (free for orders over ₹5000)

    cart.shipping = cart.subtotal >= 5000 ? 0 : 350;

    // Calculate total

    cart.total = cart.subtotal + cart.shipping;

    console.log("Cart totals updated:", {

        subtotal: cart.subtotal,

        shipping: cart.shipping,

        total: cart.total

    });

}

/\*\*

 \* Removes an item from the cart

 \* @param {string} productId - ID of product to remove

 \*/

function removeFromCart(productId) {

    const itemToRemove = cart.items.find(item => item.id === productId);

    if (itemToRemove) {

        console.log(`Removing "${itemToRemove.name}" from cart`);

    }

    cart.items = cart.items.filter(item => item.id !== productId);

    updateCartTotals();

    saveCart();

    updateCartIcon();

    renderCart();

    updateSummaryDisplay();

}

// Add a new function to update the summary fields

function updateSummaryDisplay() {

    // Update summary values

    const subtotalEl = document.querySelector('.cart-subtotal');

    const shippingEl = document.querySelector('.cart-shipping');

    const totalEl = document.querySelector('.cart-total');

    if (subtotalEl) subtotalEl.textContent = `₹${cart.subtotal.toFixed(2)}`;

    if (shippingEl) shippingEl.textContent = cart.shipping > 0 ? `₹${cart.shipping.toFixed(2)}` : 'FREE';

    if (totalEl) totalEl.textContent = `₹${cart.total.toFixed(2)}`;

}

/\*\*

 \* Updates quantity of an item in the cart

 \* @param {string} productId - ID of product to update

 \* @param {number} quantity - New quantity value

 \*/

function updateQuantity(productId, quantity) {

    const item = cart.items.find(item => item.id === productId);

    if (item) {

        item.quantity = parseInt(quantity);

        console.log(`Updated quantity for "${item.name}" to ${item.quantity}`);

        // Remove item if quantity is zero or negative

        if (item.quantity <= 0) {

            removeFromCart(productId);

            return;

        }

        updateCartTotals();

        saveCart();

        updateCartIcon();

        renderCart();

    } else {

        console.warn(`Tried to update quantity for product ID ${productId}, but it was not found in cart`);

    }

}

/\*\*

 \* Renders the cart items and totals on the cart page

 \*/

function renderCart() {

    const cartItemsContainer = document.querySelector('.cart-items');

    if (!cartItemsContainer) {

        console.warn("Cart items container not found, can't render cart");

        return;

    }

    if (cart.items.length === 0) {

        console.log("Cart is empty, showing empty message");

        cartItemsContainer.innerHTML = `

            <div class="empty-cart-message">

                Your cart is empty.

                <div style="margin-top: 20px;">

                    <a href="products-catalog.html" class="continue-shopping">

                        <i class="fas fa-arrow-left" style="margin-right: 8px;"></i>Continue Shopping

                    </a>

                </div>

            </div>

        `;

        // Make sure to update summary to zero

        updateSummaryDisplay();

        return;

    }

    console.log(`Rendering ${cart.items.length} items in cart`);

    let cartHTML = '';

    cart.items.forEach(item => {

        cartHTML += `

            <div class="cart-item" data-id="${item.id}">

                <div class="item-image">

                    <img src="${item.image}" alt="${item.name}">

                </div>

                <div class="item-details">

                    <h4>${item.name}</h4>

                    <div class="item-price">₹${item.price}</div>

                </div>

                <div class="item-quantity">

                    <button class="quantity-btn decrease">-</button>

                    <input type="number" value="${item.quantity}" min="1" class="quantity-input">

                    <button class="quantity-btn increase">+</button>

                </div>

                <div class="item-total">₹${(item.price \* item.quantity).toFixed(2)}</div>

                <button class="remove-item">

                    <i class="fas fa-trash"></i>

                </button>

            </div>

        `;

    });

    cartItemsContainer.innerHTML = cartHTML;

    // Update summary display

    updateSummaryDisplay();

    // Add event listeners to the newly created elements

    const quantityBtns = document.querySelectorAll('.quantity-btn');

    quantityBtns.forEach(btn => {

        btn.addEventListener('click', handleQuantityButtonClick);

    });

    const quantityInputs = document.querySelectorAll('.quantity-input');

    quantityInputs.forEach(input => {

        input.addEventListener('change', handleQuantityInputChange);

    });

    const removeButtons = document.querySelectorAll('.remove-item');

    removeButtons.forEach(button => {

        button.addEventListener('click', handleRemoveItemClick);

    });

    console.log("Cart rendered successfully");

}

/\*\*

 \* Renders the order summary on the checkout page

 \*/

function renderOrderSummary() {

    const orderItemsContainer = document.querySelector('.order-items');

    if (!orderItemsContainer) {

        console.warn("Order items container not found, can't render order summary");

        return;

    }

    console.log(`Rendering ${cart.items.length} items in order summary`);

    let orderHTML = '';

    cart.items.forEach(item => {

        orderHTML += `

            <div class="order-item">

                <div class="item-info">

                    <span class="item-name">${item.name}</span>

                    <span class="item-quantity">x${item.quantity}</span>

                </div>

                <div class="item-price">₹${(item.price \* item.quantity).toFixed(2)}</div>

            </div>

        `;

    });

    orderItemsContainer.innerHTML = orderHTML;

    // Update summary values

    const subtotalEl = document.querySelector('.order-subtotal');

    const shippingEl = document.querySelector('.order-shipping');

    const totalEl = document.querySelector('.order-total');

    if (subtotalEl) subtotalEl.textContent = `₹${cart.subtotal.toFixed(2)}`;

    if (shippingEl) shippingEl.textContent = cart.shipping > 0 ? `₹${cart.shipping.toFixed(2)}` : 'FREE';

    if (totalEl) totalEl.textContent = `₹${cart.total.toFixed(2)}`;

    console.log("Order summary rendered successfully");

}

/\*\*

 \* Handles click on the quantity increase/decrease buttons

 \* @param {Event} event - The click event

 \*/

function handleQuantityButtonClick(event) {

    const button = event.currentTarget;

    const cartItem = button.closest('.cart-item');

    const productId = cartItem.dataset.id;

    const quantityInput = cartItem.querySelector('.quantity-input');

    let quantity = parseInt(quantityInput.value);

    if (button.classList.contains('decrease')) {

        quantity = Math.max(1, quantity - 1);

    } else if (button.classList.contains('increase')) {

        quantity++;

    }

    quantityInput.value = quantity;

    updateQuantity(productId, quantity);

}

/\*\*

 \* Handles changes to the quantity input field

 \* @param {Event} event - The change event

 \*/

function handleQuantityInputChange(event) {

    const input = event.currentTarget;

    const cartItem = input.closest('.cart-item');

    const productId = cartItem.dataset.id;

    let quantity = parseInt(input.value);

    // If quantity is zero or less, remove the item

    if (quantity <= 0) {

        if (confirm('Are you sure you want to remove this item from your cart?')) {

            removeFromCart(productId);

        } else {

            // If user cancels, reset quantity to 1

            input.value = 1;

            updateQuantity(productId, 1);

        }

        return;

    }

    // Update with the new quantity

    updateQuantity(productId, quantity);

}

/\*\*

 \* Handles click on the remove item button

 \* @param {Event} event - The click event

 \*/

function handleRemoveItemClick(event) {

    const button = event.currentTarget;

    const cartItem = button.closest('.cart-item');

    const productId = cartItem.dataset.id;

    if (confirm('Are you sure you want to remove this item from your cart?')) {

        removeFromCart(productId);

    }

}

/\*\*

 \* Clears the entire shopping cart

 \*/

function clearCart() {

    console.log("Clearing cart");

    cart.items = [];

    updateCartTotals();

    saveCart();

    updateCartIcon();

    if (document.querySelector('.cart-items')) {

        renderCart();

    }

}

/\*\*

 \* Shows a notification message

 \* @param {HTMLElement} element - Element to show notification near

 \* @param {string} message - Message to display

 \* @param {string} type - Type of notification ('success' or 'error')

 \*/

function showNotification(element, message, type = 'success') {

    console.log(`Showing notification: "${message}" (${type})`);

    const notification = document.createElement('div');

    notification.textContent = message;

    notification.className = `notification ${type}`;

    // Style the notification

    notification.style.cssText = `

        position: fixed;

        top: 20px;

        right: 20px;

        padding: 10px 20px;

        background-color: ${type === 'success' ? '#27ae60' : '#e74c3c'};

        color: white;

        border-radius: 4px;

        z-index: 9999;

        animation: fadeInOut 3s forwards;

    `;

    // Add animation style

    const style = document.createElement('style');

    style.textContent = `

        @keyframes fadeInOut {

            0% { opacity: 0; transform: translateY(-20px); }

            10% { opacity: 1; transform: translateY(0); }

            90% { opacity: 1; transform: translateY(0); }

            100% { opacity: 0; transform: translateY(-20px); }

        }

    `;

    document.head.appendChild(style);

    document.body.appendChild(notification);

    // Remove notification after animation completes

    setTimeout(() => {

        if (notification.parentNode === document.body) {

            document.body.removeChild(notification);

        }

    }, 3000);

}

// Alternative approach to attach event listeners - try this if DOM loaded doesn't work

window.addEventListener('load', function() {

    console.log("Window loaded - backup initializer for cart");

    // Check if we need to add event listeners

    const addToCartButtons = document.querySelectorAll('.add-to-cart-btn');

    if (addToCartButtons.length > 0) {

        console.log(`Found ${addToCartButtons.length} 'Add to Cart' buttons in window.load event`);

        addToCartButtons.forEach(button => {

            // Remove any existing listeners to avoid duplicates

            button.removeEventListener('click', handleAddToCart);

            // Add fresh listener

            button.addEventListener('click', handleAddToCart);

            console.log("Added click event (window.load) to button:", button.dataset.productName || "unnamed product");

        });

    }

});

1. **Checkout.js**

document.addEventListener('DOMContentLoaded', () => {

    // Initialize checkout page

    loadCart();

    renderOrderSummary();

    setupPaymentMethodToggle();

    setupFormValidation();

    setupIndividualFieldValidation();

});

/\*\*

 \* Sets up the payment method toggle functionality

 \*/

function setupPaymentMethodToggle() {

    const paymentMethods = document.querySelectorAll('input[name="paymentMethod"]');

    const creditCardFields = document.getElementById('creditCardFields');

    const upiFields = document.getElementById('upiFields');

    paymentMethods.forEach(method => {

        method.addEventListener('change', () => {

            // Hide all payment method fields

            creditCardFields.style.display = 'none';

            upiFields.style.display = 'none';

            // Show the selected payment method fields

            if (method.value === 'creditCard') {

                creditCardFields.style.display = 'block';

            } else if (method.value === 'upi') {

                upiFields.style.display = 'block';

            }

        });

    });

}

/\*\*

 \* Sets up individual field validation with immediate feedback

 \*/

function setupIndividualFieldValidation() {

    // Full Name validation

    const fullNameInput = document.getElementById('fullName');

    fullNameInput.addEventListener('blur', () => {

        const value = fullNameInput.value.trim();

        if (value === '') {

            showFieldError(fullNameInput, 'Please enter your full name');

        } else if (value.length < 3) {

            showFieldError(fullNameInput, 'Name must be at least 3 characters');

        } else {

            clearFieldError(fullNameInput);

            showFieldSuccess(fullNameInput);

        }

    });

    // Email validation

    const emailInput = document.getElementById('email');

    emailInput.addEventListener('blur', () => {

        const value = emailInput.value.trim();

        if (value === '') {

            showFieldError(emailInput, 'Please enter your email address');

        } else if (!isValidEmail(value)) {

            showFieldError(emailInput, 'Please enter a valid email address');

        } else {

            clearFieldError(emailInput);

            showFieldSuccess(emailInput);

        }

    });

    // Phone validation

    const phoneInput = document.getElementById('phone');

    phoneInput.addEventListener('blur', () => {

        const value = phoneInput.value.trim();

        if (value === '') {

            showFieldError(phoneInput, 'Please enter your phone number');

        } else if (value.length !== 10) {

            showFieldError(phoneInput, 'Phone number must be 10 digits');

        } else {

            clearFieldError(phoneInput);

            showFieldSuccess(phoneInput);

        }

    });

    // Address validation

    const addressInput = document.getElementById('address');

    addressInput.addEventListener('blur', () => {

        const value = addressInput.value.trim();

        if (value === '') {

            showFieldError(addressInput, 'Please enter your address');

        } else if (value.length < 5) {

            showFieldError(addressInput, 'Please enter a complete address');

        } else {

            clearFieldError(addressInput);

            showFieldSuccess(addressInput);

        }

    });

    // City validation

    const cityInput = document.getElementById('city');

    cityInput.addEventListener('blur', () => {

        const value = cityInput.value.trim();

        if (value === '') {

            showFieldError(cityInput, 'Please enter your city');

        } else {

            clearFieldError(cityInput);

            showFieldSuccess(cityInput);

        }

    });

    // State validation

    const stateInput = document.getElementById('state');

    stateInput.addEventListener('blur', () => {

        const value = stateInput.value.trim();

        if (value === '') {

            showFieldError(stateInput, 'Please enter your state');

        } else {

            clearFieldError(stateInput);

            showFieldSuccess(stateInput);

        }

    });

    // Zip Code validation

    const zipCodeInput = document.getElementById('zipCode');

    zipCodeInput.addEventListener('blur', () => {

        const value = zipCodeInput.value.trim();

        if (value === '') {

            showFieldError(zipCodeInput, 'Please enter your postal code');

        } else if (value.length !== 6) {

            showFieldError(zipCodeInput, 'Postal code must be 6 digits');

        } else {

            clearFieldError(zipCodeInput);

            showFieldSuccess(zipCodeInput);

        }

    });

    // Card Number validation

    const cardNumberInput = document.getElementById('cardNumber');

    cardNumberInput.addEventListener('blur', () => {

        const value = cardNumberInput.value.trim();

        if (value === '') {

            showFieldError(cardNumberInput, 'Please enter your card number');

        } else if (value.replace(/\s/g, '').length !== 16) {

            showFieldError(cardNumberInput, 'Card number must be 16 digits');

        } else {

            clearFieldError(cardNumberInput);

            showFieldSuccess(cardNumberInput);

        }

    });

    // Expiry Date validation

    const expiryDateInput = document.getElementById('expiryDate');

    expiryDateInput.addEventListener('blur', () => {

        const value = expiryDateInput.value.trim();

        if (value === '') {

            showFieldError(expiryDateInput, 'Please enter card expiry date');

        } else if (!isValidExpiryDate(value)) {

            showFieldError(expiryDateInput, 'Invalid expiry date (MM/YY)');

        } else {

            clearFieldError(expiryDateInput);

            showFieldSuccess(expiryDateInput);

        }

    });

    // CVV validation

    const cvvInput = document.getElementById('cvv');

    cvvInput.addEventListener('blur', () => {

        const value = cvvInput.value.trim();

        if (value === '') {

            showFieldError(cvvInput, 'Please enter CVV');

        } else if (value.length < 3 || value.length > 4) {

            showFieldError(cvvInput, 'CVV must be 3-4 digits');

        } else {

            clearFieldError(cvvInput);

            showFieldSuccess(cvvInput);

        }

    });

    // Card Name validation

    const cardNameInput = document.getElementById('cardName');

    cardNameInput.addEventListener('blur', () => {

        const value = cardNameInput.value.trim();

        if (value === '') {

            showFieldError(cardNameInput, 'Please enter name on card');

        } else if (value.length < 3) {

            showFieldError(cardNameInput, 'Please enter complete name');

        } else {

            clearFieldError(cardNameInput);

            showFieldSuccess(cardNameInput);

        }

    });

    // UPI ID validation

    const upiIdInput = document.getElementById('upiId');

    upiIdInput.addEventListener('blur', () => {

        const value = upiIdInput.value.trim();

        if (value === '') {

            showFieldError(upiIdInput, 'Please enter your UPI ID');

        } else if (!isValidUpiId(value)) {

            showFieldError(upiIdInput, 'Invalid UPI ID format (username@bank)');

        } else {

            clearFieldError(upiIdInput);

            showFieldSuccess(upiIdInput);

        }

    });

}

/\*\*

 \* Shows error message for a field

 \* @param {HTMLElement} input - Input element

 \* @param {string} message - Error message to display

 \*/

function showFieldError(input, message) {

    // Clear any existing error/success

    clearFieldError(input);

    // Create error message element

    const errorElement = document.createElement('div');

    errorElement.className = 'field-error';

    errorElement.textContent = message;

    errorElement.style.color = '#e74c3c';

    errorElement.style.fontSize = '12px';

    errorElement.style.marginTop = '5px';

    // Add red border to input

    input.style.borderColor = '#e74c3c';

    input.style.boxShadow = '0 0 0 1px #e74c3c';

    // Add error message after input

    input.parentNode.appendChild(errorElement);

}

/\*\*

 \* Shows success indicator for a field

 \* @param {HTMLElement} input - Input element

 \*/

function showFieldSuccess(input) {

    // Add green border to input

    input.style.borderColor = '#27ae60';

    input.style.boxShadow = '0 0 0 1px #27ae60';

    // Add check icon (optional)

    const successIcon = document.createElement('span');

    successIcon.className = 'success-icon';

    successIcon.innerHTML = '✓';

    successIcon.style.color = '#27ae60';

    successIcon.style.position = 'absolute';

    successIcon.style.right = '10px';

    successIcon.style.top = '50%';

    successIcon.style.transform = 'translateY(-50%)';

    // Make parent relative for icon positioning

    if (input.parentNode.style.position !== 'relative') {

        input.parentNode.style.position = 'relative';

    }

    // Add success icon after input

    input.parentNode.appendChild(successIcon);

}

/\*\*

 \* Clears error/success indicators for a field

 \* @param {HTMLElement} input - Input element

 \*/

function clearFieldError(input) {

    // Remove error messages

    const errorElements = input.parentNode.querySelectorAll('.field-error');

    errorElements.forEach(el => el.remove());

    // Remove success icons

    const successIcons = input.parentNode.querySelectorAll('.success-icon');

    successIcons.forEach(el => el.remove());

    // Reset input styling

    input.style.borderColor = '';

    input.style.boxShadow = '';

}

/\*\*

 \* Sets up form validation for the checkout form

 \*/

function setupFormValidation() {

    const checkoutForm = document.querySelector('.checkout-form');

    if (checkoutForm) {

        checkoutForm.addEventListener('submit', (event) => {

            event.preventDefault();

            if (validateCheckoutForm()) {

                processOrder();

            }

        });

        // Add input validation for specific fields

        setupCardNumberValidation();

        setupExpiryDateValidation();

        setupCVVValidation();

        setupZipCodeValidation();

        setupPhoneNumberValidation();

    }

}

/\*\*

 \* Validates credit card number input

 \*/

function setupCardNumberValidation() {

    const cardNumberInput = document.getElementById('cardNumber');

    if (cardNumberInput) {

        cardNumberInput.addEventListener('input', (e) => {

            // Remove any non-digit characters

            let value = e.target.value.replace(/\D/g, '');

            // Add spaces after every 4 digits for readability

            if (value.length > 0) {

                value = value.match(/.{1,4}/g).join(' ');

            }

            // Limit to 19 characters (16 digits + 3 spaces)

            e.target.value = value.substring(0, 19);

        });

    }

}

/\*\*

 \* Validates expiry date input (MM/YY format)

 \*/

function setupExpiryDateValidation() {

    const expiryInput = document.getElementById('expiryDate');

    if (expiryInput) {

        expiryInput.addEventListener('input', (e) => {

            // Remove any non-digit characters

            let value = e.target.value.replace(/\D/g, '');

            // Format as MM/YY

            if (value.length > 2) {

                value = value.substring(0, 2) + '/' + value.substring(2, 4);

            }

            e.target.value = value.substring(0, 5);

        });

    }

}

/\*\*

 \* Validates CVV input (3-4 digits)

 \*/

function setupCVVValidation() {

    const cvvInput = document.getElementById('cvv');

    if (cvvInput) {

        cvvInput.addEventListener('input', (e) => {

            // Only allow digits

            e.target.value = e.target.value.replace(/\D/g, '').substring(0, 4);

        });

    }

}

/\*\*

 \* Validates zip code input

 \*/

function setupZipCodeValidation() {

    const zipCodeInput = document.getElementById('zipCode');

    if (zipCodeInput) {

        zipCodeInput.addEventListener('input', (e) => {

            // Allow only digits for postal code

            e.target.value = e.target.value.replace(/\D/g, '').substring(0, 6);

        });

    }

}

/\*\*

 \* Validates phone number input

 \*/

function setupPhoneNumberValidation() {

    const phoneInput = document.getElementById('phone');

    if (phoneInput) {

        phoneInput.addEventListener('input', (e) => {

            // Allow only digits for phone number

            e.target.value = e.target.value.replace(/\D/g, '').substring(0, 10);

        });

    }

}

/\*\*

 \* Validates the entire checkout form

 \* @returns {boolean} - Whether the form is valid

 \*/

function validateCheckoutForm() {

    let isValid = true;

    // Get form fields

    const fullName = document.getElementById('fullName');

    const email = document.getElementById('email');

    const phone = document.getElementById('phone');

    const address = document.getElementById('address');

    const city = document.getElementById('city');

    const state = document.getElementById('state');

    const zipCode = document.getElementById('zipCode');

    // Basic validation - trigger blur events to show errors

    if (fullName.value.trim() === '') {

        showFieldError(fullName, 'Please enter your full name');

        isValid = false;

    }

    if (email.value.trim() === '' || !isValidEmail(email.value.trim())) {

        showFieldError(email, 'Please enter a valid email address');

        isValid = false;

    }

    if (phone.value.trim() === '' || phone.value.trim().length < 10) {

        showFieldError(phone, 'Please enter a valid phone number');

        isValid = false;

    }

    if (address.value.trim() === '') {

        showFieldError(address, 'Please enter your address');

        isValid = false;

    }

    if (city.value.trim() === '') {

        showFieldError(city, 'Please enter your city');

        isValid = false;

    }

    if (state.value.trim() === '') {

        showFieldError(state, 'Please enter your state');

        isValid = false;

    }

    if (zipCode.value.trim() === '' || zipCode.value.trim().length < 6) {

        showFieldError(zipCode, 'Please enter a valid postal code');

        isValid = false;

    }

    // Payment method-specific validation

    const selectedPaymentMethod = document.querySelector('input[name="paymentMethod"]:checked').value;

    if (selectedPaymentMethod === 'creditCard') {

        const cardNumber = document.getElementById('cardNumber');

        const expiryDate = document.getElementById('expiryDate');

        const cvv = document.getElementById('cvv');

        const cardName = document.getElementById('cardName');

        if (cardNumber.value.trim() === '' || cardNumber.value.replace(/\s/g, '').length < 16) {

            showFieldError(cardNumber, 'Please enter a valid card number');

            isValid = false;

        }

        if (expiryDate.value.trim() === '' || !isValidExpiryDate(expiryDate.value.trim())) {

            showFieldError(expiryDate, 'Please enter a valid expiry date (MM/YY)');

            isValid = false;

        }

        if (cvv.value.trim() === '' || cvv.value.trim().length < 3) {

            showFieldError(cvv, 'Please enter a valid CVV');

            isValid = false;

        }

        if (cardName.value.trim() === '') {

            showFieldError(cardName, 'Please enter the name on your card');

            isValid = false;

        }

    } else if (selectedPaymentMethod === 'upi') {

        const upiId = document.getElementById('upiId');

        if (upiId.value.trim() === '' || !isValidUpiId(upiId.value.trim())) {

            showFieldError(upiId, 'Please enter a valid UPI ID');

            isValid = false;

        }

    }

    // If not valid, scroll to first error

    if (!isValid) {

        const firstError = document.querySelector('.field-error');

        if (firstError) {

            firstError.scrollIntoView({ behavior: 'smooth', block: 'center' });

        }

    }

    return isValid;

}

/\*\*

 \* Validates email format

 \* @param {string} email - Email to validate

 \* @returns {boolean} - Whether the email is valid

 \*/

function isValidEmail(email) {

    const emailRegex = /^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;

    return emailRegex.test(email);

}

/\*\*

 \* Validates expiry date format and value

 \* @param {string} expiryDate - Expiry date to validate (MM/YY format)

 \* @returns {boolean} - Whether the expiry date is valid

 \*/

function isValidExpiryDate(expiryDate) {

    // Check format

    if (!/^\d{2}\/\d{2}$/.test(expiryDate)) {

        return false;

    }

    const [month, year] = expiryDate.split('/').map(part => parseInt(part, 10));

    // Check month is between 1 and 12

    if (month < 1 || month > 12) {

        return false;

    }

    // Get current date

    const now = new Date();

    const currentYear = now.getFullYear() % 100; // Get last two digits of year

    const currentMonth = now.getMonth() + 1; // January is 0

    // Check if the card is not expired

    if (year < currentYear || (year === currentYear && month < currentMonth)) {

        return false;

    }

    return true;

}

/\*\*

 \* Validates UPI ID format

 \* @param {string} upiId - UPI ID to validate

 \* @returns {boolean} - Whether the UPI ID is valid

 \*/

function isValidUpiId(upiId) {

    // Basic UPI ID validation (username@provider)

    const upiRegex = /^[a-zA-Z0-9.\_-]+@[a-zA-Z0-9]+$/;

    return upiRegex.test(upiId);

}

/\*\*

 \* Processes the order after form validation

 \*/

function processOrder() {

    const cart = JSON.parse(localStorage.getItem('cart'));

    if (!cart || cart.items.length === 0) {

        showNotification(null, 'Your cart is empty', 'error');

        return;

    }

    // Get user and shipping information

    const user = JSON.parse(localStorage.getItem('currentUser')) || {};

    const shippingInfo = {

        fullName: document.getElementById('fullName').value.trim(),

        email: document.getElementById('email').value.trim(),

        phone: document.getElementById('phone').value.trim(),

        address: document.getElementById('address').value.trim(),

        city: document.getElementById('city').value.trim(),

        state: document.getElementById('state').value.trim(),

        zipCode: document.getElementById('zipCode').value.trim()

    };

    const paymentMethod = document.querySelector('input[name="paymentMethod"]:checked').value;

    // Generate order

    const order = {

        id: generateOrderId(),

        userId: user.id || 'guest',

        items: cart.items,

        subtotal: cart.subtotal,

        shipping: cart.shipping,

        total: cart.total,

        shippingInfo: shippingInfo,

        paymentMethod: paymentMethod,

        status: 'pending',

        orderDate: new Date().toISOString(),

        estimatedDelivery: getEstimatedDeliveryDate()

    };

    // Save order to localStorage

    saveOrder(order);

    // Clear cart

    clearCart();

    // Show success message and redirect to confirmation page

    showNotification(null, 'Order placed successfully!', 'success');

    // Create a temporary order confirmation in sessionStorage to show on next page

    sessionStorage.setItem('lastOrder', JSON.stringify(order));

    // Redirect to order confirmation page after a short delay

    setTimeout(() => {

        window.location.href = 'order-confirmation.html';

    }, 1500);

}

/\*\*

 \* Generates a unique order ID

 \* @returns {string} - Unique order ID

 \*/

function generateOrderId() {

    const timestamp = new Date().getTime();

    const random = Math.floor(Math.random() \* 1000);

    return `NSH-${timestamp}-${random}`;

}

/\*\*

 \* Calculates estimated delivery date (7-10 days from now)

 \* @returns {string} - Estimated delivery date

 \*/

function getEstimatedDeliveryDate() {

    const today = new Date();

    // Add 7-10 days for delivery

    const deliveryDays = 7 + Math.floor(Math.random() \* 4); // Random between 7-10

    const deliveryDate = new Date(today);

    deliveryDate.setDate(today.getDate() + deliveryDays);

    return deliveryDate.toISOString();

}

/\*\*

 \* Saves the order to localStorage

 \* @param {Object} order - Order object

 \*/

function saveOrder(order) {

    // Get existing orders from localStorage

    const existingOrders = JSON.parse(localStorage.getItem('orders')) || [];

    // Add new order

    existingOrders.push(order);

    // Save back to localStorage

    localStorage.setItem('orders', JSON.stringify(existingOrders));

}

/\*\*

 \* Shows a notification message

 \* @param {HTMLElement} element - Element to show notification near (null for centered)

 \* @param {string} message - Message to display

 \* @param {string} type - Type of notification ('success' or 'error')

 \*/

function showNotification(element, message, type = 'success') {

    const notification = document.createElement('div');

    notification.textContent = message;

    notification.className = `notification ${type}`;

    // Style the notification

    notification.style.cssText = `

        position: fixed;

        top: 20px;

        left: 50%;

        transform: translateX(-50%);

        padding: 15px 25px;

        background-color: ${type === 'success' ? '#27ae60' : '#e74c3c'};

        color: white;

        border-radius: 8px;

        z-index: 9999;

        box-shadow: 0 4px 15px rgba(0, 0, 0, 0.2);

        font-weight: 500;

        animation: fadeInOut 3s forwards;

        min-width: 280px;

        text-align: center;

    `;

    // Add animation style

    const style = document.createElement('style');

    style.textContent = `

        @keyframes fadeInOut {

            0% { opacity: 0; transform: translate(-50%, -20px); }

            10% { opacity: 1; transform: translate(-50%, 0); }

            90% { opacity: 1; transform: translate(-50%, 0); }

            100% { opacity: 0; transform: translate(-50%, -20px); }

        }

    `;

    document.head.appendChild(style);

    document.body.appendChild(notification);

    // Remove notification after animation completes

    setTimeout(() => {

        if (notification.parentNode === document.body) {

            document.body.removeChild(notification);

        }

    }, 3000);

}

/\*\*

 \* Loads cart data from localStorage

 \*/

function loadCart() {

    const savedCart = localStorage.getItem('cart');

    if (savedCart) {

        return JSON.parse(savedCart);

    }

    return {

        items: [],

        subtotal: 0,

        shipping: 0,

        total: 0

    };

}

/\*\*

 \* Clears cart data from localStorage

 \*/

function clearCart() {

    localStorage.removeItem('cart');

}

/\*\*

 \* Renders the order summary on the checkout page

 \*/

function renderOrderSummary() {

    const cart = loadCart();

    const orderItemsContainer = document.querySelector('.order-items');

    if (!orderItemsContainer) return;

    let orderHTML = '';

    if (cart.items.length === 0) {

        orderHTML = '<div class="empty-cart-message">Your cart is empty</div>';

        orderItemsContainer.innerHTML = orderHTML;

        // Disable the place order button

        const placeOrderBtn = document.querySelector('.place-order-btn');

        if (placeOrderBtn) {

            placeOrderBtn.disabled = true;

            placeOrderBtn.style.opacity = '0.5';

            placeOrderBtn.style.cursor = 'not-allowed';

        }

        return;

    }

    cart.items.forEach(item => {

        orderHTML += `

            <div class="order-item">

                <div class="item-info">

                    <span class="item-name">${item.name}</span>

                    <span class="item-quantity">x${item.quantity}</span>

                </div>

                <div class="item-price">₹${(item.price \* item.quantity).toFixed(2)}</div>

            </div>

        `;

    });

    orderItemsContainer.innerHTML = orderHTML;

    // Update summary values

    document.querySelector('.order-subtotal').textContent = `₹${cart.subtotal.toFixed(2)}`;

    document.querySelector('.order-shipping').textContent = cart.shipping > 0 ? `₹${cart.shipping.toFixed(2)}` : 'FREE';

    document.querySelector('.order-total').textContent = `₹${cart.total.toFixed(2)}`;

}

1. **Order-confirmation.js**

document.addEventListener('DOMContentLoaded', () => {

    // Load and display order details

    displayOrderConfirmation();

    // Set up print invoice button

    const printInvoiceBtn = document.getElementById('printInvoiceBtn');

    if (printInvoiceBtn) {

        printInvoiceBtn.addEventListener('click', printInvoice);

    }

});

function saveOrderToLocalStorage(order) {

    // Get current user

    const currentUser = JSON.parse(localStorage.getItem('currentUser'));

    // Add user ID to order

    if (currentUser && currentUser.id) {

        order.userId = currentUser.id;

        if (currentUser.email) {

            order.userEmail = currentUser.email;

        }

        console.log("Adding user ID to order:", currentUser.id);

    } else {

        console.log("No logged in user found!");

        // Create a temporary user ID if no user is logged in

        order.userId = "guest-" + Date.now();

        // Also store the email from shipping info if available

        if (order.shippingInfo && order.shippingInfo.email) {

            order.userEmail = order.shippingInfo.email;

            console.log("Added guest email from shipping info:", order.userEmail);

        }

    }

    // Get all orders from localStorage

    let allOrders = JSON.parse(localStorage.getItem('orders')) || [];

    console.log("Existing orders in localStorage:", allOrders.length);

    // Check if order already exists

    const existingOrderIndex = allOrders.findIndex(o => o.id === order.id);

    if (existingOrderIndex >= 0) {

        // Update existing order

        allOrders[existingOrderIndex] = order;

        console.log("Updated existing order in localStorage:", order.id);

    } else {

        // Add new order

        allOrders.push(order);

        console.log("Added new order to localStorage:", order.id);

    }

    // Save back to localStorage

    localStorage.setItem('orders', JSON.stringify(allOrders));

    // Verify order was saved

    const verifyOrders = JSON.parse(localStorage.getItem('orders')) || [];

    const savedOrder = verifyOrders.find(o => o.id === order.id);

    if (savedOrder) {

        console.log("Order verified in localStorage:", order.id);

    } else {

        console.error("Failed to save order to localStorage!");

    }

}

/\*\*

 \* Displays the order confirmation details

 \*/

function displayOrderConfirmation() {

    // Get order from session storage

    const orderJSON = sessionStorage.getItem('lastOrder');

    if (!orderJSON) {

        // If no order in session storage, check if order ID is in URL params

        const urlParams = new URLSearchParams(window.location.search);

        const orderId = urlParams.get('orderId');

        if (orderId) {

            // Try to load order from localStorage by ID

            const orders = JSON.parse(localStorage.getItem('orders')) || [];

            const order = orders.find(o => o.id === orderId);

            if (order) {

                displayOrderDetails(order);

            } else {

                showOrderNotFound();

            }

        } else {

            showOrderNotFound();

        }

    } else {

        // Display order from session storage

        const order = JSON.parse(orderJSON);

        displayOrderDetails(order);

    }

}

/\*\*

 \* Displays order details on the page

 \* @param {Object} order - The order object to display

 \*/

function displayOrderDetails(order) {

    // Set order information

    document.getElementById('orderIdValue').textContent = order.id;

    document.getElementById('orderDateValue').textContent = formatDate(order.orderDate);

    document.getElementById('paymentMethodValue').textContent = formatPaymentMethod(order.paymentMethod);

    document.getElementById('orderStatusValue').textContent = capitalizeFirstLetter(order.status);

    // Set shipping information

    const shippingAddressHTML = `

        ${order.shippingInfo.fullName}<br>

        ${order.shippingInfo.address}<br>

        ${order.shippingInfo.city}, ${order.shippingInfo.state} ${order.shippingInfo.zipCode}<br>

        Phone: ${order.shippingInfo.phone}<br>

        Email: ${order.shippingInfo.email}

    `;

    document.getElementById('shippingAddressValue').innerHTML = shippingAddressHTML;

    // Set order products

    const orderProductsList = document.getElementById('orderProductsList');

    let productsHTML = '';

    order.items.forEach(item => {

        productsHTML += `

            <div class="order-product">

                <div class="product-details">

                    <div class="product-name">${item.name}</div>

                    <div class="product-quantity">Quantity: ${item.quantity}</div>

                </div>

                <div class="product-price">₹${(item.price \* item.quantity).toFixed(2)}</div>

            </div>

        `;

    });

    orderProductsList.innerHTML = productsHTML;

    // Set order summary

    document.getElementById('subtotalValue').textContent = `₹${order.subtotal.toFixed(2)}`;

    document.getElementById('shippingValue').textContent = order.shipping > 0 ? `₹${order.shipping.toFixed(2)}` : 'FREE';

    document.getElementById('totalValue').textContent = `₹${order.total.toFixed(2)}`;

    // Set estimated delivery

    document.getElementById('estimatedDeliveryValue').textContent = formatDate(order.estimatedDelivery);

    // IMPORTANT: Save the order to localStorage

    saveOrderToLocalStorage(order);

}

/\*\*

 \* Shows a message when order is not found

 \*/

function showOrderNotFound() {

    const confirmationContent = document.querySelector('.confirmation-content');

    if (confirmationContent) {

        confirmationContent.innerHTML = `

            <div class="order-not-found">

                <h3>Order Not Found</h3>

                <p>We couldn't find your order information. Please check your email for order details or contact customer support.</p>

                <div class="action-buttons">

                    <a href="index.html" class="action-button">Return to Home</a>

                </div>

            </div>

        `;

    }

}

/\*\*

 \* Formats a date string into a readable format

 \* @param {string} dateString - ISO date string

 \* @returns {string} - Formatted date

 \*/

function formatDate(dateString) {

    const date = new Date(dateString);

    const options = {

        year: 'numeric',

        month: 'long',

        day: 'numeric',

        hour: '2-digit',

        minute: '2-digit'

    };

    return date.toLocaleDateString('en-IN', options);

}

/\*\*

 \* Formats payment method to be more readable

 \* @param {string} method - Payment method

 \* @returns {string} - Formatted payment method

 \*/

function formatPaymentMethod(method) {

    switch(method) {

        case 'creditCard':

            return 'Credit/Debit Card';

        case 'upi':

            return 'UPI Payment';

        case 'cod':

            return 'Cash on Delivery';

        default:

            return capitalizeFirstLetter(method);

    }

}

/\*\*

 \* Capitalizes the first letter of a string

 \* @param {string} str - String to capitalize

 \* @returns {string} - Capitalized string

 \*/

function capitalizeFirstLetter(str) {

    return str.charAt(0).toUpperCase() + str.slice(1);

}

/\*\*

 \* Prints the invoice

 \*/

function printInvoice() {

    window.print();

}

1. **Order-history.js**

document.addEventListener('DOMContentLoaded', () => {

    const container = document.querySelector('.order-history-container');

    const currentUser = JSON.parse(localStorage.getItem('currentUser'));

    const allOrders = JSON.parse(localStorage.getItem('orders')) || [];

    const userOrders = filterUserOrders(allOrders, currentUser);

    if (userOrders.length === 0) {

      container.innerHTML = `

        <div class="no-orders-message">You haven't placed any orders yet.</div>

      `;

      return;

    }

    container.innerHTML = '';

    userOrders.reverse().forEach(order => {

      const orderCard = createOrderCard(order);

      container.appendChild(orderCard);

    });

  });

  /\*\*

   \* Creates and returns an order card element

   \* @param {Object} order

   \* @returns {HTMLElement}

   \*/

  function createOrderCard(order) {

    const card = document.createElement('div');

    card.className = 'order-card';

    const statusClass = `status-${order.status.toLowerCase()}`;

    const itemSummary = order.items.slice(0, 2).map(item => `

      <div class="order-item-summary">

        <span>${item.name} x${item.quantity}</span>

        <span>₹${(item.price \* item.quantity).toFixed(2)}</span>

      </div>

    `).join('');

    const moreItems = order.items.length > 2

      ? `<div class="more-items">+${order.items.length - 2} more item(s)</div>`

      : '';

    card.innerHTML = `

      <div class="order-header">

        <div class="order-info">

          <h3>Order ID: ${order.id}</h3>

          <p class="order-date">${formatDate(order.orderDate)}</p>

        </div>

        <div class="order-status ${statusClass}">${capitalizeFirstLetter(order.status)}</div>

      </div>

      <div class="order-details">

        <div class="order-items-summary">

          ${itemSummary}

          ${moreItems}

        </div>

        <div class="order-metadata">

          <div class="metadata-item"><span class="label">Subtotal:</span><span class="value">₹${order.subtotal.toFixed(2)}</span></div>

          <div class="metadata-item"><span class="label">Shipping:</span><span class="value">${order.shipping > 0 ? `₹${order.shipping.toFixed(2)}` : 'FREE'}</span></div>

          <div class="metadata-item"><span class="label">Total:</span><span class="value">₹${order.total.toFixed(2)}</span></div>

          <div class="metadata-item"><span class="label">Delivery By:</span><span class="value">${formatDate(order.estimatedDelivery)}</span></div>

        </div>

      </div>

      <div class="order-actions">

        <button class="view-details-btn" onclick="viewOrderDetails('${order.id}')">View Details</button>

      </div>

    `;

    return card;

  }

  /\*\*

   \* Capitalizes the first letter of a string

   \* @param {string} str

   \* @returns {string}

   \*/

  function capitalizeFirstLetter(str) {

    return str.charAt(0).toUpperCase() + str.slice(1);

  }

  /\*\*

   \* Formats ISO date into a readable format

   \* @param {string} dateString

   \* @returns {string}

   \*/

  function formatDate(dateString) {

    const date = new Date(dateString);

    const options = {

      year: 'numeric',

      month: 'short',

      day: 'numeric'

    };

    return date.toLocaleDateString('en-IN', options);

  }

  /\*\*

   \* Opens the login modal

   \*/

  function openLoginModal() {

    const modal = document.getElementById('login-modal');

    if (modal) {

      modal.style.display = 'flex';

    }

  }

  /\*\*

   \* Handles "View Details" button click

   \* Redirects to order confirmation page with orderId in URL

   \* @param {string} orderId

   \*/

  function viewOrderDetails(orderId) {

    window.location.href = `order-confirmation.html?orderId=${orderId}`;

  }

  document.addEventListener('DOMContentLoaded', function() {

    // Add a diagnostic button

    const container = document.querySelector('.order-history-container');

    if (container) {

        const diagnosticBtn = document.createElement('button');

        diagnosticBtn.innerText = 'Check Order Storage';

        diagnosticBtn.style.padding = '10px 15px';

        diagnosticBtn.style.marginTop = '20px';

        diagnosticBtn.style.backgroundColor = '#3498db';

        diagnosticBtn.style.color = 'white';

        diagnosticBtn.style.border = 'none';

        diagnosticBtn.style.borderRadius = '4px';

        diagnosticBtn.style.cursor = 'pointer';

        diagnosticBtn.addEventListener('click', function() {

            const currentUser = JSON.parse(localStorage.getItem('currentUser'));

            const allOrders = JSON.parse(localStorage.getItem('orders')) || [];

            const lastOrder = JSON.parse(sessionStorage.getItem('lastOrder'));

            let message = "===== STORAGE DIAGNOSTIC =====\n";

            message += `Current User: ${currentUser ? JSON.stringify(currentUser) : 'Not logged in'}\n\n`;

            message += `Total Orders in localStorage: ${allOrders.length}\n`;

            message += `Last Order in sessionStorage: ${lastOrder ? 'Yes' : 'No'}\n\n`;

            if (allOrders.length > 0) {

                message += "Orders in localStorage:\n";

                allOrders.forEach((order, index) => {

                    message += `${index+1}. ID: ${order.id}, User: ${order.userId || 'none'}\n`;

                });

            }

            alert(message);

        });

        // Add the clear storage button

        const clearStorageBtn = document.createElement('button');

        clearStorageBtn.innerText = 'Clear Order Storage';

        clearStorageBtn.style.padding = '10px 15px';

        clearStorageBtn.style.marginTop = '20px';

        clearStorageBtn.style.marginLeft = '10px';

        clearStorageBtn.style.backgroundColor = '#e74c3c';

        clearStorageBtn.style.color = 'white';

        clearStorageBtn.style.border = 'none';

        clearStorageBtn.style.borderRadius = '4px';

        clearStorageBtn.style.cursor = 'pointer';

        clearStorageBtn.addEventListener('click', clearOrderStorage);

        container.parentNode.insertBefore(diagnosticBtn, container.nextSibling);

        container.parentNode.insertBefore(clearStorageBtn, diagnosticBtn.nextSibling);

    }

});

function filterUserOrders(allOrders, currentUser) {

    // If user is logged in, show both their user ID matched orders and any guest orders

    // that may have been created with their email

    return allOrders.filter(order => {

      // Match by user ID

      const userIdMatch = order.userId && currentUser.id && order.userId === currentUser.id;

      // Match by email (if available)

      const emailMatch = order.userEmail && currentUser.email && order.userEmail === currentUser.email;

      // Match guest orders if they contain the user's email in shipping info

      let guestEmailMatch = false;

      if (order.userId && order.userId.startsWith('guest-') && order.shippingInfo &&

          order.shippingInfo.email && currentUser.email &&

          order.shippingInfo.email === currentUser.email) {

        guestEmailMatch = true;

      }

      return userIdMatch || emailMatch || guestEmailMatch;

    });

  }

function clearOrderStorage() {

    // Clear orders from localStorage

    localStorage.removeItem('orders');

    // Clear last order from sessionStorage

    sessionStorage.removeItem('lastOrder');

    // Show confirmation message

    alert("Order storage cleared successfully!");

    // Reload the page to reflect changes

    window.location.reload();

}

1. **Script.js**

const createDynamicMenu = () => {

    const menuData = [

        { id: 'index.html', text: 'Home', icon: 'fa-home' },

        { id: 'about.html', text: 'About Us', icon: 'fa-info-circle' },

        { id: 'products.html', text: 'Products', icon: 'fa-boxes' },

        { id: 'products-catalog.html', text: 'Products Catalog', icon: 'fa-th' },

        { id: 'services.html', text: 'Services', icon: 'fa-concierge-bell' },

        { id: 'testimonials.html', text: 'Testimonials', icon: 'fa-quote-right' },

        { id: 'contact.html', text: 'Contact', icon: 'fa-envelope' },

        { id: 'login.html', text: 'Login', icon: 'fa-sign-in-alt' },

        { id: 'register.html', text: 'Register', icon: 'fa-user-plus' }

    ];

    // Create menu container

    const menuPanel = document.createElement('div');

    menuPanel.className = 'dynamic-menu';

    menuPanel.style.cssText = `

        position: fixed;

        top: 0;

        left: -300px;

        width: 300px;

        height: 100vh;

        background: rgba(44, 62, 80, 0.95);

        transition: left 0.3s ease;

        z-index: 1001;

        padding: 20px;

        box-shadow: 2px 0 5px rgba(0,0,0,0.2);

    `;

    // Create menu content

    const menuContent = menuData.map(item => `

        <a href="${item.id}" class="menu-item">

            <i class="fas ${item.icon}"></i>

            <span>${item.text}</span>

        </a>

    `).join('');

    menuPanel.innerHTML = menuContent;

    document.body.appendChild(menuPanel);

    // Add menu toggle button

    const toggleBtn = document.createElement('button');

    toggleBtn.className = 'menu-toggle';

    toggleBtn.innerHTML = '<i class="fas fa-bars"></i>';

    toggleBtn.style.cssText = `

        position: fixed;

        top: 20px;

        left: 20px;

        z-index: 1002;

        background: #e74c3c;

        color: white;

        border: none;

        padding: 10px;

        border-radius: 5px;

        cursor: pointer;

        transition: background 0.3s ease;

    `;

    document.body.appendChild(toggleBtn);

    // Add menu item styling

    const style = document.createElement('style');

    style.textContent = `

        .menu-item {

            display: flex;

            align-items: center;

            padding: 15px;

            color: white;

            text-decoration: none;

            transition: background 0.3s ease;

            border-radius: 5px;

            margin-bottom: 5px;

        }

        .menu-item:hover {

            background: #e74c3c;

        }

        .menu-item i {

            margin-right: 15px;

            width: 20px;

            text-align: center;

        }

    `;

    document.head.appendChild(style);

    // Event Handlers

    let isMenuOpen = false;

    toggleBtn.addEventListener('click', () => {

        isMenuOpen = !isMenuOpen;

        menuPanel.style.left = isMenuOpen ? '0' : '-300px';

    });

    // Close menu when clicking outside

    document.addEventListener('click', (e) => {

        if (isMenuOpen && !menuPanel.contains(e.target) && e.target !== toggleBtn) {

            isMenuOpen = false;

            menuPanel.style.left = '-300px';

        }

    });

};

// Add password visibility toggle to all password fields

function addPasswordToggle() {

    const passwordFields = document.querySelectorAll('input[type="password"]');

    passwordFields.forEach(passwordField => {

        // Create container for password field and icon

        const container = document.createElement('div');

        container.style.cssText = `

            position: relative;

            width: 100%;

            display: flex;

            align-items: center;

        `;

        // Wrap password field in container

        passwordField.parentNode.insertBefore(container, passwordField);

        container.appendChild(passwordField);

        // Style password field

        passwordField.style.cssText = `

            width: 100%;

            padding-right: 35px; /\* Make room for the icon \*/

        `;

        // Create eye icon container

        const eyeIcon = document.createElement('span');

        eyeIcon.innerHTML = '<i class="fas fa-eye"></i>';

        eyeIcon.className = 'password-toggle';

        eyeIcon.style.cssText = `

            position: absolute;

            right: 10px;

            top: 50%;

            transform: translateY(-50%);

            cursor: pointer;

            color: #666;

            padding: 5px;

            z-index: 1;

        `;

        container.appendChild(eyeIcon);

        // Toggle password visibility

        eyeIcon.addEventListener('click', () => {

            const type = passwordField.getAttribute('type');

            passwordField.setAttribute('type', type === 'password' ? 'text' : 'password');

            eyeIcon.innerHTML = type === 'password' ? '<i class="fas fa-eye-slash"></i>' : '<i class="fas fa-eye"></i>';

        });

    });

}

// =========================================

// Form Validation Implementation

// Real-time validation for all forms

// =========================================

const setupFormValidation = () => {

    const forms = document.querySelectorAll('form');

    const validationRules = {

        'text': {

            pattern: '[A-Za-z\\s]{2,}',

            message: 'Please enter at least 2 characters, letters only'

        },

        'email': {

            pattern: '[a-z0-9.\_%+-]+@[a-z0-9.-]+\\.[a-z]{2,}$',

            message: 'Please enter a valid email address'

        },

        'password': {

            pattern: '^(?=.\*[A-Za-z])(?=.\*\\d)(?=.\*[@$!%\*#?&])[A-Za-z\\d@$!%\*#?&]{8,}$',

            message: 'Password must include at least one letter, one number, and one special character (minimum 8 characters)'

        },

        'phone': {

            pattern: '^\\+?[0-9]{10,}$',

            message: 'Please enter a valid phone number with at least 10 digits'

        }

    };

    forms.forEach(form => {

        const inputs = form.querySelectorAll('input, textarea');

        inputs.forEach(input => {

            // Create validation message element

            const validationMsg = document.createElement('span');

            validationMsg.className = 'validation-message';

            validationMsg.style.cssText = `

                color: #e74c3c;

                font-size: 0.8em;

                margin-top: 5px;

                display: none;

            `;

            input.parentNode.insertBefore(validationMsg, input.nextSibling);

            // Real-time validation

            input.addEventListener('input', () => {

                const inputType = input.getAttribute('type');

                const rule = validationRules[inputType] || validationRules['text'];

                const isValid = new RegExp(rule.pattern).test(input.value);

                validationMsg.style.display = isValid ? 'none' : 'block';

                validationMsg.textContent = isValid ? '' : rule.message;

                input.style.borderColor = isValid ? '#ddd' : '#e74c3c';

                // Additional validation for confirm password

                if (inputType === 'password' && input.placeholder === 'Confirm Password') {

                    const passwordInput = input.form.querySelector('input[placeholder="Password"]');

                    if (passwordInput && input.value !== passwordInput.value) {

                        validationMsg.style.display = 'block';

                        validationMsg.textContent = 'Passwords do not match';

                        input.style.borderColor = '#e74c3c';

                    }

                }

            });

        });

    });

};

// =========================================

// User Authentication Management

// Handles user display, logout, and tracking

// =========================================

// Store current user session information

let currentUser = null;

/\*\*

 \* Updates the UI to show logged-in user information

 \* @param {string} email - User's email from login

 \*/

function updateUserUI(email) {

    // Get user info from registration data

    const registeredUsers = JSON.parse(localStorage.getItem('register\_responses') || '[]');

    const userInfo = registeredUsers.find(user => (user.email || user.Email) === email);

    if (!userInfo) return;

    // Store current user info

    currentUser = {

        name: userInfo.name || userInfo.Name || email,

        email: email,

        loginTime: new Date().toLocaleString()

    };

    // Save user info to localStorage for persistence across pages

    localStorage.setItem('currentUser', JSON.stringify(currentUser));

    // Create user info display container if it doesn't exist

    let userContainer = document.querySelector('.user-info-container');

    if (!userContainer) {

        userContainer = document.createElement('div');

        userContainer.className = 'user-info-container';

        userContainer.style.cssText = `

            position: fixed;

            top: 10%;

            left: 50%;

            transform: translate(-50%, -50%);

            background: rgba(44, 62, 80, 0.95);

            color: white;

            padding: 10px 15px;

            border-radius: 5px;

            z-index: 1000;

            display: flex;

            align-items: center;

            gap: 10px;

        `;

        document.body.appendChild(userContainer);

    }

    // Update container content

    userContainer.innerHTML = `

        <span>Welcome, ${currentUser.name}</span>

        <button id="logoutBtn" class="logout-btn" style="

            background: #e74c3c;

            color: white;

            border: none;

            padding: 5px 10px;

            border-radius: 3px;

            cursor: pointer;

        ">Logout</button>

    `;

    userContainer.style.alignItems = "center";

    // Add logout button functionality

    document.getElementById('logoutBtn').addEventListener('click', handleLogout);

    // Disable registration options

    disableRegistrationOptions();

}

/\*\*

 \* Handles user logout process

 \*/

function handleLogout() {

    if (currentUser) {

        // Save logout record

        const logoutRecord = {

            name: currentUser.name,

            email: currentUser.email,

            loginTime: currentUser.loginTime,

            logoutTime: new Date().toLocaleString()

        };

        // Save to localStorage

        const logoutRecords = JSON.parse(localStorage.getItem('logout\_records') || '[]');

        logoutRecords.push(logoutRecord);

        localStorage.setItem('logout\_records', JSON.stringify(logoutRecords));

        // Clear current user

        currentUser = null;

        localStorage.removeItem('currentUser');

        // Remove user info container

        const userContainer = document.querySelector('.user-info-container');

        if (userContainer) {

            userContainer.remove();

        }

        // Re-enable registration options

        enableRegistrationOptions();

        // Show logout notification

        const notification = document.createElement('div');

        notification.textContent = 'Logged out successfully!';

        notification.style.cssText = `

            position: fixed;

            top: 20px;

            right: 20px;

            background: #27ae60;

            color: white;

            padding: 10px 15px;

            border-radius: 5px;

            z-index: 1000;

        `;

        document.body.appendChild(notification);

        // Remove notification after 3 seconds

        setTimeout(() => notification.remove(), 3000);

        // Redirect to home page

        window.location.href = 'index.html';

    }

}

/\*\*

 \* Checks if user is logged in and updates UI accordingly

 \*/

function checkUserLogin() {

    const savedUser = localStorage.getItem('currentUser');

    if (savedUser) {

        currentUser = JSON.parse(savedUser);

        updateUserUI(currentUser.email);

        // If on login or register page, redirect to home

        const currentPage = window.location.pathname.split('/').pop();

        if (currentPage === 'login.html' || currentPage === 'register.html') {

            window.location.href = 'index.html';

        }

    }

}

/\*\*

 \* Disables registration options when user is logged in

 \*/

function disableRegistrationOptions() {

    // Disable registration links

    document.querySelectorAll('a[href="register.html"]').forEach(link => {

        link.style.pointerEvents = 'none';

        link.style.opacity = '0.5';

        link.style.textDecoration = 'line-through';

        link.setAttribute('title', 'Please log out first');

    });

    // Disable registration modals

    const registerModal = document.getElementById('register-modal');

    if (registerModal) {

        const modalTriggers = document.querySelectorAll('[data-target="#register-modal"]');

        modalTriggers.forEach(trigger => {

            trigger.style.pointerEvents = 'none';

            trigger.style.opacity = '0.5';

        });

    }

}

/\*\*

 \* Re-enables registration options after logout

 \*/

function enableRegistrationOptions() {

    // Re-enable registration links

    document.querySelectorAll('a[href="register.html"]').forEach(link => {

        link.style.pointerEvents = 'auto';

        link.style.opacity = '1';

        link.style.textDecoration = 'none';

        link.removeAttribute('title');

    });

    // Re-enable registration modals

    const registerModal = document.getElementById('register-modal');

    if (registerModal) {

        const modalTriggers = document.querySelectorAll('[data-target="#register-modal"]');

        modalTriggers.forEach(trigger => {

            trigger.style.pointerEvents = 'auto';

            trigger.style.opacity = '1';

        });

    }

}

/\*\*

 \* Displays user login/logout history

 \* @param {HTMLElement} container - Container element for displaying records

 \*/

function displayUserHistory(container) {

    const logoutRecords = JSON.parse(localStorage.getItem('logout\_records') || '[]');

    if (logoutRecords.length === 0) {

        container.innerHTML = '<p style="text-align: center; color: #666;">No login history available</p>';

        return;

    }

    let tableHTML = `

        <table style="width: 100%; border-collapse: collapse; font-size: 0.9rem;">

            <thead>

                <tr style="background: #2c3e50; color: white;">

                    <th style="padding: 10px; text-align: left; border: 1px solid #ddd;">Name</th>

                    <th style="padding: 10px; text-align: left; border: 1px solid #ddd;">Email</th>

                    <th style="padding: 10px; text-align: left; border: 1px solid #ddd;">Login Time</th>

                    <th style="padding: 10px; text-align: left; border: 1px solid #ddd;">Logout Time</th>

                </tr>

            </thead>

            <tbody>

    `;

    logoutRecords.forEach((record, index) => {

        const rowStyle = index % 2 === 0 ? 'background: #f9f9f9;' : '';

        tableHTML += `

            <tr style="${rowStyle}">

                <td style="padding: 10px; border: 1px solid #ddd;">${record.name}</td>

                <td style="padding: 10px; border: 1px solid #ddd;">${record.email}</td>

                <td style="padding: 10px; border: 1px solid #ddd;">${record.loginTime}</td>

                <td style="padding: 10px; border: 1px solid #ddd;">${record.logoutTime}</td>

            </tr>

        `;

    });

    tableHTML += `

            </tbody>

        </table>

        <div style="text-align: right; margin-top: 15px;">

            <button class="clear-history-btn" style="padding: 8px 12px; background: #e74c3c; color: white; border: none; border-radius: 4px; cursor: pointer;">

                Clear History

            </button>

        </div>

    `;

    container.innerHTML = tableHTML;

    // Add clear history functionality

    container.querySelector('.clear-history-btn').addEventListener('click', () => {

        if (confirm('Are you sure you want to clear all login history?')) {

            localStorage.removeItem('logout\_records');

            displayUserHistory(container);

        }

    });

}

// =========================================

// Form Data Local Storage Implementation

// This code handles:

// 1. Storing form submissions in localStorage

// 2. Displaying saved responses in dynamic tables

// 3. Adding "View Responses" buttons to forms

// =========================================

// Set up local storage functionality for all forms

function setupLocalStorage() {

    // Target the login, register and contact forms - BOTH modal and standalone versions

    const loginForms = document.querySelectorAll('form.auth-form');

    const contactForm = document.querySelector('.contact-form');

    // Configure each login/register form with local storage capability

    loginForms.forEach(form => {

        const isLoginForm = form.querySelector('input[placeholder="Password"]') && !form.querySelector('input[placeholder="Confirm Password"]');

        const isRegisterForm = form.querySelector('input[placeholder="Confirm Password"]');

        if (isLoginForm) {

            setupFormStorage(form, 'login\_responses');

        } else if (isRegisterForm) {

            setupFormStorage(form, 'register\_responses');

        }

    });

    // Setup contact form if it exists

    if (contactForm) setupFormStorage(contactForm, 'contact\_responses');

}

/\*\*

 \* Sets up local storage and response display for a specific form

 \* @param {HTMLElement} form - The form element to set up

 \* @param {string} storageKey - Key to use in localStorage

 \*/

function setupFormStorage(form, storageKey) {

    // Create container for displaying stored responses if not already present

    let responseContainer = form.parentNode.querySelector('.response-container');

    if (!responseContainer) {

        responseContainer = document.createElement('div');

        responseContainer.className = 'response-container';

        responseContainer.style.cssText = `

            margin-top: 20px;

            padding: 15px;

            background: #f8f9fa;

            border-radius: 5px;

            max-height: 300px;

            overflow-y: auto;

            display: none;

            box-shadow: 0 2px 4px rgba(0,0,0,0.1);

        `;

        form.parentNode.appendChild(responseContainer);

    }

    // Create "View Responses" button if not already present

    if (!form.querySelector('.view-responses-btn')) {

        const viewResponsesBtn = document.createElement('button');

        viewResponsesBtn.textContent = 'View Responses';

        viewResponsesBtn.className = 'view-responses-btn';

        viewResponsesBtn.type = 'button';

        viewResponsesBtn.style.cssText = `

            padding: 0.8rem 1.5rem;

            background: #2c3e50;

            color: white;

            border: none;

            border-radius: 4px;

            cursor: pointer;

            margin-top: 15px;

            transition: background 0.3s;

        `;

        form.appendChild(viewResponsesBtn);

        // View Responses button functionality

        viewResponsesBtn.addEventListener('click', function() {

            const isVisible = responseContainer.style.display === 'block';

            responseContainer.style.display = isVisible ? 'none' : 'block';

            if (!isVisible) {

                displayStoredResponses(responseContainer, storageKey);

            }

        });

    }

    // Add Login History button only for login forms

    if (storageKey === 'login\_responses' && !form.querySelector('.view-history-btn')) {

        const viewHistoryBtn = document.createElement('button');

        viewHistoryBtn.textContent = 'View Login History';

        viewHistoryBtn.className = 'view-history-btn';

        viewHistoryBtn.type = 'button';

        viewHistoryBtn.style.cssText = `

            padding: 0.8rem 1.5rem;

            background: #2c3e50;

            color: white;

            border: none;

            border-radius: 4px;

            cursor: pointer;

            margin-top: 15px;

            margin-left: 10px;

        `;

        form.appendChild(viewHistoryBtn);

        viewHistoryBtn.addEventListener('click', function() {

            const isVisible = responseContainer.style.display === 'block';

            responseContainer.style.display = isVisible ? 'none' : 'block';

            if (!isVisible) {

                displayUserHistory(responseContainer);

            }

        });

    }

    // Make sure the form's submit event is only attached once

    if (!form.hasAttribute('data-submit-attached')) {

        // Handle form submission

        form.addEventListener('submit', function(e) {

            e.preventDefault();

            // Validate form before proceeding

            if (!validateForm(form)) {

                return;

            }

            // For other forms (registration, contact, etc.)

            const formData = {};

            const inputs = form.querySelectorAll('input, textarea');

            inputs.forEach(input => {

                const key = input.name || input.placeholder || input.type;

                formData[key] = input.value;

            });

            formData.timestamp = new Date().toLocaleString();

            // Special handling for registration

            if (storageKey === 'register\_responses') {

                const email = form.querySelector('input[type="email"]').value;

                // Check if user already exists

                const existingUsers = JSON.parse(localStorage.getItem(storageKey) || '[]');

                const userExists = existingUsers.some(user => {

                    const userEmail = user.email || user.Email;

                    return userEmail === email;

                });

                if (userExists) {

                    showNotification(form, 'This email is already registered. Please login instead.', 'error');

                    // Automatically open login modal after 2 seconds

                    setTimeout(() => {

                        const loginModal = document.querySelector('#login-modal');

                        if (loginModal) {

                            const currentModal = form.closest('.modal');

                            if (currentModal) {

                                currentModal.style.display = 'none';

                            }

                            loginModal.style.display = 'flex';

                        }

                    }, 2000);

                    return;

                }

                // Save new registration

                saveToLocalStorage(storageKey, formData);

                showNotification(form, 'Registration successful! Please login.', 'success');

                // Refresh page after 2 seconds

                setTimeout(() => {

                    window.location.reload();

                }, 2000);

            }

            // Special handling for login form

            else if (storageKey === 'login\_responses') {

                const email = form.querySelector('input[type="email"]').value;

                const password = form.querySelector('input[type="password"]').value;

                // Get registered users from storage

                const registeredUsers = JSON.parse(localStorage.getItem('register\_responses') || '[]');

                // First check if email exists at all

                const userEmailExists = registeredUsers.some(user => {

                    const userEmail = user.email || user.Email;

                    return userEmail === email;

                });

                // Then check if credentials match

                const userExists = registeredUsers.some(user => {

                    const userEmail = user.email || user.Email;

                    const userPassword = user.password || user.Password;

                    return userEmail === email && userPassword === password;

                });

                // Add login status to formData

                formData.status = userExists ? 'Success' : 'Failed';

                // Save the login attempt regardless of success/failure

                saveToLocalStorage(storageKey, formData);

                if (!userEmailExists) {

                    // User not registered at all

                    showNotification(form, 'User not registered. Please register first.', 'error');

                    // After 2 seconds, redirect to register page

                    setTimeout(() => {

                        window.location.href = 'register.html';

                    }, 2000);

                    return;

                } else if (!userExists) {

                    // User registered but wrong password

                    showNotification(form, 'Invalid password. Please try again.', 'error');

                    return;

                } else {

                    // Successful login

                    showNotification(form, 'Login successful! Redirecting to homepage...', 'success');

                    updateUserUI(email);

                    // Close modal if present

                    const modal = form.closest('.modal');

                    if (modal) {

                        modal.style.display = 'none';

                    }

                    // Redirect to index.html after 1.5 seconds

                    setTimeout(() => {

                        window.location.href = 'index.html';

                    }, 1500);

                }

            }

            // For contact form and any other regular form

            else {

                saveToLocalStorage(storageKey, formData);

                showNotification(form, 'Your message has been sent successfully!', 'success');

            }

            form.reset();

        });

        form.setAttribute('data-submit-attached', 'true');

    }

}

/\*\*

 \* Shows a temporary notification message after form submission

 \* @param {HTMLElement} form - The form element

 \* @param {string} message - Message to display

 \* @param {string} type - Type of notification ('success' or 'error')

 \*/

function showNotification(form, message, type = 'success') {

    const notification = document.createElement('div');

    notification.textContent = message;

    // Set color based on notification type

    const bgColor = type === 'success' ? '#27ae60' : '#e74c3c';

    notification.style.cssText = `

        background: ${bgColor};

        color: white;

        padding: 10px 15px;

        border-radius: 4px;

        margin-top: 10px;

        text-align: center;

    `;

    form.appendChild(notification);

    // Remove notification after 3 seconds

    setTimeout(() => {

        notification.style.opacity = '0';

        notification.style.transition = 'opacity 0.5s ease';

        setTimeout(() => {

            if (notification.parentNode === form) {

                form.removeChild(notification);

            }

        }, 500);

    }, 3000);

}

/\*\*

 \* Validates all required fields in a form

 \* @param {HTMLElement} form - Form to validate

 \* @returns {boolean} - Whether form is valid

 \*/

function validateForm(form) {

    const inputs = form.querySelectorAll('input, textarea');

    let isValid = true;

    inputs.forEach(input => {

        if (input.hasAttribute('required') && !input.value.trim()) {

            isValid = false;

            // Highlight invalid field

            input.style.borderColor = '#e74c3c';

            // Find or create validation message

            let validationMsg = input.nextElementSibling;

            if (!validationMsg || !validationMsg.classList.contains('validation-message')) {

                validationMsg = document.createElement('span');

                validationMsg.className = 'validation-message';

                input.parentNode.insertBefore(validationMsg, input.nextSibling);

            }

            validationMsg.textContent = 'This field is required';

            validationMsg.style.display = 'block';

        }

    });

    return isValid;

}

/\*\*

 \* Saves form data to localStorage

 \* @param {string} key - Storage key

 \* @param {object} data - Data to store

 \*/

function saveToLocalStorage(key, data) {

    // Get existing data or initialize empty array

    const existingData = JSON.parse(localStorage.getItem(key) || '[]');

    // Add new data

    existingData.push(data);

    // Save back to localStorage

    localStorage.setItem(key, JSON.stringify(existingData));

}

/\*\*

 \* Displays stored responses in a table

 \* @param {HTMLElement} container - Where to display the table

 \* @param {string} storageKey - Key to retrieve data from localStorage

 \*/

function displayStoredResponses(container, storageKey) {

    const responses = JSON.parse(localStorage.getItem(storageKey) || '[]');

    if (responses.length === 0) {

        container.innerHTML = '<p style="text-align: center; color: #666;">No responses submitted yet</p>';

        return;

    }

    // Get all unique fields across all responses (except timestamp)

    const allFields = new Set();

    responses.forEach(response => {

        Object.keys(response).forEach(key => {

            if (key !== 'timestamp') allFields.add(key);

        });

    });

    // Format field names for display (capitalize, remove underscores)

    const formatFieldName = (field) => {

        return field

            .split(/[\_-]/)

            .map(word => word.charAt(0).toUpperCase() + word.slice(1))

            .join(' ');

    };

    // Create table HTML

    let tableHTML = `

    <table style="width: 100%; border-collapse: collapse; font-size: 0.9rem;">

        <thead>

            <tr style="background: #2c3e50; color: white;">

                <th style="padding: 10px; text-align: left; border: 1px solid #ddd;">Timestamp</th>

                ${Array.from(allFields).map(field =>

                    `<th style="padding: 10px; text-align: left; border: 1px solid #ddd;">${formatFieldName(field)}</th>`

                ).join('')}

            </tr>

        </thead>

        <tbody>

    `;

    // Add rows for each response

    responses.forEach((response, index) => {

        const rowStyle = index % 2 === 0 ? 'background: #f9f9f9;' : '';

        tableHTML += `

        <tr style="${rowStyle}">

            <td style="padding: 10px; border: 1px solid #ddd;">${response.timestamp}</td>

            ${Array.from(allFields).map(field =>

                `<td style="padding: 10px; border: 1px solid #ddd;">${response[field] || '-'}</td>`

            ).join('')}

        </tr>

        `;

    });

    tableHTML += `

        </tbody>

    </table>

    `;

    // Add clear button

    tableHTML += `

    <div style="text-align: right; margin-top: 15px;">

        <button class="clear-data-btn" style="padding: 8px 12px; background: #e74c3c; color: white; border: none; border-radius: 4px; cursor: pointer;">

            Clear All Data

        </button>

    </div>

    `;

    // Update container and add event listener for clear button

    container.innerHTML = tableHTML;

    // Add event listener for clear button

    const clearBtn = container.querySelector('.clear-data-btn');

    clearBtn.addEventListener('click', () => {

        if (confirm('Are you sure you want to clear all stored responses?')) {

            localStorage.removeItem(storageKey);

            displayStoredResponses(container, storageKey);

        }

    });

}

/\*\*

 \* Shows a temporary notification message after form submission

 \* @param {HTMLElement} form - The form element

 \* @param {string} message - Message to display

 \* @param {string} type - Type of notification ('success' or 'error')

 \*/

function showNotification(form, message, type = 'success') {

    const notification = document.createElement('div');

    notification.textContent = message;

    // Set color based on notification type

    const bgColor = type === 'success' ? '#27ae60' : '#e74c3c';

    notification.style.cssText = `

        background: ${bgColor};

        color: white;

        padding: 10px 15px;

        border-radius: 4px;

        margin-top: 10px;

        text-align: center;

    `;

    form.appendChild(notification);

    // Remove notification after 3 seconds

    setTimeout(() => {

        notification.style.opacity = '0';

        notification.style.transition = 'opacity 0.5s ease';

        setTimeout(() => {

            if (notification.parentNode === form) {

                form.removeChild(notification);

            }

        }, 500);

    }, 3000);

}

// =========================================

// Event Handling Implementation

// Sets up various event listeners

// =========================================

const setupEventHandlers = () => {

    // Alert on logo click

    document.querySelector('.logo').addEventListener('click', () => {

        alert('Welcome to Natural Stone Hub!');

    });

    // Confirm before form submission

    document.querySelectorAll('form').forEach(form => {

        form.addEventListener('submit', (e) => {

            if (!confirm('Are you sure you want to submit this form?')) {

                e.preventDefault();

            }

        });

    });

    // Hover effect on product cards

    document.querySelectorAll('.product-card').forEach(card => {

        card.addEventListener('mouseenter', () => {

            card.style.transform = 'scale(1.05)';

            card.style.transition = 'transform 0.3s ease';

        });

        card.addEventListener('mouseleave', () => {

            card.style.transform = 'scale(1)';

        });

    });

};

// =========================================

// Initialize all features

// =========================================

document.addEventListener('DOMContentLoaded', () => {

    createDynamicMenu();

    setupFormValidation();

    setupLocalStorage();

    setupEventHandlers();

    addPasswordToggle();

    checkUserLogin();

});