// Wait for DOM content to be fully loaded

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

// Store theme preference in localStorage

const DARK\_MODE\_KEY = 'darkMode';

const THEME\_TRANSITION\_DURATION = 500; // ms

// Elements

const header = document.getElementById("header");

const themeToggle = document.getElementById("theme-toggle");

const projectFilters = document.querySelectorAll('.filter-btn');

const projectItems = document.querySelectorAll('.project-item');

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

// Initialize theme based on user preference

initializeTheme();

// Debounce function for performance optimization

function debounce(func, wait) {

let timeout;

return function() {

const context = this;

const args = arguments;

clearTimeout(timeout);

timeout = setTimeout(() => func.apply(context, args), wait);

};

}

// Sticky header shadow on scroll with debounce for performance

const debouncedHandleScroll = debounce(handleScroll, 10);

window.addEventListener("scroll", debouncedHandleScroll);

// Initial scroll check (in case page is loaded scrolled down)

setTimeout(handleScroll, 100);

// Dark mode toggle functionality with improved animation

if (themeToggle) {

themeToggle.addEventListener("click", toggleDarkMode);

// Add tooltip to theme toggle

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

tooltip.className = 'tooltip';

tooltip.textContent = 'Toggle Dark Mode';

themeToggle.appendChild(tooltip);

// Show tooltip on hover

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

tooltip.style.opacity = '1';

tooltip.style.transform = 'translateY(0)';

});

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

tooltip.style.opacity = '0';

tooltip.style.transform = 'translateY(10px)';

});

}

// Project filtering (on projects page)

if (projectFilters.length > 0 && projectItems.length > 0) {

setupProjectFilters();

}

// Contact form handling with improved validation

if (contactForm) {

contactForm.addEventListener('submit', handleContactForm);

setupFormValidation();

}

// Smooth scrolling for anchor links

setupSmoothScrolling();

// Add animation classes when elements come into view

setupScrollAnimations();

/\*\*

\* Initialize theme based on user preference or system preference

\* with improved transition between states

\*/

function initializeTheme() {

// Check if user has a saved preference

const savedTheme = localStorage.getItem(DARK\_MODE\_KEY);

// Prepare body for smooth transition

document.body.style.transition = 'background-color 0.3s ease, color 0.3s ease';

if (savedTheme === 'true') {

document.body.classList.add('dark');

if (themeToggle) {

themeToggle.textContent = '🌞';

themeToggle.setAttribute('aria-label', 'Switch to Light Mode');

}

} else if (savedTheme === 'false') {

document.body.classList.remove('dark');

if (themeToggle) {

themeToggle.textContent = '🌙';

themeToggle.setAttribute('aria-label', 'Switch to Dark Mode');

}

} else {

// If no saved preference, check system preference

if (window.matchMedia && window.matchMedia('(prefers-color-scheme: dark)').matches) {

document.body.classList.add('dark');

if (themeToggle) {

themeToggle.textContent = '🌞';

themeToggle.setAttribute('aria-label', 'Switch to Light Mode');

}

localStorage.setItem(DARK\_MODE\_KEY, 'true');

} else {

if (themeToggle) {

themeToggle.setAttribute('aria-label', 'Switch to Dark Mode');

}

}

}

// Listen for system theme changes

if (window.matchMedia) {

window.matchMedia('(prefers-color-scheme: dark)').addEventListener('change', e => {

if (localStorage.getItem(DARK\_MODE\_KEY) === null) {

// Only auto-switch if user hasn't set a preference

if (e.matches) {

document.body.classList.add('dark');

if (themeToggle) {

themeToggle.textContent = '🌞';

themeToggle.setAttribute('aria-label', 'Switch to Light Mode');

}

} else {

document.body.classList.remove('dark');

if (themeToggle) {

themeToggle.textContent = '🌙';

themeToggle.setAttribute('aria-label', 'Switch to Dark Mode');

}

}

}

});

}

}

/\*\*

\* Handle scroll events for sticky header with improved animation

\* and performance optimization

\*/

function handleScroll() {

if (!header) return;

// Add shadow and transform effect to header when scrolled

const scrolled = window.scrollY > 10;

if (scrolled && !header.classList.contains("scrolled")) {

header.classList.add("scrolled");

// Trigger a reflow to ensure smooth animation

void header.offsetWidth;

} else if (!scrolled && header.classList.contains("scrolled")) {

header.classList.remove("scrolled");

}

// Check for elements that should animate on scroll

const animatedElements = document.querySelectorAll('.animate-on-scroll:not(.animated)');

animatedElements.forEach(element => {

if (isElementInViewport(element)) {

element.classList.add('animated');

}

});

}

/\*\*

\* Toggle dark mode with enhanced animation and accessibility

\*/

function toggleDarkMode() {

// Add transition class to trigger smooth animation for all elements

document.documentElement.classList.add('theme-transition');

// Toggle dark mode class

document.body.classList.toggle('dark');

const isDarkMode = document.body.classList.contains('dark');

// Update button icon and aria-label for accessibility

themeToggle.textContent = isDarkMode ? '🌞' : '🌙';

themeToggle.setAttribute('aria-label', isDarkMode ? 'Switch to Light Mode' : 'Switch to Dark Mode');

// Save preference to localStorage

localStorage.setItem(DARK\_MODE\_KEY, isDarkMode.toString());

// Add animation to theme toggle

themeToggle.classList.add('theme-toggle-animation');

// Flash effect on body background

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

flashElement.className = 'theme-flash';

document.body.appendChild(flashElement);

// Clean up animations after transition completes

setTimeout(() => {

themeToggle.classList.remove('theme-toggle-animation');

document.documentElement.classList.remove('theme-transition');

if (flashElement.parentNode) {

flashElement.parentNode.removeChild(flashElement);

}

}, THEME\_TRANSITION\_DURATION);

}

/\*\*

\* Setup project filtering functionality

\*/

function setupProjectFilters() {

projectFilters.forEach(button => {

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

// Remove active class from all buttons

projectFilters.forEach(btn => btn.classList.remove('active'));

// Add active class to clicked button

button.classList.add('active');

// Get filter value

const filterValue = button.getAttribute('data-filter');

// Filter projects

projectItems.forEach(item => {

if (filterValue === 'all' || item.getAttribute('data-category') === filterValue) {

item.style.display = 'grid';

// Add animation

item.classList.add('fade-in');

setTimeout(() => {

item.classList.remove('fade-in');

}, 500);

} else {

item.style.display = 'none';

}

});

});

});

}

/\*\*

\* Setup form validation with real-time feedback

\*/

function setupFormValidation() {

const formInputs = contactForm.querySelectorAll('input, textarea');

formInputs.forEach(input => {

// Create feedback element

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

feedbackElement.className = 'form-feedback';

input.parentNode.appendChild(feedbackElement);

// Add event listeners for validation

input.addEventListener('blur', () => validateInput(input, feedbackElement));

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

// Clear error when user starts typing again

if (input.classList.contains('invalid')) {

input.classList.remove('invalid');

feedbackElement.textContent = '';

feedbackElement.classList.remove('error');

}

});

});

}

/\*\*

\* Validate form input and show feedback

\* @param {HTMLElement} input - Input element to validate

\* @param {HTMLElement} feedback - Element to show feedback in

\*/

function validateInput(input, feedback) {

const value = input.value.trim();

const name = input.name;

// Don't validate empty optional fields

if (!input.required && !value) {

feedback.textContent = '';

return true;

}

// Check for required fields

if (input.required && !value) {

input.classList.add('invalid');

feedback.textContent = 'This field is required';

feedback.classList.add('error');

return false;

}

// Email validation

if (name === 'email' && value) {

const emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

if (!emailPattern.test(value)) {

input.classList.add('invalid');

feedback.textContent = 'Please enter a valid email address';

feedback.classList.add('error');

return false;

}

}

// Clear feedback if valid

feedback.textContent = '';

return true;

}

/\*\*

\* Handle contact form submission with enhanced validation

\* @param {Event} e - Form submit event

\*/

function handleContactForm(e) {

e.preventDefault();

// Get form values

const name = document.getElementById('name').value.trim();

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

const subject = document.getElementById('subject').value.trim();

const message = document.getElementById('message').value.trim();

// Validate all fields

let isValid = true;

const formInputs = contactForm.querySelectorAll('input, textarea');

formInputs.forEach(input => {

const feedbackElement = input.parentNode.querySelector('.form-feedback');

if (!validateInput(input, feedbackElement)) {

isValid = false;

}

});

if (!isValid) {

// Focus the first invalid field

contactForm.querySelector('.invalid').focus();

return;

}

// Create and show success message

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

successMessage.className = 'form-success';

successMessage.innerHTML = `

<i class="fas fa-check-circle"></i>

<p>Thank you for your message, ${name}!</p>

<p>I'll get back to you soon.</p>

`;

// Replace form with success message

contactForm.style.opacity = '0';

setTimeout(() => {

const formContainer = contactForm.parentNode;

formContainer.innerHTML = '';

formContainer.appendChild(successMessage);

// Animate success message

setTimeout(() => {

successMessage.style.opacity = '1';

successMessage.style.transform = 'translateY(0)';

}, 50);

// Reset form behind the scenes for if we want to show it again

contactForm.reset();

}, 300);

}

/\*\*

\* Setup smooth scrolling for anchor links

\*/

function setupSmoothScrolling() {

const anchorLinks = document.querySelectorAll('a[href^="#"]');

anchorLinks.forEach(link => {

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

const targetId = link.getAttribute('href');

// Skip if it's just "#"

if (targetId === '#') return;

const targetElement = document.querySelector(targetId);

if (targetElement) {

e.preventDefault();

// Get header height for offset

const headerHeight = header ? header.offsetHeight : 0;

// Scroll to element with offset for header

window.scrollTo({

top: targetElement.offsetTop - headerHeight - 20,

behavior: 'smooth'

});

}

});

});

}

/\*\*

\* Setup animations for elements when they come into view

\*/

function setupScrollAnimations() {

// Add animate-on-scroll class to elements that should animate

const elementsToAnimate = [

...document.querySelectorAll('.project-card'),

...document.querySelectorAll('.skill-category'),

...document.querySelectorAll('.timeline-item'),

...document.querySelectorAll('.education-item')

];

elementsToAnimate.forEach(element => {

element.classList.add('animate-on-scroll');

});

// Initial check for elements in viewport

handleScroll();

}

/\*\*

\* Check if an element is in the viewport

\* @param {HTMLElement} element - The element to check

\* @returns {boolean} - Whether the element is in the viewport

\*/

function isElementInViewport(element) {

const rect = element.getBoundingClientRect();

return (

rect.top <= (window.innerHeight || document.documentElement.clientHeight) \* 0.9 &&

rect.bottom >= 0

);

}

});