Advanced Frontend

This workshop is designed to integrate all the concepts we've covered, HTML5 media, CSS layout, responsive design, and core JavaScript, into a single, high-quality project: **your own professional portfolio website.**

Workshop Objectives

Topic	Feature to Implement
HTML5	Create **semantic structure**, embed the <video> element for a background, and use other **HTML5 media** and form elements.
CSS3 Layout	Use **Flexbox** for the navigation bar and internal component alignment. Use **Grid** for the main section layouts (e.g., Skills or Projects).
Responsive Design	Implement **Media Queries** to ensure the entire portfolio looks professional on mobile, tablet, and desktop screens.
JavaScript Basics	Use JS to handle a menu toggle button (for mobile), and manipulate the **DOM** to dynamically display information and handle user **events**.

Step-by-Step Instructions

Step 1: Initial HTML Structure and Media Integration (HTML5)

1. **Project Setup:** Create a folder named my-portfolio with three files: index.html, style.css, and script.js.
2. **Semantic Layout:** In index.html, define the main sections using appropriate semantic tags:
    - <header>: For the navigation and main hero area.
    - <section id="about">
    - <section id="skills">
    - <section id="projects">
    - <section id="contact">
    - <footer>
3. **Navigation (<header>):** Create a basic nav bar (<nav>) with links to the main sections. Include a button or icon for the mobile menu toggle (initially hidden on desktop).
4. **HTML5 Media:**
    - **Video Background:** Embed a short, muted, looping video (e.g., a coding animation, space scene) as a background for the hero section using the <video> tag. Ensure it's set to autoplay, loop, and muted.
    - **Audio (Optional/Creative):** If you wish to demonstrate audio skills, you can embed a small, hidden <audio> element that plays a sound on a specific interaction (e.g., clicking a button or hovering over a project).

Code Snippet Example (index.html - Hero Section):

```
<header id="hero">
  <video autoplay loop muted id="video-background">
    <!-- Use a small, low-res video file for demonstration/performance -->
    <source src="background-loop.mp4" type="video/mp4">
    Your browser does not support the video tag.
  </video>
  <nav class="navbar">
    <span class="logo">My Portfolio</span>
    <ul class="nav-links">
      <li><a href="#about">About</a></li>
      <li><a href="#projects">Projects</a></li>
      <li><a href="#contact">Contact</a></li>
    </ul>
    <button class="menu-toggle" aria-expanded="false" aria-label="Toggle navigation"
id="menu-button">
      <!-- Hamburger icon can be a simple text or SVG -->
      ☰
```

```
      </button>
    </nav>
    <div class="hero-content">
      <h1>[Your Name]</h1>
      <p>Frontend Developer & UI/UX Enthusiast</p>
    </div>
</header>
```

Bonus CSS Tip for Video Background: To ensure the video acts as a true background behind the content, you will need to use CSS positioning. Target #video-background and apply position: absolute;, top: 0;, left: 0;, width: 100%;, height: 100%;, object-fit: cover;, and z-index: -1; to place it correctly.

Step 2: Styling and Layout (CSS3: Flexbox & Grid)

1. **Link CSS:** Link your style.css file in the <head> of your HTML.
2. **Basic Styling:** Set a standard font (e.g., font-family: sans-serif;), reset margins/paddings (* { margin: 0; padding: 0; box-sizing: border-box; }), and define a clear color palette (background, text, accent).
3. **Flexbox for Navigation:**
   - Use **Flexbox** on the navigation bar (<nav>) to align items horizontally, distribute space, and center the logo/name.
   - *Property Focus:* display: flex;, justify-content: space-between;, align-items: center;.
4. **Grid for Main Content:**
   - Use **CSS Grid** for the layout of the **Projects** section. Each project should be a card. Define a grid container (display: grid;) and use grid-template-columns: repeat(auto-fit, minmax(300px, 1fr)); to create a responsive, flexible layout for project cards.
   - *Property Focus:* display: grid;, grid-template-columns, gap.
5. **Component Alignment:** Use Flexbox to center content vertically within cards or align items within your "About" or "Skills" sections.

Recommendation: Use Flexbox specifically for the **Skills** section (e.g., displaying skill logos/icons in a single, well-aligned row that wraps on smaller screens) to clearly differentiate its use from Grid in the **Projects** section.

Code Snippet Example (style.css - Flexbox and Grid):

```
/* Flexbox for Navigation (Mobile first default: Column/Hidden) */
.navbar {
    display: flex;
    flex-direction: column; /* Stacks items vertically on small screens */
    align-items: center;
    padding: 1rem;
    position: relative;
    z-index: 10;
}

.nav-links {
    list-style: none;
    display: none; /* Hide links by default on mobile */
    flex-direction: column;
    width: 100%;
    text-align: center;
}

.nav-links.open {
```

```
    display: flex; /* Show links when '.open' class is added by JS */
}

/* CSS Grid for Projects Section */
#projects-grid {
    display: grid;
    /* This creates columns that are at least 300px wide, and as many as can fit */
    grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
    gap: 2rem;
    padding: 2rem;
}

.project-card {
    background: #2c2c2c;
    border-radius: 8px;
    padding: 1.5rem;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
}
```

Step 3: Responsive Design (CSS3 Media Queries)

1. **Mobile First:** Start by styling the layout for small screens (e.g., 320px to 600px).
2. **Tablet Breakpoint:** Add your first media query (e.g., @media screen and (min-width: 768px)) to adjust the layout for tablets:
    - Increase font sizes, adjust padding, and change the grid layout if necessary (e.g., shift from 1-column to 2-column project view).
3. **Desktop Breakpoint:** Add a second media query (e.g., @media screen and (min-width: 1024px)) for desktop views:
    - Ensure the navigation bar is fully visible (hide the mobile menu button and show all links).
    - Use wider margins and potentially a more complex grid or flex layout for the main sections.

Code Snippet Example (style.css - Media Queries):

```
/* Desktop / Tablet Styles (Overrides Mobile Styles) */
@media screen and (min-width: 768px) {
  /* Flexbox for Navigation (Horizontal on larger screens) */
  .navbar {
    flex-direction: row; /* Horizontal layout */
    justify-content: space-between;
  }

  .nav-links {
    display: flex; /* Always show links */
    flex-direction: row;
    width: auto;
  }

  .menu-toggle {
    display: none; /* Hide the mobile menu button */
  }

  /* Adjust padding for main content */
  section {
    padding: 4rem 8%;
  }
}
```

Step 4: Interactivity and DOM Manipulation (JavaScript)

1. **Mobile Menu Toggle:**
   - Add a click event listener to the mobile menu button you created in Step 1.
   - When clicked, use JavaScript to:
     - Change a class on the navigation list (e.g., add .open) to make it visible (**DOM manipulation**).
     - Update the icon/text of the button (e.g., change from a hamburger to an 'X').
2. **Scroll Indicator (Optional but Recommended):**
   - Listen for the scroll event on the window object.
   - Calculate the user's scroll position and total page height.
   - Dynamically change the width of a fixed element (a progress bar) at the top of the page to show scroll progress. This requires reading and updating element styles (**DOM manipulation**).
3. **Form Validation/Handling:**
   - For the contact form in the <section id="contact">, add a submit event listener.
   - Prevent the default form submission (event.preventDefault()).
   - Check if the required fields are filled out. If so, display a custom "Thank You" message by manipulating a in the DOM. If not, display an error message next to the missing field.

Code Snippet Example (script.js - Menu Toggle):

```
const menuButton = document.getElementById('menu-button');
const navLinks = document.querySelector('.nav-links');

function toggleMenu() {
    // 1. Toggle the CSS class (controls visibility via CSS)
    navLinks.classList.toggle('open');

    // 2. Update the button text/icon for accessibility
    const isExpanded = navLinks.classList.contains('open');
    menuButton.setAttribute('aria-expanded', isExpanded);
    menuButton.innerHTML = isExpanded ? '✕' : '☰'; // X vs Hamburger
}

// Add the event handler
menuButton.addEventListener('click', toggleMenu);

// OPTIONAL: Close the menu when a link inside is clicked (for mobile UX)
navLinks.querySelectorAll('a').forEach(link => {
    link.addEventListener('click', () => {
        if (navLinks.classList.contains('open')) {
            toggleMenu(); // Closes the menu
```

```
    }
  });
});
```

Code Snippet Example (script.js - Form Submission Handling):

```
const contactForm = document.getElementById('contact-form-id'); // Assuming the form has an ID
const messageDiv = document.getElementById('form-message'); // Assuming a div for feedback

if (contactForm && messageDiv) {
  contactForm.addEventListener('submit', function(event) {
    // Stop the browser from submitting the form and refreshing the page
    event.preventDefault();

    const nameInput = document.getElementById('name').value;
    const emailInput = document.getElementById('email').value;

    if (nameInput === '' || emailInput === '') {
      messageDiv.textContent = 'Please fill out all required fields.';
      messageDiv.style.color = 'red';
    } else {
      // Successful mock submission
      messageDiv.textContent = 'Thank you for your message! I will be in touch shortly.';
      messageDiv.style.color = 'green';
      contactForm.reset(); // Clear the form fields
    }
  });
}
```

Tips & Help

Area	Tip	JavaScript Help
HTML	Use **ARIA attributes** (like aria-label and aria-expanded) on your interactive elements for accessibility.	The essential tool is document.querySelector() or document.getElementById() to grab elements.
CSS Layout	**Flexbox** is best for **1D layout** (rows OR columns, like a navbar). **Grid** is best for **2D layout** (rows AND columns, like a photo gallery/projects section).	Use element.classList.add() and element.classList.remove() to toggle visibility, which is cleaner than changing styles directly.
Responsiveness	Use **em** or **rem** for font sizes and **%** or **vw** for widths. Avoid fixed pixel values for sizes to keep everything fluid.	Use window.addEventListener('resize', ...) to handle changes to the viewport and readjust responsive CSS that might rely on JavaScript.
Debugging	Always use the browser's **Developer Tools** (F12 or right-click > Inspect). The **Console** is essential for debugging JS errors, and the **Elements** tab is critical for debugging CSS Flex/Grid issues.	Look up event.currentTarget vs event.target when dealing with deeply nested event listeners to ensure you're interacting with the element you intended.