The Ultimate Companion Guide

100 PROJECTS IN 100 DAYS

HTML, CSS & JAVASCRIPT

FREE PREVIEW (DAYS 1-10)

2,000+ GitHub Stars

Overview

Welcome to the **Companion Guide for 100 Mini Web Projects** using **HTML, CSS, and JavaScript**. Whether you're just starting out or sharpening advanced techniques, this guide delivers daily, bite-sized tasks that challenge you to customize and extend each project.

Perfect for anyone on the **100 Days of Code** journey, you'll find hands-on exercises designed to deepen your understanding and level up your skills through deliberate practice. Each day's entry comes with direct links to MDN Web Docs for on-demand reference, so you can dive deeper exactly where you need it.

What's Inside: Free Preview

- **Day 0: Before Starting Up**
  Get your environment set up and explore the boilerplate's HTML, CSS, and JavaScript structure.

- **Days 1-10: Challenges & Enhancements**
  Each day includes:
  - 🧠 **Challenges**: Modify core markup, styles, or scripts to deepen your understanding of that day's project.
  - 🧭 **Enhancements**: Advanced feature ideas, design refinements, and real-world polish suggestions.
  - **MDN Web Docs Links**: Direct pointers to documentation for every tweak, so you can dive deeper on demand.
  - **Solutions**: Available in the challenge-solutions branch of the repository.

This guide isn't just about tweaking projects. It's about building momentum, mastering fundamentals, and pushing yourself to explore, break, and rebuild.

Contents

Day 0 - Before Starting Up

Before diving into your 100-day coding journey, it's essential to lay a solid foundation. Day 0 is your prep day—use it to set up your development environment, understand the project structure, and get comfortable with the basic boilerplate.

✅ *Setup*

Follow the YouTube video or the detailed setup instructions in the GitHub repository.

You'll learn how to:

- Fork and clone the repo.
- Open it in VS Code (or your preferred editor).
- Install and use the Live Server extension to preview your work instantly.

🧠 *Challenges*

If you're new to web development, even simple setup steps can feel like blockers—and that's completely normal.

Here's what we recommend:

- Search online actively—errors are learning opportunities, not dead-ends.
- Look for video tutorials on Git, VS Code, Live Server, and basic HTML/CSS/JS setup. Sometimes a quick visual explanation unlocks what a long article can't.
- Take your time and document everything—write down each issue you hit, how you solved it, and what resources helped.

🧱 *Boilerplate Explained*

The boilerplate provides a minimal yet structured starting point for every project. It saves you time by handling the repetitive setup, letting you focus on building.

🔤 *HTML – index.html*

- `<!DOCTYPE html>` — Ensures standards-compliant rendering.
- `<meta charset="UTF-8" />` — Supports all characters (essential for modern web apps).
- `<meta name="viewport" ...>` — Makes the layout responsive by default.
- External links for CSS (`style.css`) and JS (`script.js`) are already wired up.
- Font Awesome (commented out) is included for easy enabling if you need icons.

🎨 *CSS – style.css*

- Imports **Roboto**, a clean and versatile font.
- Applies `box-sizing: border-box` universally—simplifies layout math.
- Centered flexbox layout (vertical + horizontal).
- 100vh layout with hidden overflow—great for single-view demos.
- No default margin clutter.

📜 *JS – script.js*

Empty to start—this is where your interactivity begins. Add event listeners, DOM manipulations, animations, etc.

Key Concepts

Concept	Description
Boilerplate Architecture	How to structure HTML/CSS/JS files in a minimal, reusable project layout.
Viewport & Responsive Meta Tags	Ensures your project scales correctly on mobile and desktop.
Box-Sizing	A critical CSS default for predictable element sizing.
Flexbox Centering	A clean default layout using `display: flex`, great for most UI components.
Google Fonts	Using hosted fonts via `@import`—stylish with no setup.
External Resources Linking	CSS, JavaScript, and fonts wired into the base project without redundancy.
Live Server Preview	Real-time code changes with no manual reloading—boosts iteration speed.
Semantic HTML Structure	Starts with a minimal, semantically valid HTML5 template.

🧭 *Enhancements*

- Replace Roboto with a font you like.
- Replace `<h1>My Project</h1>` with a styled `<div>` for quick visual testing.
- Add a favicon.

Since it's just Day 0, there are no hints this time—but don't worry, there's a step-by-step video to guide you!

See the solution for Day 0.

Day 1 - Expanding Cards

Welcome to your first project! You'll create a row of "Expanding Cards"—images that smoothly expand when clicked to reveal more detail, while the others shrink. This project introduces you to basic HTML for structure, CSS for styling (including rounded corners and background images), and a small amount of JavaScript to make the cards interactive. It's a fun way to see how simple code can create engaging effects.

🧠 *Challenges*

- **Change the Card Titles**: Each card currently displays "Explore the world". Open your HTML file, find these titles inside `<h3>` tags, and change them to the names of different cities or countries, such as "Canada", "Argentina", "Paris", "Tokyo", or "Brazil".
  Hint: Look for the `<h3>` elements in each card. The `<h3>` tag is used for section headings—changing its text changes the card's title.
  Learn more: HTML Heading Elements (`<h1>`–`<h6>`)
- **Adjust Card Corner Style**: The cards have rounded corners because of the `border-radius: 50px;` property in the CSS. Find the `.panel` rule in `style.css` and change `50px` to `10px` for slightly rounded corners, or `0px` for sharp corners.
  Hint: `border-radius` controls how rounded the corners of the cards appear. Try different values to see how the shape changes.
  Learn more: The `border-radius` property

🧭 *Enhancements*

- **Add a New Panel**: Add a sixth panel by copying an existing `<div class="panel">...</div>` block in your HTML. Paste it as a new sibling (another panel at the same level), and update its `background-image` URL and `<h3>` title to something new.
  Hint: In HTML, siblings share the same parent element—for example, all `<div class="panel">` blocks inside `.container` are siblings. To add a new panel, copy and paste a `<div class="panel">...</div>` block right after the others. You can find free images on Unsplash. For example, try this portrait of a smiling man from Vietnam: `https://images.unsplash.com/photo-1746105625407-5d49d69a2a47?auto=format&fit=crop&w=1350&q=80`.
  Learn more: Unsplash, The `<div>` element, The `background-image` property
- **Change Active Panel Text Color**: When a panel is active, its title appears. In `style.css`, find the `.panel.active h3` rule and add a `color` property to change the text color, such as `color: silver;`.
  Hint: The `.active` class is added to the panel you click, making it expand and show its title. Changing the `color` property in `.panel.active h3` will update the title's text color when the panel is active.
  Learn more: CSS named colors, The `class` global attribute
- **Modify Transition Speed**: The panels expand and shrink with a transition. In the `.panel` CSS rule, you'll see `transition: flex 0.7s ease-in;`. Try changing `0.7s` (0.7 seconds) to `2s` for a slower transition, or `0.2s` for a faster one. Also update the `-webkit-transition: all 700ms ease-in;` property to match (e.g., `200ms` or `2000ms`).
  Hint: Both `transition` and `-webkit-transition` control how quickly the panels animate. Make sure to update both so the animation speed is consistent across browsers. Experiment with different durations to see how the animation feels.
  Learn more: The `transition` property

See the solution for Day 1.

Day 2 - Progress Steps

In this project, you'll build a "Progress Steps" indicator, similar to those found in online forms or tutorials. This component visually displays a series of numbered steps, highlights the current step, and lets users move forward or backward using "Next" and "Prev" buttons. You'll practice using HTML for layout, CSS for styling elements (like circles and connecting lines) and their states (active or inactive), and JavaScript to control which step is active and update the display accordingly.

🧠 *Challenges*

- **Add a Fifth Step**: Right now, there are 4 steps. Update the HTML to include a 5th step by adding another `<div class="step">5</div>`. Make sure your progress indicator works correctly with the new step.
  Hint: After adding the new step, test your project to confirm that the progress bar and navigation buttons still work as expected.
  Learn more: The `<div>` element, The `class` global attribute
- **Change the Active Step Color**: The active step and progress bar are currently blue. In `style.css`, find the CSS variable `--line-border-fill` (set to `#3498db`). Change this color to something else, such as green (`#2ecc71`) or orange (`#e67e22`), and observe how it affects the active steps and progress bar.
  Hint: CSS variables (custom properties) are defined in the `:root` selector. You can use any valid hex color code. See the references for more on hex colors and CSS variables.
  Learn more: Hexadecimal color values, CSS custom properties (variables)

🧭 *Enhancements*

- **Customize Button Text**: Change the text on the navigation buttons from "Prev" and "Next" to something like "Back" and "Continue".
  Hint: Look for the `<button>` elements in your HTML file and update their text content.
  Learn more: The `<button>` element
- **Style Inactive Steps**: Inactive steps currently have a grey border and number. Try changing their background color or text color. Look for the `.step` CSS rule (not `.step.active`).
  Hint: You can adjust the `background-color` or `color` properties in the `.step` CSS rule to customize how inactive steps look.
  Learn more: The `background-color` property, The `color` property
- **Start at Step 2**: Modify the JavaScript so that the progress bar starts at step 2 instead of step 1.
  Hint: Change the initial value of the `currentActive` variable in `script.js` to `2`, and call `update()` once at the start to reflect this change.
  Learn more: Declaring variables in JavaScript, Calling functions in JavaScript

See the solution for Day 2.

Day 3 - Rotating Navigation

This project features an eye-catching navigation menu that rotates the main page content to reveal navigation links. When you click the menu icon, the page rotates away, showing the navigation menu. Clicking the close icon rotates the page back. You'll use HTML to structure the page and menu, CSS for styling and rotation animations (using `transform` and `transition`), and JavaScript to toggle the menu's visibility by adding or removing a class.

🧠 *Challenges*

- **Change Menu Icons**: The open and close buttons use Font Awesome icons (`fa-bars` and `fa-times`). Try replacing these with different icons, such as `fa-plus` for open and `fa-minus` for close. Update the `<i>` tags in the HTML to use the new icon classes.
  Hint: Browse the Font Awesome icon gallery to find other icon class names. The Font Awesome library is loaded in the `<head>` of `index.html`, which allows the icons to display. The `<i>` tag is commonly used for icons.
  Learn more: Font Awesome, The `<i>` element
- **Change the Rotation Angle**: When the navigation is shown, the page rotates by -20 degrees. In `style.css`, find the `.container.show-nav` rule and change `transform: rotate(-20deg);` to a different value, such as `-15deg` or `-30deg`. Observe how the rotation changes the appearance.
  Hint: Try different angles to see how the menu looks. Negative values rotate the page counter-clockwise (to the right), revealing the menu. Positive values would rotate it clockwise (to the left), hiding the menu. Make sure the navigation links remain visible after rotation.
  Learn more: The `rotate()` function

🧭 *Enhancements*

- **Add a Portfolio Link**: Currently, the navigation menu includes "Home", "About", and "Contact". Add a new link, such as "Portfolio", by inserting a new `<li>` item with an icon in the `<nav><ul>` section of your HTML.
  Hint: To style the new link like the others, add a CSS rule using `nav ul li + li + li + li` to target the fourth item. Since each `+ li` moves to the next `<li>`, chaining them selects the fourth one. Increase the `margin-left` and adjust `transform: translateX()` by -50% more than the previous item (for example, `-250%`).
  Learn more: The `<li>` element, The adjacent sibling combinator (+)
- **Highlight Navigation Icons**: The navigation menu icons currently use the default color. Give them a distinct color to make them stand out.
  Hint: In `style.css`, add a `color` property to the `nav ul li i` rule. For example, try `color: #ff7979;` or experiment with other colors.
  Learn more: The `color` property
- **Adjust Menu Item Transition Speed**: Menu items slide in with a delay. Try making them appear faster or slower by changing the transition speed.
  Hint: In `style.css`, look for the `transition` property in the `nav ul li` rule. Change the duration value (for example, from `0.4s` to `0.2s` or `0.6s`) to control how quickly the menu items slide in.
  Learn more: The `transition` property

See the solution for Day 3.

Day 4 - Hidden Search Widget

You'll build a "Hidden Search Widget" that starts as a simple search icon. When you click the icon, it smoothly expands into a search input field. Clicking the icon again (or a close button) will hide the input. This project helps you practice using HTML for structure, CSS for styling and transitions (like `transition` on `width`), and JavaScript to toggle an "active" class that triggers the animation.

🧠 *Challenges*

- **Change the Search Icon**: The button uses a Font Awesome search icon (`fas fa-search`). Change this to a different icon, such as a magnifying glass with a plus (`fas fa-search-plus`). Edit the `<i>` tag inside the `<button>` in your HTML.
  Hint: You can browse more Font Awesome icons on their website. Look for an icon you like and update the class name in your HTML.
  Learn more: Font Awesome, The `class` global attribute
- **Modify Expansion Width**: When the search input expands, it currently grows to `width: 200px;` . In `style.css` , find the `.search.active .input` rule and change this `width` to a new value, such as `300px` to make it wider or `150px` to make it narrower.
  Hint: If you change the input width, also update `transform: translateX(198px);` on `.search.active .btn` so the button stays aligned. The `translateX` value should be about the new width minus 2px.
  Learn more: The `width` property, The `transform` property

🧭 *Enhancements*

- **Change Placeholder Text**: The input field's placeholder text is currently "Search...". Change the `placeholder` attribute in the HTML `input` tag to something like "Type and hit enter...".
  Hint: The `placeholder` attribute controls the text shown when the input is empty.
  Learn more: The `placeholder` attribute
- **Change Background Color on Focus**: Make the input field change its background color when focused. Add a CSS rule for `.search .input:focus` and set a `background-color`.
  Hint: For example, you could use `background-color: #f0f0f0;`. Click or type in the search field to see the effect.
  Learn more: The `:focus` pseudo-class, The `background-color` property
- **Animate Button Icon Change**: Make the search icon switch to a close icon (×) when the search is active, and back to the search icon when inactive. Update the JavaScript to toggle between the Font Awesome classes `fa-search-plus` and `fa-times`.
  Hint: In `script.js`:
  (1) Get the icon element: `const icon = document.querySelector(".fas");`
  (2) In the click event listener:

```
icon.classList.toggle("fa-search-plus",
!search.classList.contains("active"));
icon.classList.toggle("fa-times",
search.classList.contains("active"));
```

  The `!` inverts the condition so that `fa-search-plus` is only added when `.active` is absent. The second parameter in `toggle()` controls whether the class is added (`true`) or removed (`false`).
  Learn more: The `toggle()` method

See the solution for Day 4.

Day 5 - Blurry Loading

You'll create a visually engaging "Blurry Loading" effect. In this project, a background image starts out very blurry and gradually becomes sharp as a percentage counter updates from 0% to 100%. This effect is often used to make loading screens feel more dynamic. You'll use HTML for the structure, CSS for the background image and blur effect, and JavaScript with `setInterval` to animate the loading process.

🧠 *Challenges*

- **Adjust the Loading Speed**: By default, the loading animation takes about 3 seconds (100 increments at 30 milliseconds each). In `script.js`, find the line `let int = setInterval(blurring, 30);`. Change `30` to `10` to make the loading much faster, or to `100` to slow it down.
  Hint: `setInterval` *repeatedly calls a function at the interval you specify (in milliseconds). Lower numbers make the animation faster, higher numbers make it slower.*
  Learn more: The `setInterval()` *method*
- **Change the Initial Blur Amount**: The background image starts with a blur of 30px, controlled by the `scale` function in `script.js`. Find the line `bg.style.filter = \`blur(${scale(load, 0, 100, 30, 0)}px)\`;`. Change the `30` (the starting blur value) to a different number, like `50` for more blur or `10` for less.
  Hint: The `scale` *function maps the loading percentage to a blur value. Changing the starting value changes how blurry the image is at the beginning.*
  Learn more: The `blur()` *function*

🧭 *Enhancements*

- **Use a Different Background Image**: Personalize your project by changing the background image. In `style.css`, update the URL in the `.bg` class to a new image. You can find free images on Unsplash.
  Hint: Look for the `background` property in the `.bg` CSS rule. Replace the existing URL with one you like.
  Learn more: Unsplash, The `background-image` property
- **Change the Loading Text Color**: Customize the color of the loading percentage text. In `style.css`, find the `.loading-text` rule and change the `color` property to any color you prefer.
  Hint: Try using a named color (like `gray`), or a hex code (like `#403d47`).
  Learn more: The `color` property
- **Control When the Loading Text Fades Out**: The loading text fades out as the percentage approaches 100%. Adjust the code to make it fade out sooner (for example, by 50%) or later (like at 90%).
  Hint: The `scale` function maps the loading percentage to opacity. Adjusting the input range changes how quickly the text becomes transparent. In `script.js`, find the line `loadText.style.opacity = scale(load, 0, 100, 1, 0);`. To fade out faster, change the input range to `scale(load, 0, 50, 1, 0)`. To fade out more slowly, try `scale(load, 0, 90, 1, 0)`. Make sure the opacity does not go below 0.
  Learn more: The `opacity` property

See the solution for Day 5.

Day 6 - Scroll Animation

This project introduces "Scroll Animation," where content boxes smoothly animate into view as you scroll down the page. Each box slides in from the side or bottom, creating a dynamic effect. You'll use HTML to structure the content, CSS for styling and animation (using `transform` and `transition`), and JavaScript to detect scroll position and add a `show` class when boxes enter the viewport.

🧠 *Challenges*

- **Make All Boxes Slide Up from the Bottom**: Right now, even-numbered boxes slide in from the left and odd-numbered ones from the right. Change the CSS so that every box slides in from the bottom instead. Update the `.box` rule to use `transform: translateY(100%);` and remove the `.box:nth-of-type(even)` rule. The `.box.show` rule should use `transform: translateY(0);`.
  Hint: `translateY` *moves elements vertically, while* `translateX` *moves them horizontally. Removing* `.box:nth-of-type(even)` *ensures all boxes use the same animation direction.*
  Learn more: The `transform` *property*
- **Change When Boxes Appear on Scroll**: Boxes currently appear when their top edge is within the bottom 80% of the window. In `script.js`, this is set by `const triggerBottom = (window.innerHeight / 5) * 4;`. Try changing this calculation to make boxes appear sooner (for example, use `(window.innerHeight / 5) * 3;` for 60%). Observe how this affects when boxes animate in.
  Hint: `window.innerHeight` *gives you the height of the visible part of the browser window.* `Element.getBoundingClientRect()` *tells you where an element is on the screen. Adjust* `triggerBottom` *to control how far down the page a box must be before it appears. Try different values to see the effect.*
  Learn more: The `innerHeight` *property, The* `getBoundingClientRect()` *method*

🧭 *Enhancements*

- **Give Each Box Unique Content**: All boxes currently display the word "Content." Change the `<h2>Content</h2>` inside each `<div class="box">` in your HTML so that each box has its own unique heading.
  Hint: Try using creative headings like "Our Services", "About Us", "Portfolio", "Contact Info", "Testimonials", "Team Members", "Latest News", or "Featured Projects".
  Learn more: HTML Heading Elements (`<h1>`–`<h6>`)
- **Change Box Background Color When Shown**: Make the background color of a box change when it receives the `.show` class.
  Hint: Add a new CSS rule to `.box.show` such as `background-color: #34113f;` to see the color change when the box appears.
  Learn more: The `background-color` property
- **Add a Fade-In Effect to Boxes**: Make the boxes fade in as well as slide in. Set `opacity: 0;` on `.box`, and `opacity: 1;` on `.box.show`. Update the `transition` property to include opacity: `transition: transform 0.4s ease, opacity 0.4s ease;`.
  Hint: `opacity: 0;` makes an element fully transparent, while `opacity: 1;` makes it fully visible. Adding opacity to the transition will create a smooth fade-in effect along with the slide.
  Learn more: The `opacity` property, The `transition` property

See the solution for Day 6.

Day 7 - Split Landing Page

You'll create a "Split Landing Page", where the screen is divided into two interactive sections—each representing a different choice (for example, "PlayStation" and "Xbox"). When you hover over one side, it smoothly expands to take up more space, while the other side shrinks. You'll use HTML to structure the two sections, CSS for layout, background images, hover effects (using CSS variables for widths and colors), and transitions. JavaScript is used to add or remove classes on hover, triggering the CSS transitions.

🧠 *Challenges*

- **Change Hover Widths**: Adjust the widths of the two sides when hovering. The active side uses the CSS variable `--hover-width` (default `75%`), and the other uses `--minimize-width` (default `25%`). These are set in the `:root` selector of your `style.css` file. Try changing them to `60%` and `40%` to see how the layout responds.
  Hint: The values for `--hover-width` and `--minimize-width` should add up to `100%` for a seamless full-screen effect. If they don't, the layout may leave gaps.
  Learn more: CSS custom properties (variables)
- **Adjust Transition Speed**: Change how quickly the sides expand and shrink by editing the CSS variable `--transition-speed` in the `:root` selector of `style.css`. For example, set it to `0.5s` for a faster animation or `2s` for a slower one.
  Hint: The `transition` property controls how quickly changes happen in CSS. Changing `--transition-speed` updates the duration of the width and background transitions.
  Learn more: The `transition` property

🧭 *Enhancements*

- **Update Section Titles**: Change the titles in each section. In your HTML file, find the `<h1>` tags inside each `<section class="split ...">` and update their text to something new, like "Mountain View" and "Ocean Breeze".
  Hint: The `<section>` element groups related content on a page, like each side of your split layout. The `<h1>` element is used for the main heading of each section. Changing its text updates what users see on the page.
  Learn more: The `<section>` element, HTML Heading Elements (`<h1>`–`<h6>`)
- **Use New Background Images**: Personalize your landing page by changing the background images. In `style.css`, update the `background: url(...)` properties for `.split.left` and `.split.right`. You can find free images on Unsplash or Pexels.
  Hint: The `background-image` property sets the image for each section. Choose images that fit your new titles or theme.
  Learn more: Unsplash, Pexels, The `background-image` property
- **Customize Button Text and Style**: Change the button text from "Buy Now" to something that matches your new theme. You can also update the button's border color in the `.btn` class (for example, change `border: #fff solid 0.2rem;`). For more customization, adjust the hover background colors using the CSS variables `--left-btn-hover-color` and `--right-btn-hover-color` in `style.css`. Try changing the overlay colors `--left-bg-color` and `--right-bg-color` to match your images.
  Hint: CSS variables like `--left-btn-hover-color` use `rgba()` values, where the last number controls transparency (from `0` for fully transparent to `1` for fully opaque). Use online tools like RGBA and Hex Color Converter to find the right color.
  Learn more: RGBA and Hex Color Converter, The `rgba()` function, The `border-color` property, The `background-color` property

See the solution for Day 7.

Day 8 - Form Wave Animation

This project creates a "Form Wave Animation" for input field labels. When you click into an input field, each letter of its label animates upward in a wave-like motion. You'll use HTML to build the form, CSS for styling and animating the labels (using `transform: translateY()` on each `<span>`), and JavaScript to wrap every letter of the label in a `<span>` with a unique transition delay. This effect helps you practice working with forms, CSS transitions, and basic DOM manipulation.

🧠 *Challenges*

- **Change the Main Title**: The form's main heading is "Please Login". Update the text inside the `<h1>` tag in your HTML file to something else.
  Hint: Look for the `<h1>` element near the top of your HTML. You can change its text to any greeting or message you like.
  Learn more: HTML Heading Elements (`<h1>`–`<h6>`)
- **Adjust Wave Animation Speed**: The speed of the wave effect is set by the `transition-delay` on each letter's `<span>` (`${idx * 50}ms`). In `script.js`, change the `50` in `${idx * 50}ms` to a smaller number (like `25`) for a faster wave, or a larger number (like `100`) for a slower wave.
  Hint: The value is in milliseconds. Try different numbers to see how the animation changes. A `<span>` is an inline container for text or other inline elements. Here, each letter's `<span>` gets a different delay, creating the wave effect.
  Learn more: The `transition-delay` property, The `<span>` element

🧭 *Enhancements*

- **Add a New Input Field**: Add a third input field to the form, such as "Username".
  Hint: Copy one of the existing `<div class="form-control">...</div>` blocks in your HTML, change the label text to "Username", and make sure the JavaScript still applies the wave effect to the new label.
  Learn more: The `<form>` element
- **Change Label Color on Focus**: When an input is focused, the label text turns "lightblue". Change this color to something else.
  Hint: In `style.css`, find the `.form-control input:focus + label span, .form-control input:valid + label span` rule. Change `color: lightblue;` to your preferred color. The `+` selector means the label is immediately after the input.
  Learn more: The `color` property, The `:focus` pseudo-class, The adjacent sibling combinator (+)
- **Modify Button Appearance**: Change the "Login" button's background color or text. You can edit the `background: lightblue;` in the `.btn` CSS rule or change the text "Login" in the `<button>` tag in HTML.
  Hint: For a consistent look, you may also want to update other uses of `lightblue` in your CSS, such as the link color in `.container a` and the border color in `.form-control input:focus, .form-control input:valid`. The `<a href="#">` element creates a clickable link; `href="#"` is a placeholder.
  Learn more: The `background-color` property, The `<button>` element, The `<a>` element

See the solution for Day 8.

Day 9 - Sound Board

You'll build a simple "Sound Board" that lets you play fun sound effects with a click. You'll use HTML to add `<audio>` elements for each sound, style the buttons with CSS, and use JavaScript to generate a button for each sound. When you click a button, its sound plays—if another sound is already playing, it stops first. This project helps you practice working with the DOM, events, and basic audio controls.

🧠 *Challenges*

- **Change Button Text**: By default, each button displays the sound file name (like "applause" or "boo"). In `script.js`, find where `btn.innerText = sound;` is set. Try changing the text to uppercase with `btn.innerText = sound.toUpperCase();` or add a label, such as `btn.innerText = "Play " + sound;`.
  Hint: Try using `btn.innerText = "Play " + sound;` to make the button text more descriptive. You can also use `toUpperCase()` to make the text stand out.
  Learn more: The `innerText` property, The `toUpperCase()` method
- **Adjust Button Styling**: The buttons use `background-color: rebeccapurple;` in the `.btn` CSS rule. In `style.css`, try changing this to another color you like. You can also experiment with the `color` property (for the text) and adjust `border-radius` to change the button's shape.
  Hint: Try different values for `background-color`, `color`, and `border-radius` to see how they affect the look of your buttons.
  Learn more: The `border-radius` property, The `background-color` property, The `color` property

🧭 *Enhancements*

- **Add a New Sound**: Find a short `.mp3` sound effect and add it to your `sounds/` folder. Next, add its name to the `sounds` array in `script.js`. Then, create a matching `<audio>` tag in `index.html` with the correct `id` and `src`. The script will automatically create a button for your new sound.
  Hint: You can find free sound effects on sites like Pixabay, Freesound, or Zapsplat. Make sure your file is in `.mp3` format and use a simple, descriptive name (like `doorbell.mp3`). The filename will be used for the button text.
  Learn more: The `<audio>` element, The `id` global attribute, Pixabay Sound Effects, Freesound, Zapsplat
- **Add Visual Feedback When Playing**: Make the button change appearance while its sound is playing. You'll need to update your JavaScript to track which sound is playing and add or remove a class (like `is-playing`) to the button.
  Hint: In `style.css`, add a rule like `.btn.is-playing { background-color: rebeccapurple; }` for the playing state.
  In `script.js`:
  (1) Add `btn.classList.add("is-playing")` when a sound starts.
  (2) Store the audio element in a variable for reuse.
  (3) Use `audio.onended = () => btn.classList.remove("is-playing")` to reset the button when the sound finishes.
  (4) In `stopSounds()`, remove `is-playing` from all buttons with `document.querySelectorAll(".btn").forEach(btn => btn.classList.remove("is-playing"));`
  Learn more: The `ended` event, The `add()` method, The `remove()` method

See the solution for Day 9.

Day 10 - Dad Jokes

Get ready to laugh with the "Dad Jokes" generator! This project displays a random dad joke fetched from an online API. Click the button to get a new joke. You'll use HTML for the layout, CSS for styling, and JavaScript to fetch a joke from icanhazdadjoke.com using `fetch` and `async/await`. You'll then parse the JSON response and show the joke on the page.

🧠 *Challenges*

- **Style the Joke Text**: The joke text uses a specific font size and spacing. In `style.css`, find the `.joke` class. Try changing the `font-size` to make the text larger or smaller, or adjust the `line-height` for different spacing. You can also add a `color` property.
  Hint: Try using `rem` units instead of `px` for `font-size`. For example, `font-size: 1.875rem;` is the same as `font-size: 30px;` (since 1rem = 16px by default). `rem` units scale with user preferences, making your site more accessible. You can use online tools like PX to REM Converter to help convert values.
  Learn more: The `font-size` property, The `line-height` property, The `color` property, PX to REM Converter, CSS relative length units
- **Handle Network Failure Gracefully**: Turn off your internet connection and refresh the page. What do you see? By default, the placeholder says "// Joke goes here"—but that's not very friendly. Change this placeholder text in the HTML (inside `<div id="joke" class="joke">`) to something like "Hmm, our joke delivery service seems to be on a coffee break."
  Hint: This is called graceful degradation—making sure your app still works (or at least fails politely) when things go wrong. After you change the placeholder, try going offline to see your new message. Then turn your internet back on and check that jokes load as expected.
  Learn more: The `<div>` element, The `id` global attribute

🧭 *Enhancements*

- **Check API Response Status**: APIs don't always return what we expect. In `script.js`, after you fetch the joke, check if `res.status === 200` before displaying the joke. Use this code: `jokeEl.innerHTML = res.status === 200 ? data.joke : "No joke found!";`
  Hint: The `?` and `:` form a ternary operator—a shortcut for if/else. It works like this: `condition ? valueIfTrue : valueIfFalse`. So, if `status` is 200, show the joke; otherwise, show "No joke found!". To test, change the URL to `https://icanhazdadjoke.com/nonexistent` (which returns a 404) and see your error message. Remember to change it back afterward!
  Learn more: Icanhazdadjoke API documentation, The conditional (`?:`) operator, HTTP status codes
- **Prevent Multiple Clicks**: The API is fast, but users might click the button several times. Disable the button while fetching a new joke and change its text to show it's working. In `script.js`, inside `generateJoke()`, add `jokeBtn.disabled = true;` and `jokeBtn.innerText = "Loading...";` before the fetch call. After the joke is displayed, set `jokeBtn.disabled = false;` and `jokeBtn.innerText = "Get Another Joke";`. In `style.css`, add a `.btn:disabled` rule to style the disabled button (for example, `opacity: 0.6;` and `cursor: not-allowed;`).
  Hint: Disabling the button prevents users from making multiple API calls and gives feedback that something is happening. If you want to practice using the ternary operator, create a "helper function" to manage the button's state. A helper function is a small function that does one specific job. For example: `function setButtonState(isLoading) { jokeBtn.disabled = isLoading; jokeBtn.innerText = isLoading ? "Loading..." : "Get Another Joke"; }`. You could then call `setButtonState(true)` at the start of `generateJoke()` and `setButtonState(false)` at the end.
  Learn more: The `disabled` property, The `:disabled` pseudo-class

See the solution for Day 10.

Congratulations on completing Day 10!
Watch the recap video and review the slides.

Get the Full Guide

This is a free preview of the **full companion guide PDF** for the “100 Projects in 100 Days” series.

Ready for more after your first 10 projects?

Continue your journey with **Volume 1: Essentials (Days 1–50) on Amazon** — unlock 40 more hands-on projects to sharpen your skills and build your portfolio.

Want early access to future releases?

Volume 2: Advanced (Days 51–100) and an **All-in-One Edition** are coming soon.

Join the notification list to be the first to know when they're available!