

DATASOFTIXS

WEB DEVELOPMENT PROJECT

INTERN ID: WD100258

WEEK-1

CREATING YOUR DIGITAL IDENTITY

PROJECT 2: STORYTELLER'S HAVEN

Overview

The code provided creates a simple, responsive blog template suitable for aspiring writers or bloggers. It includes a header with navigation links, a main section displaying blog post previews, and a footer with social media links.

HTML Code Breakdown

1. Document Structure:

- The document starts with a **<!DOCTYPE html>** declaration, indicating that it is an HTML5 document.
- The **<html>** tag wraps the entire content, with a **lang** attribute set to "en" for English.

2. Head Section:

- The **<head>** section contains metadata about the document, including character encoding (**UTF-8**), viewport settings for responsive design, the title of the page, and a link to an external CSS stylesheet (**styles.css**).

3. Body Section:

- The **<body>** section contains the visible content of the webpage, structured into three main parts: **header**, **main**, and **footer**.

4. Header:

- The **<header>** contains a logo and a navigation bar (**<nav>**).
- The navigation bar is an unordered list (****) with list items (****) that link to different sections of the blog (Home, About, Categories, Contact).

5. Main Section:

- The **<main>** section includes a **<section>** for blog posts, which is styled to display multiple posts in a grid layout.
- Each blog post is represented by an **<article>** element, which includes:
 - An image (****) for visual appeal.
 - A title (**<h2>**) for the post's name.
 - An excerpt (**<p>**) providing a brief overview of the post's content.
 - A "Read More" link (**<a>**) that directs users to the full post.

6. Footer:

- The **<footer>** contains social media links and a copyright notice.
- Social media links are styled as anchor tags (**<a>**), allowing users to connect with the blog on various platforms.

CSS Code Breakdown

1. Global Styles:

- The universal selector (*) applies a box-sizing model to all elements, ensuring that padding and borders are included in the total width and height of elements. It also resets margins and paddings to zero for a consistent layout.

2. Body Styles:

- The body is styled with a default font family (Arial) and a line height for improved readability.

3. Header Styles:

- The header has a dark background color, white text, and uses flexbox to align the logo and navigation items. Padding is added for spacing.

4. Navigation Styles:

- The navigation list is displayed as a horizontal row using flexbox. List items have left margins for spacing, and links are styled to remove underlines and set the text color to white.

5. Main Section Styles:

- The main section has padding to create space around the content.

6. Blog Posts Section:

- The blog posts are arranged in a responsive grid layout using CSS Grid. The grid automatically adjusts the number of columns based on the available width, with a minimum column width of 300 pixels.

7. Individual Post Styles:

- Each blog post has a light gray border, rounded corners, and centered text. Overflow is hidden to ensure that any content exceeding the post's boundaries is not displayed.

8. Post Image Styles:

- Images within posts are set to take the full width of their container while maintaining their aspect ratio.

9. Post Title and Excerpt Styles:

- The title has a specific font size and vertical margins, while the excerpt has horizontal padding for better readability.

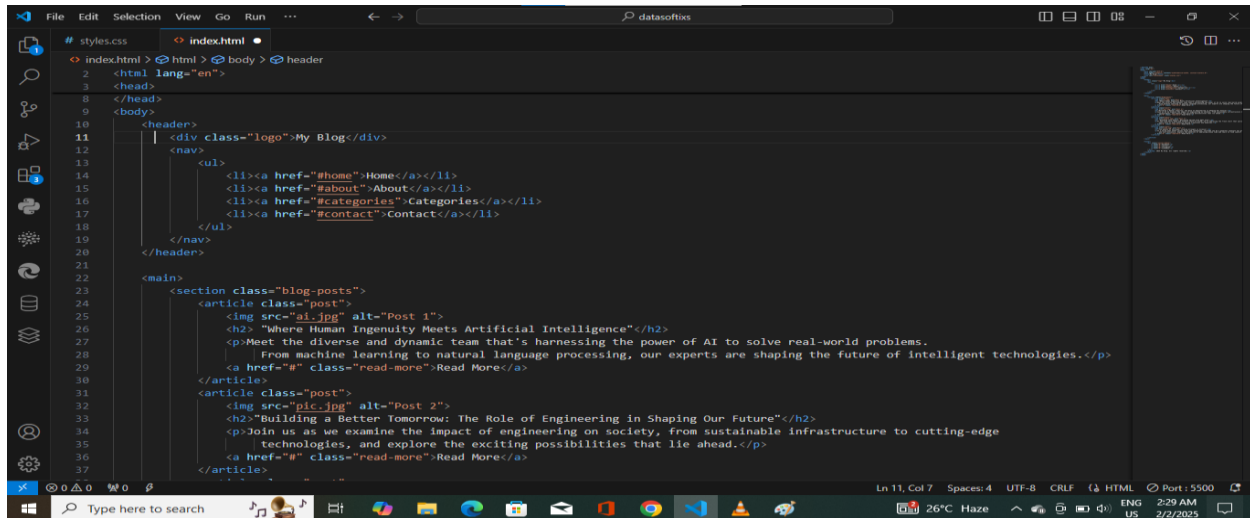
10. Read More Button Styles:

- The "Read More" button is styled as an inline-block element with padding, a dark background, white text, and rounded corners. It is designed to stand out and encourage users to click through to the full post.

11. Footer Styles:

- The footer has a dark background and white text, with social media links styled for visibility and spacing.

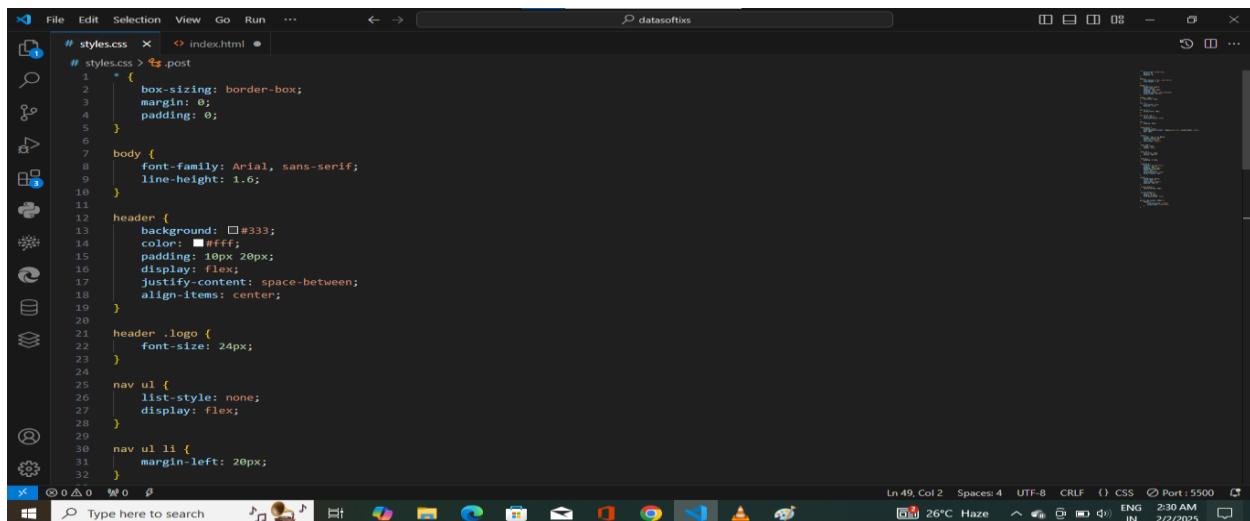
CODE SCREENSHOTS:



```

# styles.css
index.html
index.html > html > body > header
2 <html lang="en">
3 <head>
8 </head>
9 <body>
10
11 <header>
12 | <div class="logo">My Blog</div>
13
14 <nav>
15 <ul>
16 <li><a href="#home">Home</a></li>
17 <li><a href="#about">About</a></li>
18 <li><a href="#categories">Categories</a></li>
19 <li><a href="#contact">Contact</a></li>
20 </ul>
21 </nav>
22 </header>
23
24 <main>
25 <section class="blog-posts">
26 <article class="post">
27 
28 <h2>Where Human Ingenuity Meets Artificial Intelligence</h2>
29 <p>Meet the diverse and dynamic team that's harnessing the power of AI to solve real-world problems.
30 From machine learning to natural language processing, our experts are shaping the future of intelligent technologies.</p>
31 <a href="#" class="read-more">Read More</a>
32 </article>
33 <article class="post">
34 
35 <h2>Building a Better Tomorrow: The Role of Engineering in Shaping Our Future</h2>
36 <p>Join us as we examine the impact of engineering on society, from sustainable infrastructure to cutting-edge
37 technologies, and explore the exciting possibilities that lie ahead.</p>
38 <a href="#" class="read-more">Read More</a>
39 </article>
40 </main>
41 </body>
42 </html>

```

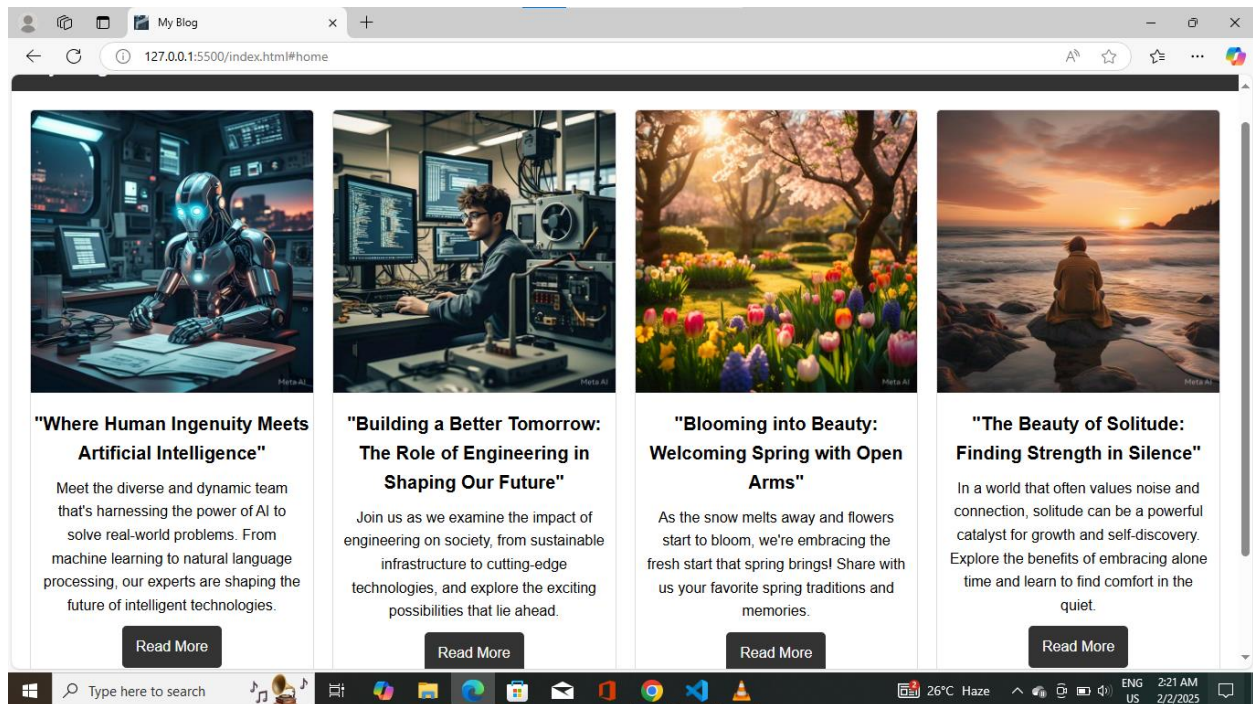


```

# styles.css
# styles.css > .post
1
2 {
3   box-sizing: border-box;
4   margin: 0;
5   padding: 0;
6 }
7
8 body {
9   font-family: Arial, sans-serif;
10  line-height: 1.0;
11 }
12
13 header {
14   background: #f3f3f3;
15   color: #fff;
16   padding: 10px 20px;
17   display: flex;
18   justify-content: space-between;
19   align-items: center;
20 }
21
22 header .logo {
23   font-size: 24px;
24 }
25
26 nav ul {
27   list-style: none;
28   display: flex;
29 }
30
31 nav ul li {
32   margin-left: 20px;
33 }

```

OUTPUT SCREENSHOTS:



WEEK 2 INTERACTIVE EXPERIENCES

PROJECT 3: TASK MASTER

HTML (todo.html)

1. **DOCTYPE and HTML Structure:** The document starts with the `<!DOCTYPE html>` declaration, followed by the opening `<html>` tag. The `<head>` section includes metadata, the title of the page, and a link to the CSS stylesheet.
2. **Body Content:**
 - A `<div>` with the class `container` wraps the main content of the application.
 - An `<h1>` element displays the title "To-Do List".
 - An `<input>` field with the ID `taskInput` allows users to type in new tasks.
 - A `<button>` with the ID `addTaskBtn` is provided for users to add the task they entered in the input field.
 - An unordered list (``) with the ID `taskList` will display the list of tasks.

3. **Script Inclusion:** The `<script>` tag at the end of the body includes the JavaScript file (**script.js**), which contains the logic for the application.

CSS (sty.css)

1. **Body Styles:** The body has a light gray background, and the font is set to Arial. Padding is added for spacing.
2. **Container Styles:** The container has a maximum width, centered alignment, white background, padding, rounded corners, and a subtle shadow for a card-like appearance.
3. **Heading Styles:** The heading is centered.
4. **Input and Button Styles:** The input field and button are styled for a clean look. The button changes color on hover to indicate interactivity.
5. **List Styles:** The unordered list has no default styling, and each list item (``) is styled to display flexibly with padding and a bottom border.
6. **Completed Task Styles:** Completed tasks have a line-through text decoration and a lighter color to indicate they are done.
7. **Delete Button Styles:** The delete button is styled similarly to the add button but with a red background, indicating a destructive action.

JavaScript (scripts.js)

1. **Event Listener for DOMContentLoaded:** The script starts by adding an event listener that triggers when the DOM is fully loaded. This calls the **loadTasks** function to populate the task list from local storage.
2. **Element References:** The script retrieves references to the input field, button, and task list using **document.getElementById**.
3. **Add Task Functionality:**
 - The **addTask** function is triggered when the "Add Task" button is clicked.
 - It retrieves the value from the input field, checks if it's not empty, and then calls **createTaskElement** to create a new task element.
 - The task is also saved to local storage using **saveTaskToLocalStorage**.
4. **Creating Task Elements:**
 - The **createTaskElement** function creates a new list item (``) with a span for the task text and a delete button.
 - If the task is marked as completed, it adds the **completed** class to the list item.
5. **Handling Task Clicks:**

- The **handleTaskClick** function listens for clicks on the task list.
- If a task's text is clicked, it toggles the completed state and updates local storage.
- If the delete button is clicked, it removes the task from both the DOM and local storage.

6. Local Storage Functions:

- **saveTaskToLocalStorage**: Saves a new task to local storage.
- **updateTaskInLocalStorage**: Updates the completed status of a task in local storage.
- **deleteTaskFromLocalStorage**: Removes a task from local storage when it is deleted.

Code screenshot:

The image displays two screenshots of a VS Code editor window showing JavaScript code for a task manager application. The top screenshot shows the initial setup, and the bottom screenshot shows the complete implementation.

Top Screenshot Code:

```

1  const taskInput = document.getElementById('task-input');
2  const addTaskButton = document.getElementById('add-task-button');
3  const taskList = document.getElementById('task-list');
4
5  // Load tasks from local storage
6  let tasks = JSON.parse(localStorage.getItem('tasks')) || [];
7
8  // Function to render tasks
9  function renderTasks() {
10     taskList.innerHTML = '';
11     tasks.forEach((task, index) => {
12         const li = document.createElement('li');
13         li.textContent = task.text;
14         if (task.completed) {
15             li.classList.add('completed');
16         }
17
18         // Create a complete button
19         const completeButton = document.createElement('button');
20         completeButton.textContent = task.completed ? 'Undo' : 'Complete';
21         completeButton.addEventListener('click', () => {
22             task.completed = !task.completed;
23             saveTasks();
24             renderTasks();
25         });
26
27         // Create a delete button
28         const deleteButton = document.createElement('button');
29         deleteButton.textContent = 'Delete';
30         deleteButton.classList.add('delete-button');
31         deleteButton.addEventListener('click', () => {
32             tasks.splice(index, 1);

```

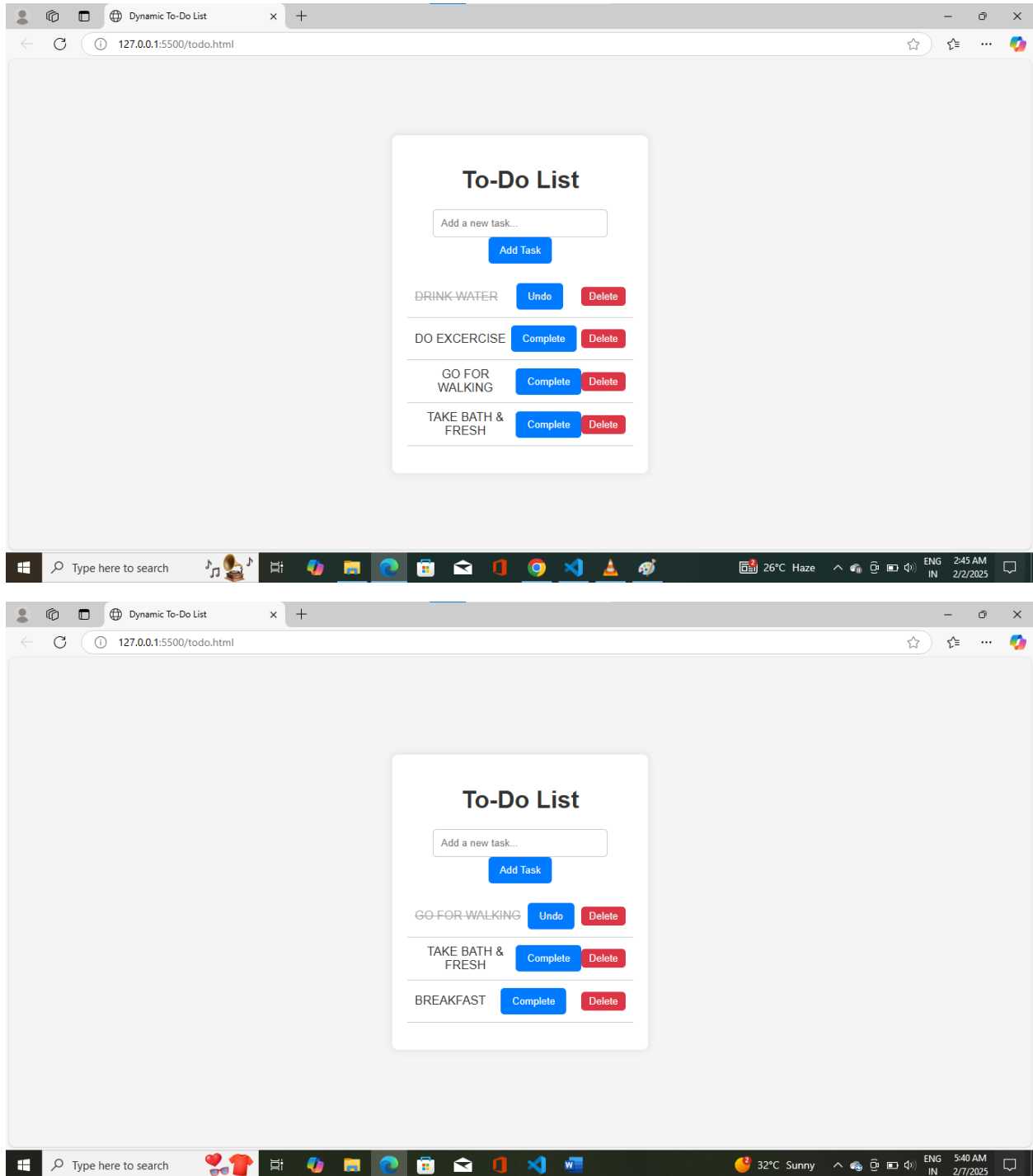
Bottom Screenshot Code:

```

9  function renderTasks() {
10     tasks.forEach((task, index) => {
31         deleteButton.addEventListener('click', () => {
32             tasks.splice(index, 1);
33             saveTasks();
34             renderTasks();
35         });
36
37         li.appendChild(completeButton);
38         li.appendChild(deleteButton);
39         taskList.appendChild(li);
40     });
41 }
42
43 // Function to save tasks to local storage
44 function saveTasks() {
45     localStorage.setItem('tasks', JSON.stringify(tasks));
46 }
47
48 // Event listener for adding a task
49 addTaskButton.addEventListener('click', () => {
50     const taskText = taskInput.value.trim();
51     if (taskText) {
52         tasks.push({ text: taskText, completed: false });
53         taskInput.value = '';
54         saveTasks();
55         renderTasks();
56     }
57 });
58
59 // Initial render
60 renderTasks();

```

Output screenshot:



WEEK 3 BRINGING IDEA'S TO LIFE

PROJECT 6: THE ART EXPLORER

HTML Structure: The HTML consists of a filter section, a gallery section with images, and a modal for displaying larger images.

CSS Styles: The CSS styles create a responsive grid layout for the gallery, add hover effects, and style the modal.

JavaScript Functionality: The JavaScript handles the opening and closing of the modal when images are clicked, as well as the filtering of images based on categories. When an image is clicked, the modal displays the larger version of the image, and clicking the close button hides the modal. The filter buttons allow users to show or hide images based on their categories, enhancing the user experience by making it easy to navigate through different types of images.

HTML Structure

1. **Document Declaration:** The document starts with the `<!DOCTYPE html>` declaration, indicating that this is an HTML5 document.
2. **Head Section:**
 - The `<head>` section includes metadata such as character set and viewport settings for responsive design.
 - The title of the page is set to "Responsive Image Gallery".
 - A link to an external CSS file (`styles.css`) is included for styling.
3. **Body Section:**
 - **Filter Buttons:** A set of buttons is created to filter images by categories. Each button has a `data-filter` attribute that corresponds to the category it filters.
 - **Gallery:** The main gallery is structured using a series of `div` elements, each containing an image. Each image is wrapped in a `gallery-item` div, and categories are assigned as classes (e.g., `category1`, `category2`).
 - **Modal:** A modal structure is defined to display larger images when clicked. It includes a close button (`×`) and an image element to show the selected image.

CSS Styles (`styles.css`)

1. **Body Styles:** The body uses a simple font-family for readability.
2. **Filter Buttons:** The filter buttons are styled for spacing and cursor indication, making them visually appealing and interactive.
3. **Gallery Layout:**

- The gallery is set up as a grid using CSS Grid Layout, allowing for a responsive design that adjusts the number of columns based on the screen size.
 - Each gallery item has a relative position and overflow hidden to ensure that hover effects are contained within the item.
4. **Image Hover Effects:**
- The images scale up slightly when hovered over, creating a zoom effect. This is achieved using CSS transitions.
5. **Modal Styles:**
- The modal is initially hidden (**display: none**) and covers the entire viewport when displayed.
 - The modal background is semi-transparent black, and the modal content is centered with a maximum width and height.
6. **Close Button:** The close button is styled to be prominent and easy to click, positioned in the top-right corner of the modal.

JavaScript Functionality (scripts.js)

1. **Event Listener for DOMContentLoaded:** The script waits for the DOM to fully load before executing any JavaScript, ensuring that all elements are available for manipulation.
2. **Modal Functionality:**
 - An event listener is added to each gallery image. When an image is clicked, the modal is displayed, and the clicked image's source is set as the modal image's source.
 - The modal can be closed by clicking the close button, which sets the modal's display to **none**.
3. **Filtering Functionality:**
 - Event listeners are added to each filter button. When a button is clicked, the script checks the **data-filter** attribute to determine which images to show or hide.
 - If the filter is "all", all images are displayed. Otherwise, only images that match the selected category are shown.

Summary

This code creates a fully functional image gallery that is responsive and user-friendly. The key features include:

- **Responsive Design:** The gallery adapts to different screen sizes using CSS Grid.
- **Hover Effects:** Images have a zoom effect on hover, enhancing interactivity.

- **Modal View:** Clicking an image opens a modal displaying a larger version of the image, improving the viewing experience.
- **Filtering:** Users can filter images by categories, allowing for easy navigation through different types of content.

Output screenshots:

