Build Modal Dialog

Building a Modal Dialog

A **modal dialog** is a popup window that appears on top of the main content, typically used for alerts, confirmations, forms, or additional information. In this assignment, we will create a modal dialog that can be opened and closed using JavaScript.

📌 Features

- A **button** to open the modal.
- A **close button** inside the modal.
- Clicking outside the modal (on the background overlay) will also close it.
- Smooth **CSS transitions** for better user experience.

🛠️ Steps to Build the Modal

We'll break it down into three parts:

1. **HTML** – The structure of the modal.
2. **CSS** – Styling for the modal and animations.
3. **JavaScript** – Handling user interactions.

1️⃣ HTML (Structure of the Modal)

```
html


<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Modal Dialog Example</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>

    <!-- Button to Open Modal -->
    <button id="openModalBtn">Open Modal</button>

    <!-- Modal Structure -->
    <div id="modalOverlay" class="modal-overlay">
        <div class="modal">
            <span class="close-btn">&times;</span>
            <h2>Modal Title</h2>
            <p>This is a simple modal dialog box.</p>
        </div>
    </div>

    <script src="script.js"></script>
```

```
</body>
</html>
```

📌 Explanation

- `#openModalBtn` → The button that will trigger the modal.
- `#modalOverlay` → The full-screen background that contains the modal.
- `.modal` → The actual modal box.
- `.close-btn` → The "×" (close) button to hide the modal.

2️⃣ CSS (Styling and Animations)

```
css

/* Reset styles */
body {
    font-family: Arial, sans-serif;
    text-align: center;
    margin: 0;
    padding: 0;
}

/* Button styling */
#openModalBtn {
    padding: 10px 20px;
    font-size: 18px;
    cursor: pointer;
    background-color: #007bff;
    color: white;
    border: none;
    border-radius: 5px;
    margin-top: 20px;
}

/* Modal overlay (Background) */
.modal-overlay {
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background: rgba(0, 0, 0, 0.5);
    display: flex;
    align-items: center;
    justify-content: center;
    opacity: 0;
    visibility: hidden;
    transition: opacity 0.3s ease, visibility 0.3s ease;
}

/* Modal box */
.modal {
    background: white;
    padding: 20px;
    width: 300px;
    border-radius: 8px;
    text-align: center;
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.3);
    position: relative;
    transform: translateY(-50px);
    transition: transform 0.3s ease;
}

/* Close button */
```

```
.close-btn {
    position: absolute;
    top: 10px;
    right: 15px;
    font-size: 24px;
    cursor: pointer;
}

/* Show modal when active */
.modal-overlay.show {
    opacity: 1;
    visibility: visible;
}

.modal-overlay.show .modal {
    transform: translateY(0);
}
```

📌 Explanation

- The **modal is hidden by default** (`opacity: 0; visibility: hidden;`).
- When the `show` class is added via JavaScript:
  - The overlay appears (`opacity: 1; visibility: visible;`).
  - The modal **animates in smoothly** from above (`transform: translateY(0);`).
- The close button (`×`) is positioned **at the top right** inside the modal.

3️⃣ JavaScript (Adding Functionality)

```
js

// Select elements
const openModalBtn = document.getElementById('openModalBtn');
const modalOverlay = document.getElementById('modalOverlay');
const closeModalBtn = document.querySelector('.close-btn');

// Function to open the modal
openModalBtn.addEventListener('click', () => {
    modalOverlay.classList.add('show');
});

// Function to close the modal
closeModalBtn.addEventListener('click', () => {
    modalOverlay.classList.remove('show');
});

// Close modal when clicking outside the modal box
modalOverlay.addEventListener('click', (event) => {
    if (event.target === modalOverlay) {
        modalOverlay.classList.remove('show');
    }
});
```

📌 Explanation

1. **Selecting Elements**
   - We grab the **button**, **modal overlay**, and **close button** using `document.getElementById` and `document.querySelector`.

2. **Opening the Modal**
    - When the button is clicked, the `show` class is added to `modalOverlay`, making it visible.
3. **Closing the Modal**
    - Clicking on the close button (`×`) removes the `show` class, hiding the modal.
4. **Clicking Outside Closes the Modal**
    - If the user clicks anywhere outside the `.modal` box (on the overlay), it closes the modal.

📌 Summary

Component	Functionality
HTML	Defines the structure of the modal and the button
CSS	Styles the modal, overlay, and animations
JavaScript	Handles user interaction for opening and closing the modal

---

🎯 Key Learnings

✅ **DOM Manipulation** – Using `getElementById` and `querySelector` to interact with elements.
✅ **Event Handling** – `addEventListener()` to detect button clicks and perform actions.
✅ **CSS Transitions** – Smooth animations using `transform` and `opacity`.
✅ **Conditional Logic** – Closing the modal only when clicking outside of it.

🚀 Full Code (For Easy Copy-Paste)

📌 HTML (index.html)

```
html
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Modal Dialog Example</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>

    <button id="openModalBtn">Open Modal</button>

    <div id="modalOverlay" class="modal-overlay">
        <div class="modal">
            <span class="close-btn">&times;</span>
            <h2>Modal Title</h2>
            <p>This is a simple modal dialog box.</p>
        </div>
    </div>

    <script src="script.js"></script>
</body>
</html>
```

📌 CSS (styles.css)

```
css

body { font-family: Arial, sans-serif; text-align: center; margin: 0; padding: 0; }
#openModalBtn { padding: 10px 20px; font-size: 18px; cursor: pointer; background-color: #007bff; co
.modal-overlay { position: fixed; top: 0; left: 0; width: 100%; height: 100%; background: rgba(0, 0
.modal { background: white; padding: 20px; width: 300px; border-radius: 8px; text-align: center; bo
.close-btn { position: absolute; top: 10px; right: 15px; font-size: 24px; cursor: pointer; }
.modal-overlay.show { opacity: 1; visibility: visible; }
.modal-overlay.show .modal { transform: translateY(0); }
```

📌 JavaScript (script.js)

```
js

const openModalBtn = document.getElementById('openModalBtn');
const modalOverlay = document.getElementById('modalOverlay');
const closeModalBtn = document.querySelector('.close-btn');

openModalBtn.addEventListener('click', () => modalOverlay.classList.add('show'));
closeModalBtn.addEventListener('click', () => modalOverlay.classList.remove('show'));
modalOverlay.addEventListener('click', (event) => { if (event.target === modalOverlay) modalOverlay
```

Let me know if you need improvements or explanations! 🚀

📌 Implementing Debouncing in JavaScript

Debouncing is a technique that ensures a function is executed **only after a certain delay** when an event (like typing in a search box) stops occurring. This prevents unnecessary function calls, reducing **API requests** and improving **performance**.

🔥 Why Use Debouncing?

- **Prevents excessive API calls** while typing.
- **Enhances performance** by limiting function execution.
- **Improves user experience** by reducing unnecessary data fetching.

🛠️ Features of Our Project

✔️ A **search input box** where users type queries.
✔️ **Debouncing function** to delay API requests.
✔️ **Fetch mock results** from an API (e.g., GitHub Users API).
✔️ **Display fetched results** dynamically.

1️⃣ Understanding Debouncing

A debounce function **delays** the execution of another function until after a specified period **has passed without further calls**.

- **Example Scenario:**

- Without **debouncing**: If a user types "Hello," an API request is triggered **5 times** (once for each keystroke).
- With **debouncing**: The API request is triggered **only once**, after the user stops typing for a set time (e.g., 500ms).

2️⃣ Project Structure

```
bash

/debounce-project
├── index.html      (HTML file)
├── styles.css      (CSS file)
└── script.js       (JavaScript file)
```

3️⃣ HTML (index.html)

```
html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Debounced Search</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>

    <h2>Search Users</h2>
    <input type="text" id="searchInput" placeholder="Type to search..." />

    <ul id="results"></ul>

    <script src="script.js"></script>
</body>
</html>
```

📌 Explanation

- `<input>` **box** → Users will type search queries here.
- `<ul>` **list** → Display fetched search results dynamically.
- `script.js` → Handles debouncing and API calls.

4️⃣ CSS (styles.css)

```css
body {
    font-family: Arial, sans-serif;
    text-align: center;
    padding: 20px;
}

#searchInput {
    width: 80%;
    padding: 10px;
    font-size: 18px;
    border: 2px solid #007bff;
    border-radius: 5px;
    margin-bottom: 10px;
}

ul {
    list-style: none;
    padding: 0;
    max-width: 400px;
    margin: auto;
}

li {
    padding: 10px;
    background: #f8f9fa;
    margin: 5px 0;
    border-radius: 5px;
    transition: background 0.3s;
}

li:hover {
    background: #e2e6ea;
}
```

📌 Explanation

- Styled the **search bar** for a clean look.
- Used a **list (`<ul>`)** to display search results.
- Added a **hover effect** on search results.

5️⃣ JavaScript (script.js)

```js
// Select input box and result list
const searchInput = document.getElementById("searchInput");
const resultsList = document.getElementById("results");

// Debounce function to delay API call
function debounce(func, delay) {
    let timer;
    return function (...args) {
        clearTimeout(timer);
        timer = setTimeout(() => func(...args), delay);
    };
}

// Fetch API results
async function fetchResults(query) {
    if (query.trim() === "") {
        resultsList.innerHTML = ""; // Clear results if input is empty
        return;
```

```
    }

    try {
        const response = await fetch(`https://api.github.com/search/users?q=${query}`);
        const data = await response.json();
        displayResults(data.items);
    } catch (error) {
        console.error("Error fetching results:", error);
    }
}

// Display fetched results
function displayResults(users) {
    resultsList.innerHTML = "";
    users.slice(0, 5).forEach(user => {
        const li = document.createElement("li");
        li.textContent = user.login;
        resultsList.appendChild(li);
    });
}

// Apply debounce to API call
const debouncedSearch = debounce(fetchResults, 500);

// Listen for input events
searchInput.addEventListener("input", (event) => {
    debouncedSearch(event.target.value);
});
```

6️⃣ 📌 Explanation of JavaScript Code

🔹 Step 1: Selecting Elements

```
js

const searchInput = document.getElementById("searchInput");
const resultsList = document.getElementById("results");
```

- `searchInput` → Captures user input.
- `resultsList` → Displays fetched search results.

🔹 Step 2: Implementing Debounce Function

```
js

function debounce(func, delay) {
    let timer;
    return function (...args) {
        clearTimeout(timer);  // Clear previous timer
        timer = setTimeout(() => func(...args), delay); // Call function after delay
    };
}
```

- Uses `setTimeout()` to **delay** function execution.
- `clearTimeout(timer)` ensures only the **latest call** is executed.

🔹 Step 3: Fetching API Results

```
js

async function fetchResults(query) {
    if (query.trim() === "") {
        resultsList.innerHTML = ""; // Clear results if input is empty
        return;
    }

    try {
        const response = await fetch(`https://api.github.com/search/users?q=${query}`);
        const data = await response.json();
        displayResults(data.items);
    } catch (error) {
        console.error("Error fetching results:", error);
    }
}
```

- Calls the **GitHub Users API** with the search query.
- **Clears results** if input is empty.
- **Handles API errors** using `` `try...catch` ``.

🔹 Step 4: Displaying Results

```
js

function displayResults(users) {
    resultsList.innerHTML = "";
    users.slice(0, 5).forEach(user => {
        const li = document.createElement("li");
        li.textContent = user.login;
        resultsList.appendChild(li);
    });
}
```

- Clears previous results.
- **Shows top 5 users** based on the query.
- Creates **list items dynamically**.

🔹 Step 5: Applying Debounce

```
js

const debouncedSearch = debounce(fetchResults, 500);
```

- The `` `fetchResults` `` function is **wrapped inside debounce** with a `` `500ms` `` delay.

🔹 Step 6: Listening for Input Changes

```
js

searchInput.addEventListener("input", (event) => {
    debouncedSearch(event.target.value);
```

```
});
```

- Calls `debouncedSearch` **only after user stops typing**.

7️⃣ Final Code (For Easy Copy-Paste)

📌 HTML (index.html)

```
html


<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Debounced Search</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>

    <h2>Search Users</h2>
    <input type="text" id="searchInput" placeholder="Type to search..." />

    <ul id="results"></ul>

    <script src="script.js"></script>
</body>
</html>
```

📌 CSS (styles.css)

```
css


body { font-family: Arial, sans-serif; text-align: center; padding: 20px; }
#searchInput { width: 80%; padding: 10px; font-size: 18px; border: 2px solid #007bff; border-radius
ul { list-style: none; padding: 0; max-width: 400px; margin: auto; }
li { padding: 10px; background: #f8f9fa; margin: 5px 0; border-radius: 5px; transition: background (
li:hover { background: #e2e6ea; }
```

📌 JavaScript (script.js)

```
js


const searchInput = document.getElementById("searchInput");
const resultsList = document.getElementById("results");

function debounce(func, delay) {
    let timer;
    return function (...args) {
        clearTimeout(timer);
        timer = setTimeout(() => func(...args), delay);
    };
}

async function fetchResults(query) {
    if (query.trim() === "") { resultsList.innerHTML = ""; return; }
    try { const response = await fetch(`https://api.github.com/search/users?q=${query}`);
        const data = await response.json();
        displayResults(data.items);
    } catch (error) { console.error("Error fetching results:", error); }
}
```

```
function displayResults(users) {
    resultsList.innerHTML = "";
    users.slice(0, 5).forEach(user => {
        const li = document.createElement("li");
        li.textContent = user.login;
        resultsList.appendChild(li);
    });
}

const debouncedSearch = debounce(fetchResults, 500);
searchInput.addEventListener("input", (event) => debouncedSearch(event.target.value));
```

This is **optimized, efficient, and a great learning exercise**! 🚀 Let me know if you need any improvements.