Task Tracker App

Table of Contents

Introduction

Welcome to the Task Tracker App development assignment! In this assignment, you'll be building a simple task management system using HTML, CSS, and JavaScript. The goal is to create an interactive application where users can input tasks, set deadlines, mark tasks as complete, and remove tasks.

Let's get started!

Getting Started

Create a new **Cursor Project** called for HTML/CSS/JS and ensure you have three files named: `index.html`, `script.js`, and `style.css`.

Now, you're ready to start building the Task Tracker App!

Building the Task Tracker App

Section 1: Setting Up HTML Structure

Open the `index.html` file and set up the basic HTML structure for the Task Tracker App.

```
<!-- index.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="style.css">
    <title>Task Tracker</title>
</head>
<body>
    <div class="container">
        <h1>Task Tracker</h1>
        <form id="taskForm">
            <!-- Task form inputs go here -->

        </form>
        <table id="taskTable">
            <!-- Task table content here -->
        </table>
    </div>
    <script src="script.js"></script>
</body>
</html>
```

Your Turn: This is a basic HTML structure for the app. Your first task is to add the necessary form inputs.

Add the following form inputs inside the `taskForm` element:

- A label and input field for task name with an id of `taskName`.
- A label and text area for task description with an id of `taskDescription`.
- A label and input field for task deadline with an id of `taskDeadline`.

```
<form>
  <label for="taskName">Task name:</label><br>
  <input type="text" id="taskName" name="taskName"><br>
  <label for="taskDescription">Description:</label><br>
  <textarea id="taskDescription" name="taskDescription"> </textarea><br><br>
 <label for="taskDeadline">Task deadline:</label><br>
  <input type="date" id="taskDeadline" name="taskDeadline"><br>
<button type="submit">Add Task</button>
</form>
```

Test your changes by opening the `index.html` file in your web browser from the live server.

Section 2: Basic Styling with CSS

Open the `style.css` file and add basic styling to make your app visually appealing.

```
/* style.css */
body {
    font-family: 'Arial', sans-serif;
    margin: 0;
    padding: 0;
    background-color: #f4f4f4;
}

.container {
    max-width: 800px;
    margin: 50px auto;
    padding: 20px;
    background-color: #fff;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

form {
    margin-bottom: 20px;
}

/* Add more styles as needed */
```

Explanation: This bit of code should have added some default styling to the form with some margin at the bottom for better spacing.

Test your changes and commit your work:

Section 3: JavaScript Implementation

Open the `script.js` file so we can implement some JavaScript functionality. Remember that you can add Javascript to your websites by adding a script tag to your HTML files. We have one `<script/>` tag in the file now that is sourcing the javascript from this file we're creating. We'll break the code into multiple sections for better understanding.

Section 3.1: TODOs

Add the following code to the top of your `script.js` file:

```
// script.js

// Section 1: TODOs
// TODO: Register submissions from the user on the form.
// TODO: Determine the value of the data submitted and add it to a JavaScript array calle
// TODO: Call the render function to update the table with the new tasks.
```

Here we have added some comments to the top of the file to help guide us through the implementation of the JavaScript functionality. We'll be implementing the functionality in the next sections.

Section 3.2: App State Variables

Add this to the bottom of your `script.js` file:

```
// script.js

// Section 2: App State Variables
let tasks = [];
```

This code initializes an app state variable `tasks` as an empty array. We'll use this array to keep track of the tasks and update the table with the new tasks.

Section 3.3: Cached Element References

Add the following code to the bottom of your `script.js` file:

Your Turn:

Use the `document.getElementById` method to get the form and table elements and assign them to the `taskForm` and `taskTable` variables respectively.

Update your lines so instead of setting them to null we are getting references to the two elements:

```
const taskForm = document.getElementById("taskForm")
const taskTable = document.getElementById("taskTable")
```

To test your changes, you can add a `console.log` statement to log the values of `taskForm` and `taskTable` to the console. If the values are `null`, it means the elements are not being selected correctly. If the values are not `null`, and hovering over them in the console shows the correct elements, then you have successfully selected them!

Test your changes by opening the `index.html` file in your web browser from the live server.

Section 3.4: Functions and Event Listeners

We're going to add some code to the bottom of your `script.js` file to implement the functions and event listeners we need for the Task Tracker App.

Add the following code to the bottom of your `script.js` file:

```
// Function to handle form submissions
function handleSubmission(event) {
    event.preventDefault(); // this function stops our form from reloading the page

    // TODO: Get form input values

    // TODO: Validate input fields

    // TODO: Update the tasks array

    render();
}

// Function to render tasks in the table
function render() {
    // TODO: Use array methods to create a new table row of data for each item in the arr
}

// Function to initialize the table
function init() {
    taskTable.innerHTML = ''; // Clear the table
    tasks = []; // Reset the tasks array
    render(); // Call the render function
}
```

These are the functions we'll be using to handle form submissions, render tasks in the table, and initialize the table. We'll be implementing the functionality for these functions in the next steps.

First we'll add the `handleSubmission` function to handle form submissions. This function will capture form input values, validate them, update the `tasks` array, and call the `render` function to update the table with the new tasks. Functions that are used to handle events are called event handlers, so this function is an event handler for our form submissions.

Inside the `handleSubmission` function, we'll need to get the form input values, validate them, update the `tasks` array, and call the `render` function.

Under the `// TODO: Get form input values` comment, we'll use the `document.getElementById` method to get the form input values and assign them to variables. You can use the `document.getElementById` method to get the form input values and assign them to variables. For

example, to get the value of the task name input field, you can use the following code: `const taskName = document.getElementById('taskName').value`

This code gets the value of the task name input field and assigns it to the `taskName` variable. Do the same for the task description and task deadline input fields.

Under the `// TODO: Validate input fields` comment, we'll validate the input fields. If the task name and deadline are not filled out, we'll display an alert message to the user and return from the function.

First, we'll need to validate the input fields. If the task name and deadline are not filled out, we'll display an alert message to the user and return from the function. Add an `if` statement to check if the task name and deadline are not filled out. If they are not filled out, use the `alert` method to display an alert message to the user and return from the function.

For example: `alert('Task name and deadline are required!')`

Finally under the `// TODO: Update the tasks array` comment, we'll update the `tasks` array with the new task by pushing an object with the task name, description, and deadline to the array.

Use the `push` method to add a new task to the `tasks` array. For example: `tasks.push({ name: taskName, description: taskDescription, deadline: taskDeadline })`

Then we'll call the `render` function to update the table with the new tasks.

Here's what it should look like finally:

```
  function handleSubmission(event) {
      event.preventDefault();

      // TODO: Get form input values
        const taskName = document.getElementById('taskName').value
        const taskDescription = document.getElementById('taskDescription').value
        const taskDeadline = document.getElementById('taskDeadline').value

      // TODO: Validate input fields
        if(taskName == "" || taskDeadline == "") {
           alert('Task name and deadline are required!')
        }

      // TODO: Update the tasks array

  tasks.push({ name: taskName, description: taskDescription, deadline: taskDeadline })

      render();
```

```
}
```

Next, we'll work on the `render` function to render tasks in the table. This function will use array methods to create a new table row of data for each item in the array. We'll be using the `map` method to create a new table row for each task in the `tasks` array. You can learn more about the `map` method here.

Under the `// TODO: Use array methods to create a new table row of data for each item in the array` comment, use the `map` method to create a new table row for each task in the `tasks` array. You can use the `map` method to create a new table row for each task in the `tasks` array. For example:

```
taskTable.innerHTML = tasks.map(task => `
    <tr>
        <td>${task.name}</td>
        <td>${task.description}</td>
        <td>${task.deadline}</td>
        <td><button onclick="markTaskComplete(this)">Complete</button></td>
        <td><button onclick="removeTask(this)">Remove</button></td>
    </tr>
`).join('');
```

This is a sophisticated piece of code that uses the `map` method to create a new table row for each task in the `tasks` array. The `map` method creates a new array by calling a provided function on every element in the array. In this case, we're using the `map` method to create a new table row for each task in the `tasks` array. We're going to create the table row as a string, and we're using the `join` method to join the string elements and create a new array of HTML strings which is passed into our table. Super complex but we can talk about it together tomorrow!

After this function, you should see a function called `init` to initialize the table at the end of your javascript file. This function will clear the table, reset the tasks array, and call the render function to update the table with the new tasks.

In order to make the app interactive, we'll add an event listener to listen for form submissions and trigger the `handleSubmission` function.

Event listeners are used to listen for events on a specific element and trigger a function when the event occurs. In this case, we're adding an event listener to listen for form submissions and trigger the `handleSubmission` function we implemented earlier.

Add the following code to the bottom of your `script.js` file:

```
// Event listener for form submission
taskForm.addEventListener('submit', handleSubmission);
```

This code adds an event listener to listen for form submissions and trigger the `handleSubmission` function.

Lastly, we need to call the `init` function to set up the initial state of the app when the page loads.

At the very bottom of your `script.js` file, add the following code:

```
// Call the init function to set up the initial state of the app
init();
```

Javascript files are executed in the order they are loaded, so the `init` function will be called after the `handleSubmission` function and the `render` function are defined. This will set up the initial state of the app when the page loads. Because we're calling the `init` function at the bottom of the file, it will be called once the page has loaded and the DOM is ready. Because we've added our element references, event listeners, and functions to the file, the app will be set up and ready to go when the `init` function is called.

You should now have a complete `script.js` file with the necessary functions and event listeners to handle form submissions and render tasks in the table. Test your changes by opening the `index.html` file in your web browser from the live server and try adding a new task. You should see the new task added to the table.

If you're having trouble, you can use the console in your browser's developer tools to check for errors and debug your code. You can also use `console.log` statements to log values to the console and check if your code is working as expected.

```
// Section 4: Functions and Event Listeners

// Function to handle form submissions
function handleSubmission(event) {
    event.preventDefault();

    // TODO: Get form input values
    const taskName = document.getElementById('taskName').value;
    const taskDescription = document.getElementById('taskDescription').value;
    const taskDeadline = document.getElementById('taskDeadline').value;

    // TODO: Validate input fields
    if (!taskName|| !taskDeadline) {
```

```
            alert('Task name and deadline are required!');
            return;
        }

        // TODO: Update the tasks array
    tasks.push({ name: taskName, description: taskDescription, deadline: taskDeadline });

        // TODO: Call the render function
        render();
    }

    // Function to render tasks in the table
    function render() {
        // TODO: Use array methods to create a new table row of data for each item in the arr
        taskTable.innerHTML = tasks.map(task => `
            <tr>
                <td>${task.name}</td>
                <td>${task.description}</td>
                <td>${task.deadline}</td>
                <td><button onclick="markTaskComplete(this)">Complete</button></td>
                <td><button onclick="removeTask(this)">Remove</button></td>
            </tr>
        `).join('');
    }

    // Function to initialize the table
    function init() {
        taskTable.innerHTML = ''; // Clear the table
        tasks = []; // Reset the tasks array
        render(); // Call the render function
    }

    // Event listener for form submission
    taskForm.addEventListener('submit', handleSubmission);

    // Call the init function to set up the initial state of the app
    init();
```

Congratulations! You've successfully implemented the JavaScript functionality for the Task Tracker App. Feel free to explore additional improvements and features to enhance your Task Tracker App further. Happy coding!