Js Projects

Creating a **To-Do App** like this is a great project to learn JavaScript, HTML, and CSS. Let me walk you through the steps to build and understand this project.

Step 1: Setting Up the Project Structure

1. **Create a folder for your project** and name it something like `todo-app`.
2. Inside this folder:
    - Create three files: `index.html`, `styles.css`, and `script.js`.

Step 2: Building the HTML

The **HTML** provides the structure of your app.

1. Copy the code provided for `index.html`.
2. Understand its components:
    - The `div` with the class `container` wraps the entire app.
    - Inside this container:
        - An `h1` element displays the title.
        - An `input` field and a button (`Add Task`) allow users to add tasks.
        - An `ul` element serves as a placeholder for the task list.
    - The `<script>` tag at the end includes the JavaScript logic from `script.js`.

Step 3: Styling with CSS

The **CSS** styles your app to make it look attractive and provide a dark mode theme.

1. Copy the code provided for `styles.css`.
2. Understand the styles:
    - The `body` sets the background color, font, and central alignment.
    - `.container` styles the app card with padding, border-radius, and a shadow.
    - The `input` and `button` are styled for consistent spacing and a dark mode look.
    - `ul` and `li` define the list layout and completed tasks are styled with a `line-through`.

Step 4: Adding JavaScript Functionality

The **JavaScript** makes the app interactive.

1. Copy the code for `script.js`.
2. Understand the key functionalities:
    - **Loading Existing Tasks:**

```javascript
let tasks = JSON.parse(localStorage.getItem("tasks")) || [];
```

- It fetches tasks from `localStorage` (a browser storage system) or initializes an empty list.

- **Adding a Task:**

```javascript
const newTask = { id: Date.now(), text: taskText, completed: false };
tasks.push(newTask);
saveTasks();
renderTask(newTask);
todoInput.value = "";
```

- When a user enters a task and clicks `Add Task`, the task is saved, rendered on the page, and stored in `localStorage`.

- **Rendering a Task:**

```javascript
function renderTask(task) {
  const li = document.createElement("li");
  li.setAttribute("data-id", task.id);
  li.innerHTML = `<span>${task.text}</span><button>delete</button>`;
}
```

- Dynamically adds the task to the `ul` element with a delete button.

- **Completing a Task:**

```javascript
li.addEventListener("click", (e) => {
  if (e.target.tagName === "BUTTON") return;
  task.completed = !task.completed;
  li.classList.toggle("completed");
  saveTasks();
});
```

- Toggles the `completed` state and applies the appropriate styling.

- **Deleting a Task:**

```javascript
li.querySelector("button").addEventListener("click", (e) => {
  e.stopPropagation();
  tasks = tasks.filter((t) => t.id !== task.id);
  li.remove();
  saveTasks();
});
```

- Removes the task from the `tasks` array, updates `localStorage`, and removes it from the UI.

- **Saving Tasks:**

```
javascript


  function saveTasks() {
    localStorage.setItem("tasks", JSON.stringify(tasks));
  }
```

- Ensures that tasks persist even after refreshing the browser.

Step 5: Testing the App

1. Open the folder in a code editor (e.g., VS Code).
2. Start a local server to preview your app (e.g., by installing the **Live Server** extension in VS Code).
3. Open the app in the browser and test:
   - Adding tasks.
   - Marking tasks as complete.
   - Deleting tasks.
   - Refreshing the browser to confirm persistence.

Step 6: Iterate and Improve

- **Enhancements to Consider:**
  - Add an edit option for tasks.
  - Include categories or priorities for tasks.
  - Style completed tasks differently (e.g., change their background color).
  - Add animations when tasks are added or removed.

If you face any challenges or want to extend the app, feel free to ask for further guidance!

Writing your own `script.js` step by step is a fantastic way to learn! Here's a guide to help you create the functionality piece by piece:

Step 1: Set Up Basic Event Listeners

Start by linking your `script.js` file to the `index.html` and adding an event listener to check if everything is working.

```
javascript


  document.addEventListener("DOMContentLoaded", () => {
    console.log("Script is connected!");
  });
```

- **Test**: Open your browser's developer tools (F12), and you should see "Script is connected!" in the console.

Step 2: Access Key HTML Elements

Identify and grab references to the input, button, and task list elements using their `id`.

```
javascript


 document.addEventListener("DOMContentLoaded", () => {
   const todoInput = document.getElementById("todo-input");
   const addTaskButton = document.getElementById("add-task-btn");
   const todoList = document.getElementById("todo-list");

   console.log(todoInput, addTaskButton, todoList);
 });
```

- **Test**: When you open the console, you should see the references to these elements.

Step 3: Handle the Add Task Button Click

Write functionality to read the input and append a task to the list.

1. Add an event listener for the button.
2. Create a new list item (`li`) dynamically.
3. Add the task text to the list item.
4. Append the `li` to the `ul`.

```
javascript


 addTaskButton.addEventListener("click", () => {
   const taskText = todoInput.value.trim();
   if (taskText === "") return;

   const li = document.createElement("li");
   li.textContent = taskText;

   todoList.appendChild(li);
   todoInput.value = ""; // Clear the input
 });
```

- **Test**: Add a task and see if it shows up in the list.

Step 4: Save Tasks in Memory

Use a `tasks` array to keep track of the added tasks.

1. Create a `tasks` array.
2. Push each new task into the array.
3. Log the array to confirm it's working.

```javascript
const tasks = [];

addTaskButton.addEventListener("click", () => {
  const taskText = todoInput.value.trim();
  if (taskText === "") return;

  const newTask = {
    id: Date.now(),
    text: taskText,
    completed: false,
  };

  tasks.push(newTask);
  console.log(tasks);

  const li = document.createElement("li");
  li.textContent = taskText;

  todoList.appendChild(li);
  todoInput.value = "";
});
```

- **Test**: Check the console to see if tasks are being added to the `tasks` array.

Step 5: Render Tasks with Delete and Completion

Update the task rendering to include:

1. A "delete" button.
2. A way to mark tasks as completed by clicking.

```javascript
function renderTask(task) {
  const li = document.createElement("li");
  li.setAttribute("data-id", task.id);
  li.innerHTML = `
    <span>${task.text}</span>
    <button>delete</button>
  `;

  // Mark as completed
  li.addEventListener("click", (e) => {
    if (e.target.tagName === "BUTTON") return; // Ignore if the delete button is clicked
    task.completed = !task.completed;
    li.classList.toggle("completed");
  });

  // Delete task
  li.querySelector("button").addEventListener("click", (e) => {
    e.stopPropagation(); // Prevent marking as completed
    tasks = tasks.filter((t) => t.id !== task.id); // Remove task from array
    li.remove(); // Remove from UI
  });

  todoList.appendChild(li);
}

addTaskButton.addEventListener("click", () => {
  const taskText = todoInput.value.trim();
  if (taskText === "") return;
```

```
    const newTask = { id: Date.now(), text: taskText, completed: false };
    tasks.push(newTask);
    renderTask(newTask);
    todoInput.value = "";
  });
```

- **Test**: Add tasks and try marking them as complete or deleting them.

Step 6: Persist Tasks with Local Storage

1. Save tasks to `localStorage` whenever they are updated.
2. Load tasks from `localStorage` when the app starts.

Save Tasks to Local Storage

Add a helper function:

```
javascript


  function saveTasks() {
    localStorage.setItem("tasks", JSON.stringify(tasks));
  }
```

Call `saveTasks()` whenever tasks are modified:

```
javascript


  addTaskButton.addEventListener("click", () => {
    const taskText = todoInput.value.trim();
    if (taskText === "") return;

    const newTask = { id: Date.now(), text: taskText, completed: false };
    tasks.push(newTask);
    saveTasks();
    renderTask(newTask);
    todoInput.value = "";
  });
```

Load Tasks from Local Storage

At the start, load and render existing tasks:

```
javascript


  document.addEventListener("DOMContentLoaded", () => {
    const savedTasks = JSON.parse(localStorage.getItem("tasks")) || [];
    savedTasks.forEach(renderTask);
  });
```

- **Test**: Refresh the page and check if tasks persist.

Final Step: Refine and Test

- Add some dummy tasks for testing.
- Experiment with deleting, completing, and adding new tasks.
- Check browser local storage to confirm tasks are saved.

If you follow these steps and understand each block of code, you'll learn to build the script entirely on your own! 🚀

Is this conversation helpful so far?