To-Do List App

Let's break this assignment into well-structured steps so that you can learn the most from it.

Step 1: Understanding the Core Features

The To-Do List application should have:

1. An input field to add tasks.
2. A button to submit tasks.
3. A list to display tasks.
4. Each task should have an "Edit" and "Delete" option.
5. Tasks should be stored in **local storage** so that they persist even after refreshing the page.

Concepts You Will Learn

- **DOM Manipulation**: To dynamically update the UI.
- **Event Handling**: To respond to user interactions.
- **Local Storage**: To store and retrieve tasks.
- **Array Methods**: To manage task operations efficiently.

Step 2: Building the HTML Structure

We need a basic structure for the application.

HTML Code

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>To-Do List</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="container">
        <h1>To-Do List</h1>
        <div class="todo-input">
            <input type="text" id="taskInput" placeholder="Add a new task">
            <button id="addTaskBtn">Add Task</button>
        </div>
        <ul id="taskList"></ul>
    </div>
    <script src="script.js"></script>
</body>
</html>
```

Explanation

- `input` **field**: Allows users to type a task.
- `button`: Adds the task when clicked.
- `ul` **(unordered list)**: Holds all the tasks.
- **CSS & JavaScript files linked**: `styles.css` for styling and `script.js` for functionality.

Step 3: Styling the To-Do List

Let's make the UI look clean.

CSS Code (styles.css)

```
css


body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f4;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    margin: 0;
}

.container {
    background: white;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);
    width: 350px;
    text-align: center;
}

h1 {
    margin-bottom: 15px;
}

.todo-input {
    display: flex;
    gap: 10px;
}

input {
    flex: 1;
    padding: 8px;
    border: 1px solid #ccc;
    border-radius: 5px;
}

button {
    padding: 8px 15px;
    border: none;
    background: #28a745;
    color: white;
    cursor: pointer;
    border-radius: 5px;
}

button:hover {
    background: #218838;
}

ul {
```

```
    list-style: none;
    padding: 0;
    margin-top: 20px;
}

li {
    background: #fff;
    display: flex;
    justify-content: space-between;
    align-items: center;
    padding: 10px;
    margin-bottom: 5px;
    border-radius: 5px;
    border: 1px solid #ddd;
}

.edit-btn {
    background: #ffc107;
    color: black;
}

.delete-btn {
    background: #dc3545;
}

button.edit-btn:hover {
    background: #e0a800;
}

button.delete-btn:hover {
    background: #c82333;
}
```

Explanation

- The `container` centers the to-do app.
- `input` and `button` are styled for a clean look.
- `ul` styles list items (`li`) with padding and a border.
- Different colors for **Add, Edit, and Delete** buttons.

Step 4: Implementing the Functionality

Now, we will implement JavaScript to handle adding, editing, deleting, and storing tasks.

JavaScript Code (script.js)

```
js


// Select elements
const taskInput = document.getElementById("taskInput");
const addTaskBtn = document.getElementById("addTaskBtn");
const taskList = document.getElementById("taskList");

// Retrieve tasks from local storage
let tasks = JSON.parse(localStorage.getItem("tasks")) || [];

// Function to render tasks
function renderTasks() {
    taskList.innerHTML = ""; // Clear the list before rendering
    tasks.forEach((task, index) => {
        const li = document.createElement("li");
        li.innerHTML = `
            <span>${task}</span>
```

```
            <div>
                <button class="edit-btn" onclick="editTask(${index})">Edit</button>
                <button class="delete-btn" onclick="deleteTask(${index})">Delete</button>
            </div>
        `;
        taskList.appendChild(li);
    });
}

// Function to add a task
function addTask() {
    const taskText = taskInput.value.trim();
    if (taskText !== "") {
        tasks.push(taskText);
        localStorage.setItem("tasks", JSON.stringify(tasks));
        taskInput.value = ""; // Clear input field
        renderTasks(); // Update UI
    }
}

// Function to delete a task
function deleteTask(index) {
    tasks.splice(index, 1);
    localStorage.setItem("tasks", JSON.stringify(tasks));
    renderTasks();
}

// Function to edit a task
function editTask(index) {
    const newTask = prompt("Edit task:", tasks[index]);
    if (newTask !== null && newTask.trim() !== "") {
        tasks[index] = newTask.trim();
        localStorage.setItem("tasks", JSON.stringify(tasks));
        renderTasks();
    }
}

// Event Listeners
addTaskBtn.addEventListener("click", addTask);
window.addEventListener("load", renderTasks);
```

Step 5: Understanding the Code

1. Storing and Retrieving Data

- `localStorage.setItem("tasks", JSON.stringify(tasks))` → Saves the array as a string.
- `JSON.parse(localStorage.getItem("tasks"))` → Retrieves the stored tasks and converts them back to an array.

2. Rendering Tasks

- `renderTasks()`:
  - Clears the list before updating it.
  - Loops through `tasks[]` and creates `li` elements.
  - Adds **Edit** and **Delete** buttons dynamically.

3. Adding Tasks

- `addTask()`:

- Gets text from the input field.
- Pushes the new task into the `tasks[]` array.
- Updates **local storage**.
- Calls `renderTasks()` to refresh the UI.

4. Editing Tasks

- `editTask(index)`:
  - Uses `prompt()` to let users modify the task.
  - Updates **local storage** after editing.

5. Deleting Tasks

- `deleteTask(index)`:
  - Removes the task from the `tasks[]` array.
  - Updates **local storage**.

6. Event Handling

- `addTaskBtn.addEventListener("click", addTask)` → Calls `addTask()` when the button is clicked.
- `window.addEventListener("load", renderTasks)` → Ensures tasks are loaded when the page refreshes.

Step 6: Testing the Application

- Open the file in a browser.
- Try adding tasks.
- Refresh the page (tasks should persist).
- Try editing and deleting tasks.
- Ensure local storage is working (`F12 > Application > Local Storage`).

Final Thoughts

This assignment covers essential **JavaScript skills**: ✅ **DOM Manipulation**
✅ **Event Handling**
✅ **Local Storage**
✅ **Array Methods**

Try modifying the code! Some ideas:

- Add a **"Mark as Completed"** feature.
- Use **CSS animations** for adding/removing tasks.

- Implement **drag-and-drop sorting**.

Would you like me to add more features? 🚀