JavaScript Project - To-Do List

Back to JS page

Table of Contents

- JavaScript Project - To-Do List
  - To-Do List Overview
  - You Will Learn
  - First: Create the HTML
    - Example 1
  - Add a Display Function
    - Example 2
  - Add an Add Task Function
  - Add a Remove Task Function
  - Add a Clear All Function
  - Function to Save a Task
  - Function to Load the Tasks
  - Final Project
    - Example 3
  - Exercises & Solutions
    - Example 4
  - Bonus Challenges (Level Up)
    - Example 5
  - Document
  - Reference

To-Do List Overview

In this project, you will build a functional **To-Do List**.

- You will be able to **add**, **remove**, and **clear all** tasks.
- The tasks will be kept synchronized within an **Array** saved in **localStorage**.

You Will Learn

- How to store items in an array structure
- How to add and remove items dynamically from an array
- How to generate and update HTML lists based on array values
- How to serialize and load JSON data using localStorage

First: Create the HTML

Start by creating an HTML file with:

- An `<input>` field with `id="task"`.
- An "Add" button calling `addTask()`.
- An unordered list `<ul>` with `id="list"`.
- A "Clear All" button calling `clearAll()`.
- A script tag initializing an empty `tasks` array.

```
<!DOCTYPE html>
<html>
<body>

<h2>To-Do List</h2>
<p>From w3schools.com, Experiment by Teeratus_R</p>

<input id="task" placeholder="New task">
<button onclick="addTask()">Add</button>
<ul id="list"></ul>
<button onclick="clearAll()">Clear All</button>

<script>
// Create a task Array
let tasks = [];
</script>

</body>
</html>
```

Example 1

Result View Example

Add a Display Function

The `displayTasks()` function loops through the array items, wraps each item in `<li>` elements alongside remove buttons, and prints the result inside the list element:

```
// Function to display the list
function displayTasks() {
  let html = "";
  for (let i = 0; i < tasks.length; i++) {
    html += "<li>" + tasks[i] +
    " <button onclick='removeTask(" + i + ")'>x</button></li>";
  }
  document.getElementById("list").innerHTML = html;
}
```

Example 2

Result View Example

Add an Add Task Function

Retrieves the input value, pushes it to the tasks array, resets input text, and updates list views:

```
// Function to Add a task
function addTask() {
  let taskInput = document.getElementById("task");
  let text = taskInput.value;
  if (text === "") {
    return;
  }
  tasks.push(text);
  taskInput.value = "";
  saveTasks();
  displayTasks();
}
```

Add a Remove Task Function

Removes tasks by index using the JavaScript array `splice()` method:

```
// Function to Remove a task
function removeTask(i) {
  tasks.splice(i, 1);
  saveTasks();
  displayTasks();
}
```

> [!NOTE]
> `tasks.splice(i, 1)` deletes exactly 1 item at index position `i`.

Add a Clear All Function

Empties the array and refreshes the displays:

```
// Function to Clear all tasks
function clearAll() {
  tasks = [];
  saveTasks();
  displayTasks();
}
```

Function to Save a Task

Converts the tasks array into text format and saves it inside `localStorage`:

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

> [!IMPORTANT]
> `localStorage` only handles text keys and values. We must use `JSON.stringify(tasks)` to convert the tasks array into a string before saving.

Function to Load the Tasks

Retrieves string data and parses it back into an array structure:

```
function loadTasks() {
  let saved = localStorage.getItem("tasks");
  if (saved !== null) {
    tasks = JSON.parse(saved);
  }
}
```

Final Project

Combines HTML code structure and state persistence logics together:

```
<!DOCTYPE html>
<html>
<body>

<h2>To-Do List</h2>
<p>From w3schools.com, Experiment by Teeratus_R</p>

<input id="task" placeholder="New task">
<button onclick="addTask()">Add</button>
<ul id="list"></ul>
<button onclick="clearAll()">Clear All</button>

<script>
let tasks = [];

function displayTasks() {
  let html = "";
  for (let i = 0; i < tasks.length; i++) {
    html += "<li>" + tasks[i] +
    " <button onclick='removeTask(" + i + ")'>x</button></li>";
  }
  document.getElementById("list").innerHTML = html;
}

function addTask() {
  let taskInput = document.getElementById("task");
  let text = taskInput.value;
  if (text === "") {
    return;
  }
  tasks.push(text);
  taskInput.value = "";
  saveTasks();
  displayTasks();
}

function removeTask(i) {
  tasks.splice(i, 1);
  saveTasks();
  displayTasks();
}

function clearAll() {
  tasks = [];
  saveTasks();
  displayTasks();
}

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

```
}

function loadTasks() {
  let saved = localStorage.getItem("tasks");
  if (saved !== null) {
    tasks = JSON.parse(saved);
  }
}

loadTasks();
displayTasks();
</script>

</body>
</html>
```

Example 3

Result View Example

Exercises & Solutions

Exercise 1

Show an alert if the user tries to add an empty task.

Exercise 2

Allow the user to press Enter to add a task.

Exercise 3

Add a "Saved!" message to show when saving.

Solutions Code

```
<input id="task" placeholder="New task" onkeydown="handleKey(event)">
<button onclick="addTask()">Add</button>
<ul id="list"></ul>
<button onclick="clearAll()">Clear All</button>
<p id="status" style="color: green; font-weight: bold;"></p>

<script>
// (Inside script)
function addTask() {
  let taskInput = document.getElementById("task");
  let text = taskInput.value.trim();
  if (text === "") {
    alert("Please enter a task! Task cannot be empty.");
    return;
  }
  tasks.push(text);
  taskInput.value = "";
  saveTasks();
  displayTasks();
}

function handleKey(event) {
  if (event.key === "Enter") {
    addTask();
  }
}

function saveTasks() {
  localStorage.setItem("tasks", JSON.stringify(tasks));
  showStatus("Saved!");
}

function showStatus(msg) {
  let statusEl = document.getElementById("status");
  statusEl.innerHTML = msg;
  setTimeout(function () {
    statusEl.innerHTML = "";
  }, 2000);
}
</script>
```

Example 4

Result View Example

Bonus Challenges (Level Up)

Improve task workflows by adding premium styling, status checks, and filtering controls:

- **Checkbox markers:** Check off tasks as done (triggers text strikethrough).
- **Filters:** Toggle list views by active state categories (All / Active / Completed).
- **Inline Editing:** Modify list names and save task changes dynamically.
- **Sorting:** Sort list items alphabetically in one click.

Example 5

Result View Example

Document

Document in project

You can Download PDF file.

Reference

- W3Schools JavaScript To-Do List Project