Todo App Project

The whole series in one app: add, complete and delete tasks, clean rendering, and localStorage persistence proven with a refresh test.

Introduction

This project gathers the whole JavaScript series into one small, real app: the **DOM** (3.13), **events** (3.14), **forms** (3.15), **arrays of objects** (3.8 and 3.9), **JSON and localStorage** (3.23). Nothing new is invented here — every line is a lesson you already know, finally working together.

One design rule holds the whole app together: **one array is the single truth**, and the page is always **rendered from it**. Learn that rule and you can build far bigger apps than a todo list. *(Roman Urdu: Ye poori series ka aik chhota app hai — aik hi array asal sach hai, page hamesha usi se banta hai.)*

Key points

- **One array of task objects is the single truth** — `tasks` holds objects shaped `{ id, text, done }`, for example `{ id: 1, text: "Buy milk", done: false }`. Everything the user sees comes from this array, and nothing else.
- **One `render()` function draws the whole list** — it turns the array into page HTML with `map` and `join("")`, or shows the empty-state message when there are no tasks. You never edit the DOM by hand in five different places — change the array, call `render()`, the page obeys.

  (Roman Urdu: Page ko haath se har jagah na badlo — array badlo, render() chalao, page khud theek ho jayega.)
- **Add: read, push, save, render** — read the input value, push a new object with `Date.now()` as a fresh `id`, then **save** and **render**. Four small steps, always in that order.
- **Complete: toggle `done`** — find the task by its `id` and flip `done` from `false` to `true` (or back). The render gives done tasks a `done` class — a line-through style. The data changes; the style follows.
- **Delete: filter the id out** — `tasks = tasks.filter(task => task.id !== id)` keeps every task **except** the deleted one, then save and render. No hunting through the page for the element.
- **Save and load are one line each** — save: `localStorage.setItem("tasks", JSON.stringify(tasks))`. Load on start: `JSON.parse(localStorage.getItem("tasks")) || []` — the `|| []` gives a clean empty array on the very first visit, when the key does not exist yet (3.23's `null` trap, disarmed).

  (Roman Urdu: Save matlab array ko JSON bana kar localStorage mein rakhna — refresh par wapas parse karke wahi array mil jata hai.)
- **The refresh test proves it** — add tasks, press **F5**: variables die, `localStorage` does not, so the app loads the saved array and renders the same list. If your list survives F5, your persistence is real.
- **The empty state is part of the design** — when `tasks.length === 0`, render a friendly message like `No tasks yet. Add your first task!` instead of a blank hole in the page.

Code examples

Example 1 — The complete Todo App (single HTML file)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Todo App — Code Scope</title>
  <style>
    body { font-family: Arial, sans-serif; background: #faf9f6; color: #0b0f19; padding: 20px;
}
    .done { text-decoration: line-through; color: gray; }
    li { margin: 8px 0; }
    li span { cursor: pointer; }
    button { margin-left: 8px; }
  </style>
</head>
<body>
  <h1>My Todo List</h1>

  <input type="text" id="taskInput" placeholder="Write a task">
  <button onclick="addTask()">Add Task</button>

  <ul id="list"></ul>

  <script>
    let tasks = JSON.parse(localStorage.getItem("tasks")) || [];

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

    function render() {
      const list = document.getElementById("list");

      if (tasks.length === 0) {
        list.innerHTML = "<li>No tasks yet. Add your first task!</li>";
        return;
      }

      list.innerHTML = tasks.map((task) => `
        <li>
          <span class="${task.done ? "done" : ""}"
onclick="toggleTask(${task.id})">${task.text}</span>
```

```
          <button onclick="deleteTask(${task.id})">Delete</button>
        </li>
      `).join("");
    }

    function addTask() {
      const input = document.getElementById("taskInput");
      const text = input.value.trim();

      if (text === "") {
        alert("Please write a task first");
        return;
      }

      tasks.push({ id: Date.now(), text: text, done: false });
      saveTasks();
      render();
      input.value = "";
      input.focus();
    }

    function toggleTask(id) {
      const task = tasks.find((t) => t.id === id);
      if (task) {
        task.done = !task.done;
      }
      saveTasks();
      render();
    }

    function deleteTask(id) {
      tasks = tasks.filter((t) => t.id !== id);
      saveTasks();
      render();
    }

    render();
  </script>
</body>
</html>
```

What it does: This is the whole app in one file — copy it, save it as an HTML file, open it in a browser, and it works. The page has one input, one Add button and one list. The script holds the `tasks` array (loaded from `localStorage` at the start), the `saveTasks()` saver, the single `render()` that draws the list from the array

with `map` / `join`, and the three actions — `addTask`, `toggleTask`, `deleteTask` — each of which changes the array, saves, and re-renders. Clicking a task's text toggles it complete; its Delete button removes it.

What you will see: the page opens with the heading `My Todo List`, an empty input, and the message `No tasks yet. Add your first task!`. Type `Buy milk` and click Add Task: the message is replaced by `Buy milk` with a Delete button beside it, and the input clears. Add `Study JavaScript`: it appears below the first task. Click the words `Buy milk`: the text gains a line through it and turns gray — click it again and the line disappears. Click Delete next to `Study JavaScript`: it vanishes. Now press **F5**: after the refresh, `Buy milk` is still there, still crossed out. That is the refresh test, passed.

Example 2 — Add, complete and delete: the three actions up close

```
function addTask() {
  const input = document.getElementById("taskInput");
  const text = input.value.trim();

  if (text === "") {
    alert("Please write a task first");
    return;
  }

  tasks.push({ id: Date.now(), text: text, done: false });
  saveTasks();
  render();
  input.value = "";
  input.focus();
}

function toggleTask(id) {
  const task = tasks.find((t) => t.id === id);
  if (task) {
    task.done = !task.done;
  }
  saveTasks();
  render();
}

function deleteTask(id) {
  tasks = tasks.filter((t) => t.id !== id);
  saveTasks();
  render();
}
```

What it does: These are the same three actions from the full app, shown alone so the pattern is easy to see. Every action follows the identical recipe: **change the array, save, render**. Add pushes a new object (empty text is refused first), toggle finds the task by `id` with `find` and flips its `done`, delete rebuilds the array with

`filter` keeping everything except that `id`. None of them touches the page directly — `render()` is the only function that writes HTML.

What you will see: trace it by hand with the list from Example 1. With tasks `Buy milk` (id saved) and `Study JavaScript`, calling `toggleTask` with the first task's id flips only its `done` to `true` — after `render()`, only that line is crossed out. Calling `deleteTask` with the second task's id leaves a one-task array, and after `render()` only `Buy milk` remains on the page. Add an empty input: the alert `Please write a task first` appears and nothing changes — no blank task can ever enter the array.

Example 3 — The localStorage layer: save, load, and the refresh test

```
// Load once, at the very start of the script:
let tasks = JSON.parse(localStorage.getItem("tasks")) || [];

// Save after every single change:
function saveTasks() {
  localStorage.setItem("tasks", JSON.stringify(tasks));
}
```

What it does: Two small pieces do all the remembering in the app. At startup, `getItem("tasks")` reads the saved JSON text and `JSON.parse` turns it back into the array of objects — or, on the first visit ever, `getItem` returns `null`, `JSON.parse(null)` gives `null`, and `|| []` steps in with an empty array. After that, `saveTasks()` is called inside every action, so the saved copy can never fall behind the real array.

What you will see: do the refresh test step by step. (1) Open the app, add two tasks and complete one of them. (2) In the browser, open DevTools, go to the Application tab, and look at Local Storage for this file: the key `tasks` holds text like `[{"id":1730000000000,"text":"Buy milk","done":true},{"id":1730000005000,"text":"Study JavaScript","done":false}]` — your exact ids will differ, because they come from `Date.now()`. (3) Press **F5**: both tasks return exactly as you left them, the completed one still crossed out. (4) Delete both tasks and refresh again: the empty-state message `No tasks yet. Add your first task!` is back, and the stored key now holds `[]`. The page never remembered anything — the drawer did, and the load line read it.

Quick recap

- **One array is the single truth** — `{ id, text, done }` objects; one **`render()`** draws the page from it with `map` / `join`, including the empty-state message.
- Every action follows the same recipe: **change the array → save → render** (add pushes, toggle flips `done`, delete filters the `id` out).
- Persistence is two lines: save with **`JSON.stringify`** into `localStorage`, load with **`JSON.parse(...) || []`** — proven by the F5 refresh test.

Common mistakes

1. **Editing the DOM by hand in every action** — add appends an element here, delete removes one there, toggle styles one directly — and soon the page and the array disagree, so a refresh resurrects deleted tasks. The fix: actions change **only the array**, then call the one `render()`.
2. **Forgetting to save after one of the actions** — adds survive a refresh but completions or deletes do not, because only `addTask` called `saveTasks()`. The fix: save inside **every** action, with no exceptions — change, save, render, every time.
3. **Loading without the `|| []` safety** — on the first visit `getItem("tasks")` is `null`, and code that expects an array crashes on `tasks.push` or `tasks.length`. The fix: `JSON.parse(localStorage.getItem("tasks")) || []` from day one.

Practice

1. Add an **Edit** button next to each task: clicking it puts the task's current text back into the input and removes the old task from the array (so Add re-creates it edited), then save and render. Test the full cycle — edit, refresh, confirm the edited text survived.
2. Add a counter line above the list that `render()` updates every time, in this exact shape: `2 tasks, 1 done`. Then add a **Clear completed** button that filters out every task with `done === true`, saves, and re-renders. Refresh after clearing and confirm the done tasks stay gone.

Practice quiz

1. What is the single truth of the Todo App?

- The HTML on the page
- One tasks array of objects shaped { id, text, done }; everything the user sees comes from this array
- localStorage alone
- The input field

Answer: One tasks array of objects shaped { id, text, done }; everything the user sees comes from this array

2. What recipe does every Todo action follow?

- Edit the DOM by hand, then hope the array agrees
- Change the array, then save, then render; add pushes, toggle flips done, delete filters the id out
- Save first, then change, then render
- Render first, then change

Answer: Change the array, then save, then render; add pushes, toggle flips done, delete filters the id out

3. How does the Todo App load safely on the very first visit?

- JSON.parse(localStorage.getItem("tasks")) alone, which crashes on null
- JSON.parse(localStorage.getItem("tasks")) || [], so a missing key gives a clean empty array

- It starts with the tasks of the last user
- It never loads, it starts empty by luck

Answer: JSON.parse(localStorage.getItem("tasks")) || [], so a missing key gives a clean empty array

---

Watch the Code Scope channel: https://www.youtube.com/@CodeScopePK — this lesson's own video joins the channel when it is recorded.

Read online: https://scopetreasure.com/notes/js-todo-app-project · More free tools and notes at https://scopetreasure.com