**Project Overview**

You created a **responsive To-Do List Web App** where users can:

* Add tasks
* Mark tasks as completed
* Delete tasks
* Automatically save tasks using **localStorage**

**HTML Summary**

* Structured with a container .todoapp that holds:
  + A heading with an icon
  + An input field and add button
  + A task list inside a <ul> tag
* JavaScript handles dynamic task creation and interaction.

**CSS Features**

* Applied **linear gradient background** using background: linear-gradient(...).
* Centered the To-Do app using margin and max-width.
* Styled the input and button inside a .row with flexbox.
* Added **hover effects** for buttons and delete icons.
* Used **custom images** (checked.png, unchecked.png) for task status toggling via pseudo-element ::before.
* Enhanced UI using:
  + box-shadow, border-radius
  + transition, hover, and responsive padding

**JavaScript Functionality**

1. **Add Task**
   * Validates input.
   * Creates a new <li> and a close <span>.
   * Appends to the task list.
2. **Delete Task**
   * Clicking the ❌ (span) removes the task.
3. **Mark as Completed**
   * Clicking the <li> toggles the .checked class and changes icon.
4. **Save Data**
   * Uses localStorage.setItem() to save the task list on every action.
5. **Load Data**
   * When the page loads, showTask() fetches tasks from localStorage.

**Technologies Used**

* **HTML** for structure
* **CSS** for styling and responsiveness
* **JavaScript** for interactivity and data persistence
* **LocalStorage** for saving user tasks permanently in the browser

**CSS Summary**

1. **Global Reset** with \* selector.
2. **Background**:
   * Full-screen gradient using linear-gradient.
3. **To-do App Box**:
   * Centered box with rounded corners and padding.
   * Responsive width (max-width: 540px).
4. **Input Row**:
   * Flexbox used to align input and button.
   * Stylish button with red background and rounded edges.
5. **Task List**:
   * Tasks styled with padding and hover effects.
   * Custom **checkbox icons** using ::before.
   * checked tasks shown with line-through and checked icon.
6. **Delete Button (span)**:
   * Positioned to the right of each task.
   * Hover effect for delete interaction.

**JavaScript Summary**

## 1. getElementById()

**Usage in your code:**

const inputBox = document.getElementById("input-box");

const listContainer = document.getElementById("list-container");

What it does:

It fetches the HTML element with the specified id.

Why it's used:

You need to access the input box (where user types task) and the container (<ul>) where tasks are displayed. Without this, you can't interact with the UI using JavaScript.

2. addEventListener()

**Usage in your code:**

listContainer.addEventListener("click", function(e) {

...

}, false);

What it does:

Attaches a function to run when a specific event (like a click) happens on an element.

Why it's used:

You are listening for clicks on the list (ul) to:

* Mark a task as complete when clicking on a <li>
* Delete a task when clicking on the <span> (×)

3. createElement()

**Usage in your code:**

let li = document.createElement("li");

let span = document.createElement("span");

What it does:

It creates a new HTML element (like <li> or <span>) dynamically using JavaScript.

Why it's used:

Whenever a user adds a new task, your app dynamically builds the <li> and adds it to the task list.

4. innerHTML

**Usage in your code:**

li.innerHTML = inputBox.value;

span.innerHTML = "\u00d7";

What it does:

Gets or sets the **HTML content** inside an element.

Why it's used:

* To insert the user’s task text into the <li>.
* To insert the close icon × using Unicode (\u00d7).

5. appendChild()

**Usage:**

li.appendChild(span);

listContainer.appendChild(li);

What it does:

Adds a new child element into a parent element.

Why it's used:

* You first add the delete icon (<span>) inside the task item (<li>).
* Then you add the complete <li> inside the <ul> (task list).

6. classList.toggle()

**Usage:**

e.target.classList.toggle("checked");

What it does:

Adds the class if it’s not there, removes it if it is.

Why it's used:

To add/remove the checked class when a task is clicked — this controls the style for marking the task as complete.

7. localStorage.setItem() and localStorage.getItem()

**Usage:**

localStorage.setItem("data", listContainer.innerHTML);

listContainer.innerHTML = localStorage.getItem("data");

What it does:

* setItem(key, value) saves data in browser storage
* getItem(key) fetches it later

Why it's used:

To save the list of tasks even after refreshing or reopening the browser.

8. innerText vs innerHTML (not used directly, but important)

* innerText: Just the visible text
* innerHTML: Includes HTML tags (like <span>)

You used innerHTML because tasks include a delete <span> icon, not just plain text.

9. e.target (Event Object)

**Usage:**

if (e.target.tagName === "LI") { ... }

What it does:

Gives you the exact element that was clicked.

Why it's used:

To check whether user clicked on a <li> (mark complete) or <span> (delete).

10. function and Event Binding

**Example:**

function addTask() { ... }

You attached this function to the button:

<button onclick="addTask()">ADD</button>

What it does:

Defines reusable logic — here, to add a new task into the list.