1. Overview of the Solution

A responsive To-Do app built with HTML, CSS, and JS

---

2. HTML Approach

I created a simple and semantic layout dividing the page in three sections:

- **Header:** Contains the page title, which introduces the purpose of the web app. This section could be reused across additional pages if the project expands.
- **Main:** Holds all the task-management features, allowing users to create, complete, and delete their own tasks and to filter which tasks should be shown using the options all, active and completed. These tasks are saved in localStorage through JavaScript, ensuring they persist between sessions.
- **Footer:** Currently displays a placeholder text (“General footer”). In the future, this area could be updated to show links to other pages or external resources useful to users.

---

3. CSS Approach

Design strategy:

- **Responsive interface:** To build a responsive interface, I used percentage-based widths throughout and applied a CSS grid layout for the task elements. Combined with `@media` queries, this allows the number of columns to adjust automatically based on the user's screen size.
- **Code reusability and maintainability:** I defined multiple CSS variables and reusable classes to improve maintainability and ensure consistent styling across the project.
- **Color palette:** For color styling, I selected the HSL (Hue, Saturation, Lightness) model to improve readability and make collaboration easier for other developers. The palette is based on various shades of black for the page background, lighter shades for elements to create a “light-from-above” effect, and different white tones to distinguish between primary and secondary text.
- **Typography:** For typography, I chose safe, widely available fonts so the design renders correctly across browsers and on most devices.

---

4. JavaScript Logic (Main Functionality)

This section explains the core logic behind the application:

- **Selecting DOM elements**
  I used a mix of `getElementById()` and `querySelector()` to efficiently target and interact with specific HTML elements.
- **How the “Add” button works**
  I attached a click event listener to the Add button. When triggered, it checks whether the input field is empty.
  - If it is empty, an error message (“Please enter a task”) is displayed.
  - If the input contains text, any previous error message is hidden and the `addTask()` function is executed. This function creates a new `<li>` element that contains the task description and its associated controls. The updated list of tasks is then saved to `localStorage` by overwriting the value assigned to the `"tasks"` key, using `JSON.stringify()` to convert the array into plain text. After generating the task, I add event listeners to the checkbox and delete button so users can modify the task’s state or remove it permanently. In the future, this could be connected to a database to track deleted tasks or generate statistics.
- **Validating empty input**
  To ensure the user enters valid text, I retrieve the input value, apply `.trim()`, and check that it is not an empty string using the `!==` operator. Additionally, I enforce a maximum length of 250 characters.
- **Creating tasks dynamically**
  Whenever the user clicks “Add” and the input is valid, the `addTask()` function creates a new <li> element and appends it to the <ul> that holds all tasks. During this process, the function also generates the required child elements (checkbox, text container, delete button) and assigns appropriate classes for styling.
- **Handling “complete” and “delete” actions**
  Once a task is created, I use `querySelector()` to target the newly added checkbox and attach an event listener. When clicked, it triggers the `changeTaskState()` function, which checks whether the task is currently marked as completed.
  - If it becomes checked, a line-through style is applied to the task description and the `checked` attribute is set.
  - If it is unchecked, the attribute is removed and the text-decoration resets to normal. Similarly, the delete button removes the task from both the DOM and `localStorage`.
- **Filtering tasks**
  I added an event listener to the filter `<select>` element so that each time its value changes, the function retrieves the selected filter option and uses a `switch` statement to determine which tasks should be shown or hidden.

5. Challenges

- Making sure tasks persist with localStorage
- Handling event listeners on dynamically added items
- Keeping code organized

---

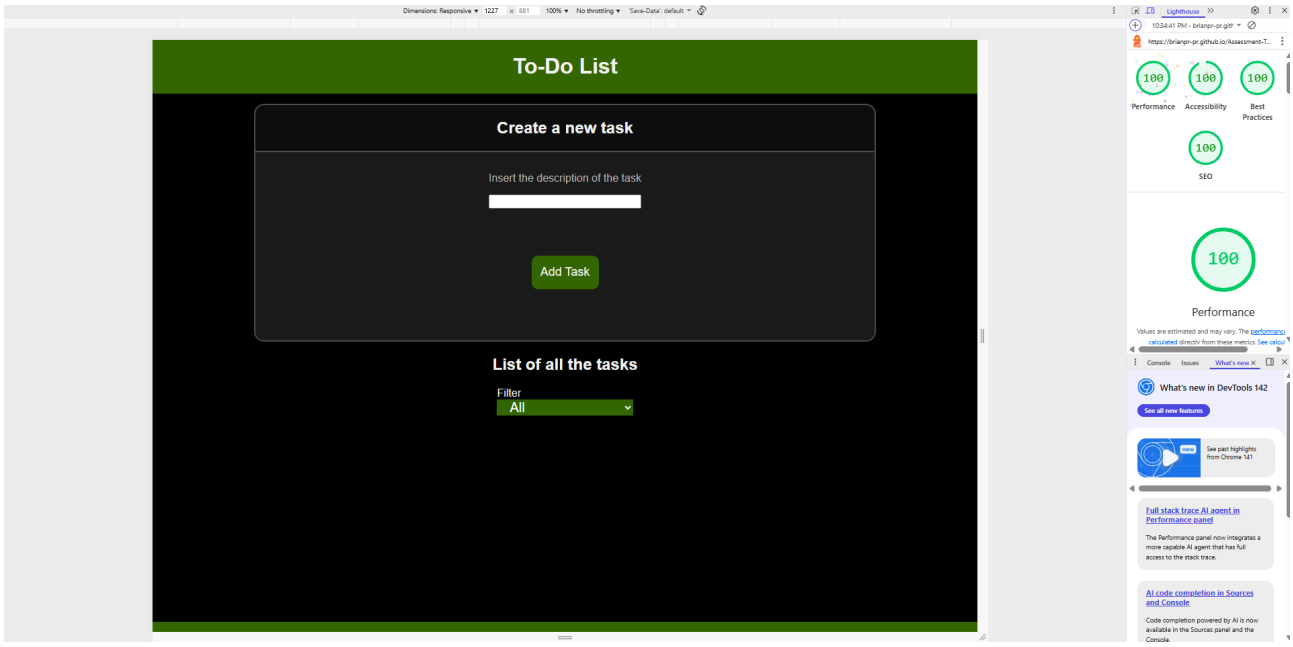
6. Future Improvements

- Adding animations
- Add an ‘Edit Task’ feature, display the creation date and time for each task, and include sorting options such as alphabetical order, creation date, and ascending or descending order.
- Improving accessibility (keyboard navigation, ARIA roles)

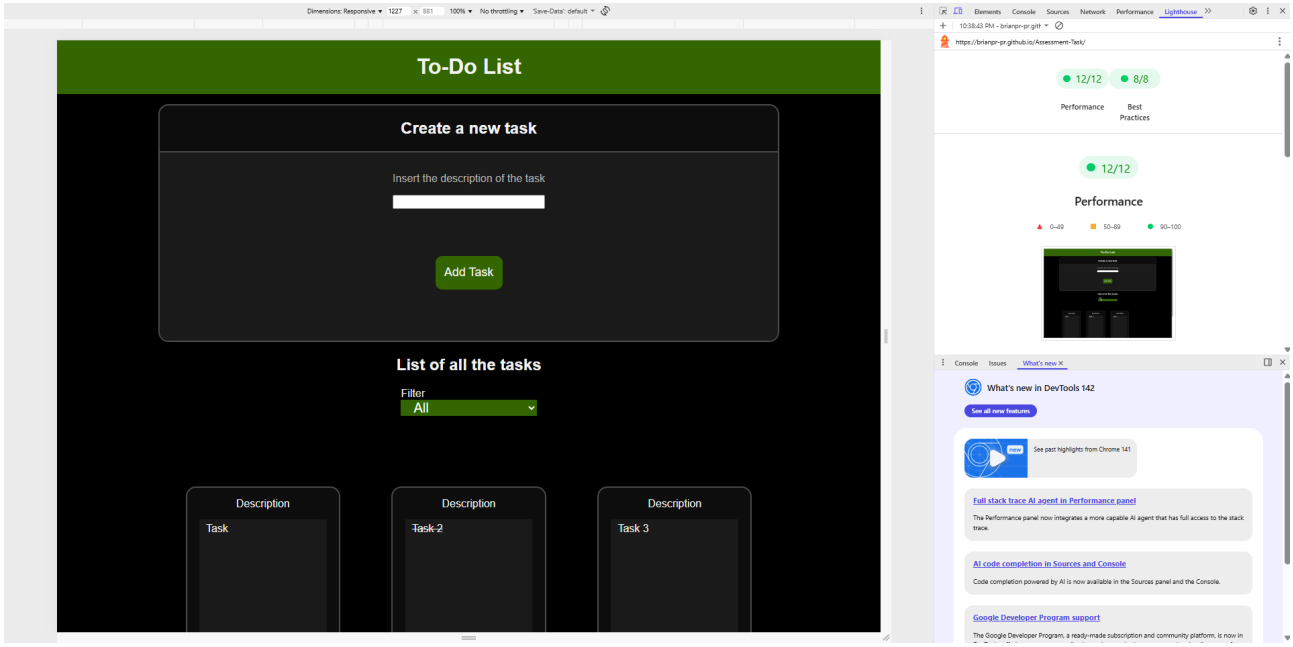
7. Final remarks

The web page passes all of the lighthouse tests:

- **Navigation (Default)**



- **Timespan**



- **Snapshot**