Hands-on Lab: Create a To-Do List Using JavaScript

Skills Network

Estimated time needed: 30 minutes

What you will learn

In this To-Do List application lab, you will explore the process of building a functional task management interface using HTML, CSS, and JavaScript. Follow step-by-step instructions to understand the implementation of fundamental features, including adding tasks, dynamically displaying a task list, marking tasks as completed, clearing completed tasks, and implementing live search functionality. Throughout this process, you will grasp key concepts such as DOM manipulation, event handling, array manipulation for task management, and filtering tasks based on user input.

Learning objectives

After completing this lab, you will be able to:

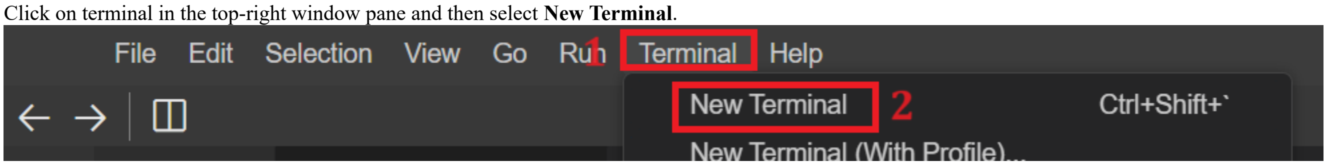
- **Task management implementation:** Learn the process of creating a functional task management interface by implementing features like adding tasks, displaying a task list dynamically, toggling task completion status, and clearing completed tasks using HTML, CSS, and JavaScript.
- **DOM manipulation proficiency:** Gain proficiency in manipulating the Document Object Model (DOM) using JavaScript to dynamically create and modify elements within the webpage, enabling real-time updates and interactions within the To-Do List application.
- **Event handling and user interaction:** Explore event-driven programming by implementing event listeners for user actions such as adding tasks, toggling completion, and filtering tasks based on search input, fostering a responsive and interactive user experience.
- **Understanding front-end principles:** Comprehend fundamental front-end development principles, including UI design considerations, CSS styling for layout and aesthetics, and integrating JavaScript functionalities to create a cohesive and user-friendly To-Do List application interface.

Prerequisites

- Basic knowledge of HTML.
- Basic understanding of arrays, arrays methods and properties, DOM manipulation.
- Web browser with a console (Chrome DevTools, Firefox Console, and so on).

Step 1: Setting up the environment

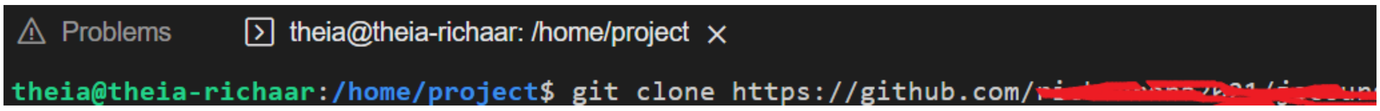
1. Firstly, you need to clone your main repository in **Skills Network Environmemnt**. Follow given steps:
   - Click on terminal in the top-right window pane and then select **New Terminal**.



   - Perform `git clone` command by writing the given command in the terminal.

```
git clone <github-repository-url>
```

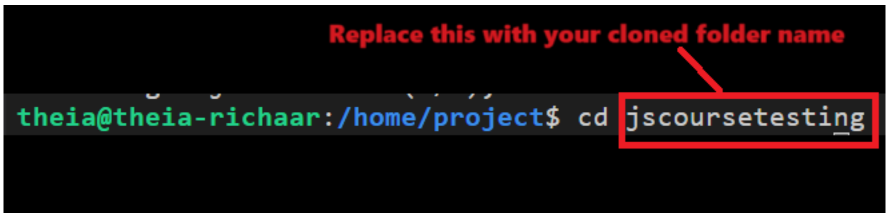
Note: Put your own GitHub repository link instead of `<github-repository-url>` which you created in first lab.



```
theia@theia-richaar:/home/project$ git clone https://github.com/[redacted]
```

   - Above step will clone the folder for your GitHub repository under project folder in explorer. You will also see multiple folders inside the cloned folder.
   - Now you need to navigate inside the cloned folder. For this write the given command in the terminal:

```
cd <repository-folder-name>
```



```
theia@theia-richaar:/home/project$ cd jscoursetesting
```

Note: Write your cloned folder name instead of `<repository-folder-name>`‘. Perform `git clone` if you have logged out of **Skills Network Environment** and you cannot see any files or folder after you logged in.

2. Now select **cloned Folder Name** folder, right-click on it and click on **New Folder**. Enter folder name as **TodoList**. It will create the folder for you. Then select **TodoList** folder, right-click and select **New File**. Enter file named as **todo_list.html** and click OK. It will create your HTML file.

3. Create template structure of a HTML file by adding the following content:

```
    <!DOCTYPE html>
    <html>
<head>
    <title>ToDo List</title>
</head>
<body>
    <h1>ToDo List</h1>
    <input type="text" id="taskInput" placeholder="Add a new task">
    <button id="addTaskBtn">Add Task</button>
    <!-- <input type="text" id="searchInput" placeholder="Search..."> -->
    <ul id="taskList">
        <!-- Tasks will be displayed here -->
    </ul>
    <button id="clearCompletedBtn">Clear Completed</button>
    </body>
    </html>
```

4. The above HTML code includes the given points:

- This code snippet represents a simple web page layout. It includes the following elements:
    - An `<h1>` heading indicates a "To-Do List." An `<input>` field with the ID taskInput allows users to input new tasks, and a button with the ID addTaskBtn is present to add these tasks.
    - Task list display: An unordered list `<ul>` with the ID taskList displays the tasks users add. Initially, it is empty and meant to populate as users add tasks.
    - Clear completed button: Lastly, a button with the ID clearCompletedBtn intended to clear completed tasks from the list.

Note: When you have pasted the code, save your file.

5. Now select **TodoList** folder again, right click and select **New File**. Enter the file named **todo_list.js** and click OK. It will create your JavaScript file.

6. To include js file in **todo_list.html**, You can use the script tag in the HTML file just above the `</body>` tag. You can use the given code to include and save the script file.

```
<script src="./todo_list.js"></script>
```

Step 2: Defining variables to access data

1. Declare a variable named **taskInput** and initialize it as follows:
    - Write the given code in the **todo_list.js** file.

    ```
    const taskInput = document.getElementById("taskInput");
    ```

    - This line retrieves the HTML element with the ID "taskInput" from the document, assigning it to the constant variable **taskInput**. It enables access to the input field where users can enter new tasks using JavaScript in the Todo List application.

2. Similarly, add given code in **todo_list.js** file:

```
const addTaskBtn = document.getElementById("addTaskBtn");
const taskList = document.getElementById("taskList");
const clearCompletedBtn = document.getElementById("clearCompletedBtn");
```

- In the above lines of code, document.getElementById() is used to retrieve specific elements from the HTML document by their unique IDs, such as:
    - addTaskBtn fetches the button element responsible for adding tasks.
    - taskList retrieves the unordered list element where tasks are displayed.
    - clearCompletedBtn accesses the button used to clear completed tasks.

3. Declare an empty array with a variable named **tasks**.

```
let tasks = [];
```

Step 3: Defining various functions to access data

1. Create the **addTask** function by including the given code in the JavaScript file.

```
function addTask() {
            const taskText = taskInput.value.trim();
            if (taskText !== "") {
                tasks.push({ text: taskText});
                taskInput.value = "";
                displayTasks();
            }
        }
```

- The above code includes:
    - **taskText** variable to retrieve the value entered into the taskInput HTML element by the user, trimming any trailing whitespace.
    - A conditional statement that uses an `if` block to check if the taskText is not an empty string; if not, it creates a new task object with the entered text.
    - Addition of this new task object using the `push` array method to the tasks array, representing the ToDo List.
    - Resetting the value of the taskInput field to an empty string after adding the task, clearing the input for the next task entry.
    - Calling the **displayTasks** function to display entered todo tasks, which you will create in the next step.

2. Create the **displayTasks** function by including the given code in the JavaScript file.

```
function displayTasks() {
            taskList.innerHTML = "";
            tasks.forEach((task, index) => {
                const li = document.createElement("li");
                li.innerHTML = `<input type="checkbox" id="task-${index}" ${task.completed ? "checked" : ""}>
```

```
                <label for="task-${index}">${task.text}</label>`;
            li.querySelector("input").addEventListener("change", () => toggleTask(index));
            taskList.appendChild(li);
        });
    }
```

- The above code includes the following:
  - `taskList.innerHTML = "";` to clear the existing content within the taskList element by setting its innerHTML to an empty string.
  - `tasks.forEach` iterates through the tasks array using forEach, creating a list item `<li>` for each task.
  - It constructs HTML content for each task by assigning it to `li.innerHTML`, which includes a checkbox, a label displaying the task text, and corresponding IDs.
  - Then, with the help of `li.querySelector`, it sets up an event listener for each checkbox within the task list `<li>` element. When the checkbox state changes, it triggers the toggleTask() function, which you will create in the next step.
  - Then appends the newly created list item containing the task details in the To-Do List interface using the `appendChild` method.

3. Create the **toggleTask** function and include the given code:

```
function toggleTask(index) {
    tasks[index].completed = !tasks[index].completed;
    displayTasks();
}
```

  - This **toggleTask** function toggles the completion status of a specific task in the tasks array based on the provided index.
  - It helps by selecting the checkbox regardless. If selected, then it will mark that particular task as completed.
  - For this, you need to call one more function called the **clearCompletedTasks** function.

4. Create a **clearCompletedTasks** function:

```
function clearCompletedTasks() {
        tasks = tasks.filter(task => !task.completed);
        displayTasks();
    }
```

  - In the above code, the filter method filters the task array, which has the list of tasks entered by users.
  - `tasks.filter(task => !task.completed);` code filters the tasks array to retrieve only the tasks that are not marked as completed (task.completed is false), returning a new array excluding completed tasks.

5. Perform addEventListener for addTask and clearCompletedTasks buttons to listen for clicks after clicking the **Add Task** and **Clear Completed** buttons.

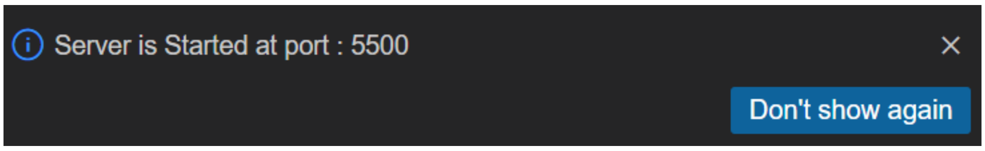
```
addTaskBtn.addEventListener("click", addTask);
clearCompletedBtn.addEventListener("click", clearCompletedTasks);
```

6. The function calls the displayTasks function to show the entered todo task after clicking the **Add Task** button.

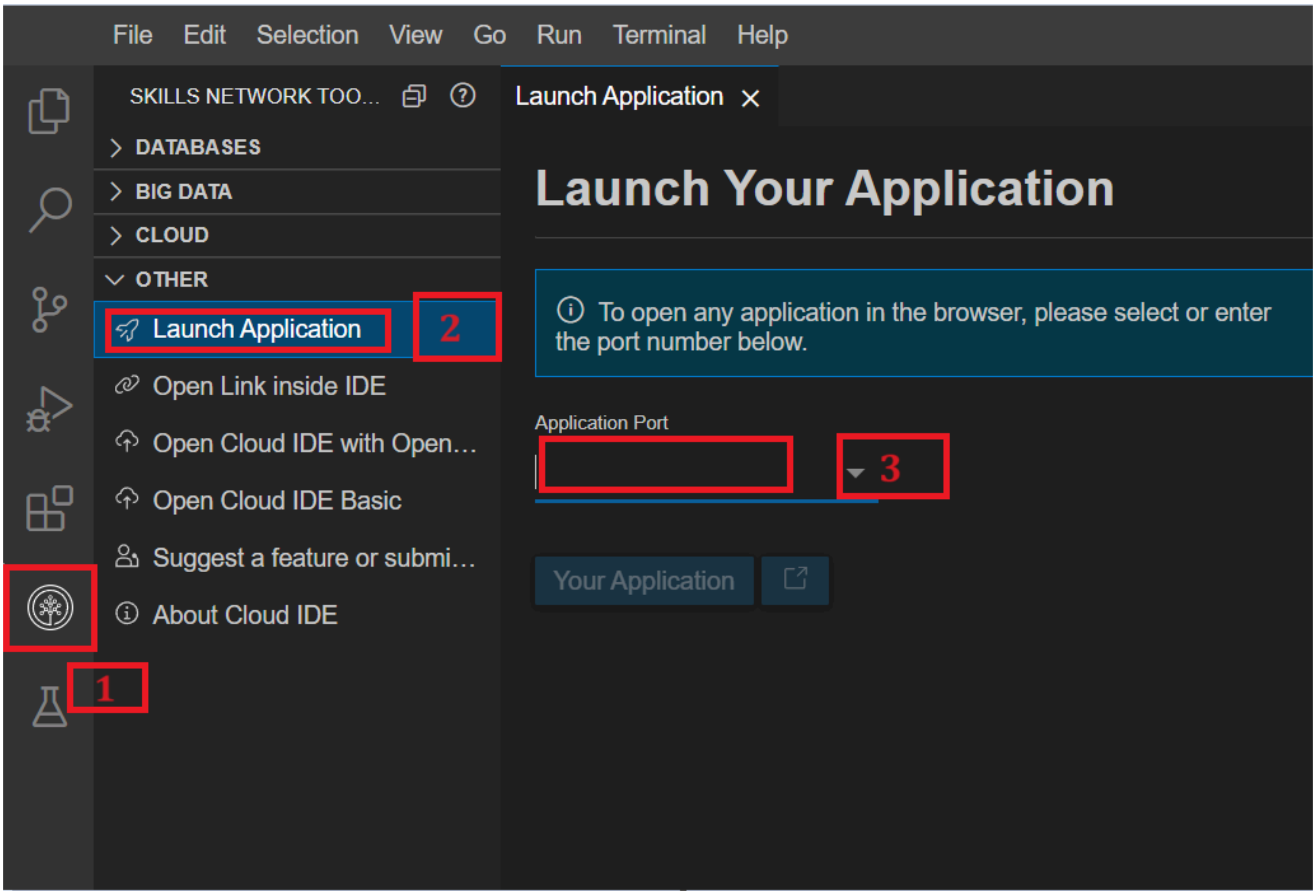
```
displayTasks();
```

Step 4: Check the output

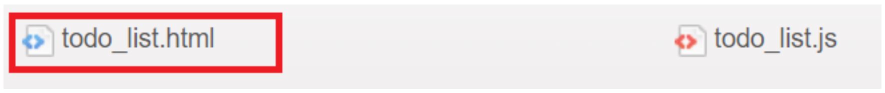
1. To view your HTML page, right-click the **todo_list.html** file after selecting this file, then select "Open with Live Server."
   - The server should start on port 5500, indicated by a notification on the bottom right side.


Server is Started at port : 5500

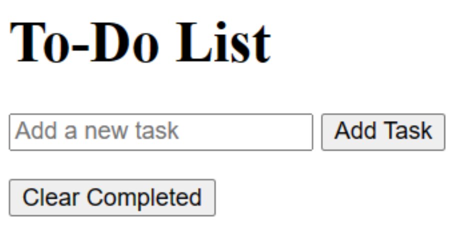
2. Click the Skills Network button on the left (refer to number 1). It will open the "Skills Network Toolbox". Next, click "Launch Application" (refer to number 2). From there, you enter port number 5500 at number 3 and click this button .



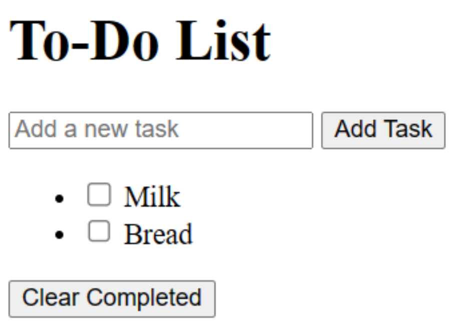
3. It will open your default browser where you will see **cloned-folder-name** folder name. Click on that **cloned-folder-name** folder name. After clicking you will see multiple folders name, among those folders name click on **TodoList** folder. You will see files related to this folder where again you will click on **todo_list.html** file as shown below.



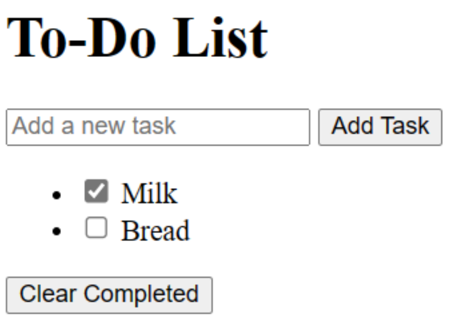
3. It will open the HTML page and show the output as shown below:



4. Then, enter any task in the input box and click the **Add Task** button; you will see the task on the front page as shown below.



5. You can then select that particular task by clicking the check box and the **Clear Completed** button. You will see that the task is no longer visible on the front page.



To-Do List

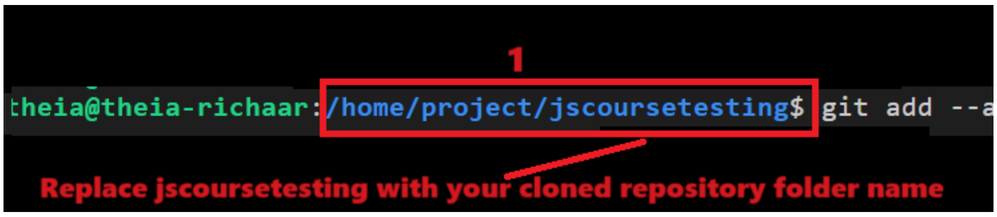
- ☑ Milk
- ☐ Bread

Step 5: Perform Git commands

1. Perform `git add` to add the latest files and folder in the git environment.

   ```
   git add --a
   ```

   - Make sure the terminal has the path as follows:



```
theia@theia-richaar:/home/project/jscoursetesting$ git add --a
```

Replace jscoursetesting with your cloned repository folder name

2. Then perform `git commit` in the terminal. While performing `git commit`, terminal can show message to set up your `git config --global` for user.name and user.email. If yes, then you need to perform `git config` command as well for `user.name` and `user.email` as given below.

   ```
   git config --global user.email "you@example.com"
   ```

   ```
   git config --global user.name "Your Name"
   ```

   Then perform git commit as given:

   ```
   git commit -m "message"
   ```

3. Next, perform `git push` just by writing given command in terminal.

   ```
   git push origin
   ```

   - After the push command, the system will prompt you to enter your username and password. Enter the username for your GitHub account and the password that you created in the first lab. After entering the credentials, all of your latest folders and files will be pushed to your GitHub repository.

Practice task

1. In this practice task, you need to include a button to clear all tasks at once.

2. For this you need to create a button named **Clear All Tasks** in the HTML file.

3. Then create a function to clear the tasks list, which you have stored in **tasks** array in lab, and call this function at the time the **Clear All Tasks** button is clicked.

   Hint: you can empty the entire array when user clicks on the button.

Don't forget to perform `git add`, `git commit` and `git push` commands again after completing the task.

Summary

1. **HTML structure:** Created essential HTML elements like title, To-Do List heading, input field, and buttons for adding tasks and clearing completed ones. Set up an empty list to display tasks.

2. **Defined variables:** To capture HTML elements using a document.getElementById variables were declared. Initialized an empty array of tasks to manage tasks.

3. **Task management:** addTask gathered user input, updated tasks, and refreshed the displayed list through displayTasks. This function dynamically generated HTML elements for each task. toggleTask and clearCompletedTasks managed task completion and removal.

4. **Event handling:** Set event listeners for the "Add Task" and "Clear Completed" buttons, triggering respective functions. Initially displayed existing tasks through displayTasks. This setup formed the foundation for a dynamic To-Do List, allowing smooth task addition, viewing, completion, and clearing.