TASK 1: DESIGN AND DEVELOPMENT OF A WEB APPLICATION (TASK MANAGER)

(Final Phase Report)

Module: Software Engineering (DLMCSPSE01)

Table of Contents

Abstract

The task manager is a coherent web application that is lightweight, responsive, and a productivity application that assists the user in managing daily activities using a simple and user-friendly interface. This application has enabled the user to add, edit, and delete tasks as well as search the tasks with priorities and due dates. Also, the application has a Dark Mode that improves the user interface and ease of access. The main agenda of developing this project is to develop a client-side application with the help of HTML, CSS, and JavaScript, which helps to locally store the data via Web Storage API (LocalStorage), as well as not depend on the backend server. This final phase report is established to deliver a coherent overview of the implementation process, technical design, testing process, lessons learned and improvements that can be made. The application has been effective in illustrating how to use the concepts of DOM manipulation, local storage persistence and responsive design; thus, it is a viable and scalable solution for task organisation and time management.

1. Introduction

In this project, the researcher develops a browser-based web application whose name is Task Manager. This application has helped in planning, tracking, and managing users' daily activities. The application is developed using HTML5, CSS3, and JavaScript (ES6), and the system helps users add, delete, and search tasks. Also, double-clicking on the task confirms that the task is complete and local storage has been used to save the data, as well as maintaining data persistence. This final phase report is developed to show the detailed implementation, development strategies and testing applicability of the application. The project is expected to illustrate the principles of software engineering with the help of a functional, user-friendly, and attractive web application. It was stressed on simplicity, responsiveness, and maintainability so that users should be able to manage tasks intuitively and developers should be able to extend or maintain the system easily in order to enhance it in the future.

2. System Design and Architecture

It has been underscored from the previous discussion such as which is established on the conception phase and development phase, that the Task Manager Web Application has been designed with the help of a client-side, single-page architecture that is all developed using the core web technologies of HTML, CSS and JavaScript. The simplicity, lightweight, and compatibility with modern browsers without having to depend on external frameworks and back-end servers are guaranteed by this structure.

2.1 Architecture Overview

2.1.1 Frontend Layer

The user interface is designed with the help of HTML that identifies all the necessary items like text input fields, buttons, task lists, and priority selectors. CSS has helped to accelerate the display in terms of layout, spacing, colour scheme and cogent visual transitions. It is also responsive and offers accessibility in dark mode.

2.1.2 Logic Layer

For the logic layer, the researcher has been using JavaScript, which has helped to handle the dynamic functionality of the application. It deals with the interactions with the users, including adding, deleting, editing and completing tasks and confirms that the updates are reflected immediately in the DOM.

2.1.3 Data Layer

The researcher has used Web Storage API, such as Local Storage, for maintaining data persistence. With the help of this method, it is possible to save the tasks in the browser as well as confirm that data will not be dependent on the server even after closing the browser or rebooting their computer, and the data will still be accessible.

2.2 Key Components

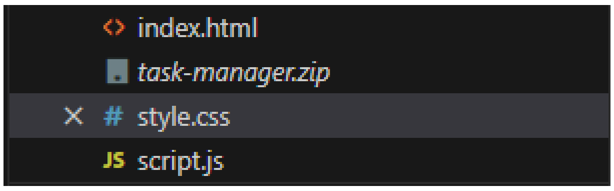


Figure 1: Key Components

index.html

- It has helped to manage layout and input structure.

style.css

- Visual styling, priority indicators (red, orange, green), and dark mode themes.

script.js

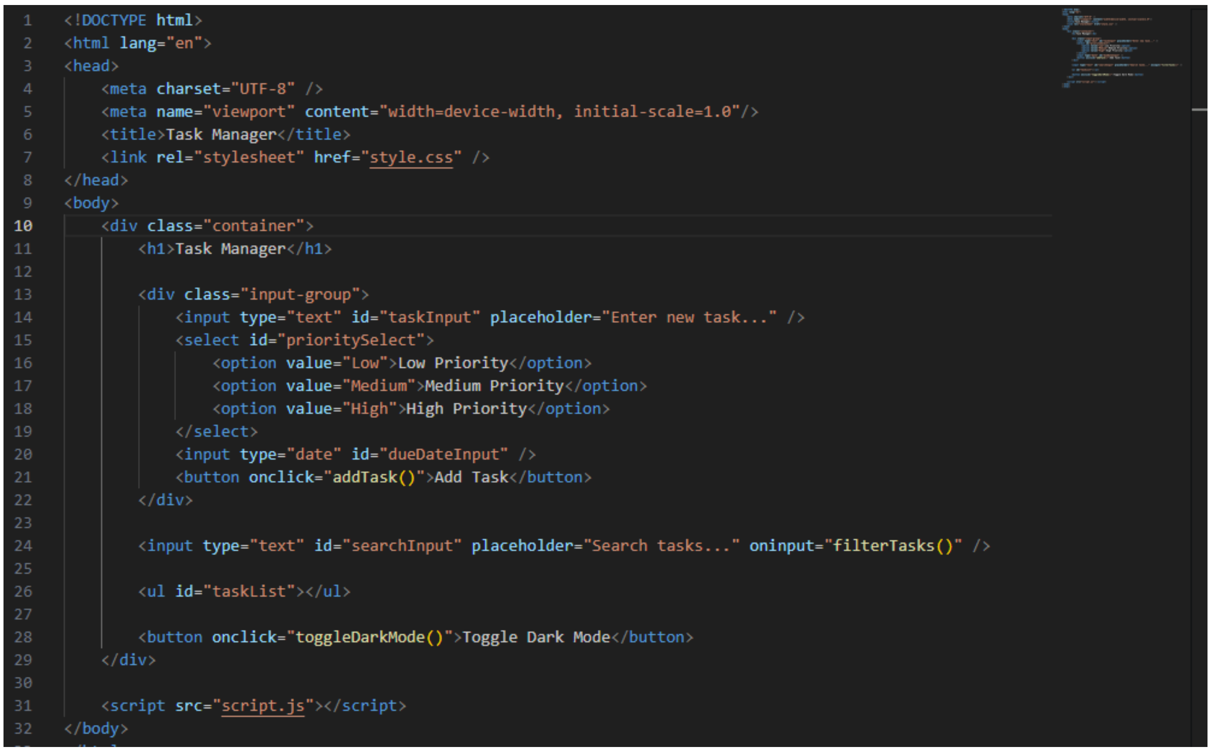
- Contains all event-driven logic, task management, and synchronisation with Local Storage.

2.3 Flow

Upon loading the page, the tasks that were stored in the Local Storage are read. Users are able to create, edit or delete tasks, and each update causes the saveTasks() function to be used to save the tasks. The functions filterTasks and toggleDarkMode increase the usability and accessibility. This coherent design confirms maintainability, scalability, as well engaging user experience.

3. Implementation Details

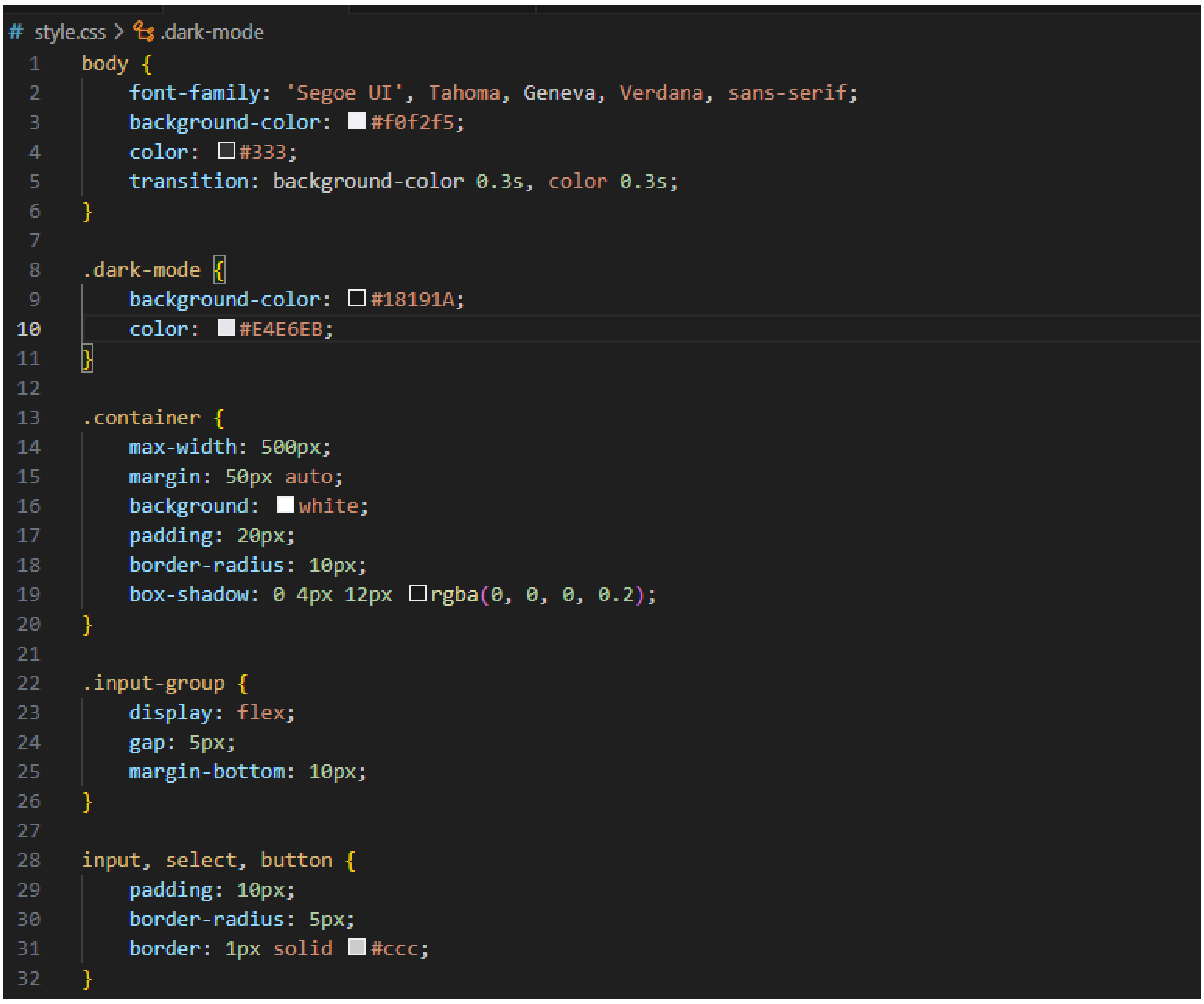
3.1 Implementation of HTML, CSS and JavaScript



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
    <title>Task Manager</title>
    <link rel="stylesheet" href="style.css" />
</head>
<body>
    <div class="container">
        <h1>Task Manager</h1>

        <div class="input-group">
            <input type="text" id="taskInput" placeholder="Enter new task..." />
            <select id="prioritySelect">
                <option value="Low">Low Priority</option>
                <option value="Medium">Medium Priority</option>
                <option value="High">High Priority</option>
            </select>
            <input type="date" id="dueDateInput" />
            <button onclick="addTask()">Add Task</button>
        </div>

        <input type="text" id="searchInput" placeholder="Search tasks..." oninput="filterTasks()" />

        <ul id="taskList"></ul>

        <button onclick="toggleDarkMode()">Toggle Dark Mode</button>
    </div>

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

Figure 2: Index.HTML file

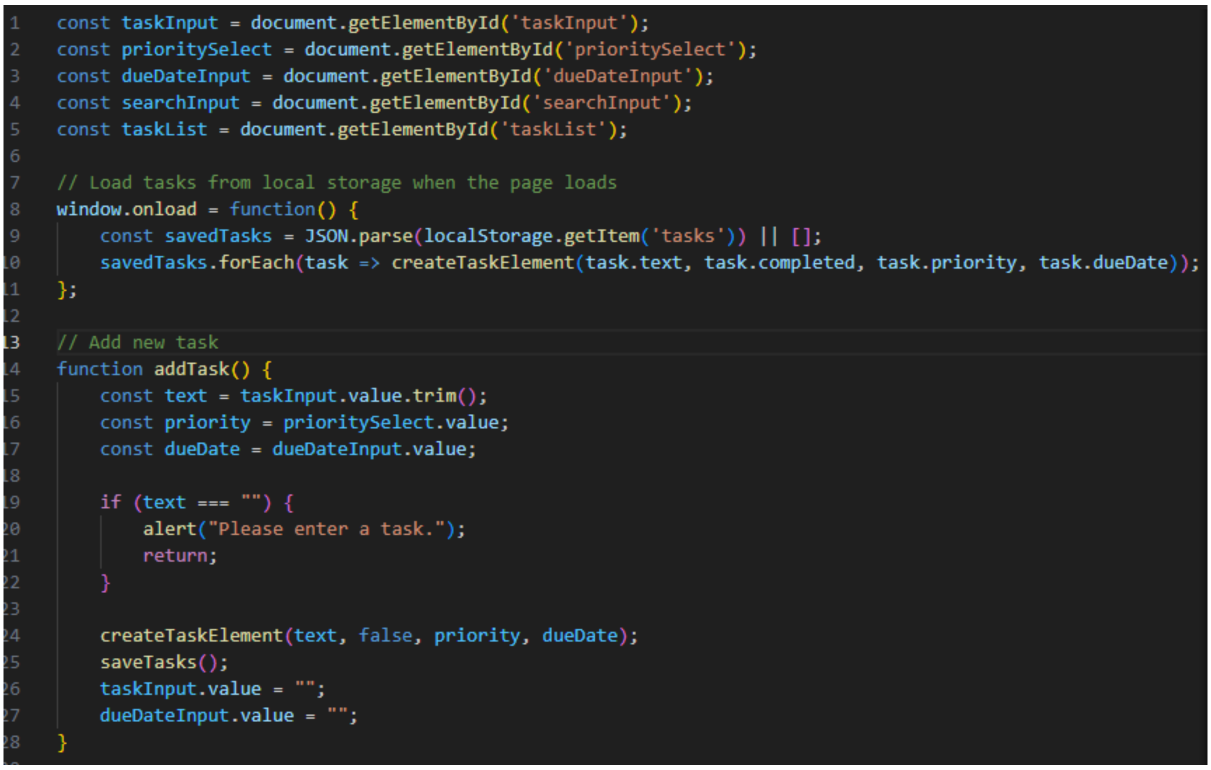
It has been underscored from the above figure (2) that the researcher develops an HTML file to structure the Task Manager web application. Here, the researcher uses input fields for adding tasks, selecting priority, and setting due dates. Also, it has been seen from Figure 2 that the researcher creates a search bar to help users find the task, as well as buttons to help add tasks and toggle dark mode. Also, the task list has been shown with the help of JavaScript, which has helped to develop an interactive, user-friendly task management interface.


style.css > .dark-mode

```css
body {
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    background-color: #f0f2f5;
    color: #333;
    transition: background-color 0.3s, color 0.3s;
}

.dark-mode {
    background-color: #18191A;
    color: #E4E6EB;
}

.container {
    max-width: 500px;
    margin: 50px auto;
    background: white;
    padding: 20px;
    border-radius: 10px;
    box-shadow: 0 4px 12px rgba(0, 0, 0, 0.2);
}

.input-group {
    display: flex;
    gap: 5px;
    margin-bottom: 10px;
}

input, select, button {
    padding: 10px;
    border-radius: 5px;
    border: 1px solid #ccc;
}
```

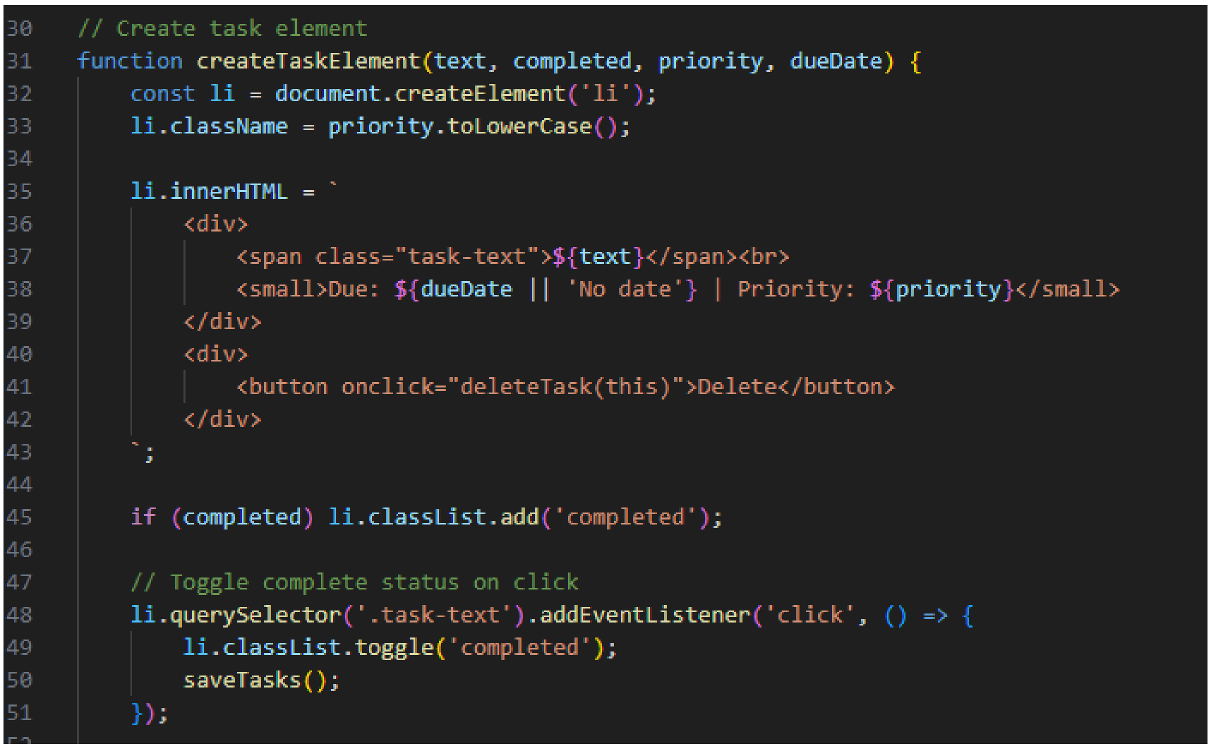
Figure 3: Style.CSS

The Task Manager has a clean, responsive, and user-friendly interface as a result of the CSS styling. It employs the mild palette of colours, smooth transitions, and clear typography to read. Priorities are colour-coded (red (high), orange (medium), and green (low)) to identify the tasks conveniently. The design has hover effects, rounded elements and a dark mode option, which improves usability and appeal to various devices and screen sizes.



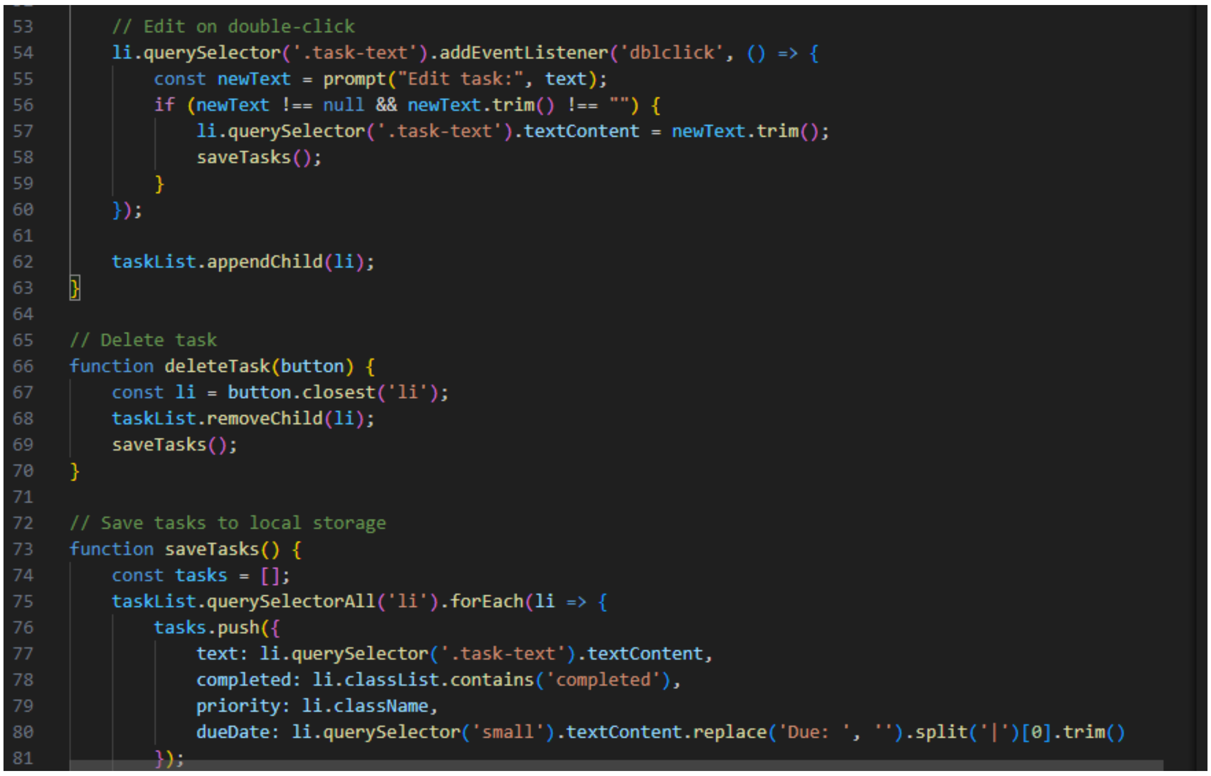
```
const taskInput = document.getElementById('taskInput');
const prioritySelect = document.getElementById('prioritySelect');
const dueDateInput = document.getElementById('dueDateInput');
const searchInput = document.getElementById('searchInput');
const taskList = document.getElementById('taskList');

// Load tasks from local storage when the page loads
window.onload = function() {
    const savedTasks = JSON.parse(localStorage.getItem('tasks')) || [];
    savedTasks.forEach(task => createTaskElement(task.text, task.completed, task.priority, task.dueDate));
};

// Add new task
function addTask() {
    const text = taskInput.value.trim();
    const priority = prioritySelect.value;
    const dueDate = dueDateInput.value;

    if (text === "") {
        alert("Please enter a task.");
        return;
    }

    createTaskElement(text, false, priority, dueDate);
    saveTasks();
    taskInput.value = "";
    dueDateInput.value = "";
}
```

Figure 4: Load tasks from local storage when the page loads, and add a new task (script.js)



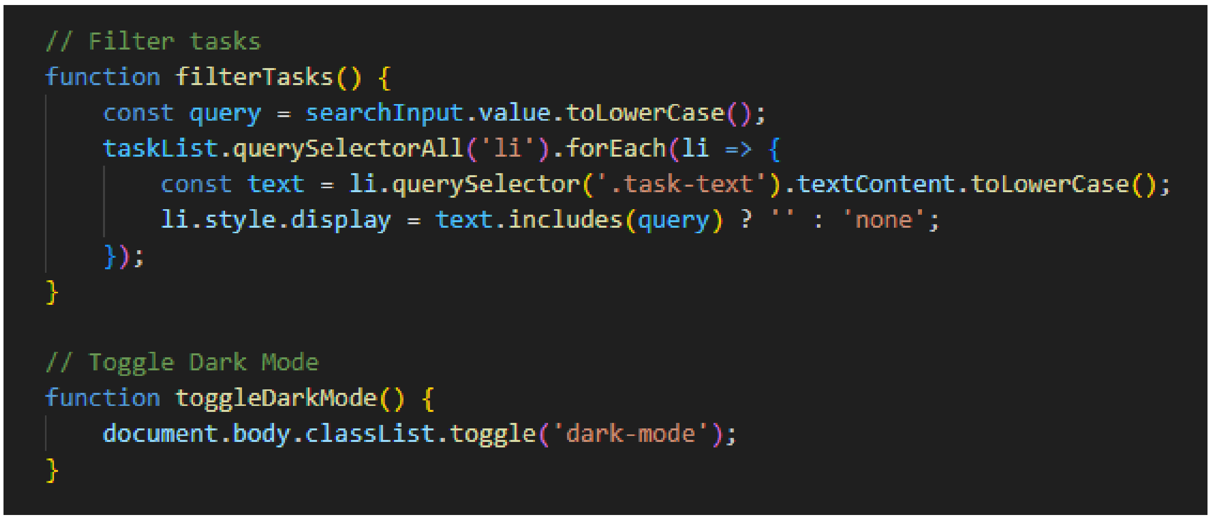
```
// Create task element
function createTaskElement(text, completed, priority, dueDate) {
    const li = document.createElement('li');
    li.className = priority.toLowerCase();

    li.innerHTML = `
        <div>
            <span class="task-text">${text}</span><br>
            <small>Due: ${dueDate || 'No date'} | Priority: ${priority}</small>
        </div>
        <div>
            <button onclick="deleteTask(this)">Delete</button>
        </div>
    `;

    if (completed) li.classList.add('completed');

    // Toggle complete status on click
    li.querySelector('.task-text').addEventListener('click', () => {
        li.classList.toggle('completed');
        saveTasks();
    });
```

Figure 5: Create a task element and Toggle



```
    // Edit on double-click
    li.querySelector('.task-text').addEventListener('dblclick', () => {
        const newText = prompt("Edit task:", text);
        if (newText !== null && newText.trim() !== "") {
            li.querySelector('.task-text').textContent = newText.trim();
            saveTasks();
        }
    });

    taskList.appendChild(li);
}

// Delete task
function deleteTask(button) {
    const li = button.closest('li');
    taskList.removeChild(li);
    saveTasks();
}

// Save tasks to local storage
function saveTasks() {
    const tasks = [];
    taskList.querySelectorAll('li').forEach(li => {
        tasks.push({
            text: li.querySelector('.task-text').textContent,
            completed: li.classList.contains('completed'),
            priority: li.className,
            dueDate: li.querySelector('small').textContent.replace('Due: ', '').split('|')[0].trim()
        });
```

Figure 6: Edit on double-click, Delete task, Save tasks to local storage



```
// Filter tasks
function filterTasks() {
    const query = searchInput.value.toLowerCase();
    taskList.querySelectorAll('li').forEach(li => {
        const text = li.querySelector('.task-text').textContent.toLowerCase();
        li.style.display = text.includes(query) ? '' : 'none';
    });
}

// Toggle Dark Mode
function toggleDarkMode() {
    document.body.classList.toggle('dark-mode');
}
```

Figure 7: Filter tasks and Toggle Dark Mode

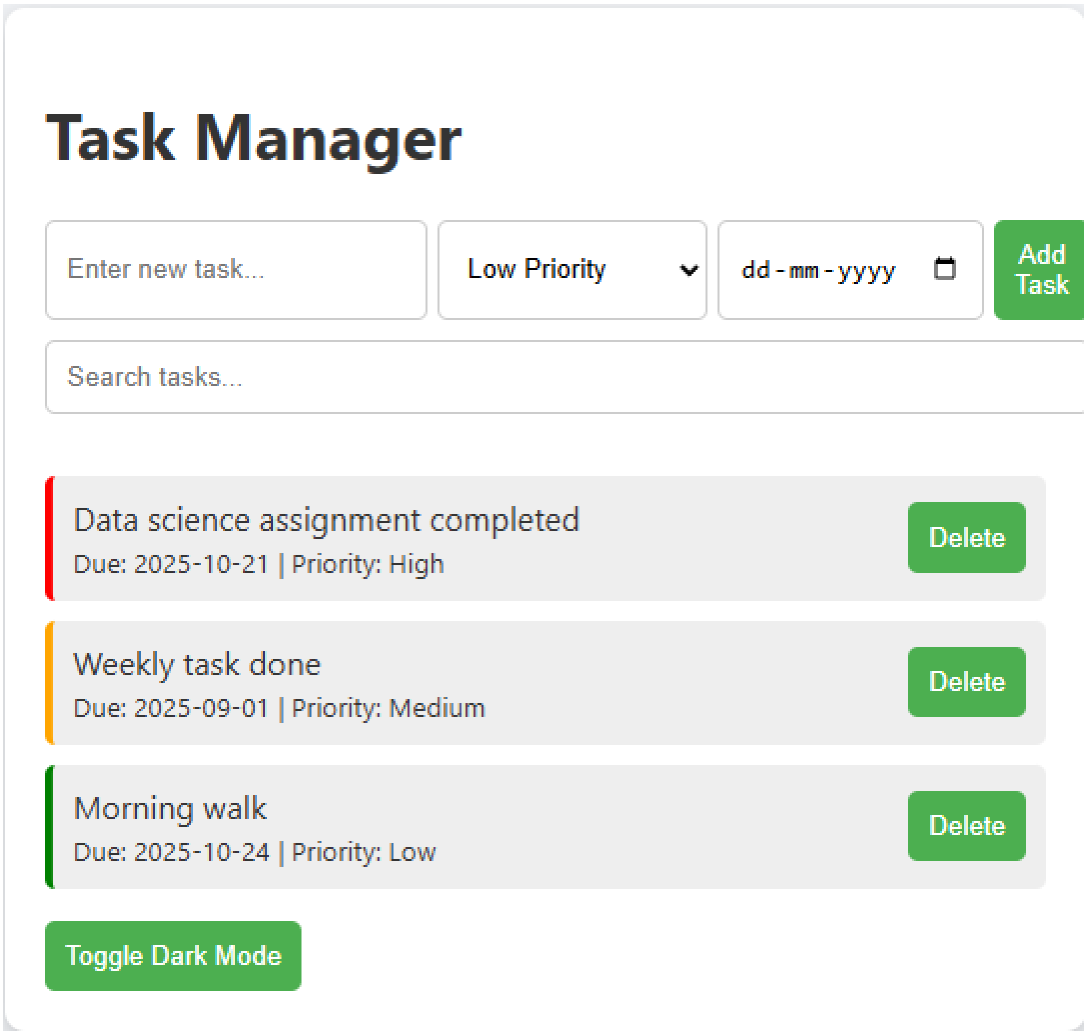
It has been underscored from the aforementioned figure (4,5,6, and 7) that the use of JavaScript the researcher has accelerated the functionality of the Task Manager's application via DOM manipulation and local storage integration. It enables users to add, complete and delete tasks, as well as search them. Every task has a name, priority, and due date, and all of them are stored in the Local Storage of the browser to become persistent. Event listeners make it interactively; when one clicks, a task is complete, and when one clicks two times, one can edit it. The saveTasks () is used to update the stored data with each change made. A filter feature (filterTasks()) offers real-time search, and a toggle of DarkMode () offers the light and dark themes. It can be concluded that the script delivers a coherent user interaction and the ability to manage tasks effectively without being dependent on the backend.

3.2 Tools Used

- Visual Studio Code (IDE)
- Google Chrome Developer Tools (for debugging and testing)
- GitHub

4. Testing and Evaluation

4.1 Testing


Task Manager

Enter new task... | Low Priority | dd-mm-yyyy | Add Task

Search tasks...

Data science assignment completed
Due: 2025-10-21 | Priority: High
Delete

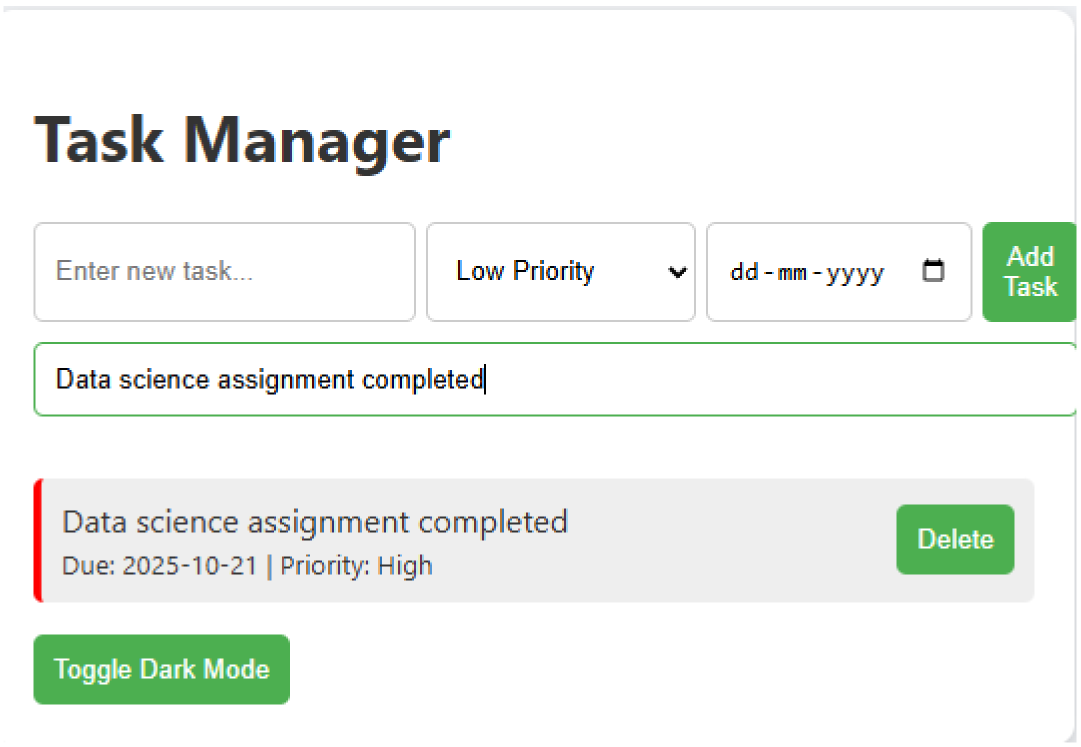
Weekly task done
Due: 2025-09-01 | Priority: Medium
Delete

Morning walk
Due: 2025-10-24 | Priority: Low
Delete

Toggle Dark Mode

Figure 8: User interface with testing by adding various tasks

The accurate task management functionality has been underscored by the user interface. Here, the researcher added tasks with various priorities, such as high, medium, and low, as well as the colour-coded has help with recognition. The successful addition of tasks with corresponding priorities states the application's correctness.


Task Manager

Enter new task...

Low Priority

dd-mm-yyyy

Add Task

Data science assignment completed

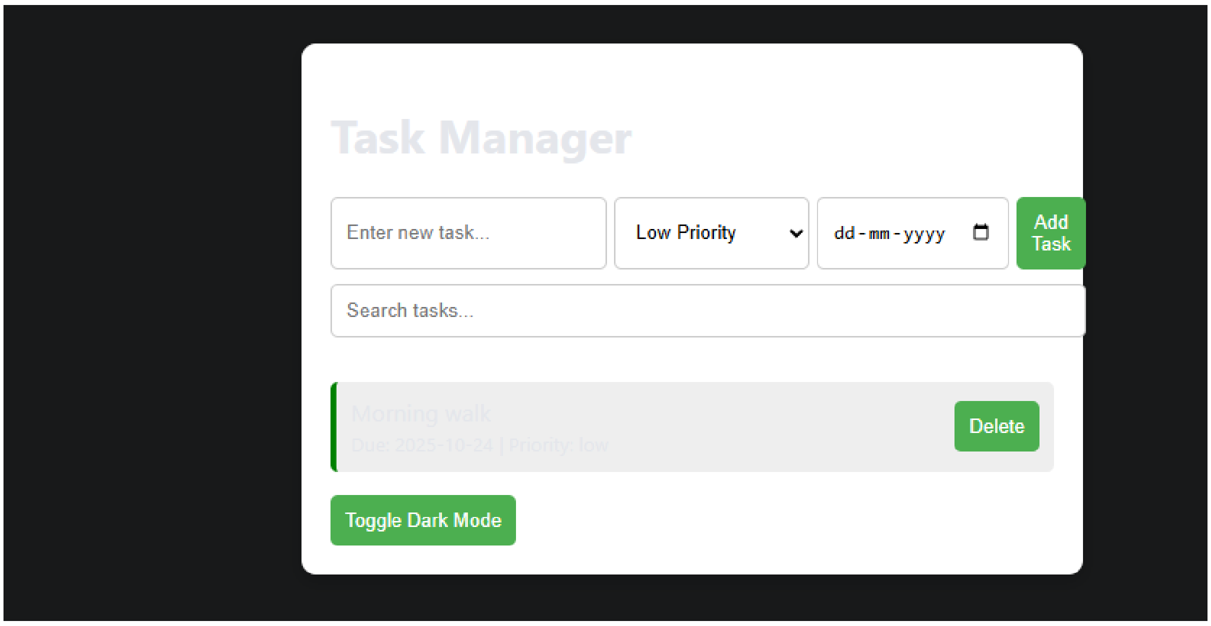
Data science assignment completed
Due: 2025-10-21 | Priority: High

Delete

Toggle Dark Mode

Figure 9: Testing by searching tasks

The above figure (9) tests the application search functionality, such as users can easily filter and locate tasks according to text input, as well as confirming coherent task management.


Task Manager

Enter new task...

Low Priority

dd-mm-yyyy

Add Task

Search tasks...

Morning walk
Due: 2025-10-24 | Priority: low

Delete

Toggle Dark Mode

Figure 10: Test Toggle Dark mode

It is underscored from the figure (10) that the interface switching between light and dark themes boosts accessibility and user comfort in various lighting environments.

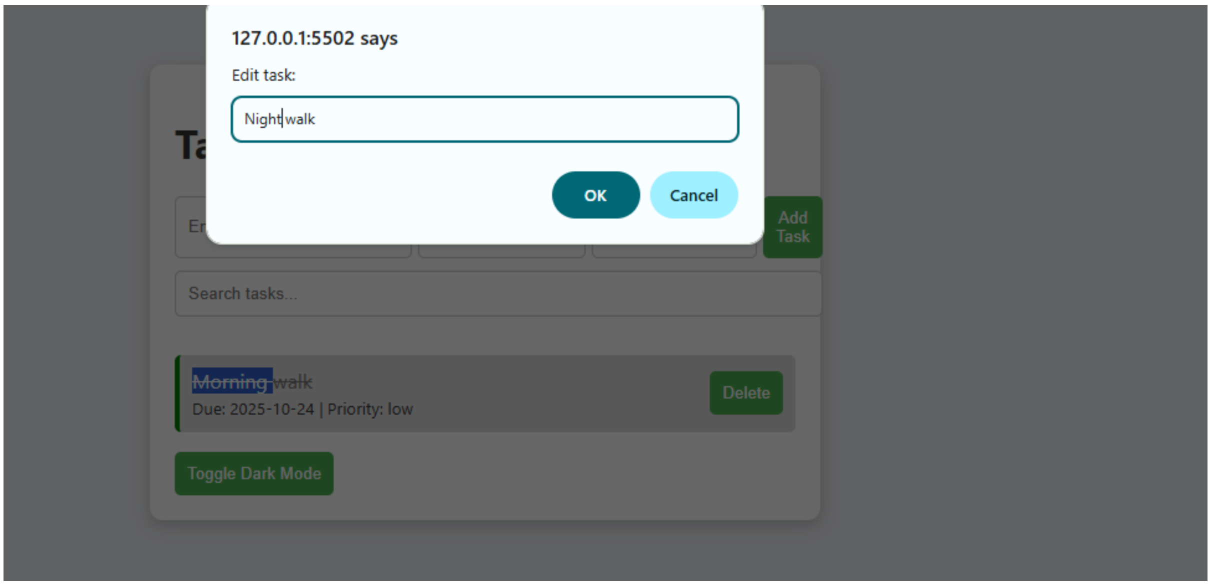


Figure 11: Test for Edit task

The above figure (11) shows the testing of the editing feature, which helps the user to edit the existing task, such as double-clicking on the task; this editing option has appeared.

4.2 Evaluation

Test Case	Expected Result	Status
Add task with all fields filled	Task appears in the list	Passed
Edit existing task	Updates reflected in UI	Passed
Delete task	Task removed from list & LocalStorage	Passed
Refresh page	Data persists	Passed
Search query	Tasks filtered dynamically	Passed
Dark mode toggle	Colour scheme changes	Passed

Table 1: Test evaluation

5. Lessons Learned

In designing the Task Manager web application, various effective lessons were learnt that enhanced technical and conceptual knowledge. The exercise improved the mastery of the DOM manipulation that enabled it to efficiently select and update HTML elements using pure JavaScript without any external libraries. The focus on modular coding with well-designed functions, such as addTask and saveTasks, enhanced the readability, maintainability, and debugging efficiency. Local Storage has been used to show how data persistence at the client-side can successfully substitute lightweight databases in small applications. The balance of designing based on the UI/UX ensured a clean, responsive, user-friendly interface, which enhanced accessibility and interaction. Also, browser-based testing demonstrated the importance of the presence of cross-browser compatibility and responsive design performance. It can be concluded, this project has given a practical understanding of the concepts of front-end development, event-driven programming and how to integrate storage mechanisms into a cohesive entity, which led to a greater insight into how to make web applications efficient, maintainable, and visually attractive.

6. Challenges and Solutions

There were a number of difficulties that were experienced during the development process, which had to be debugged and solved through iterations. The initial problem was the data persistence, where the tasks could not be saved properly after editing them because the Local Storage was not synchronised with the updates made to the DOM. This was resolved out by calling the saveTasks() function after each modification of the DOM, so that the data would be saved in a similar way. The second problem was that the inconsistency of style when switching between light and dark modes created contrast and readability problems. In a bid to correct this, CSS variables were added to the text and background colour, which confirms equal visibility in both themes. The third issue was edit functionality, where the priority and due date fields would be reset in case of an edit. This was solved by optimising the JSON format by adding all the necessary features of the task and re-rendering them correctly on page reload. The issues of overcoming these challenges helped to improve the debugging, problem-solving, and interpretation of the principles of event handling, data management, and responsive interface design.

7. Future Enhancements

Although the existing solution is fully satisfactory in terms of core requirements, future enhancements can be achieved through various kinds of methods, which are outlined in the section below.

Cloud Synchronization

- The multi-device approach is to store tasks using Firebase or a Node.js backend.

Notifications

- Adding reminders for upcoming due dates.

Category Tagging

- It should allow categorising tasks in projects or subjects.

Drag-and-Drop Sorting

- Enhancing user interaction.

Progress Statistics

- Bar charts of the completion rate of tasks.

These improvements would make the Task Manager more of an active and user-friendly productivity package.

8. Conclusion

It can be concluded that the Task Manager application is a coherent web application that delivers an accurate, user-friendly task organisation application based on web technology. The project shows the incorporation of the responsive design, persistent storage, and accessible UI principles. This Final Phase is the completion of the conceptual design, development and implementation into a product in the real world. This experience provided technical knowledge of HTML, CSS and JavaScript, as well as hands-on experience of debugging, documentation and deployment. Also, this project stands as a coherent milestone in front-end web development mastery.

References

Mozilla Developer Network (MDN). *Web Storage API Documentation*. https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API

W3Schools. *JavaScript DOM Tutorial. https://www.w3schools.com/js/js_htmldom.asp*

FreeCodeCamp. *Local Storage and Web Storage Explained. https://www.freecodecamp.org/news/web-storage-localstorage-vs-sessionstorage-in-javascript/*

CSS-Tricks. *Implementing Dark Mode in CSS and JS. https://css-tricks.com/come-to-the-light-dark-side/*

Appendix: GitHub Link

https://github.com/Hriday-Dewan-10246379/Dewan-Hriday-10246379-PSE-DLMCSPSE01