Task Manager Code Explanation

This Document Provides a detail explanation of the Task Manager web application code. The application is a simple but complete task managment system built with HTML, CSS, and JavaScript

Table of Contents

1. HTML Structure
2. CSS Structure
3. JavaScript Functionality
4. Data Management
5. Event Handling

HTML structure

The HTML structure defines the user interface of the Task Manager Application:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Task Manager Using local storage</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <!--Outside container with classname container-->
    <div class="container">
        <h1> Task Manager</h1>
        <!--input section-->
        <div class="input-section">
            <input type="text" id="task-input" placeholder="Add a new task..." />
            <button id="add-button">Add</button>
        </div>
        <!-- task list-->
        <ul id="task-list" class="task-list">
            <!-- Tasks will be added here dynamically -->
        </ul>
        <!-- No task -->
            <div id="no-tasks" class="no-tasks">
                No tasks yet! Add a task to get started
            </div>
            <!-- Status bar -->
            <div class="status-bar">
                <span id="tasks-count">Total 0 tasks</span>
                <span id="completed-count">Completed: 0</span>
            </div>

            <!-- button to clear all tasks -->
            <button id="clear-all" class="clear-all">Clear All Tasks</button>

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

Key HTML Compontents:

- A container `div` that wraps the entire application
- A heading that displays the title
- An input section with a text field and an "Add " button
- An empty unordered list (`ul`) where tasks will be displayed
- A message that shows when there are no tasks
- A status bar showing task counts
- A "Clear All Tasks" button

CSS Styling

The CSS defines the visual appearence of the Task Manager

```
/* Basic Reset */
*{
    margin: 0;
    padding: 0;
    box-sizing: border-box;
    font-family: Arial, Helvetica, sans-serif;
}

body{
    background-color: #f5f5f5;
    padding: 20px;
}
.container{
    max-width: 600px;
    margin: 0 auto;
    background-color: white;
    border-radius: 8px;
```

```
        padding: 20px;
    }
    h1{
        text-align: center;
        margin-bottom: 20px;
        color: #333;
    }
    .input-section{
        display: flex;
        margin-bottom: 20px;
    }

    #task-input{
        flex: 1;
        padding: 10px;
        font-size: 16px;
    }
    #add-button{
        color: white;
        background-color: #4caf04;
        cursor: pointer;
        font-size: 16px;
    }

    .task-list{
        list-style-type: none;
    }

    .task-item{
        display: flex;
        justify-content: space-between;
        align-items: center;
        padding: 12px;
        margin-bottom: 8px;
    }
    .task-text{
        flex:1;
        margin-left: 10px;
    }

    .clear-all{
        display: block;
        width: 100%;
        padding: 10px;
        margin-top: 20px;
        background-color: #ff9800;
        color: white;
        cursor: pointer;
        font-size: 16px;
```

```
}

.no-tasks{
    text-align: center;
    color: #888;
    font-size: italic;
    padding: 20px;
}

.status-bar{
    display: flex;
    justify-content: space-between;
    margin-top: 15px;
    padding-top: 15px;
    font-size: 14px;
    color: #666;

}

.delete-btn
{
    background-color: #f44336;
    color: white;
    cursor: pointer;
    margin-left: 10px;

}

.completed{
    text-decoration: line-through;
    color: #888;

}
```

Key CSS features:

1. **Reset Styles** : Sets default margins, padding, and box-sizing for all elements.
2. **Container styling** : Creates a centered, white card with rounded corners and subtle shadow.
3. **Input section** : uses flexbox to position the input field and **ADD** button side by side.
4. **Task items**: Styles each task with background color, spacing and flexbox layout.
5. **Button styles**: Defines apperance for ADD, Delete and Clear All buttons with hover effects
6. **Status indicators**: Styles for completed tasks (strikethrough) and status bar for tasks count.
7. **Responsive Design** : Uses relative units and max-width to ensure responsive behavior.

JavaScript Functionality

The JavaScript code handles all the dynamic behavior of the Task Manager:

DOM Elements

```
// Dom Elements
onst taskInput= document.getElementById('task-input');
const addButton = document.getElementById('add-button');
const taskList= document.getElementById('task-list');
const noTaskMessage = document.getElementById('no-tasks');
const clearAllButton = document.getElementById('clear-all');
const tasksCountElement= document.getElementById('tasks-count');
const completedCountElement= document.getElementById('completed-count');



let tasks=[]; // initail blank array with name tasks
```

This sections selects all the necessary DOM elements that will be manipulated and initializes and empty tasks array.

Data Management

```
function loadTasks(){
    // Load Tasks from local Storage when pages loads
    const savedTasks = localStorage.getItem('tasks');
    // if tasks exist in local storage, parse them into tasks array
    if(savedTasks){
        tasks = JSON.parse(savedTasks);
        renderTasks();
    }

}
// save Tasks to localStorage
function saveTasks(){
    localStorage.setItem('tasks', JSON.stringify(tasks));
}
```

Thesed functions handle persistent storage:

- `loadTasks()` : Retrieves tasks from localStorage when the page loads.
- `saveTasks()` : Saves the current Tasks to localStorage whenever changes are made.

Task Operations

```
// Add a new task

function addTask(){
    const taskText = taskInput.value.trim();
// check if task is not empty
    if(taskText){
        // Create a new task object
        const newTask = {
            id: Date.now(),// generatews a unique id using timestamp
            text: taskText,
            completed:false,
            createdAt: new Date().toISOString()

        };
        // Add text to array

// tasks.push(taskText);
tasks.push(newTask);
console.log(tasks);

    }
    saveTasks();

    // Clear input
    taskInput.value='';

    //Update ui
    renderTasks();
}
function saveTasks(){
    localStorage.setItem('tasks', JSON.stringify(tasks));
}

// step 5
function loadTasks(){
    // Try to get tasks from local Storage
    const savedTasks = localStorage.getItem('tasks');
    // if tasks exist in local storage, parse them into tasks array
    if(savedTasks){
        tasks = JSON.parse(savedTasks);
        renderTasks();
    }

}
function deleteTask(taskId)
{
    // filter out the task for givenid
    tasks= tasks.filter(function(task){
            return task.id !== taskId;
        });

        // Save updated task to localStorage
        saveTasks();
        //Update Ui
        renderTasks();
    }
    //Clear all task
    function clearAllTask(){
        //comfirm before clearing
        if(tasks.length >0)

        {
            const confirmed= confirm("Are you sure you want to delete all tasks?!");
            if(confirmed){
                tasks=[];
                saveTasks();
                renderTasks();
            }
        }
    }
```

These function implemen the core task operations:

- `addTask()` : Creates a new task object, adds it to the array, and upadts the UI
- `deleteTask(taskId)` : Removes a specififc task by ID using array filtering
- `toggleTaskCompletion(taskId)` : Toggles the completed status of a task
- `clearAllTasks()` : Remove all tasks after confirmation.

UI Rendering

```
    function renderTasks(){

        // Clear Current list
        taskList.innerHTML = '';

        // show/hide the "no tasks" message
        if(tasks.length === 0)
        {
            noTaskMessage.style.display = 'block';
        }else{
            noTaskMessage.style.display = 'none';
        }

        // Create task elements
        tasks.forEach(function(task){

            const li = document.createElement('li');
            li.className = 'task-item';
            taskList.appendChild(li);
            // create checkbox
            const checkbox = document.createElement('input');
            checkbox.type = 'checkbox';
            checkbox.checked = task.completed;
            checkbox.addEventListener('change', function(){
                toggleTaskCompletion(task.id);
            });

            //Create task text span
            const span = document.createElement('span')
            span.className = task.completed ? 'task-text completed': 'task-text';
            span.textContent = task.text;
            // create delete button
            const deleteButton = document.createElement('button');
            deleteButton.className = 'delete-btn';
            deleteButton.textContent =  'Delete';
            deleteButton.addEventListener('click', function(){
                deleteTask(task.id);
            });

            // Add Elements to list items
            li.appendChild(checkbox);
            li.appendChild(span);
            li.appendChild(deleteButton);

            taskList.appendChild(li);
        });
        updateTaskCounts();

    }
```

These Functions handle UI updates:

- `renderTask()` : Recreates the entire task list in the DOM based on the current data
- `updateTaskCounts()` : Calculates and displays the total and completed tasks counts.

Event Handling

```
//EVent Listeners
addButton.addEventListener('click', addTask);
taskInput.addEventListener('keypress', function(e){
// Add task when enter key is pressed
if(e.key == 'Enter')
{
    addTask();
}
});

clearAllButton.addEventListener('click', clearAllTask)

//Initialize app
loadTasks();
```

The final sections sets up event listeners:

- click handler for the Add button
- KeyPress handler for the Enter Key in the input field
- Click handler for the Clear All button
- Initial call to `loadTasks()` to load saved tasks when the page loads.

Data Management

The Application uses a simple but effective data structure

1. **Task Object Structure**:

```
{
        id: Date.now(),// generates a unique id using timestamp
        text: taskText,
        completed:false,
        createdAt: new Date().toISOString()

    };
```

2. Storage Method:

- The application uses `localStorage` for persistent Storage.
- Tasks are stored as a JSON string and parsed back to an array when needed.
- This allows tasks to persist even when the browser is closed and reopened.

Event Flow

The typical flow of operation is :

1. User adds a task -> `addTask()` -> `saveTasks()` -> `renderTasks()`
2. User toggles completion -> `toggleTaskCompletion()` -> `saveTasks()` -> `renderTasks()`
3. User Deletes a task -> `deleteTasks()` -> `saveTasks()` -> `renderTasks()`
4. User clears all tasks -> `clearAllTasks()` -> `saveTasks()` -> `renderTasks`

this patterns ensures that:

1. The data model(tasks array) is updated first
2. Changes are persisted in localStorage
3. The UI is updated to reflect the current state