Documentation: Interactive Task Manager Web Application

Student Name: Gilbert Hara

Student ID: BScICT 24/200/015

1. System Description

The Interactive Task Manager is a client-side web application developed using HTML, CSS, and JavaScript. It enables users to create, manage, and track personal tasks without requiring any server-side technologies. The application fulfills all minimum functional requirements and implements two features: task categories and a task counter.

The user interface consists of three main sections:

- ✓ A form to input new tasks (with name, optional due date, and category)
- ✓ A statistics panel showing total and completed task counts
- ✓ A dynamic list that displays all tasks with interactive controls

All data is stored exclusively in a JavaScript array. No external libraries, frameworks, or backend services are used. Task persistence across browser sessions is achieved using the browser's localStorage API, which is a client-side storage mechanism and complies with the "client-side only" requirement.

2. Implementation Details

- **HTML Structure**

The document uses semantic elements as required: <header> for the title, <main> containing three <section> elements (task input, statistics, and task list), and a <footer> for copyright information. The form includes a text input for the task name (marked as required), a dropdown for category selection, an optional date input, and a submit button.

- **CSS Design**

The layout is built with Flexbox to ensure logical vertical stacking of sections on all screen sizes. Buttons include hover pseudo-class styles that change background color on mouse over, satisfying the requirement for interactive visual feedback. Task items are styled with borders, padding, and subtle background shading. Completed tasks are visually distinguished by a green left border and strikethrough text. Long task names are prevented from overflowing their container through the use of `word-wrap: break-word` and `word-break: break-all`.

- **JavaScript Logic**

All interactivity is handled in `script.js`. On page load, the application attempts to retrieve previously saved tasks from `localStorage`. If none exist, it initializes an empty array called `tasks`.

When the user submits the form:

- The task name is trimmed and validated. If empty, an alert is shown and submission is cancelled.
- A new task object is created with properties: id (generated using Date. Now), text, category, due Date (or null if not provided), and completed (initially false).
- The object is added to the tasks array.
- The array is saved to localStorage using JSON.string.
- The task list and counters are re-rendered.

- The render Tasks function clears the current list in the DOM, then iterates over the tasks array using for each. For each task, it creates an <li> element, applies the completed class if applicable and populates its content with the task text, category, and optional due date. Two buttons are added: Complete (or "Undo" if already completed) and Delete. Event listeners are attached to these buttons to call toggle complete (id) or delete Task (id).

- The toggle complete function finds the task by ID, flips it's completed Boolean, saves the updated array, and re-renders the list.

- The delete Task function prompts for confirmation, filters the array to remove the task, saves the change, and re-renders.

- The task counter is updated by the update Counters function, which calculates:
    - Total tasks = tasks. Length
    - Completed tasks = number of tasks where `completed === true (using filter)

These values are written to the corresponding elements in the statistics section.

3. Challenges Faced

- ✓ **Text Overflow**
- During testing, entering a very long string without spaces (e.g., repeated digits) caused horizontal overflow, breaking the layout. This was resolved by applying CSS properties that force long words to wrap within their container, ensuring consistent appearance regardless of input length.
- ✓ **Data Persistence**
- Since JavaScript arrays are reset on page reload, tasks would be lost without persistence. The solution was to use `localStorage` to save and load the task array as a JSON string. This approach maintains full client-side operation while improving user experience.
- ✓ **Real-Time Counter Accuracy**
- Ensuring the task counters reflected the current state after every user action required careful function sequencing. The solution was to call update Counters at the end of every render cycle, guaranteeing up-to-date statistics.

This application fully satisfies the project requirements. It uses semantic HTML, Flexbox layout, external CSS and JavaScript, form validation, array-based data storage, DOM manipulation, loops, and conditionals. The two optional features—task categories and a live task counter—have been implemented effectively. The solution is entirely client-side, responsive, and robust against common user input issues. The code demonstrates a clear understanding of core front-end web development concepts.