**High-Level Design (HLD) Document**

**1. Introduction**

**1.1 Purpose**

The purpose of this High-Level Design document is to outline the architecture and components of the To-Do List Application. It provides a high-level overview of how the system is structured and how its various parts interact.

**1.2 Scope**

This document covers the architectural design, key components, and interactions required to build the To-Do List Application.

**2. Architecture Overview**

**2.1 System Architecture**

The To-Do List Application will follow a **Client-Server Architecture**. However, since it is a client-side web application, the server-side component is not present. The entire application will run in the user's browser, utilizing **HTML**, **CSS**, and **JavaScript**.

**Architectural Diagram**

+---------------------------------+

| Web Browser |

| |

| +---------------------------+ |

| | To-Do List App | |

| | | |

| | +-------------------+ | |

| | | User Interface | | |

| | | (HTML/CSS) | | |

| | +-------------------+ | |

| | | |

| | +-------------------+ | |

| | | JavaScript | | |

| | | (Business Logic)| | |

| | +-------------------+ | |

| | | |

| | +-------------------+ | |

| | | Local Storage | | |

| | | (Data Storage) | | |

| | +-------------------+ | |

| +---------------------------+ |

+---------------------------------+

**3. Components**

**3.1 User Interface (UI)**

* **Description**: The UI is responsible for user interactions. It is built using HTML and CSS, ensuring that the application is visually appealing and user-friendly.
* **Components**:
  + Header: Displays the title of the application.
  + Input Area: Contains an input field for new tasks and an "Add" button.
  + Task List: Displays all tasks with options to edit, complete, or delete.
  + Footer: Contains copyright and contact information.

**3.2 Business Logic (JavaScript)**

* **Description**: The JavaScript code manages the application's functionality, including adding, editing, deleting, and completing tasks.
* **Functions**:
  + addTask(): Adds a new task to the list.
  + editTask(): Edits an existing task in-line.
  + deleteTask(): Deletes a selected task.
  + toggleComplete(): Marks a task as complete or incomplete.
  + saveTasks(): Saves tasks to local storage.
  + loadTasks(): Loads tasks from local storage when the application initializes.

**3.3 Data Storage**

* **Description**: The application uses the browser's **Local Storage** to persist tasks across sessions.
* **Data Structure**:

json

[

{

"name": "Task 1",

"completed": false

},

{

"name": "Task 2",

"completed": true

}

]

**4. Interaction Flow**

**4.1 Use Case Scenarios**

1. **Adding a Task**:
   * User enters a task in the input field and clicks the "Add" button.
   * The application validates the input, adds the task to the list, and updates local storage.
2. **Editing a Task**:
   * User clicks the "Edit" button next to a task.
   * The task text is replaced with an input field pre-filled with the current task name.
   * User edits the text and clicks the "Save" button.
   * The application updates the task in the list and local storage.
3. **Completing a Task**:
   * User clicks the "Complete" button next to a task.
   * The task is marked as complete, and the UI updates accordingly.
4. **Deleting a Task**:
   * User clicks the "Delete" button next to a task.
   * The application removes the task from the list and local storage.

**4.2 Sequence Diagram for Adding a Task**

User UI JavaScript Local Storage

| | | |

|-- Enter Task -->| | |

| |-- Validate Input -->| |

| |-- Add Task -->|-- Save to Local Storage -->|

| | |<-- Confirmation --|

|<-- Update UI --| | |

**5. Conclusion**

This High-Level Design document provides an overview of the architecture, components, and interaction flow of the To-Do List Application. By following this design, developers can build a structured, maintainable, and functional web application that meets user requirements.