Software Design Document: Hero & Monster Manager

1. Introduction

This document provides a high-level overview of the design for the "Hero & Monster Manager" application. It describes the application's purpose, architecture, key features, data management, and user interface design.

2. Purpose and Scope

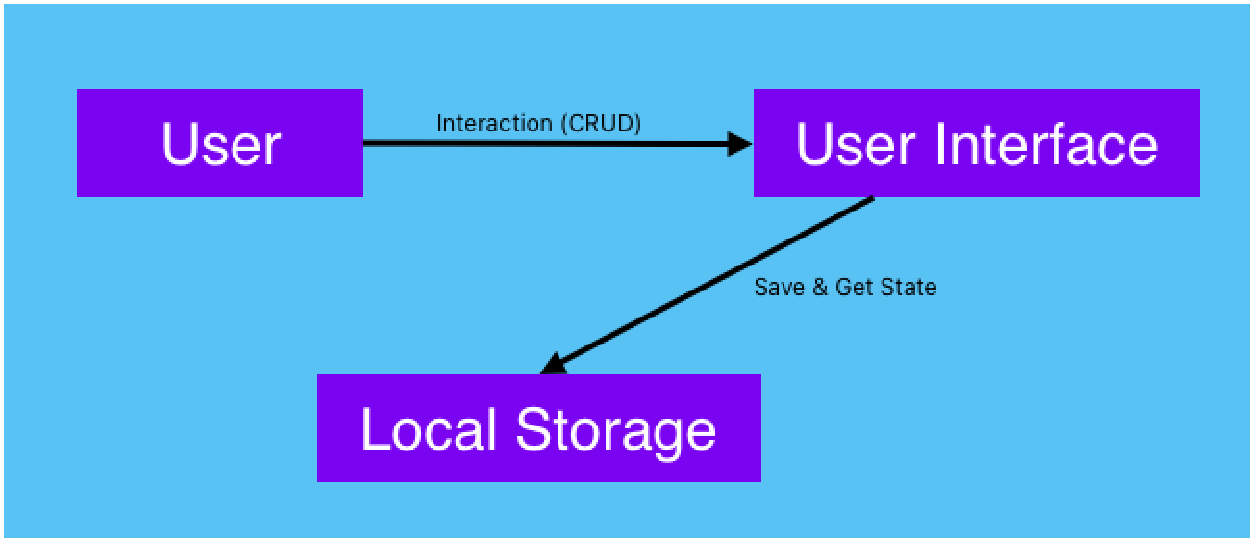
The "Hero & Monster Manager" is a web-based application designed for managing character entities, specifically categorizing them as either "Heroes" or "Monsters." Users can create, view, edit, and delete characters, with data persistence handled locally. The primary goal is to demonstrate fundamental web development concepts including HTML structure, CSS styling (using Tailwind CSS), and JavaScript for dynamic content manipulation and local storage interaction.

3. Architecture Overview

The application follows a client-side, single-page application (SPA) architecture.

- **User Interface (Client-Side):** The entire application logic and rendering are handled within the user's web browser using HTML, CSS, and JavaScript.
- **Data Storage:** Character data is persisted using the browser's localStorage.
- **Simulated Backend:** A simulateCharacterAPI function is used to mimic asynchronous API calls for creating and updating characters, introducing a simulated network delay and occasional failure for demonstration purposes.

A simplified block diagram of the system is as follows:



4. Key Features

- **Character Creation:** Users can add new characters by providing a name, health, attack power, type (Hero or Monster), and an icon.
- **Character Listing:** Characters are displayed in two distinct rosters: "Heroes Roster" and "Monsters Den," dynamically updated upon changes.
- **Character Editing:** Existing characters can be edited, updating their properties. The form intelligently switches between "Add New" and "Editing" modes.
- **Character Deletion:** Users can remove characters from the system.
- **Form Validation:** Basic client-side validation for input fields.
- **Responsive Design:** The layout adapts to various screen sizes (mobile, tablet, desktop) using Tailwind CSS.
- **Local Data Persistence:** Character data is saved and loaded from localStorage, ensuring data persists across browser sessions.
- **User Feedback:** Messages (success, error, info) are displayed to the user for actions like character creation, update, or deletion.
- **Loading Indicator:** A "Processing... please wait!" message is shown during simulated API calls.

5. Data Model

The core data entity is a "Character," represented by a JavaScript object with the following structure:

```
{
  id: number,                // Unique identifier for the character
  name: string,              // Name of the character (e.g., "Superman", "Dragon")
  health: number,            // Current health points
  attack: number,            // Attack power
  type: "Hero" | "Monster",          // Type of character
  icon: string,              // Emoji or character symbol (e.g., "🦸", "🐉")
  maxHealth: 100             // Maximum health (fixed at 100 in current implementation)
}
```

This data is stored in the browser's localStorage as two separate JSON stringified arrays: heroesData and monstersData.

6. User Interface Design

The UI is built using plain HTML and styled extensively with Tailwind CSS, ensuring a clean, modern, and responsive aesthetic.

- **Overall Layout:** A central container with a gradient background, housing a character creation/edit form and two distinct roster sections.
- **Form Section:**
  - Dynamic title ("Add New Character" or "Editing [Character Name]").
  - Input fields for Name, Health, Attack Power, Character Type (dropdown), and Character Icon.
  - A single "Create Character" or "Save Changes" button.
  - A loading indicator and a message box for user feedback.
- **Roster Sections ("Heroes Roster" and "Monsters Den"):**
  - Each roster displays characters in a grid layout, adapting column counts based on screen size.
  - **Character Card:** Each character is presented in a card format, displaying:
    - Icon and Name
    - Current Health and Max Health with a health bar (green for healthy, red for low).
    - Attack Power.
    - "Edit" and "Delete" buttons for individual character actions.

7. Technical Implementation Details

- **HTML Structure:** Semantic HTML5 elements are used for clarity and accessibility.
- **CSS Styling:**
  - **Tailwind CSS:** Used for most styling, providing utility classes for layout, spacing, colors, typography, borders, shadows, and responsive adjustments.
  - **Custom CSS:** Minimal custom CSS is used for specific effects like character card hover transitions and the defeated state.
- **JavaScript Logic:**
  - **DOM Manipulation:** Direct manipulation of the Document Object Model (DOM) to render, update, and remove character cards.
  - **Event Listeners:** Used to handle user interactions (form submission, button clicks).
  - **Data Management Functions:** Functions to saveDataToLocalStorage(), getDataToLocalStorage(), addOrUpdateCharacter(), editCharacter(), and deleteCharacter().
  - **Simulated API:** The simulateCharacterAPI function demonstrates asynchronous operations with Promises, setTimeout, and random success/failure.

    - **Unique ID Generation:** A simple counter (nextCharacterId) generates unique IDs for new characters.
    - **Message Box:** A showMessage() function controls the display and styling of user feedback messages.

8. Future Considerations

- **Robust Input Validation:** Enhance form validation to cover more edge cases (e.g., non-numeric input for health/attack).
- **Backend Integration:** Replace the simulated API with actual API calls to a server-side backend and a persistent database (e.g., Node.js with Express and MongoDB/PostgreSQL) for a true full-stack application.
- **Error Handling:** Implement more sophisticated error handling and display for network failures or invalid data.
- **Advanced Features:**
    - Search/Filter functionality for characters.
    - Character abilities/skills.
    - "Battle" simulation between heroes and monsters.
    - User authentication.