# **Software Design Document – Use Case 4: Clear Data**

## **1. System Overview**

This use case allows the user to permanently delete all previously added ideas. The operation is triggered via a button located on the Home page (index.tsx). Upon activation, the process resets both the application state (ideas) and the localStorage, returning the application to its initial, empty state. The Singleton Design Pattern is applied for managing this functionality.

## **2. System Context**

The application is entirely self-contained and does not interact with any external services. The architecture includes:

* **User**: The only actor in the system. In this use case, the user performs the data reset action.
* **Browser**: The platform where the application runs.
* **LocalStorage**: The storage mechanism for persisting user ideas, and the target of the data-clearing operation.

## **3. Key Functions and Functionalities**

This use case provides the following capabilities:

* Displays a “Clear All” button on the Home page.
* When clicked, invokes getDeleteAllIdeasSingleton().
* Resets the in‑memory ideas state via setIdeas([]).
* Clears the ideapulse\_ideas key from localStorage using saveIdeasToStorage([]).
* Automatically updates the UI to reflect an empty list.

## **4. Assumptions and Dependencies**

### **Assumptions:**

* The user is using a modern web browser.
* JavaScript is enabled.
* localStorage is available and unrestricted in the browser.

### **Dependencies:**

* **React / Next.js**: Framework used for component-based UI.
* **TypeScript**: Ensures type safety.
* **Tailwind CSS**: Used for UI styling.
* **Next-pwa**: Enables PWA features.
* **utils/storage.ts**: Contains storage logic and helper functions.
* **StorageManager.ts**: Implements the Singleton pattern for centralized storage access.

## **5. Architectural Design**

### **Layered Architecture Structure:**

* **UI Layer**: index.tsx
* **Controller Layer**: utils/ideaFactory.ts
* **Storage Layer**: utils/storage.ts

## **6. Component Design**

| **Component** | **Responsibility** |
| --- | --- |
| index.tsx (Home page) | Renders the “Clear All” button and displays the idea list. |
| utils/ideaFactory.ts | Exposes getDeleteAllIdeasSingleton() for delete logic. |
| utils/storage.ts | Implements saveIdeasToStorage() and related helpers. |

## **7. Data Design**

### **Core Data:**

All ideas are stored in localStorage under the key ideapulse\_ideas. When the user clears data, this key is removed.

### **Data Flow:**

User clicks button → getDeleteAllIdeasSingleton() → setIdeas([]) → saveIdeasToStorage([]) →UI re‑render

## **8. Design Patterns**

The **Singleton Pattern** is applied in this use case to ensure there is only one instance of the storage controller (StorageManager) throughout the application.

* StorageManager.getInstance().clearIdeas() is used to remove all ideas.
* Centralized access ensures consistency across all components.

## **9. Implementation Notes**

* The user interface presents a single button labeled "Clear All Data".
* When clicked, the clearIdeasFromStorage() function is triggered.
* The state is updated using setIdeas([]), and the UI refreshes to reflect the empty list.
* No confirmation dialog is implemented by default, but it can be added for safety.

## **10. User Interface Design**

**Home Page** includes:

* A prominently placed **“Clear All”** button.
* The idea list view, which becomes empty after clearing.
* Optionally, a toast or banner can confirm successful data deletion.

## **11. External Interfaces**

* No APIs or third-party services are used.
* All logic and data interactions remain within the local browser context.

## **12. Performance Considerations**

* Data clearing should execute in under 50ms.
* UI should reflect the cleared state immediately.
* Minimal memory usage and efficient localStorage operations are guaranteed.

## **13. Design for Testability**

* clearAll() (via getDeleteAllIdeasSingleton()) and saveIdeasToStorage() are pure functions—ideal for unit tests.
* The Home page component can be tested with React Testing Library by mocking localStorage and verifying state resets on button click.