Software Design Document - Use Case 2: View Ideas

1. System Overview

This use case allow users to see previously added ideas as a list. Each idea contains content, mood, tags and timestamp. To implement this use case Observer pattern was chosen. React's useState and useEffect automatically updates the UI when the data changes, this provides a dynamic and responsive experience.

2. System Context

The application does not integrate with any external systems. Its architecture is as follows:

- User: The only actor in the system, responsible for creating and saving ideas. In this use case user also requests to see saved ideas.
- Browser: The platform where the application is executed
- LocalStorage: The mechanism used to store data locally within the user's browser. Also loads and monitors saved ideas.

3. Key Functions and Functionalities

This use case covers the following features:

- Automatically loads all ideas stored in localStorage on page load.
- Displaying each idea's content, mood, and tags.
- Presenting the ideas in a chronologically ordered list.
- Automatically updating the list when a new idea is added or an existing idea is deleted.

4. Assumptions and Dependencies

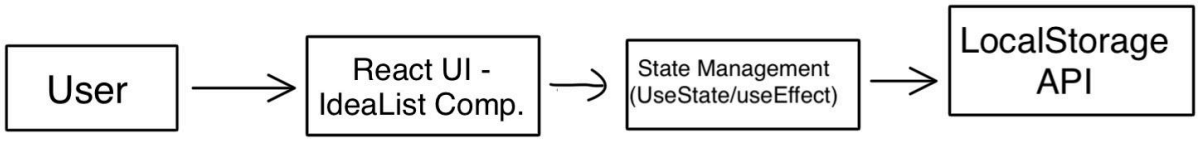
Assumptions:

- User has a modern web browser
- JavaScript is enabled in the browser
- The browser supports the localStorage API and it is not restricted or disabled

Dependencies:

- Next.js / React: Core framework for component based user interface
- Typescript: static type checking to reduce errors during development
- Tailwind CSS: A utility first CSS framework.
- Uuid: Library for generating unique id for each idea.
- Next-pwa: used to easily integrate paw features

5. Architectural Design



Layered Architecture Structure:

- UI Layer: IdeaList.tsx
- State/Controller: pages/index.tsx
- Storage: utils/storage.ts

6. Component Design
Subsystems and Responsibilities

Component	**Responsibility**
IdeaList.tsx	Renders the list of stored ideas
pages/index.tsx	passes idea state to IdeaList Component
storage.ts	Loads ideas from localStorage

7. Data Design

The core data entity is the Idea object. Each idea contains the following fields:

- id: A unique identifier (string, typically a UUID).
- content: The textual content of the idea.
- mood: The emotional state associated with the idea (inspired, neutral, tired, excited).
- tags: An array of strings used to categorize the idea.,
- timestamp: The creation date and time in ISO 8601 format.

Storage

The application uses localStorage to store and retrieve user ideas under the key ideapulse_ideas. On application startup (in index.tsx), the stored data is retrieved (getIdeasFromStorage() function).

When user adds, edits or deletes an idea setIdeas() and saveIdeasToStorage() functions update the state and trigger a UI change.

Data Flow:

Flow is from storage to UI

On page load —> getIdeasFromStorage() —> useState() —> IdeaList

8. Design Patterns

Main goal in this use case is to update the UI when the idea list changes. To achieve this Observer Pattern was chosen.

To achieve Observer Pattern Reacts UseState and UseEffect functions are used.

- useState(ideas) manages observable state.
- useEffect observes and re-renders when ideas are modified.
- This allows IdeaList.tsx to always show the updated list without manual refresh.

With that real-time synchronization happens.

9. Implementation Notes

View ideas use case is implemented using a modular React component architecture. The core logic consists of:

- Retrieving data from localStorage
- Storing the retrieved data in a useState variable named ideas
- Passing the ideas state as a prop to the IdeaList component
- Rendering the ideas dynamically with conditional logic to handle empty states or filters

10. User interface Design

View Ideas use case presents ideas in a list form. Each idea is displayed with:

- Content
- Tags
- Mood
- Timestamp

This UI also includes real-time updates when ideas are added, edited, or deleted and optional filters by mood or tag for better usability

11.External Interfaces

There is no external APIs or backend services. No third-party services are used for data storage or retrieval. All logic and data remain local to the user's device

12.Performance Considerations

Ideas must be retrieved and rendered within milliseconds after page load.
UI updates should be fast
Storage and memory usage are minimal due to efficient JSON structure.

13.Design for Testability

The application is structured to support modular and component-based testing:

- IdeaList accepts props and can be tested with mock data.
- utils/storage.ts contains pure functions suitable for unit testing.
- State and props are separated, allowing isolated testing of logic and UI.
- Jest or React Testing Library can be integrated to test rendering and data flow.