

# Software Design Document – IdeaPulse: Add Idea Use Case

**Authors:** Mustafa Çevik, Halil Melih Akça, Aylin Şahin, Alp Toksöz

## 1. System Overview

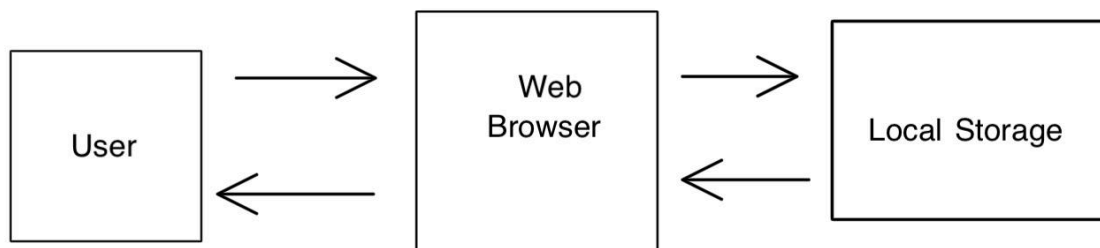
IdeaPulse is a web-based Progressive Web Application (PWA) designed to help users capture their ideas instantly and effortlessly. This document outlines the first core use case of the project: Add Idea.

In this feature, users can input text to describe their idea, assign a “mood” that reflects the idea’s context or category, and attach relevant tags for better organization. All ideas are stored locally.

## 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.
- Browser: The platform where the application is executed
- LocalStorage: The mechanism used to store data locally within the user’s browser



## 3. Key Functions and Functionalities

The user starts by selecting “Add New Idea” from the navigation bar. In this section they can:

- Type a description of their idea
- Add relevant tags to categorize the idea
- Choose a mood that reflects the idea
- Finally, click the Save button to store the new idea locally

## 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

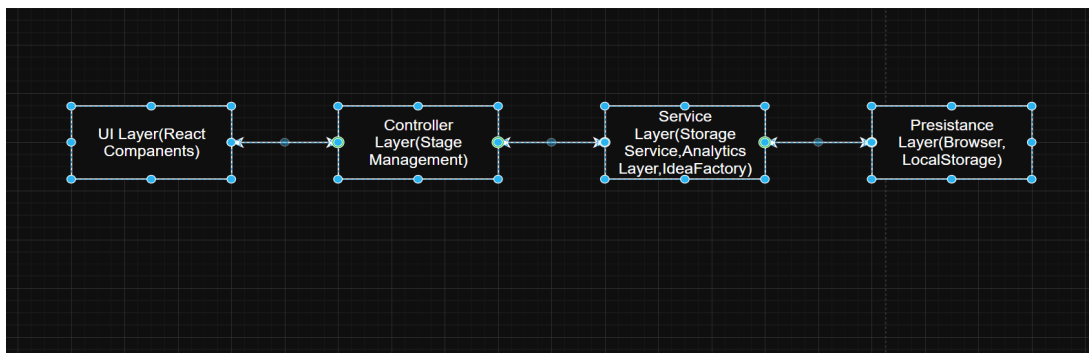
Dependencies:

- 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

### 5.1 System Architecture Diagram (High-Level)

The application uses Client-Side Layered Architecture that separates responsibilities:



### 5.2. Layered Architecture Structure:

- This structure is suitable for small and medium-sized client applications and is based on the principle of Separation of Responsibilities (SoC):
- UI Layer: Responsible only for the appearance and user interaction.
- Controller/Service Layer: Manages business logic and data flow.
- Persistence Layer: Stores data in resources such as localStorage, abstracted from external layers.

Thanks to this architecture:

Code becomes more readable and testable.

For example, StorageService can be tested independently of UI.

Changes are made easier thanks to the independence between layers.

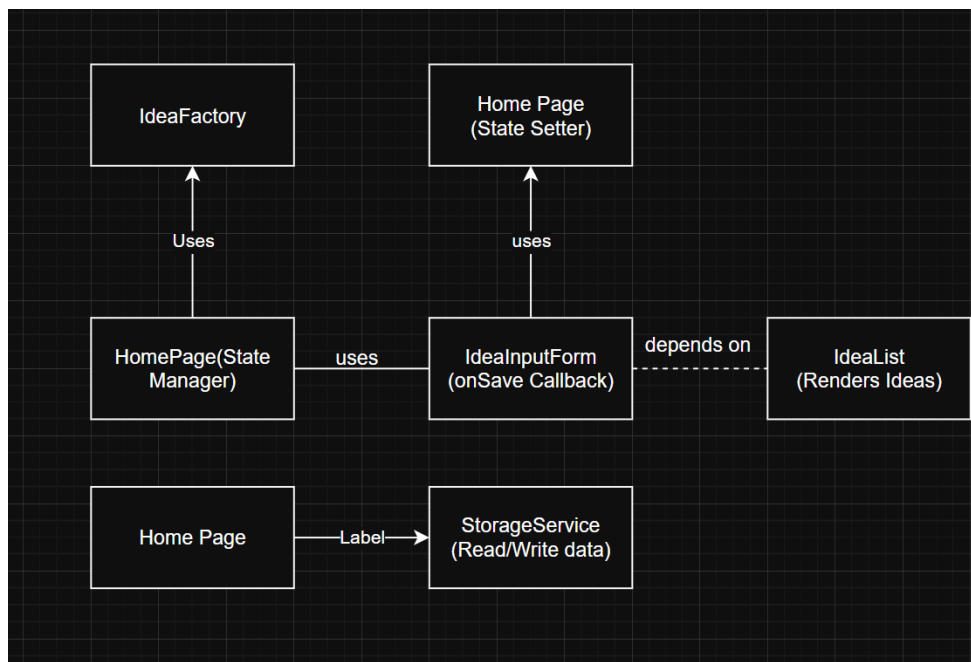
## 6. Component Design

## 6.1. Subsystems and Modules

### Core Components and Modules for the “Add Idea” Function:

- HomePage (Component): It is the home page of the application. It contains other components and provides general state management.
- IdeaInputForm (Component): It is the form component where the user enters ideas, moods and labels.
- IdeaList (Component): It is the component that lists the saved ideas.
- IdeaFactory (Module/Service): It is the factory module that creates new Idea objects according to user inputs.
- StorageService (Module/Service): It is the service layer that abstracts data reading and writing operations with localStorage.

## 6.2. Component Diagram (UML)



## 6.3. Interfaces Between Components

- IdeaInputForm → HomePage: When the user clicks the "Save" button, the IdeaInputForm component passes the data to the HomePage component via an onSave callback prop.
- HomePage → IdeaList: HomePage passes the idea list it reads from localStorage or newly added to the IdeaList component via the ideas prop

## 7. Data Design

## 7.1. Data Model

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

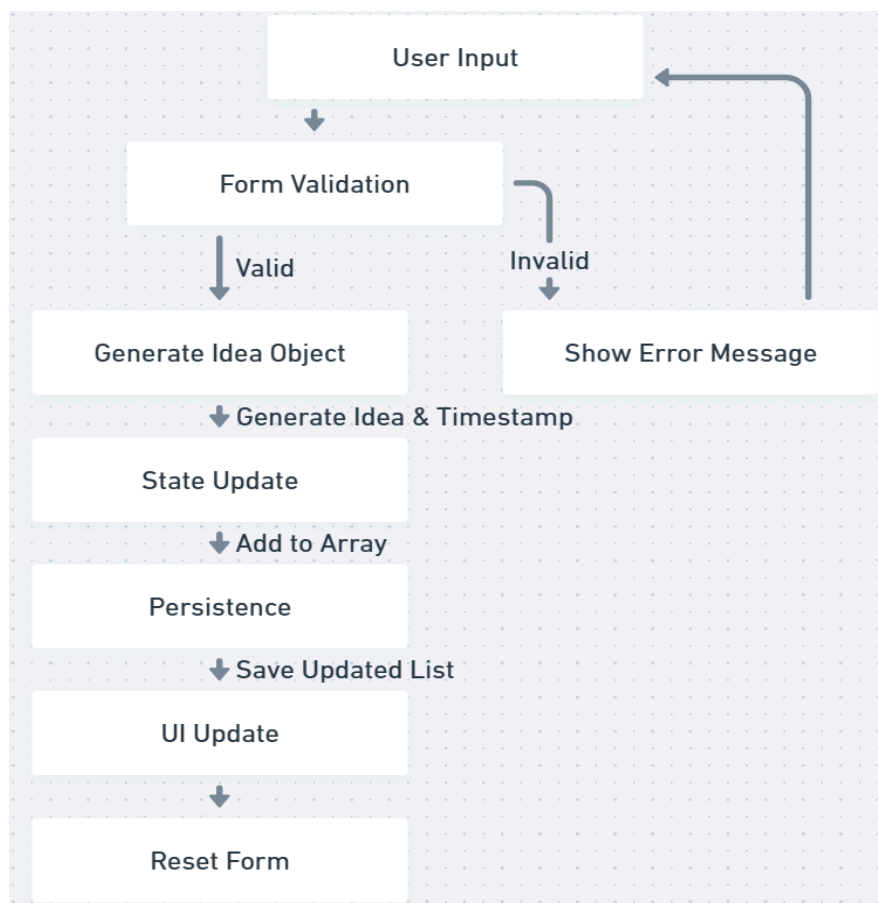
- **id**: A unique identifier (string, typically a UUID).
- **content**: The textual content of the idea.
- **mood**: The emotional state associated with the idea (inspired, neutral, tired, excited).
- **tags**: An array of strings used to categorize the idea.,
- **timestamp**: The creation date and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

This model supports flexible categorization, mood tracking, and time-based analysis.,

## 7.2 Data Storage

Ideas are stored in the browser using localStorage. All ideas are serialized as a JSON array and saved under a designated key. The user's theme preference is also stored in localStorage. While IndexedDB is checked for more complex needs, localStorage remains the primary storage mechanism for its simplicity and offline accessibility.

## 7.3. Data Flow Diagram ( For “Add Idea”)



## 7.4. Data Validation Rules

- **Content:** Must not be empty or contain only whitespace.
- **Tags:** Optional; if provided, they are split by commas, trimmed, and filtered to remove empty entries.
- **Mood:** Must be one of the predefined mood options.
- **ID:** Must be unique for each idea (typically generated using UUID).
- **Timestamp:** Must be a valid ISO 8601 date-time string representing the creation time.

## 8. Design Patterns

### 8.1. Factory Pattern

#### **Context and Justification:**

Creating a new Idea object involves more than just using user input. It also requires generating a unique id and assigning a createdAt timestamp. Separating this creation logic from UI components (such as IdeaInputForm or HomePage) makes the code cleaner and more maintainable. The IdeaFactory is responsible for this logic.

#### **Implementation:**

A createIdea(content, mood, tags) function is defined. This function generates a unique ID using uuid, creates a timestamp using new Date().toISOString(), and combines all the data into a valid Idea object which it then returns.

Benefit:

If the structure of the Idea object changes in the future (e.g., a new field is added), only the IdeaFactory needs to be updated. UI components remain unaffected by such changes.

### 8.2. Observer Pattern

#### **Context and Justification:**

When a user adds a new idea, the idea list (IdeaList) needs to be automatically updated. Continuously checking localStorage for changes would be inefficient. The Observer pattern allows components (observers) to be automatically notified when a change occurs in the data source (subject).

#### **Implementation:**

This pattern is implicitly implemented using React's state management (useState, useEffect).

- **Subject:** The ideas state variable in the HomePage component.
- **Observer:** The IdeaList component, which receives the ideas state as a prop.
- **Notification Mechanism:** When a new idea is added, the setIdeas function in HomePage is called. React detects the state change and re-renders all components that depend on that state, including IdeaList.

Benefit:

This design decouples the data layer from the presentation layer. IdeaList doesn't need to know where the data comes from or how it's updated; it simply renders what it receives. This leads to a reactive and easily maintainable UI.




## 9. Implementation Notes

### 9.1. Home Page Structure:

The homepage consists of two main sections arranged vertically:

1. IdeaInputForm — Positioned at the top of the page.
2. IdeaList — Displayed below the form, showing a list of idea cards stacked vertically.

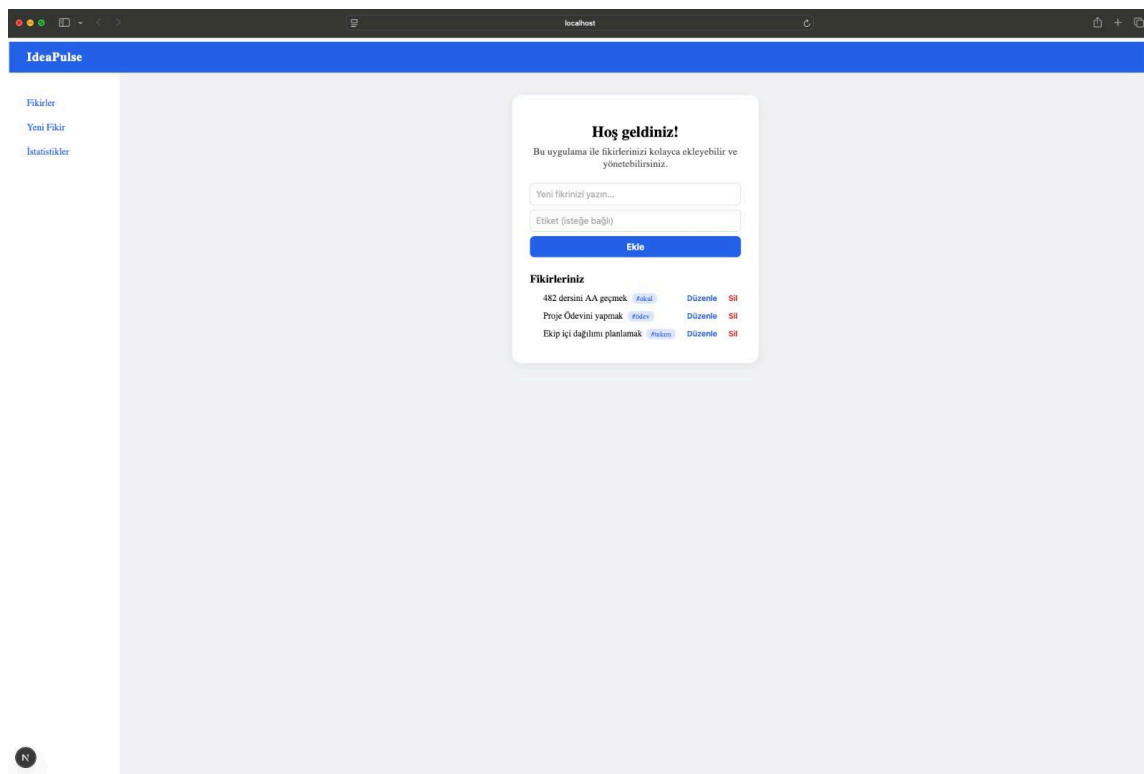
### 9.2. IdeaInputForm Component:

- A large textarea for entering the main idea content.
- Below the textarea, there are stylized mood selection buttons, such as:
  -  Inspired
  -  Happy
  -  Thoughtful
- An input field (`<input type="text">`) for tags.
  - Users can separate tags using commas or spaces.
- A prominent “Save” button located at the bottom-right corner of the form.

### 9.3. IdeaList Component:

- Displays each idea as a card.
- Each card contains the following elements:
  - The content of the idea.
  - The creation date (formatted timestamp).
  - The selected mood, represented visually (e.g., emoji or label).
  - The associated tags, displayed as badges or labels.

## 10. User Interface Design



## 11. Other Considerations

### External Interfaces:

There are no interactions with external APIs or systems for this use case.

### Performance Considerations:

Although **localStorage** operations are synchronous, they are expected to be performant given the limited data volume (a few thousand ideas at most). List updates are expected to complete in under 500 milliseconds.

### Error Handling and Logging:

Error handling is limited to basic user input validation—such as preventing submission of empty ideas. No server-side logging is implemented at this stage.

### **Design for Testability:**

Modules like IdeaFactory and StorageService are designed as pure functions or simple classes. This makes them easy to unit test independently of the UI components.

### **Change Log**

#### **v1.0 (01/07/2025):**

- Initial design and implementation of the “Add Idea” use case completed.
- Factory and Observer patterns integrated.

### **Future Work / Open Issues**

- Add support for editing existing ideas.
- Add the ability to delete ideas.