System Architecture - Expense Sharing System

# 1. System Overview

The Expense Sharing System is a web-based application designed to simplify and streamline the process of splitting shared expenses among participants in collaborative group activities or one-time arrangements.  
The system ensures that costs are allocated fairly based on individual consumption and contributions, including support for a general participation fee to cover shared overhead costs.  
Users can create and manage events, add participants and categories of expenses, calculate balances, and review how much each person should pay or receive.  
The system also supports persistent storage of both draft and completed events, enabling users to resume editing at any stage and ensure transparency and flexibility throughout the process.

# 2. System Components

The system is composed of several modular components, each responsible for a distinct aspect of functionality. This separation of concerns promotes clarity, maintainability, and future extensibility.

| **Component** | **Description** |
| --- | --- |
| **User Interface (UI)** | Handles interaction with the user, including receiving input (event details, categories, participant data) and presenting output (balances and suggested payments). The UI may include forms, tables, and navigation options. |
| **Application Logic** | Contains the core business logic of the system, such as participation fee deduction, cost distribution per category, and balance calculations for each participant. |
| **Data Model** | Represents the core data structures, such as Event and Participant, along with their attributes and relationships. |
| **Storage Layer** | Responsible for the persistent saving and loading of events, including both draft and finalized events. Events may be stored in local files using formats such as JSON. |
| **Event Manager** | Manages the creation, editing, and navigation of events and participants during the data entry phase. It enables returning to and modifying individual participants or entire events before finalization. |

# 3. Class Responsibilities

| **Class** | **Responsibility** |
| --- | --- |
| **Event** | Represents an individual event. Stores the event name, list of categories, participation fee, list of participants, and the event status (DRAFT or COMPLETED). Provides methods for managing participants and categories. |
| **Participant** | Represents a single participant in an event. Stores the participant’s name, a map of expenses per category, and a list of consumed categories. |
| **BalanceCalculator** | Performs the main expense-sharing logic, including participation fee deduction, per-category cost distribution, and final balance calculation per participant. |
| **EventStorage** | Handles saving and loading events from persistent storage (e.g., JSON files). Responsible for converting event objects to and from a file-based format. |
| **EventManager** | Manages the creation and editing of events and participants during the data entry process. Allows revisiting and modifying existing participants or event details prior to finalization. |

# 4. Data Flow Description

1. **Event Initialization**:  
   The user chooses to create a new event or continue editing an existing draft.  
   The system either creates a new *Event* object or loads it from storage via *EventStorage*.
2. **Data Entry**:  
   The user inputs event details, including categories, participation fee, and participant data.  
   Data is handled by the UI and passed to the *EventManager*, which updates the corresponding *Event* and *Participant* objects.
3. **Draft Saving (Optional)**:  
   At any point, the user may choose to save the current event as a draft.  
   The *EventManager* triggers the *EventStorage* to serialize the event and save it as a draft file.
4. **Final Calculation**:  
   When the user completes data entry and chooses to calculate balances, the *BalanceCalculator* processes the event:
   * Deducts participation fees from categories.
   * Distributes per-category costs among consumers.
   * Calculates each participant’s final balance.
5. **Output Display**:  
   The results are passed back to the UI, which displays each participant’s balance and suggested transfers.
6. **Final Saving (Optional)**:  
   The user can choose to save the event as "completed", updating its status and storing it via *EventStorage*.
7. **Event Retrieval and Editing (Optional)**:  
   The user may later reopen a saved event (draft or completed) to review or modify data.

# 5. Design Principles and Considerations

**Modularity**: The system is divided into distinct components, each with a single responsibility (UI, Logic, Data, Storage), allowing for easier testing, debugging, and maintenance.

**Separation of Concerns**: Business logic is kept separate from data representation and user interaction. This ensures that changes in one part of the system do not affect others unnecessarily.

**Extensibility**: The architecture supports future additions such as export modules, authentication, recurring events, and cloud storage with minimal disruption to existing components.

**Flexibility in Data Entry**: The system allows editing of events and participants at any stage prior to finalization, including saving events as drafts and resuming them later.

**Simplicity**: The user experience and internal system structure are designed to remain as simple and intuitive as possible, making the system accessible even to non-technical users.

**Transparency**: The system includes features such as tabular breakdown of calculations and suggested transfers to ensure clarity and fairness in cost sharing.