**Lab Report: Lab 1 - Requirements Elicitation & Modeling**

**(KanbanFlow)**

## Contents

---

## 1. Cover Page

Title: Elicitation and Modeling Requirements for the KanbanFlow Project Management Tool

Course Info: Software Architecture

Student Details:

| Họ và Tên | MSSV |
|---|---|
| Trần Thành Long | 23010070 |
| Nguyễn Xuân Mạnh | 23010045 |

Date: Dec, 06 2025

## 2. Abstract/Summary

The KanbanFlow project aims to develop a streamlined collaborative task management tool based on the Kanban methodology.

This initial lab focused on requirements elicitation and modeling. We documented essential Functional, Non-Functional, and Architecturally Significant Requirements (ASRs) in a structured format.

The core system behavior was formally modeled using a UML Use Case Diagram, clearly defining system boundaries, external actors, and key behavioral relationships.

This foundation establishes the scope and the critical architectural drivers that will be addressed in subsequent labs, particularly the design of the Layered Architecture in next Lab 2.

## 3. Lab Specific Section: I. Requirements Elicitation & Modeling

This section documents the deliverables for Lab 1, as specified in the report requirements.

### 3.1 Software Requirements Specifications (SRS)

The following tables document the elicited requirements, which collectively form the basis for the KanbanFlow design.

#### 3.1.1 Functional Requirements (FRs)

These requirements define the specific behaviors and functions the KanbanFlow system must provide.

| ID | Description | Priority |
|---|---|---|
| **FR-01** | The system must allow a Registered User to create new Boards and manage Lists (columns) within those boards (e.g., To Do, In Progress, Done). | High |
| **FR-02** | The system must allow Users to create Cards (Tasks) within lists and edit their details (title, label, attachment, description, due date, assign members, checklist). | High |
| **FR-03** | The system must support drag-and-drop functionality to move Cards between Lists to update their status. | Critical |
| **FR-04** | The system must allow a Board Admin to invite other users to join a board via email or username. | Medium |
| **FR-05** | The system must allow Users to add comments to a specific Card for collaboration. | Medium |
| **FR-06** | The system must allow Admin to change the visibility of the board (limit to team or public so other users on platform can see) | Medium |

### 3.1.2 Non-Functional Requirements (NFRs)

These requirements specify criteria that can be used to judge the operation of the system, not specific functions[7].

| ID | Attribute | Description | Impact |
|---|---|---|---|
| NFR-01 | Performance (Response Time) | API response time for CRUD operations on Cards and Lists must be < 300ms under normal load conditions (50 concurrent users). | High |

| ID | Attribute | Description | Impact |
|---|---|---|---|
| NFR-02 | Security (Access Control) | Only users who are explicitly added as members of a Board can view or edit the cards within that board. All requests must be authenticated and authorized on the server-side. | Critical |
| NFR-03 | Reliability (Data Integrity) | The system must ensure that simultaneous moves of the same card by different users do not result in data loss or corruption, using optimistic locking or conflict resolution mechanisms. | Critical |
| NFR-04 | Availability | The system must maintain a minimum uptime of 99% during business hours (8 AM - 6 PM). | Medium |
| NFR-05 | Scalability | The system must be capable of supporting horizontal scaling to handle growth from 100 to 1,000 concurrent active boards without degrading response time. | High |
| NFR-06 | Usability (Responsiveness) | The user interface must be responsive and function correctly across desktop and mobile. | Medium |

### 3.1.3 Architecturally Significant Requirements (ASRs)

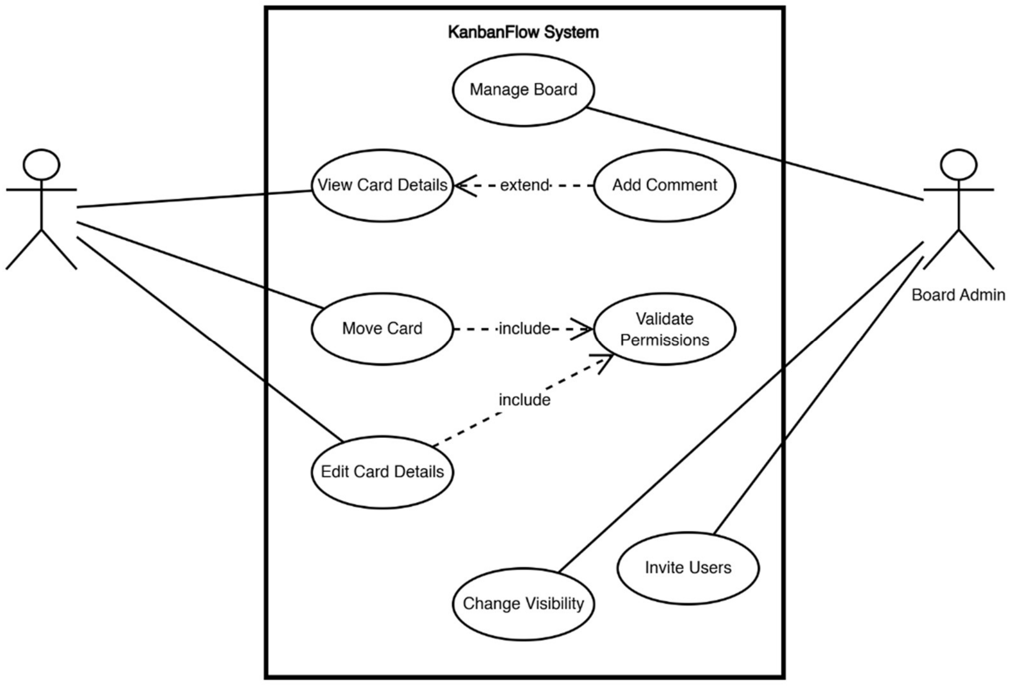
These are the non-functional requirements that have the most profound influence on the system's architecture, driving the selection of patterns.

| **ASR ID** | **Quality Attribute** | **Requirement Statement** | **Architectural Rationale** |
|---|---|---|---|
| ASR-1 | Modifiability | The core Business Logic must be strictly decoupled from the Web Interface so that a future Mobile Application (iOS/Android) can be developed using the same backend services without modifying existing code. | Enforces the principle of Separation of Concerns. Drives the design toward a layered structure where the Presentation Layer is physically separated from the Business Layer, allowing multiple interface types to share the same logic. |
| ASR-2 | Security | All write operations (Create, Update, Delete) on Cards and Lists must be validated against the user's Board Membership permissions on the server-side. | Necessitates a dedicated Security middleware to enforce access control at the Business Logic layer, protecting data regardless of the UI used. |
| ASR-3 | Reliability | The system must implement optimistic locking or conflict resolution to handle concurrent card modifications without data loss. | Requires the Persistence Layer to implement versioning or timestamps on card entities, and the Business Logic Layer to handle conflict detection and resolution. |
| ASR-4 | Scalability | The system must support stateless API design to enable horizontal scaling from 100 to 1,000 concurrent active boards. | Suggests avoiding stateful sessions in the web server and encourages RESTful API design within the Layered Architecture to |

| **ASR ID** | **Quality Attribute** | **Requirement Statement** | **Architectural Rationale** |
|---|---|---|---|
| | | | facilitate future load balancing. |
| ASR-5 | Performance | API endpoints must respond within 300ms for standard CRUD operations under normal load (50 concurrent users). | Drives efficient database query design, proper indexing in the Persistence Layer, and potential caching strategies at the Business Logic Layer. |

## 3.2 Modeling Artifact: UML Use Case Diagram

The UML Use Case Diagram models the functional scope of the KanbanFlow system, showing the system boundary, the external Actors, and the core Use Cases.



- **Actors:** The diagram clearly identifies the two primary actors: **Team Member** (Standard User) and **Board Admin**.
- **System Boundary:** The box delineates the scope of the KanbanFlow application.
- **Core Use Cases:** Key high-level functions, such as **Manage Board**, **Move Card**, and **Edit Card Details**, are shown.
- **Critical Path Flow:** The diagram details the **Move Card** (Task Progression) process, which is the core value proposition of the system.
- **Relationships Flow:**
    - **Include Relationship:** The "Validate Permissions" Use Case is a mandatory part of the "Move Card" process, modeled using the <<include>> relationship.
    - **Extend Relationship:** The "Add Comment" Use Case is an optional behavior that extends the "View Card Details" process, modeled using the <<extend>> relationship.

## 4. Architectural Design (note: Problem analysis of next lab)

The requirements elicited in Lab 1 have a direct impact on the design challenge of Lab 2: Layered Architecture Design.

### 4.1 The Problem Statement

The problem is to design an architecture that satisfies the core Kanban requirements while enforcing separation of concerns and strict dependency rules. The Layered Architecture pattern is chosen as the first structural approach to address the Modifiability, Security, Reliability, Scalability, and Performance ASRs identified in Lab 1.

### 4.2 Impact of ASRs on Layered Architecture

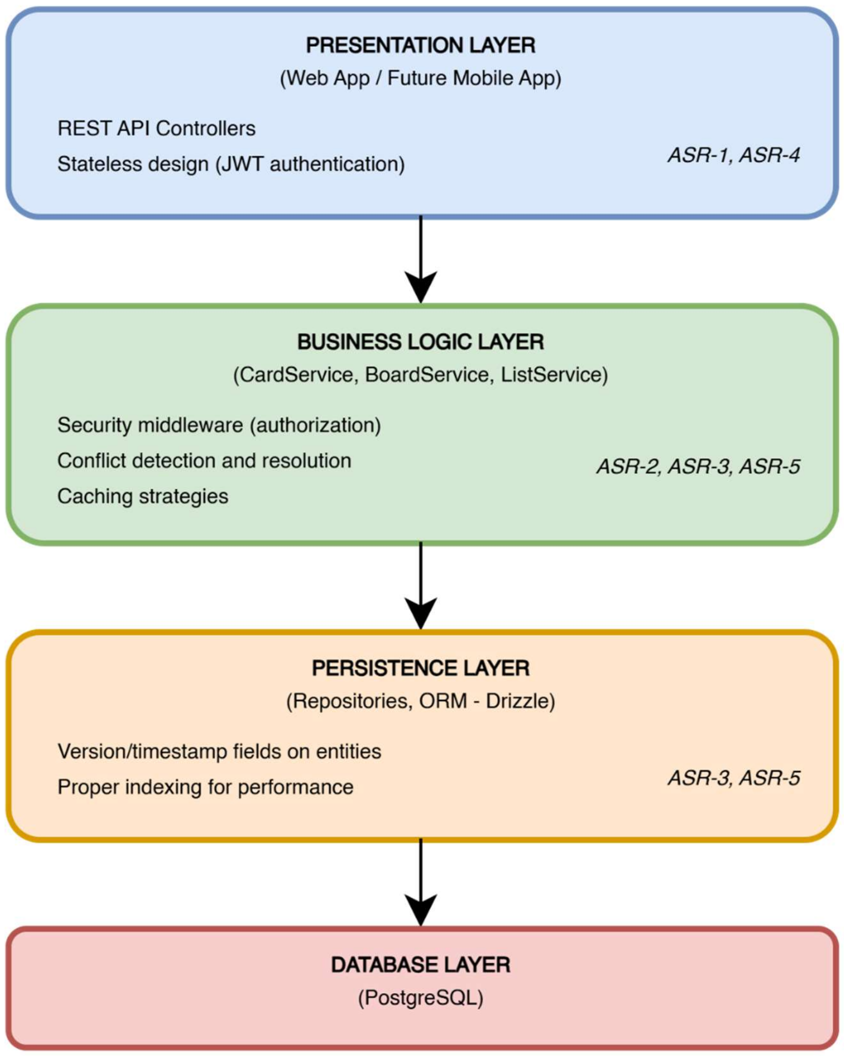
- ASR-1 (Modifiability) → Layered Structure: ASR-1 demands that changes to card types or external integrations remain isolated. The Layered Architecture inherently supports this by strictly defining four layers (Presentation, Business Logic, Persistence, Database) with downward dependencies.
    - *This means:* The Business Logic (where card movement rules reside) can be modified without impacting the Presentation layer (UI). This also enables future Mobile App development using the same backend services.

- ASR-2 (Security) → Business Logic Layer: ASR-2 requires rigorous authorization for board modifications.
    - In the Layered Architecture, this logic is enforced in the Business Logic Layer through a dedicated Security middleware.
    - This ensures that security checks are applied to the core services (e.g., CardService, BoardService), protecting them from unauthorized access regardless of whether the request originates from the Web App or a Mobile API.
- ASR-3 (Reliability) → Persistence Layer & Business Logic Layer: ASR-3 requires handling concurrent card modifications without data loss.
    - The Persistence Layer must implement version control or timestamp fields on Card entities to detect conflicts.
    - The Business Logic Layer must implement conflict detection and resolution mechanisms (optimistic locking) to handle simultaneous updates safely.
    - *This means:* When two users move the same card simultaneously, the system will detect the conflict and resolve it without corrupting data.
- ASR-4 (Scalability) → Presentation Layer: ASR-4 requires stateless API design to enable horizontal scaling.
    - The Presentation Layer must avoid storing server-side sessions, using token-based authentication (e.g., JWT) instead.
    - RESTful API patterns are enforced to support growth from 100 to 1,000 concurrent active boards through load balancing.
    - *This means:* Multiple server instances can handle requests without session synchronization overhead.
- ASR-5 (Performance) → Persistence Layer & Business Logic Layer: ASR-5 requires API response times under 300ms.
    - The Persistence Layer must implement proper database indexing strategies on frequently queried fields (e.g., board_id, column_id).
    - The Business Logic Layer may implement caching mechanisms for frequently accessed data (e.g., board metadata, user permissions).

- *This means:* Database queries are optimized and repeated data fetches are minimized through caching.

The next step (Lab 2) will use these requirements to formally define the four architectural layers and model the logical components (Controller, Service, Repository) and their interfaces within those layers.

## 4.3 Layered Architecture Overview


PRESENTATION LAYER
(Web App / Future Mobile App)
REST API Controllers
Stateless design (JWT authentication)
ASR-1, ASR-4
BUSINESS LOGIC LAYER
(CardService, BoardService, ListService)
Security middleware (authorization)
Conflict detection and resolution
Caching strategies
ASR-2, ASR-3, ASR-5
PERSISTENCE LAYER
(Repositories, ORM - Drizzle)
Version/timestamp fields on entities
Proper indexing for performance
ASR-3, ASR-5
DATABASE LAYER
(PostgreSQL)

## 5. Conclusion & Reflection

The requirements elicitation phase for KanbanFlow successfully defined the project scope through detailed Functional, Non-Functional, and Architecturally Significant Requirements. The UML Use Case Diagram provides a clear visual model of user interactions and the critical path. The identified ASRs especially those relating to Modifiability and Security directly necessitate the adoption of a structured pattern like the Layered Architecture for the subsequent design phase (next Lab 2), ensuring the architecture can sustainably support the required system behaviors.