Lab Report: Lab 2 - Layered Architecture Design (Logical View)

KanbanFlow

Contents

1. Cover Page

Title: Layered Architecture Design 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, 11 2025

2. Abstract/Summary

This lab focuses on designing the Layered Architecture for the KanbanFlow application, representing the system's static structure and component relationships. Building upon the requirements elicited in Lab 1, this design phase formally defines the four architectural layers and their responsibilities.

The architecture is driven by the Architecturally Significant Requirements (ASRs) identified previously, particularly Modifiability (ASR-1), Security (ASR-2), Reliability (ASR-3), Scalability (ASR-4), and Performance (ASR-5). The Layered Architecture pattern enforces strict separation of concerns and downward-only dependencies, enabling future extensibility (e.g., Mobile App development) while maintaining security and data integrity.

Key deliverables include the formal layer definitions, component identification for the core "Move Card" feature, and a UML Component Diagram modeling the logical view of the system.

3. Lab Specific Section: II. Layered Architecture Design

3.1 Activity Practice 1: Defining Layers and Responsibilities

Goal: Formally define the four layers and their roles within the KanbanFlow monolithic structure.

3.1.1 The Four Layers Definition

Adhering to the Layered Pattern Rule (a layer can only interact with the layer directly below it):

Layer	Purpose/Responsibility	Output/Artifact
1. Presentation Layer (UI/API)	Handles HTTP requests, JWT token authentication, session management, and serves the RESTful API endpoints. Routes incoming requests to appropriate controllers. Enforces stateless design (ASR-4).	Controllers: BoardController, CardController, ListController

2. Business Logic Layer (Service/Domain)	Contains the core business rules, validation logic, permission checks (ASR-2), optimistic locking conflict resolution (ASR-3), and transaction management. Orchestrates data access and enforces Kanban workflow rules.	Services: BoardService, CardService, ListService, AuthService
3. Persistence Layer (Data Access)	Responsible for mapping business objects to database entities and executing CRUD operations. Implements versioning/timestamps for conflict detection (ASR-3). Manages database indexing strategies (ASR-5).	Repositories: BoardRepository, CardRepository, ListRepository, UserRepository
4. Data Layer	The physical database storage system (PostgreSQL). Contains the actual tables, indexes, and constraints that persist the Kanban data.	Database Schema (Tables: boards, lists, cards, users, board_members, comments)

3.1.2 Data Flow Definition

Scenario: A Team Member moves a Card from "In Progress" to "Done" list.

Request Flow:

3.2 Activity Practice 2: Component Identification (Kanban Board Feature)

Goal: Break down the core Kanban features into concrete components that reside in the top three layers.

3.2.1 Component Identification for Core Features

Feature 1: Move Card (Critical - FR-03)

- Layer 1 (Presentation):
  - Component Name: CardController
  - Responsibility: Receives request PATCH /cards/{id}/move. Validates authentication token. Calls Business Logic Layer. Returns JSON response.
- Layer 2 (Business Logic):
  - Component Name: CardService
  - Responsibility: Receives card ID, target list ID, and position. Validates user has board membership. Checks version for optimistic locking (ASR-3). Updates card's list assignment.
- Layer 3 (Persistence):
  - Component Name: CardRepository
  - Responsibility: Translates request into database query with version check: UPDATE cards SET list_id = ?, position = ?, version = version + 1 WHERE id = ? AND version = ?. Returns updated entity.

Feature 2: Manage Board (FR-01)

- Layer 1 (Presentation):
  - Component Name: BoardController
  - Responsibility: Receives requests like POST /boards, GET /boards/{id}, PUT /boards/{id}. Handles request validation and routing.
- Layer 2 (Business Logic):
  - Component Name: BoardService
  - Responsibility: Enforces board creation rules. Validates user permissions for board access (ASR-2). Manages board visibility settings (FR-06).
- Layer 3 (Persistence):
  - Component Name: BoardRepository
  - Responsibility: Executes CRUD operations: INSERT INTO boards, SELECT * FROM boards WHERE id = ?, etc.
  -

Feature 3: Manage Lists (FR-01)

- Layer 1 (Presentation):
  - Component Name: ListController
  - Responsibility: Receives requests like POST /boards/{boardId}/lists, PUT /lists/{id}, DELETE /lists/{id}.
- Layer 2 (Business Logic):
  - Component Name: ListService
  - Responsibility: Validates list operations within board context. Enforces ordering logic. Checks user board membership.
- Layer 3 (Persistence):
  - Component Name: ListRepository
  - Responsibility: Translates to queries: INSERT INTO lists, UPDATE lists SET position = ?, DELETE FROM lists WHERE id = ?.

Feature 4: Manage Cards (FR-02)

- Layer 1 (Presentation):
  - Component Name: CardController
  - Responsibility: Receives requests for card CRUD: POST /lists/{listId}/cards, GET /cards/{id}, PUT /cards/{id}.
- Layer 2 (Business Logic):
  - Component Name: CardService
  - Responsibility: Enforces card validation rules (title required, due date format). Manages card details (labels, attachments, checklists, assignees).
- Layer 3 (Persistence):
  - Component Name: CardRepository
  - Responsibility: Executes card CRUD with proper indexing on list_id and board_id for performance (ASR-5).

Feature 5: Authentication & Authorization (ASR-2)

- Layer 1 (Presentation):
  - Component Name: AuthMiddleware
  - Responsibility: Intercepts all requests. Validates JWT tokens. Extracts user context and attaches to request.
- Layer 2 (Business Logic):
  - Component Name: AuthService
  - Responsibility: Handles login/logout logic. Generates JWT tokens (stateless for ASR-4). Validates user credentials.
  - Component Name: PermissionService

    - Responsibility: Validates board membership. Checks if user has required role (Member vs Admin).
- Layer 3 (Persistence):
    - Component Name: UserRepository
    - Responsibility: Queries user data: SELECT * FROM users WHERE email = ?.
    - Component Name: BoardMemberRepository
    - Responsibility: Queries membership: SELECT * FROM board_members WHERE board_id = ? AND user_id = ?.

3.2.2 Interface Definitions

3.2.2.1 Interfaces provided by Business Logic Layer to Presentation Layer:

ICardService Interface

```
interface ICardService {
  getCardDetails(cardId: string, userId: string): Promise<Card>
  createCard(listId: string, cardData: CreateCardDTO, userId: string):
Promise<Card>
  updateCard(cardId: string, cardData: UpdateCardDTO, userId: string):
Promise<Card>
  moveCard(cardId: string, targetListId: string, position: number, userId:
string, version: number): Promise<Card>
  deleteCard(cardId: string, userId: string): Promise<void>
}
```

IBoardService Interface

```
interface IBoardService {
  createBoard(boardData: CreateBoardDTO, userId: string): Promise<Board>
  getBoardDetails(boardId: string, userId: string): Promise<Board>
  updateBoard(boardId: string, boardData: UpdateBoardDTO, userId: string):
Promise<Board>
  inviteMember(boardId: string, inviteeEmail: string, userId: string):
Promise<void>
  updateVisibility(boardId: string, visibility: 'private' | 'public', userId:
string): Promise<Board>
}
```

IListService Interface

```
interface IListService {
  createList(boardId: string, listData: CreateListDTO, userId: string):
Promise<List>
  updateList(listId: string, listData: UpdateListDTO, userId: string):
Promise<List>
  reorderLists(boardId: string, listOrder: string[], userId: string):
Promise<List[]>
  deleteList(listId: string, userId: string): Promise<void>
}
```

IAuthService Interface

```
interface IAuthService {
  login(email: string, password: string): Promise<AuthToken>
  validateToken(token: string): Promise<User>
  logout(token: string): Promise<void>
}
```

3.2.2.2 Interfaces provided by Persistence Layer to Business Logic Layer:

ICardRepository Interface

```
interface ICardRepository {
  findById(cardId: string): Promise<CardEntity | null>
  findByListId(listId: string): Promise<CardEntity[]>
  create(cardEntity: CardEntity): Promise<CardEntity>
  update(cardEntity: CardEntity): Promise<CardEntity>
  updateWithVersionCheck(cardId: string, updates: Partial<CardEntity>,
expectedVersion: number): Promise<CardEntity | null>
  delete(cardId: string): Promise<void>
}
```

IBoardRepository Interface

```
interface IBoardRepository {
  findById(boardId: string): Promise<BoardEntity | null>
  findByUserId(userId: string): Promise<BoardEntity[]>
  create(boardEntity: BoardEntity): Promise<BoardEntity>
  update(boardEntity: BoardEntity): Promise<BoardEntity>
  delete(boardId: string): Promise<void>
}
```

IListRepository Interface

```
interface IListRepository {
  findById(listId: string): Promise<ListEntity | null>
  findByBoardId(boardId: string): Promise<ListEntity[]>
  create(listEntity: ListEntity): Promise<ListEntity>
  update(listEntity: ListEntity): Promise<ListEntity>
  delete(listId: string): Promise<void>
}
```

IBoardMemberRepository Interface

```
interface IBoardMemberRepository {
  findByBoardAndUser(boardId: string, userId: string): Promise<BoardMemberEntity
| null>
  findByBoardId(boardId: string): Promise<BoardMemberEntity[]>
  create(memberEntity: BoardMemberEntity): Promise<BoardMemberEntity>
  delete(boardId: string, userId: string): Promise<void>
}
```

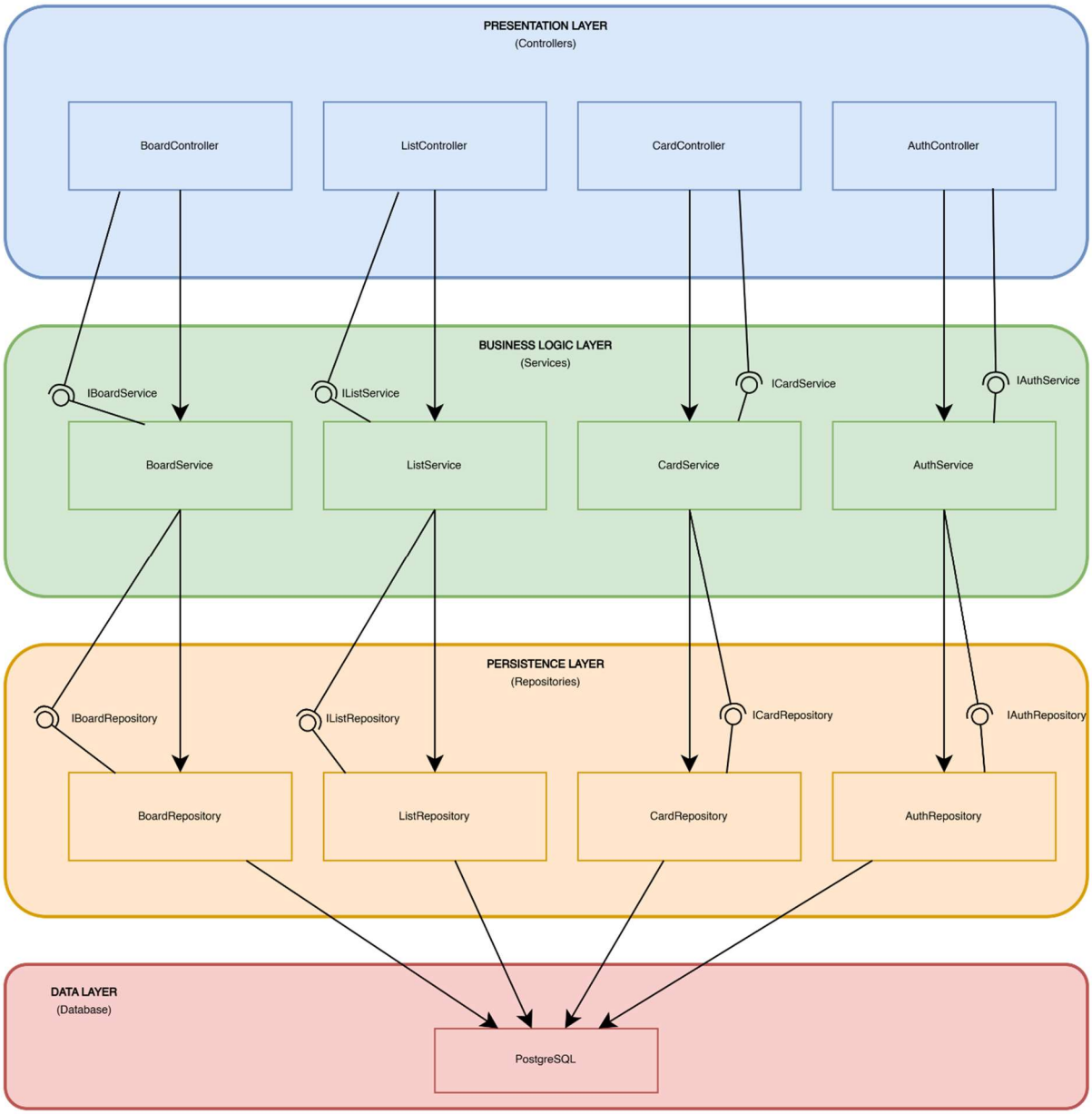
3.3 Activity Practice 3: Component Diagram Modeling

Goal: Create a visual model of the Layered Architecture using a UML Component Diagram.

3.3.1 Component Diagram Description

The UML Component Diagram for KanbanFlow illustrates the three application layers with their components and interfaces:

3.3.2 Diagram Key Elements



1. Layers: Three stacked rectangles representing Presentation, Business Logic, and Persistence layers, plus the Data Layer at the bottom.
2. Components: Each component is represented with the UML component notation (rectangle with two small rectangles on the left side).
3. Provided Interfaces (Lollipop ○—):
    - BoardService provides IBoardService

- ListService provides IListService
- CardService provides ICardService
- AuthService provides IAuthService
- Each Repository provides its respective interface

4. Required Interfaces (Socket ◠):
    - BoardController requires IBoardService
    - ListController requires IListService
    - CardController requires ICardService
    - Services require their corresponding Repository interfaces
5. Strict Dependency Rule: All dependency arrows point strictly downward, enforcing the Layered Architecture constraint that a layer can only depend on the layer directly below it.

3.3.3 ASR Mapping to Components

ASR	**Layer(s) Affected**	**Component(s)**	**Implementation Strategy**
ASR-1 (Modifiability)	All Layers	All Components	Strict layer separation enables future Mobile App to use same Services
ASR-2 (Security)	Business Logic	AuthService, PermissionService	All write operations validated through PermissionService before processing
ASR-3 (Reliability)	Persistence + Business	CardRepository, CardService	Version field in CardEntity; updateWithVersionCheck() method; conflict handling in CardService
ASR-4 (Scalability)	Presentation	AuthMiddleware	Stateless JWT authentication; no server-side sessions
ASR-5 (Performance)	Persistence	All Repositories	Database indexes on board_id, list_id, user_id; efficient query patterns

4. Conclusion & Reflection

The Layered Architecture design for KanbanFlow successfully translates the requirements from Lab 1 into a structured, maintainable system architecture. Key accomplishments include:

1. Layer Definition: Four distinct layers were formally defined with clear responsibilities, adhering to the strict downward dependency rule.
2. Component Identification: Core components for the Kanban features (Board, List, Card management) were identified across Presentation, Business Logic, and Persistence layers.
3. Interface Design: Well-defined interfaces between layers ensure loose coupling and enable future extensibility (e.g., Mobile App development as per ASR-1).
4. ASR Alignment: Each Architecturally Significant Requirement is addressed by specific components:
    - Modifiability through layer separation
    - Security through AuthService and PermissionService
    - Reliability through optimistic locking in CardRepository
    - Scalability through stateless JWT design
    - Performance through indexed database queries
5. UML Component Diagram: A visual model clearly illustrates the system's logical structure with proper interface notation (provided/required).

This foundation prepares the team for Lab 3: Implementation, where these components will be realized as actual code modules following the designed architecture.

Note: The Component Diagram should be created in draw.io (Diagrams.net) using proper UML notation as described in Activity Practice 3. The ASCII representation above serves as a guide for the visual diagram creation.