**Lab2: Layered Architecture Design**

Table of Contents

[1. Cover page 1](#_Toc216514789)

[Objectives 3](#_Toc216514790)

[Technologies & Tools 3](#_Toc216514791)

[Nozie Project Structure 3](#_Toc216514792)

[Activity Practice 1: Defining Layers and Responsibilities 3](#_Toc216514793)

[Step-by-Step Instructions 4](#_Toc216514794)

[Activity Practice 2: Identifying Components (Movie Detail Feature) 5](#_Toc216514795)

[Step-by-Step Instructions 5](#_Toc216514796)

[Activity Practice 3: Component Diagram Modeling 6](#_Toc216514797)

[Conclusion 9](#_Toc216514798)

# 1. Cover page

· **Title**: Elicitation and Modeling Requirements for the NoZie Online Streaming Application

· **Course Info:** Software Architecture

· **Student Details:**

|  |  |
| --- | --- |
| Họ Tên | MSSV |
| Nguyễn Duy Minh | 23010359 |
| Nghiêm Đức Việt | 23010636 |
| Nguyễn Văn Nhật | 23010887 |
| Đặng Nhất Nhất | 23010345 |

· **Date**: December 6, 2025

# **Objectives**

1. Understand the principles and constraints of the Layered Architecture pattern.
2. Define the four main layers and their responsibilities.
3. Identify key components within each layer for the Movie Detail feature.
4. Model the logical view using a UML Component Diagram.

# **Technologies & Tools**

|  |  |  |
| --- | --- | --- |
| **Tool** | **Purpose** | **Setup Instructions** |
| draw.io (Diagrams.net) | Create professional UML Component Diagrams | Access online via browser or use the desktop app |
| Whiteboard/Pen & Paper | Initial brainstorming and sketching of layers/components | Standard brainstorming tools |

# **Nozie Project Structure**

The Nozie project is organized according to a Feature-First architecture combined with Layered Architecture:

src/

└── lib/

├── app/ # App configuration

├── core/ # Core components (shared)

│ ├── repositories/ # Core repositories

│ ├── services/ # Core services

│ ├── models/ # Core models

│ ├── widgets/ # Shared widgets

│ └── utils/ # Utilities

├── features/ # Feature modules

│ ├── movie/ # Movie feature

│ │ ├── presentation/ # Layer 1: UI (Screens, Widgets)

│ │ ├── services/ # Layer 2: Business Logic

│ │ └── data/ # Layer 3: Repositories

│ ├── auth/ # Authentication feature

│ ├── home/ # Home feature

│ └── ...

└── routes/ # Routing configuration

Each feature follows a 3-layer structure:

* **presentation/**: Presentation Layer (UI)
* **services/**: Business Logic Layer
* **data/**: Persistence Layer

# **Activity Practice 1: Defining Layers and Responsibilities**

**Goal:** Formally define the four layers and their roles within the monolithic structure of the Nozie application (Movie Streaming App).

### Step-by-Step Instructions

#### 1. Define the Four Layers

Document the name and primary purpose of each layer, strictly adhering to the Layered Pattern Rule (A layer can only interact with the layer directly beneath it).

|  |  |  |
| --- | --- | --- |
| **Layer** | **Purpose/Responsibility** | **Output/Artifact** |
| **1. Presentation Layer (UI)** | Handles user interactions, routing, state management (Riverpod), and rendering the user interface (Flutter Widgets). | Screens, Widgets (e.g., MovieDetailScreen) |
| **2. Business Logic Layer (Service/Domain)** | Contains core business rules, validation logic, and business logic management. Coordinates data access. | Services, Notifiers (e.g., MovieWatchService, RatingsService) |
| **3. Persistence Layer (Data Access)** | Responsible for mapping business objects to database entities and executing CRUD operations (Create, Read, Update, Delete). | Repositories (e.g., MovieRepository) |
| **4. Data Layer** | Physical database storage system (Firebase Firestore, Firebase Storage, Firebase Auth). | Firestore Collections, Documents |

#### 2. Define Data Flow

Outline the typical request flow for a user viewing the movie detail page.

**Processing Flow:**

User Action (Tap on Movie)

→ Layer 1 (Presentation: MovieDetailScreen)

→ Layer 2 (Business Logic: MovieWatchService, RatingsService)

→ Layer 3 (Persistence: MovieRepository)

→ Layer 4 (Data: Firebase Firestore)

→ Layer 3 (Persistence: MovieRepository)

→ Layer 2 (Business Logic: Services)

→ Layer 1 (Presentation: MovieDetailScreen)

→ UI Update (Display Movie Details)

**Detailed Description:**

* **User Action:** User taps on a movie in the list, navigating to /movie/{id}.
* **Layer 1 (Presentation):** MovieDetailScreen receives the movieId from the route and uses Riverpod providers to fetch data.
* **Layer 2 (Business Logic):**
  + MovieWatchService checks access rights (free movie, purchased, or subscribed).
  + RatingsService handles rating logic.
* **Layer 3 (Persistence):** MovieRepository converts the request into a Firestore query (get movie document by ID).
* **Layer 4 (Data):** Firebase Firestore executes the query and returns the movie data.
* **Layer 3 → 2 → 1:** Data is transformed and passed back up through the layers.
* **UI Update:** The Presentation layer updates the UI to display movie info, trailer, ratings, etc.

# Activity Practice 2: Identifying Components (Movie Detail Feature)

**Goal:** Decompose the Movie Detail feature into specific components located within the three layers above.

### Step-by-Step Instructions

#### 1. Identify Components

For the "View Movie Details" feature, identify specific software components (classes or modules) that will reside in Layer 1, 2, and 3.

##### Layer 1 (Presentation):

* **Component Name:** MovieDetailScreen
* **Location:** lib/features/movie/presentation/screens/movie\_detail\_screen.dart
* **Responsibility:**
  + Receive movieId from route parameters (/movie/{id}).
  + Use Riverpod providers to fetch movie data.
  + Render UI: poster, movie info, trailer, ratings, similar movies.
  + Handle user interactions (play video, add to wishlist, purchase).
  + Call the Business Logic Layer via providers.

##### Layer 2 (Business Logic):

* **Component Name:** MovieWatchService
* **Location:** lib/features/movie/services/movie\_watch\_service.dart
* **Responsibility:**
  + Check movie access rights (free, purchased, or subscribed).
  + Handle movie watching logic (increment view count, add watch history).
  + Manage business rules related to watching movies.
* **Component Name:** RatingsService
* **Location:** lib/features/movie/services/ratings\_service.dart
* **Responsibility:**
  + Handle movie rating logic.
  + Calculate average ratings.
  + Manage user ratings.

##### Layer 3 (Persistence):

* **Component Name:** MovieRepository
* **Location:** lib/core/repositories/movie\_repository.dart
* **Responsibility:**
  + Convert requests into Firestore queries (e.g., collection('movies').doc(id).get()).
  + Execute CRUD operations on Firestore.
  + Map between Movie models and Firestore documents.
  + Return Stream/Future to the Business Logic Layer.
  + Handle complex queries (filter by genre, search, similar movies).

#### 2. Define Interfaces

Identify the main interface (method signatures) provided by the Business Logic Layer to the Presentation Layer, and by the Persistence Layer to the Business Logic Layer.

MovieWatchService Interface (for Layer 1):

Future<bool> hasAccess(String movieId)

Future<void> incrementView(String movieId)

Future<void> addWatchHistory(String movieId)

**Description:**

* hasAccess: Checks if the user has the right to watch the movie.
* incrementView: Increments the view count of the movie.
* addWatchHistory: Adds the movie to the user's watch history.

MovieRepository Interface (for Layer 2):

Future<Movie?> getMovieDetail(String id)

Stream<MovieItem?> streamById(String id)

Stream<List<MovieItem>> streamSimilar(String movieId, {int limit = 10})

**Description:**

* getMovieDetail: Retrieves movie details by ID (async).
* streamById: Streams movie info by ID (real-time updates).
* streamSimilar: Streams a list of similar movies.

# Activity Practice 3: Component Diagram Modeling

Request Processing Flow

A diagram of a company

AI-generated content may be incorrect.

Layered Architecture

A diagram of a software company

AI-generated content may be incorrect.

#### Component Diagram with Interfaces

A diagram of a business logistic

AI-generated content may be incorrect.

# Conclusion

Lab 2 has helped the group understand and apply the Layered Architecture pattern to the Nozie application (Movie Streaming App). By:

1. Clearly defining the four layers and their responsibilities:
   * **Presentation Layer:** Flutter Screens and Widgets using Riverpod for state management.
   * **Business Logic Layer:** Services handling business rules (MovieWatchService, RatingsService).
   * **Persistence Layer:** Repositories accessing Firebase Firestore.
   * **Data Layer:** Firebase services (Firestore, Storage, Auth).
2. Identifying specific components within each layer for the Movie Detail feature:
   * MovieDetailScreen (Presentation)
   * MovieWatchService, RatingsService (Business Logic)
   * MovieRepository (Persistence)
   * Firebase Firestore (Data)
3. Modeling the architecture using a UML Component Diagram with clear interfaces and dependencies.

The group now has a solid foundation to implement the system according to a layered architecture, ensuring modularity, maintainability, and scalability. This architecture is well-suited for a Flutter app using Riverpod state management and a Firebase backend.