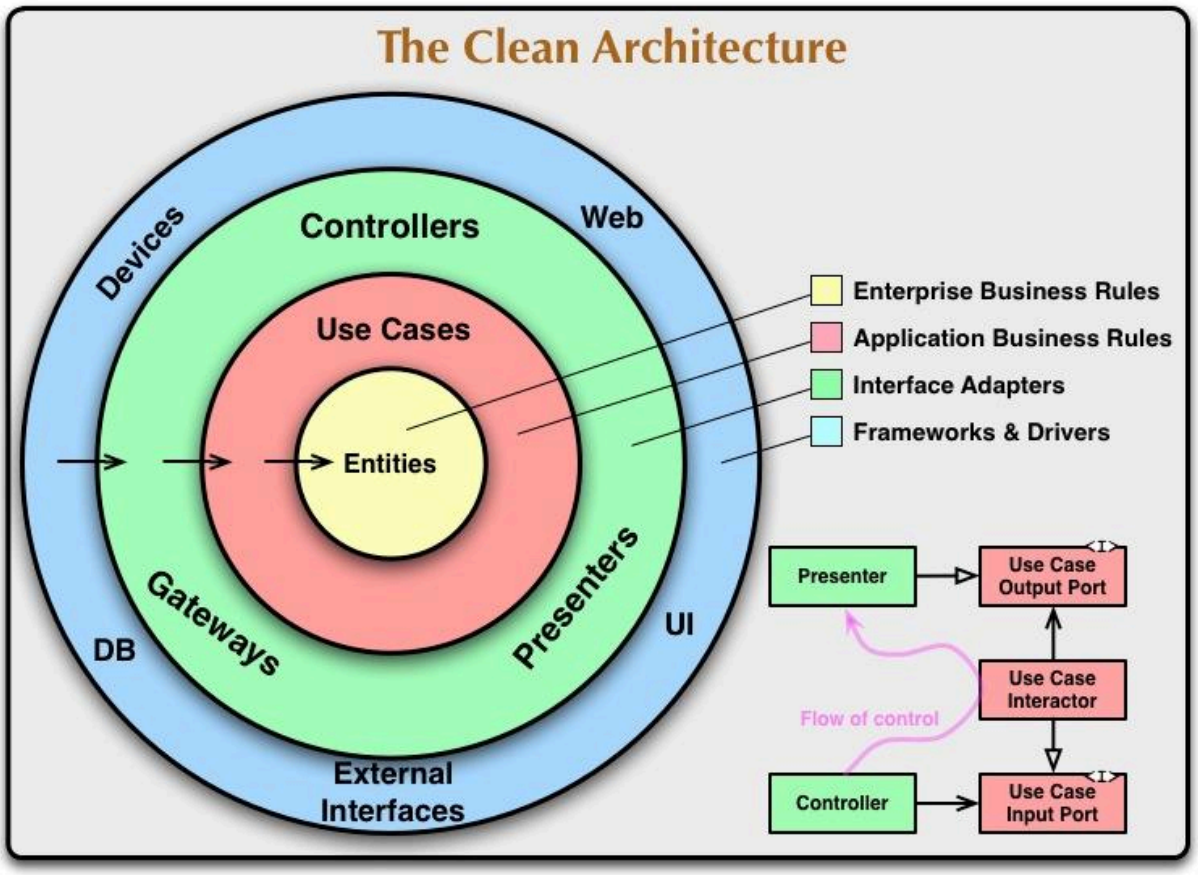
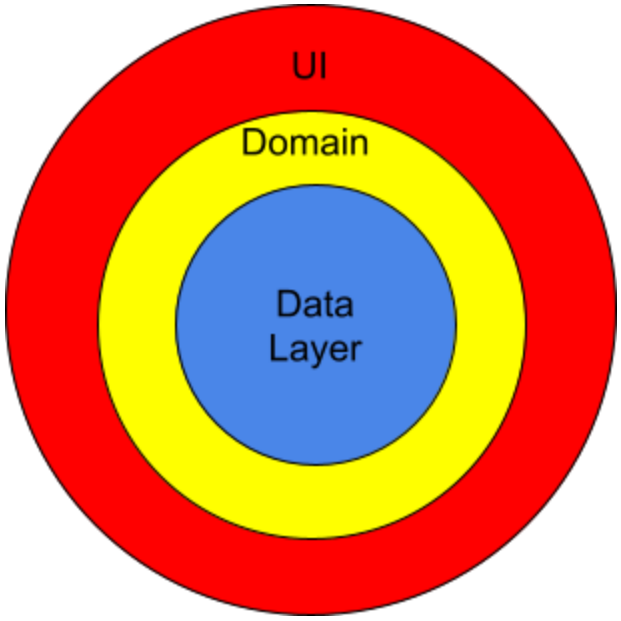
Clean Architecture is all about the separation of Concerns or Division of Responsibility in code.

In Clean Architecture pattern separation of concern is Implemented in 4 layers.



But in Android Clean Architecture is Implemented in 3 Layers.

1. Presentation Layer or UI layer
2. Domain or Business Layer
3. Data Layer



As you see in this Diagram Data layer is Innermost layer.

Key Requirements to follow to implement Clean Architecture pattern ?

1. Interaction between Layers
    a. Presentation layer need data to display and data layer holds , arrange or create whatever data for the application,
        i. But the Presentation Layer means (Screens, ViewModels etc) can't communicate with the data layer directly.
        ii. All the data UI needs are Provided through domain Layer or we can say Repo interfaces, use cases etc.
        iii. And Data layer Implement those interfaces or use cases to send the data needed.
        iv. All use cases are injected into the viewmodel.

1. Project Sample one
    a. Pagination with Cache??
        i.
        ii.

```
@Composable
fun ItemRow(item: SeriesEntity) {
```

iii. This composable function depend on Data layer class called SeriesEntity which will break the separation of Concern here similarly.

iv.
```
val seriesFlow = pager.flow.cachedIn(viewModelScope)
```

1. This pager return Flow <PagingData<SeriesEntity>> which breaks again series flow should not depend on data layer class.

v. Now let's fix and refactor this code adding new layer called domain layer

Important 👍

👋 Domain layer Should not depend on any concrete Implementation Details, Like Paging, Databases and Network Libraries.

Best Practises 🙂

- Dependency injection: Use Hilt/Dagger to manage dependencies
- Coroutines: Handle asynchronous operations efficiently
- Single Responsibility: Each component should have one reason to change
- Dependency Rule: Dependencies only point inward
- Interface Segregation: Define specific interfaces for different use cases

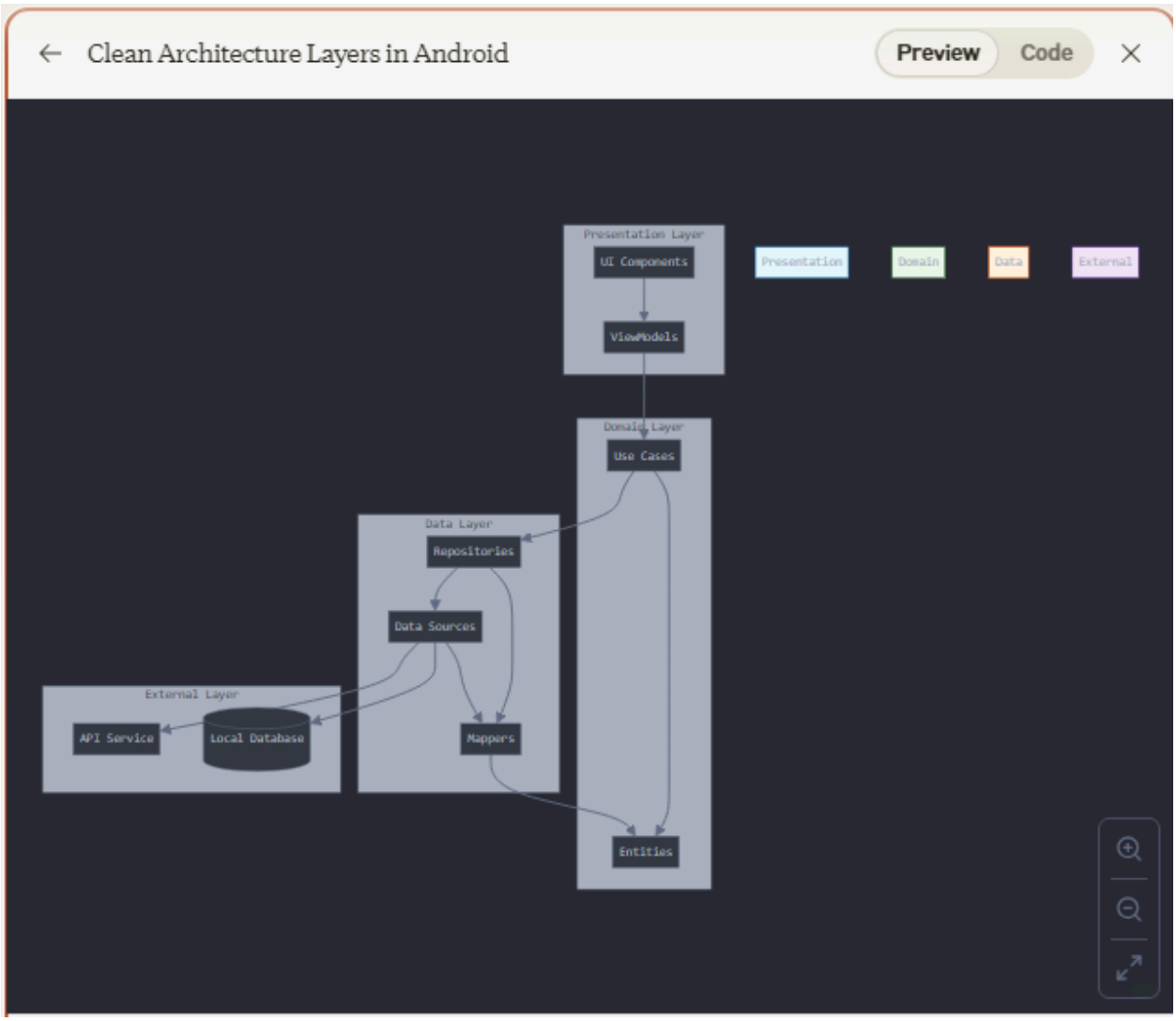
Responsibility of each layer 😀

1. Domain Layer (Core Business Logic)
    - Entities: Pure Kotlin data classes representing core business objects
    - Use Cases: Single-responsibility classes containing business logic
    - Repository Interfaces: Abstract definitions of data operations

2. Data Layer (Data Operations)

- Repository Implementations: Concrete implementations of repository interfaces
- Data Sources: Classes handling data from different sources (API, database)
- Mappers: Classes that convert between data and domain models

3. Presentation Layer (UI)
    - UI Components: Activities, Fragments, Composables
    - ViewModels: Handle UI logic and state management
    - UI Models: Data classes specifically formatted for UI


Clean Architecture Layers in Android
Preview
Code
Presentation Layer
UI Components
ViewModels
Presentation
Domain
Data
External
Domain Layer
Use Cases
Data Layer
Repositories
Data Sources
External Layer
API Service
Local Database
Mappers
Entities