**The Clean Architecture**

Clean Architecture is a software design philosophy introduced by Robert C. Martin (also known as Uncle Bob) that emphasises the separation of concerns and the independence of architectural elements within a software system. It provides a set of principles and guidelines for structuring applications in a way that promotes maintainability, testability, and scalability.

The main components of Clean Architecture are:

**1. Data Layer:**

* The Data Layer is responsible for managing data and interacting with external data sources such as databases, web services, and local storage.It typically includes
* components such as data models, repositories, data access objects (DAOs), and network clients.
* The primary goal of the Data Layer is to abstract the details of data storage and retrieval, allowing higher layers to interact with data consistently and efficiently.

**2. Domain Layer:**

* The Domain Layer contains the business logic and rules that govern the behaviour of the application.
* It defines the core entities, use cases, and business operations of the system, independent of any specific implementation details or external dependencies.
* The Domain Layer is the heart of the application and should be agnostic to any external frameworks, libraries, or user interfaces.

**3. Presentation Layer:**

* The Presentation Layer is responsible for presenting data to the user and handling user interactions.
* It includes components such as user interface (UI) screens, view models, controllers, and presenters.
* The Presentation Layer communicates with the Domain Layer to retrieve and display data, but it should not contain any business logic or domain-specific rules.

**4. Dependency Injection (DI):**

* Dependency Injection is a technique for managing dependencies between different components of an application.
* It involves providing objects (dependencies) to a component rather than having the component create or manage its dependencies directly.
* DI promotes loose coupling between components, making the application more modular, testable, and maintainable.

In Clean Architecture, these layers are organised hierarchically, with each layer depending only on the layers beneath it. This creates a clear separation of concerns and allows for changes to be made in one layer without affecting other layers. Additionally, the architecture places a strong emphasis on the use of interfaces and abstractions to decouple components and facilitate dependency inversion. Overall, Clean Architecture provides a flexible and scalable approach to building software systems that are robust, maintainable, and adaptable to change.

The clean architecture layer:

* **Structure A**

**Features ->**

**1> Data**

1.1 Source

1.1.1 Local

Dao, database, entity

1.1.2 Remote

1.1.2.1 PoJo

data classes

1.1.2.1 Service

Retrofit instance, API service.

1.2 Reporsitory\_Implement

LoginRepositoryImplement

RegisterRepositoryImplement

**2> Domain**

2.1 Model

data classes

2.2 Repository (Interface)

LoginRepository

RegisterRepository

2.3 Use Case

LoginUseCase

RegisterUseCase

**3> presentation**

3.1 login

loginViewScreen

LoginViewModel

LoginState

3.2 Register

RegisterViewScreen

RegisterViewModel

RegisterState

**4> Dependency Injection**

4.1 Component

AppComponent (class)

4.2 Module

AppModule (Object)

* **Structure B**

**Features ->**

1. **Login**

**1> Data**

1.1 Source

1.1.1 Local

Dao, database, entity

1.1.2 Remote

1.1.2.1 PoJo

Data classes

1.1.2.1 Service

Retrofit instance, API service.

1.2 Reporsitory\_Implement

LoginRepositoryImplement

**2> Domain**

2.1 Model

Data classes

2.2 Repository (Interface)

LoginRepository

2.3 Use Case

LoginUseCase

**3> presentation**

loginViewScreen

LoginViewModel

LoginState

**4> Dependency Injection**

4.1 Component

AppComponent (class)

4.2 Module

AppModule (Object)

1. **Register**

**1> Data**

1.1 Source

1.1.1 Local

Dao, database, entity

1.1.2 Remote

1.1.2.1 PoJo

Data classes

1.1.2.1 Service

Retrofit instance, API service.

1.2 Reporsitory\_Implement

RegisterRepositoryImplement

**2> Domain**

2.1 Model

Data classes

2.2 Repository (Interface)

RegisterRepository

2.3 Use Case

RegisterUseCase

**3> presentation**

RegisterViewScreen

RegisterViewModel

RegisterState

**III) Dependency Injection**

4.1 Component

AppComponent (class)

4.2 Module

AppModule (Object)

project/

│

├── app/

│ ├── data/

│ │ ├── source/

│ │ │ ├── local/

│ │ │ │ ├── dao/

│ │ │ │ │ └── UserDao.kt

│ │ │ │ └── database/

│ │ │ │ └── AppDatabase.kt

│ │ │ │

│ │ │ ├── remote/

│ │ │ │ ├── api/

│ │ │ │ │ └── ApiService.kt

│ │ │ │ └── models/

│ │ │ │ └── ResponseModel.kt

│ │ │ │

│ │ │ └── network/

│ │ │ └── RetrofitInstance.kt

│ │ │

│ │ ├── repositories\_implement/

│ │ │ ├── LoginRepository.kt

│ │ │ └── RegisterRepository.kt

│ │ │

│ │ └── DependencyProvider.kt

│ │

│ ├── domain/

│ │ ├── model/

│ │ │ ├── User.kt

│ │ │ └── RegistrationData.kt

│ │ │

│ │ ├── repository/

│ │ │ ├── LoginRepository.kt

│ │ │ └── RegisterRepository.kt

│ │ │

│ │ ├── usecase/

│ │ │ ├── LoginUseCase.kt

│ │ │ └── RegisterUseCase.kt

│ │ │

│ │ └── DependencyProvider.kt

│ │

│ └── presentation/

│ ├── login/

│ │ ├── LoginActivity.kt

│ │ ├── LoginViewModel.kt

│ │ └── LoginState.kt

│ │

│ └── register/

│ ├── RegisterActivity.kt

│ ├── RegisterViewModel.kt

│ └── RegisterState.kt

│

└── di/

├── component/

│ └── AppComponent.kt

└── module/

└── AppModule.kt

A black screen with white text

Description automatically generated