Repository vai servir apenas para interfaciar os dados para o domain onde tem nossas regras de negocios

O viewmodel vai ficar somente com a responsáblidade de gerenciar o estado da nossa UI...

Pode ser um Mobx, provider, etc.

Core / domain

Infra

data

Entity

Responsável para pegar os dados externos: API, youtube, etc

Repository

Controller

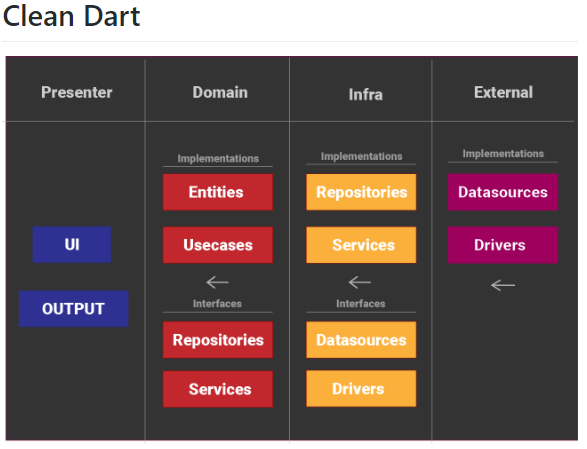
ViewModel

UI

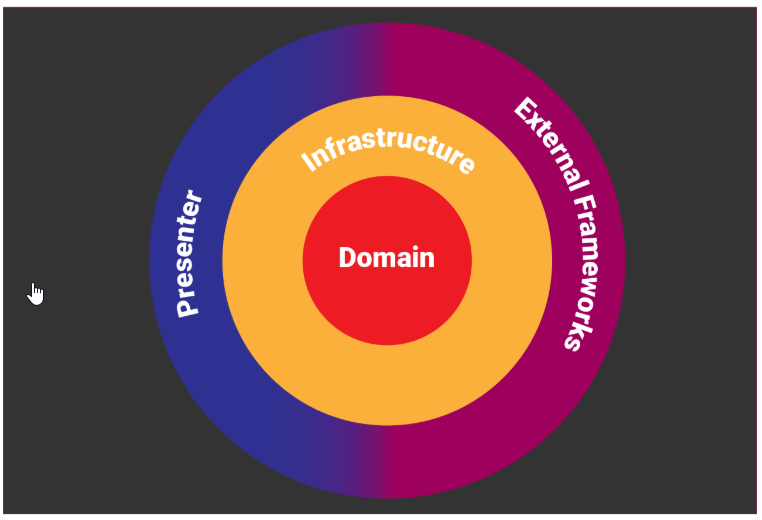
Errors

Presenter

Todo nosso domain: regra de negocio, entidades, etc.



Usando o Flutter como exemplo teremos então quatro camadas mantendo a “Arquitetura de Plugin”, com foco principal no Domínio da Aplicação, camada esta que hospeda as 2 Regras de Negócio principais, estamos falando das **Entidades** e dos **Casos de Uso**



**Presenter**

A Camada **Presenter** fica responsável por declarar as entradas, saídas e interações da aplicação.

Usando o Flutter como exemplo, hospedaremos os Widgets, Pages e também Alguma Gerência de Estado, já no backend como exemplo, seria nesta camada onde colocaríamos os Handlers ou Commands da nossa API.

**Domain**

A camada de **Domain** hospedará as Regras de Negócio Corporativa(Entity) e da Aplicação(Usecase).

Nossas Entidades devem ser Objetos simples podendo conter regras de validação dos seus dados por meio de funções ou ValueObjects. **A Entidade não deve usar nenhum Objeto das outras camadas.**

Os **Casos de Uso** devem executar a lógica necessária para resolver o problema. Se o **Caso de Uso** precisar de algum acesso externo então esse acesso deve ser feito por meio de contratos de interface que serão implementados em uma camada de mais baixo nível.

Na camada **Domain** dever ser responsável apenas pela execução da lógica de negócio, não deve haver implementações de outros Objetos como Repositories ou Services dentro do **Domain**.

Tomando um Repository como exemplo, teremos que ter apenas o contrato de interfaces(Abstrações) e a responsabilidade de implementação desse objeto deverá ser repassado a outra camada mais baixa.

**Infrastructure (Infra)**

Está camada dá suporte a camada **Domain** implementando suas interfaces. Para isso, essa camada se propõem a adaptar os dados externos para que possa cumprir os contratos do domínio.

Muito provavelmente nessa camada iremos implementar alguma interface de um Repository ou Services que pode ou não depender de dados externos como uma API ou acesso a algum Hardware como por exemplo Bluetooth.

Para que o Repository possa processar e adaptar os dados externos devemos criar contratos para esses serviços visando passar a responsabilidade de implementação para a camada mais baixa da nossa arquitetura.

Como sugestão, iremos criar objetos de **DataSource** quando quisermos acessar um dado externo, uma BaaS como Firebase ou um Cache Local usando SQLite por exemplo. Outra sugestão seria criar objetos denominados **Drivers** para interfacear a comunicação com algum Hardware do dispositivo.

Os acessos externos como Datasources e Drivers devem ser implementados por outra camada, ficando apenas os Contratos de Interface nesta camada de Infra.

**External**

Aqui começaremos a implementar os acessos externos e que dependem de um hardware, package ou acesso muito específico.

Basicamente a camada External deve conter tudo aquilo que terá grandes chances de ser alterado sem que o programador possa intervir diretamente no projeto.

No Flutter por exemplo, para cache local usamos o SharedPreferences, mas talvez em alguma estágio do projeto a implementação do SharedPreferences não seja mais suficiente para a aplicação e deve ser substituída por outro package como Hive, nesse ponto a única coisa que precisamos fazer é criar uma nova classe, implementando o Contrato esperado pela camada mais alta (que seria a **Infra**) e implementarmos a Lógica usando o Hive.

Um outro exemplo prático seria pensar em um Login com Firebase Auth, porém outro produto deseja utilizar um outro provider de autenticação. Bastaria apenas implementar um datasource baseado no outro provider e “Inverter a Dependência” substituindo a implementação do Firebase pela nova quando for necessário.

Os Datasources devem se preocupar apenas em “descobrir” os dados externos e enviar para a camada de Infra para serem tratados.

Da mesma forma os objetos **Drivers** devem apenas retornar as informações solicitadas sobre o Hardware do Device e não devem fazer tratamento fora ao que lhe foi solicitado no contrato.