Demonstração projeto desafio Hexagon

Dependências importantes do projeto

Dagger Hilt

Dagger Hilt é uma biblioteca de injeção de dependência para Android, baseada no Dagger 2, que simplifica significativamente o processo de injeção de dependência.
Você pode usá-la para injetar dependências em diferentes partes do seu aplicativo, o que torna o código mais limpo, modular e testável.

Jetpack Compose

Jetpack Compose é o kit de ferramentas moderno e declarativo do Google para a criação de interfaces de usuário em aplicativos Android. Ele simplifica o desenvolvimento

de interfaces de usuário, permitindo que você construa interfaces de forma mais rápida e intuitiva, utilizando código Kotlin mais limpo e conciso em comparação com as abordagens tradicionais baseadas em XML.

Room

Room é uma biblioteca de persistência do Android que oferece uma camada de abstração sobre o SQLite, permitindo que você trabalhe com bancos de dados de forma mais
fácil e eficiente. Com o Room, você pode definir facilmente esquemas de banco de dados usando anotações Java/Kotlin e realizar operações CRUD (criar, ler, atualizar, excluir) de forma mais simples e segura.

ConstraintLayout

ConstraintLayout é um layout manager poderoso e flexível do Android que permite criar layouts complexos e responsivos de forma mais eficiente do que os layouts tradicionais,

como RelativeLayout e LinearLayout. Ele facilita a criação de interfaces de usuário com designs flexíveis e adaptação automática a diferentes tamanhos de tela e orientações de dispositivo.

Arquitetura

Como se pode ver, esse projeto foi construido seguindo um princípio orientado à features.

O Core ficando responsável por gerenciar a árvore de navegação, a injeção de dependências,
e a abstração do database para acesso às fontes de dados armazenadas localmente com Room, outras futuras implementações a nível de projeto, também entram nessa camada.

As camadas ListEmployee e EditEmployee representam as 2 principais features do app, a exibição de todos os empregados numa tela
com filtro de atividade ou inatividade e a livre edição de um empregado já salvo ou novo.

Clean Architecture é utilizado em cada feature, incluindo o Core, separando cada uma das features

em 3 camadas (data, domain e presentation).

O padrão MVVM também é utilizado nas features em conjunto com clean, separando as regras de negócio, dos acessos ás fontes de dados da apresentação.

A abordagem utilizada, também foi muito inspirada numa solução proposta por Levinzom Roman para maior desacoplamento das regras
de UI da ViewModel, problema muito encontrado no compose devido à remoção quase que total das regras de UI das views graças ao
novo formato declarativo e reativo.

Ideias utilizadas

Clean Architecture

Clean Architecture é um padrão de arquitetura de software que promove a separação de preocupações e a independência de frameworks. Ele organiza o código em camadas distintas (como a camada de apresentação, camada de domínio e camada de dados), cada uma com sua própria

responsabilidade e dependências unidirecionais. Isso facilita a manutenção, teste e evolução do aplicativo ao longo do tempo.

MVVM (Model-View-ViewModel)

MVVM é um padrão de arquitetura de software que separa a lógica de apresentação da lógica de negócios e da lógica de persistência de dados.
Ele consiste em três componentes principais: Model (que contém a lógica de negócios e os dados), View (que representa a interface do usuário) e ViewModel (que atua como intermediário entre o Model e a View, fornecendo os dados necessários para a interface do usuário e manipulando eventos do usuário).
MVVM facilita a manutenção, teste e reutilização do código.

Abordagem de criação de nova camada Coordinator e de contratos de estado e ação

Com essa proposta de arquitetura, respeitando um fluxo unidirecional de dados, utilizo os conceitos padrão do MVVM
mas adaptados para funcionar de maneira eficaz, testável e escalável com abstrações das ações e do estado, utilizando
fluxos de dados para representar os estados, consigo uma maior reatividade e com a integração das ações, num só objeto, consigo
reduzir a quantidade de parâmetros passados para os componentes, melhorando a legibilidade e escalabilidade do código.
Não posso deixar de mencionar também o Coordinator, que embora nesse projeto tenha ficado com poucas responsabilidades,
principalmente devido à baixa complexidade dessa solução, tem um papel fundamental em dividir regras referentes à
reatividade da UI com o estado salvo e observado na ViewModel.

Fonte: https://levinzon-roman.medium.com/jetpack-compose-ui-architecture-a34c4d3e4391