Android TechTalk
Cássio Bruzasco

Uso de arquitetura moderna em Android

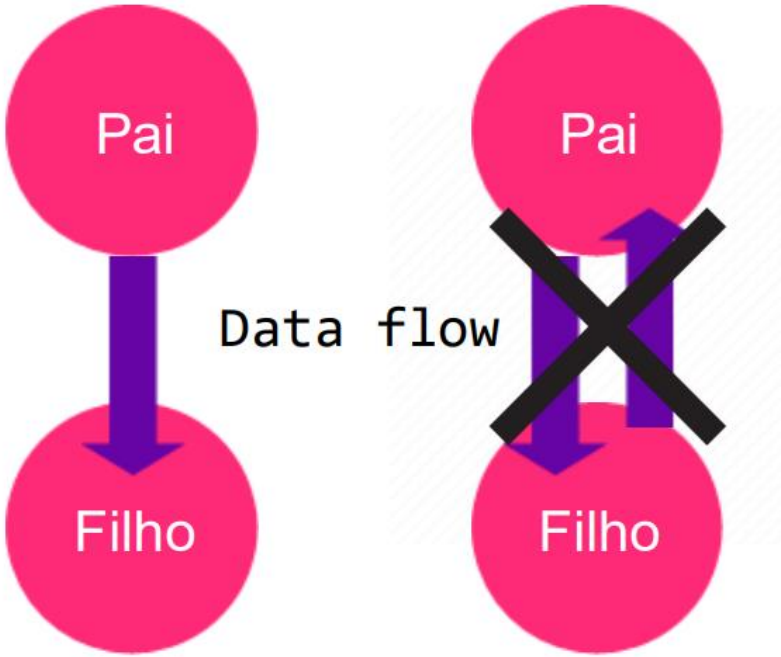
Credenciais

- 10 anos trabalhando com desenvolvimento Android
- *Product Owner, Tech lead, Digital Architect e Software Engineer*
- 5 anos de Venturus entre idas e vindas

MVVM

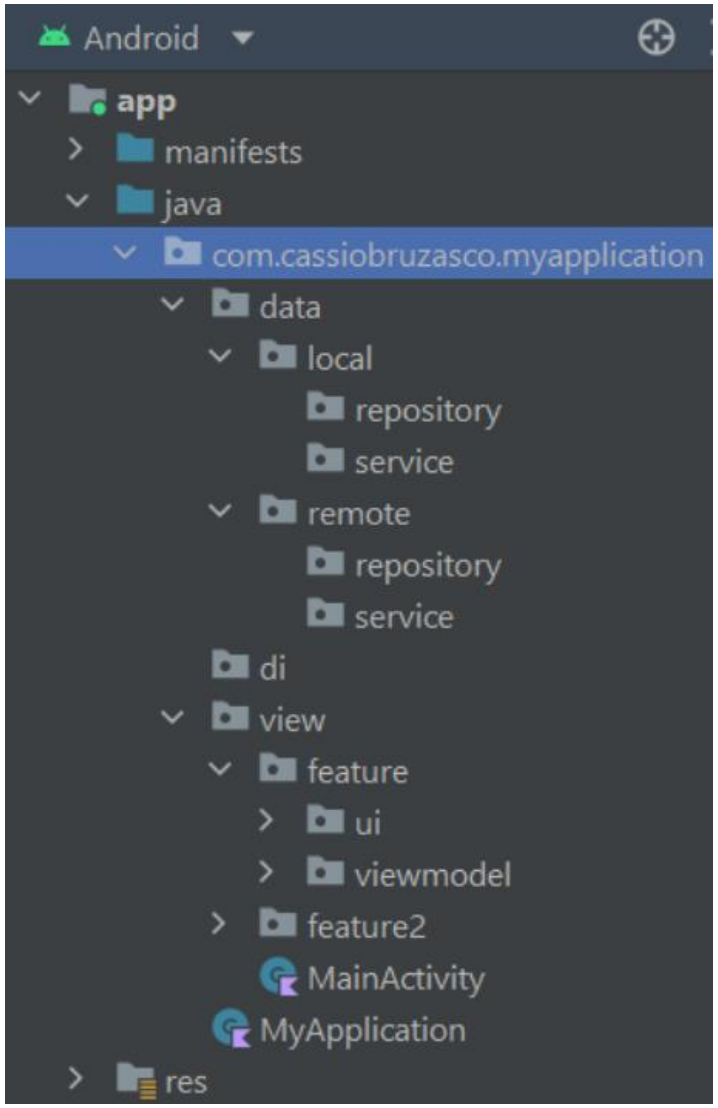
Model – View - ViewModel

- Manutenibilidade
- Separar responsabilidades
- *Unidirectional Data Flow*
- Entre outros motivos mais profundos...

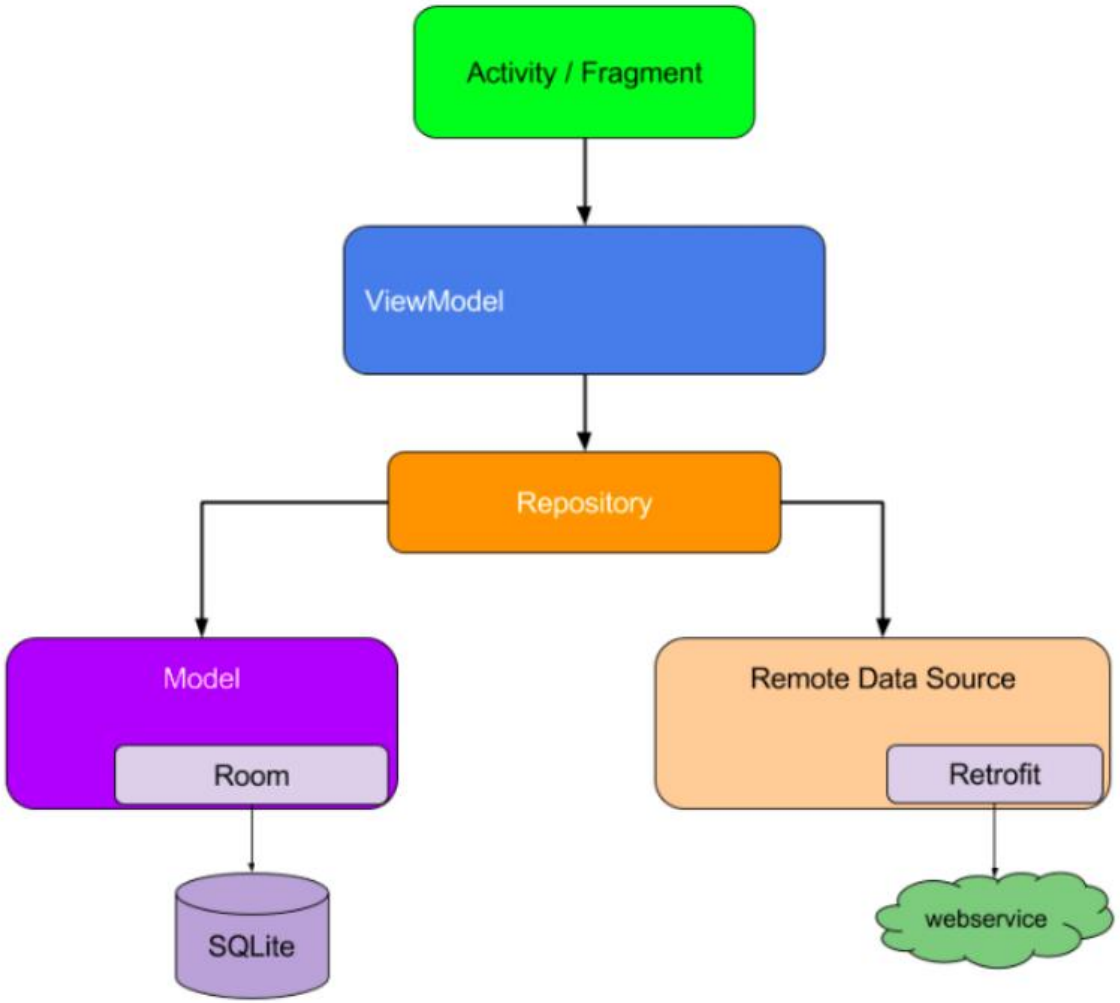


Model – View - ViewModel

- **data** – tudo relacionado a dados (Camada Model)
- **local** – requisições e modelar o banco de dados interno
- **remote** – requisições e modelar serviços externos (API)
- **repository** – interface/implementação que faz a comunicação do DB ou API com a camada do VM
- **di** – módulos para implementação do injetor de dependência
- **view** – pacote de apresentação (ou unificar por *feature*)
- **viewmodel** – pacote para relacionar seu *fragment/activity* com um VM



Model – View - ViewModel



Diving into ViewModel

ViewModel

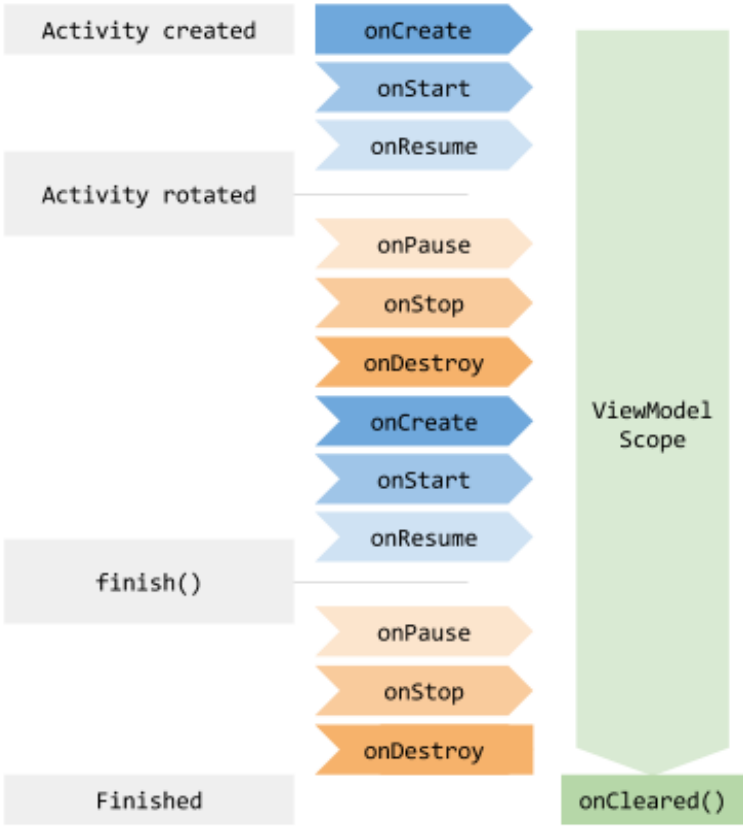
•Persistir o *"current state"* da UI

• Acessar lógica de negócio

• *Lifecycle aware*

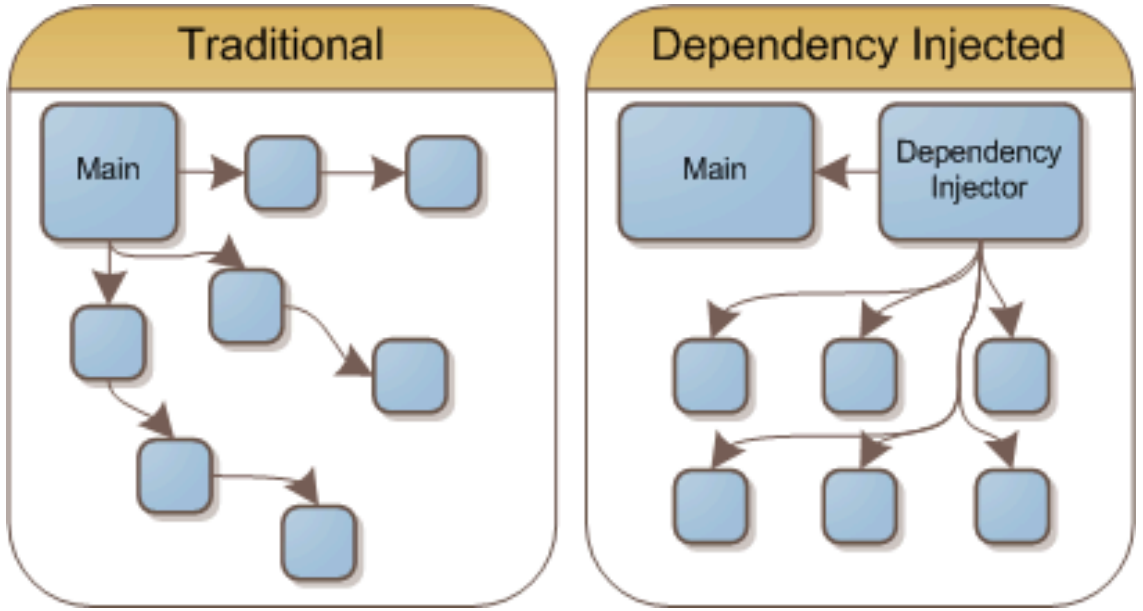
• *ViewModel* nunca poderá ter referência de uma *view*, *lifecycle* ou contexto. Caso tenha temos grandes riscos de *memory leak*

• O *lifecycle* do *ViewModel* é amarrado diretamente com seu escopo. Então se uma *activity* é finalizada, o *viewmodel* também, o mesmo acontece se o *viewmodel* está atrelado a um *fragment* e esse *fragment* é *detached*



Injeção de dependência

Injeção de dependência



- Koin não é um injetor de dependência. Koin utiliza da estratégia de *Service Locator Pattern*
- Grande ganho em testes unitários
- Uso mais inteligente de memória
- Facilidade de comunicação entre submódulos

	build time	runtime performance
Koin	no impact	impact
Dagger	impact	no impact

Observer Pattern

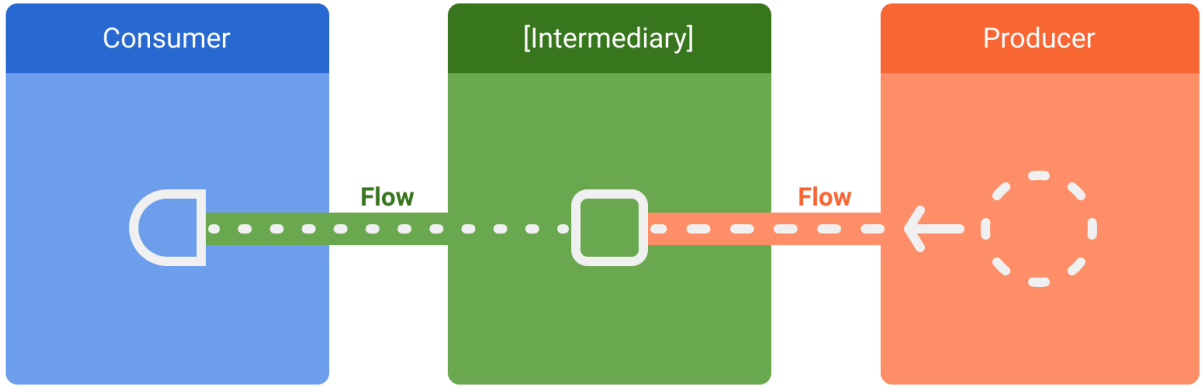
Extraindo o máximo de Flows

- Flow é parecido com um Iterator, emite valores em sequência porém de forma assíncrona.

- Emit -> Adiciona um dado a stream de dados

  Operadores -> Modificam um valor emitido para ser coletado posteriormente (OnEach, map, ...)

- Collect -> Operador final que coleta todos os dados da stream.



Producer: Produz a stream de dados de forma assíncrona

Intermediary: Modifica essa stream conforme desejado (Opcional)

Consumer: Consome o dado da stream de dados

Operadores úteis

filter() -> filtra resultado de um flow pelo predicado desejado

merge() -> junta o resultado de um flow em outro flow sem preservar a ordem

zip() -> faz dois ou mais flows acontecerem em paralelo

OnEach() -> executa após um valor ser emitido no upstream

retry() -> tenta executar o flow novamente x vezes se a exception for a desejada

catch() -> processa uma exception que ocorreu na execução do flow

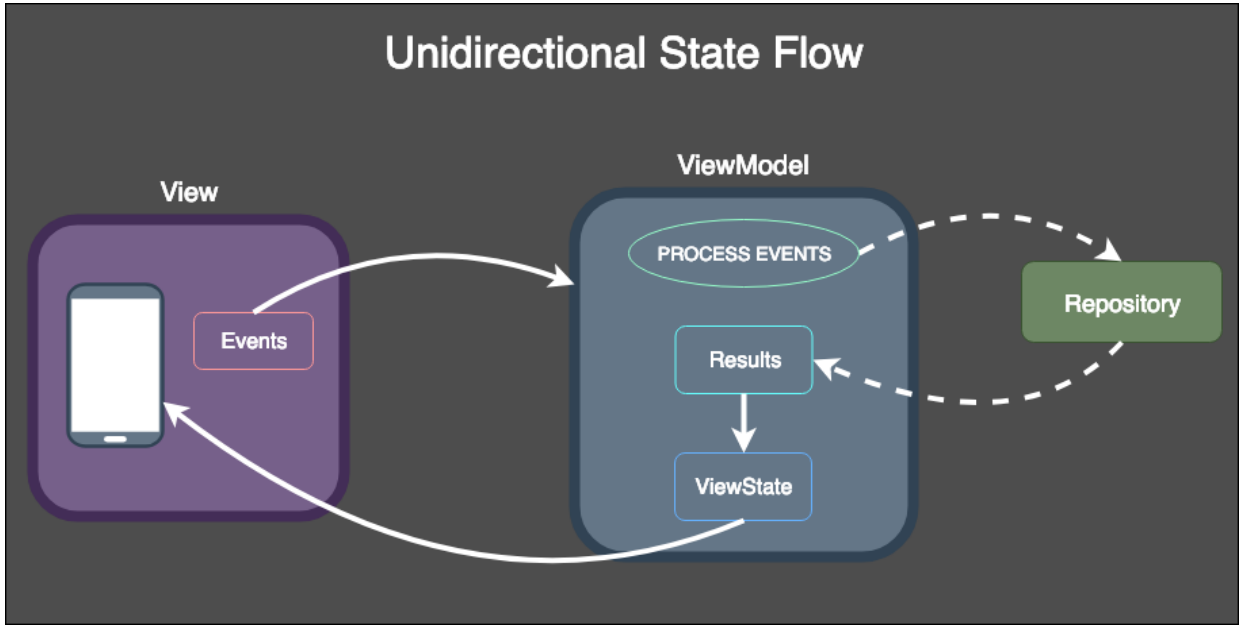
collect() -> Operador terminal que coleta o valor emitido ignorando todo o resto

collectLatest() -> Grande diferença do collect() é que se for emitido um valor novo enquanto esse operador foi chamado, ele bloqueia o antigo emit e só coleta o novo valor.

e muito mais...

Programação reativa

- Uso de LiveData não é aconselhado em projetos atuais, principalmente se utilizar Compose.
- Coroutines e Flows são altamente recomendados pelo Google para se comunicar entre as camadas da aplicação.
- *StateFlow é state aware* e retorna valores em sequência.
- *Flow no geral tem muitos operadores que podemos aproveitar e facilitar nossa vida. Como visto anteriormente.*



Guidelines

Use a clearly defined data layer.

Strongly recommended

The data layer exposes application data to the rest of the app and contains the vast majority of business logic of your app.

- You should create repositories even if they just contain a single data source.
- In small apps, you can choose to place data layer types in a `data` package or module.

Use lifecycle-aware UI state collection.

Strongly recommended

Collect UI state from the UI using the appropriate lifecycle-aware coroutine builder: `repeatOnLifecycle` in the View system and `collectAsStateWithLifecycle` in Jetpack Compose.

Read more about `repeatOnLifecycle`.

Read more about about `collectAsStateWithLifecycle`.

Do not send events from the ViewModel to the UI.

Strongly recommended

Process the event immediately in the ViewModel and cause a state update with the result of handling the event. More about UI events here.

[Latest Google recommendations for android architecture]

Demonstração

[Repository]

[LinkedIn]