TI-2600 Laboratório de Desenvolvimento Multiplataforma

Antonio Carvalho - Treinamentos

KMP NBR (Network Bound Resource)

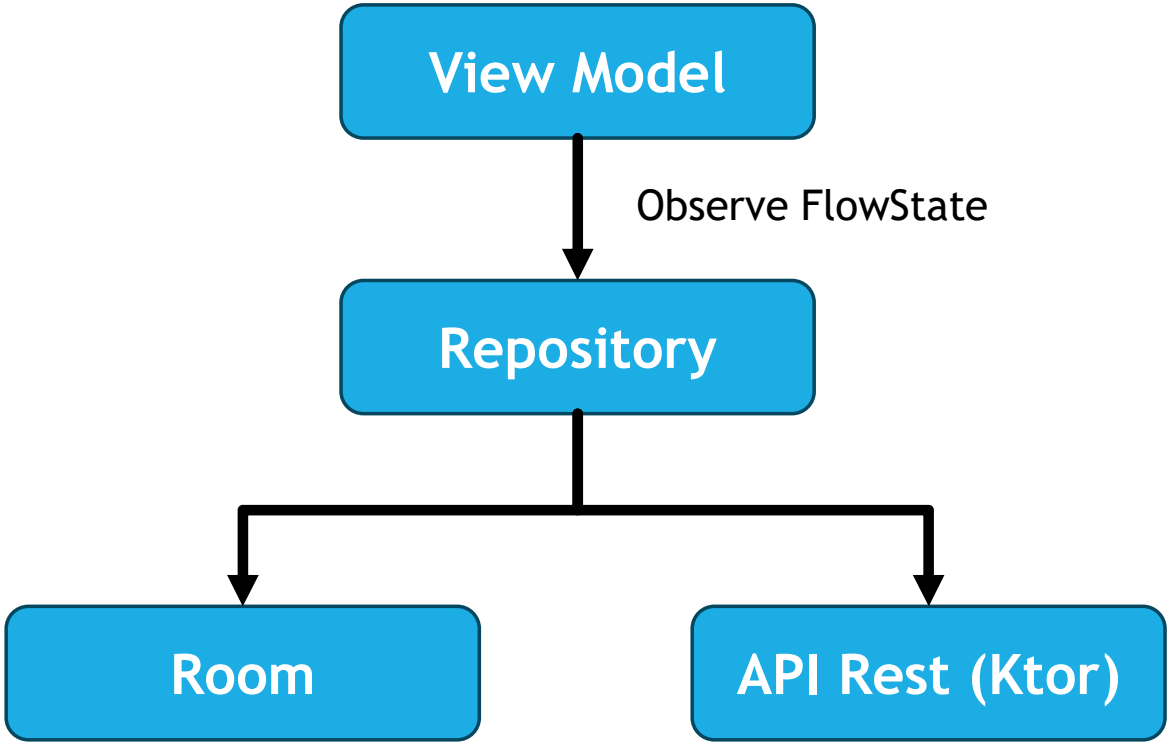
KMP Arquitetura NBR (Network Bound Resource)

- Network Bound Resource (ou NBR) é um padrão de arquitetura muito usado no Android para combinar dados de cache local com dados remotos, de forma eficiente e segura, oferecendo:
  - Melhor desempenho (cache local)
  - Experiência offline
  - Atualização transparente quando há conexão com a internet

KMP Arquitetura NBR (Network Bound Resource)

- Network Bound Resource consiste em uma estratégia para controlar como os dados são carregados em uma aplicação.
- O princípio consiste em **Usar sempre o cache local como fonte principal, mas decidir dinamicamente se e quando buscar da rede, e atualizar o cache com os dados remotos.**

KMP Arquitetura NBR – Diagrama



KMP Arquitetura NBR – Fluxo Típico

- Buscar do banco local (Room)
- Verificar se precisa buscar da API (por tempo, erro, ausência de dados etc.)
- Se precisar, buscar da API
  - Salvar os dados no banco local (cache)
  - Emitir os dados atualizados para o consumidor (ViewModel/UI)

KMP Arquitetura NBR – Decisões tomadas

- Quando é necessário buscar dados da rede?
  - Dados estão vazios
  - Cache está velho
  - Usuário solicitou atualização
- Como lidar com erro de rede?
  - Continuar com o cache
  - Mostrar mensagem de erro
- Como atualizar a UI?
  - Emitindo dados do banco assim que forem atualizados

KMP Arquitetura NBR – Decisões tomadas

- Implementação genérica e simplificada da função NBR:

```
fun <T> networkBoundResource(
    query: () -> Flow<T>,
    fetch: suspend () -> T,
    saveFetchResult: suspend (T) -> Unit,
    shouldFetch: (T) -> Boolean = { true }
): Flow<T> = flow {
    val data = query().first()
    emit(data) // Emite os dados do Cache local primeiro
    if (shouldFetch(data)) {
        try {
            val fetched = fetch()
            saveFetchResult(fetched)
        } catch (e: Exception) {
            e.printStackTrace()
        }
    }
    // reemite os dados atualizados (Room detecta mudanças e atualiza o Flow)
    emitAll(query())
}
```

KMP Room com API Rest usando arquitetura NBR

KMP Room + API Rest + NBR

- Esta combinação permitirá ter:
  - Acesso offline aos dados.
  - Minimizar chamadas desnecessárias à API.
  - Melhorar o tempo de resposta usando cache local com sincronização.

KMP Room + API Rest + NBR

- Será criada uma nova camada chamada Repository e haverá mudanças na camada ViewModel e DAO
  - ContatoRepository
  - ContatoViewModel
  - ContatoDAO

KMP Room + API Rest + NBR

- Camada Repository com **ContatoRepository:**

```
class ContatoRepository(
    private val dao: ContatoDao,
    private val api: ContatoApi
) {
    fun getContatos(): Flow<List<Contato>> = flow {
        emitAll(dao.getAll()) // Emite o cache local primeiro
        try {
            val contatosRemotos = api.getAll()
            dao.insertAll(contatosRemotos) // Atualiza cache
        } catch (e: Exception) {
            // Se houver erro de rede, segue com o cache.
            Log.e("AGENDA_CONTATO","Erro: ",e)
        }
    }
}
```

KMP Room + API Rest + NBR

- Camada ViewModel atualização com **ContatoViewModel:**

```
class ContatoViewModel(
    private val repository: ContatoRepository
) : ViewModel() {
    val contatos: StateFlow<List<Contato>> =
        repository.getContatos()
         .stateIn(
             scope = viewModelScope,
             started = SharingStarted.WhileSubscribed(5000),
             initialValue = emptyList()
         )
}
```

KMP Room + API Rest + NBR

- Camada DAO atualização com **ContatoDAO:**

```
import androidx.room.Dao
import androidx.room.Delete
import androidx.room.Insert
import androidx.room.Query

@Dao
interface ContatoDAO {

    @Query("Select * from Contato")
    suspend fun getAll() : Flow<List<Contato>>
    @Insert
    suspend fun insert(contato : Contato)

    @Delete
    suspend fun delete(contato : Contato)
}
```

KMP Room + API Rest + NBR

Atualização na camada UI:

```
@Composable
fun ContatoUI(viewModel: ContatoModel = viewModel()) {
    val contatos by viewModel.contatos.collectAsState()

    LazyColumn {
        items(contatos) { contato ->
            Text(text =
                "${contato.nome} - ${contato.email}")
        }
    }
}
```