Documentacao de Desenvolvimento - ToDo List com Firebase

Projeto: ToDo List com Firebase Authentication e Firestore
Grupo: Luccas Asaphe, Matheus Nascimento

Indice

Arquiteturas Implementadas

1. Sincronizacao Firestore com callbackFlow + snapshot listener

Implementacao: TaskRepositoryImpl.kt
Complexidade: Alta - gerenciamento de listener com callbackFlow
Descricao: Converte Firebase snapshot listener em Flow usando callbackFlow. Mantem listener ativo enquanto Flow estiver sendo observado. Implementa ordenacao em memoria para evitar indices compostos do Firestore.

2. Reactive ViewModel com flatMapLatest + stateIn

Implementacao: TaskViewModel.kt
Complexidade: Alta - padrao avancado de Coroutines
Descricao: Usa flatMapLatest para automaticamente reagir a mudancas de userId. Transforma Flow em StateFlow com stateIn e SharingStarted.Lazily para mantem listener ativo enquanto ViewModel existir.

3. Firebase Auth State Tracking com AuthStateListener

Implementacao: AuthViewModel.kt
Complexidade: Media - listener de estado de autenticacao
Descricao: Configura AuthStateListener no init do ViewModel para reagir a mudancas de sessao (login/logout). Essencial para sincronizar UI com estado de autenticacao.

4. UI com Validacoes e Estados

Implementacao: LoginScreen.kt, SignUpScreen.kt, TaskListScreen.kt
Descricao: Telas com Jetpack Compose usando StateFlow.collectAsState() para observar ViewModel. Implementam validacoes de email, senha, e CRUD de tarefas com AlertDialogs.

Desafios Encontrados

1. Sincronizacao de Tarefas em Tempo Real

Problema: Tarefas desapareciam apos aparecerem brevemente na UI

Causa Raiz: TaskViewModel original usava collect() manual com launch, cancelando o listener quando a coroutine era cancelada.

Solucao: Usar flatMapLatest + stateIn para mantem listener ativo continuamente enquanto ViewModel existe.

Aprendizado: Listeners devem ser gerenciados via StateFlow + stateIn, nao via manual collect() + launch

2. PERMISSION_DENIED apos Logout/Login

Problema: Apos logout e novo login, recebia PERMISSION_DENIED apos ~35 segundos

Causa Raiz: getCurrentUser() usava flowOf(), que emite uma vez e completa. Nao reagia a mudancas de estado de autenticacao.

Solucao: Implementar callbackFlow com AuthStateListener para reagir continuamente a mudancas de autenticacao.

Aprendizado: Firebase nao notifica mudancas automaticamente. AuthStateListener e essencial para sincronizar estado de sessao com a UI.

3. Firestore Rules e Snapshot Listeners

Problema: Mesmo com rules corretas, snapshot listeners falhavam com PERMISSION_DENIED

Causa Raiz: Validacoes continuas de snapshot listeners nao suportam resource.data em read rules de forma confiavel.

Solucao: Usar read rules simples (so autenticacao) e fazer filtragem na query Android com whereEqualTo().

Aprendizado: Snapshot listeners precisam de read rules muito simples. Filtragem deve ser feita na query, nao na security rule.

4. Validacao de Email e Checkbox UI

Problema: Email invalido passava validacao; checkbox nao atualizava ao marcar

Solucao: Usar Patterns.EMAIL_ADDRESS para validacao. Usar getBoolean() explicitamente ao deserializar Firestore.

5. Erro Persistindo ao Navegar entre Telas

Problema: Mensagem de erro continuava exibida ao navegar para outra tela

Solucao: Chamar viewModel.resetState() antes de navegar para limpar estado anterior.

Aprendizado: Gerenciar ciclo de vida de estados ao navegar entre telas e importante para UX.

Aprendizados Principais

Arquitetura e Padroes

Clean Architecture + MVVM

- Separacao entre camadas (Presentation/Domain/Data) melhora testabilidade
- ViewModels gerenciam estado, nao logica de negocio
- Repositories abstraem detalhes de implementacao

StateFlow vs LiveData

- StateFlow e mais funcional e permitem composicao (.map, .filter, .flatMapLatest)
- Melhor integracao com Kotlin Coroutines

Firebase

Authentication

- Firebase gerencia sessoes automaticamente

- AuthStateListener e essencial para reagir a mudancas de sessao

Firestore

- Snapshot listeners sao eficientes para sincronizacao em tempo real
- Read rules devem ser simples (so autenticacao)
- Filtragem deve ser feita na query, nao na rule

Kotlin Coroutines - Padroes Criticos

callbackFlow

- Converte callbacks Firebase (listeners) em Flow reativo
- awaitClose() limpa listener automaticamente quando Flow e cancelado

flatMapLatest

- Muda automaticamente para novo Flow quando dependencia (ex: userId) muda
- Cancela Flow anterior automaticamente (limpeza de recurso)

stateIn

- Converte Flow em StateFlow (cacheado com ultimo valor)
- SharingStarted.Lazily mantem listener ativo enquanto ha observers

Jetpack Compose

- Declarativo - UI definida como funcao da state
- Recomposicao automatica ao observar StateFlow.collectAsState()
- Sem findViewById/binding - mais conciso e seguro

Debugging

- Logs estrategicos em pontos-chave (entry, snapshot received, errors)
- Logcat com filtros por tag ajuda muito na investigacao
- Testes em device real sao essenciais

Conclusao

Este projeto envolveu tres componentes complexos:

- Estrutura de sincronizacao em tempo real com Firestore
- Padroes avancados de Kotlin Coroutines (flatMapLatest + stateIn)
- Integracao completa de Firebase com Compose

Aprendizados chave:

- Entender o "por que" das decisoes e mais importante que copiar codigo
- Listeners devem ser gerenciados via reactive patterns (StateFlow + stateIn)
- Firestore rules devem ser simples; filtragem na query
- Testes em device real sao essenciais para deteccao de problemas