Documentação Técnica: To-Do List App (Android/Kotlin)

1. Modelo de Dados (Data Model)

O aplicativo utiliza uma abordagem **NoSQL** (Document-Oriented) para garantir flexibilidade e escalabilidade na nuvem. A entidade central do sistema é a Tarefa (Todo), desenhada para ser serializável e compatível com o formato JSON do Firebase.

Entidade de Domínio

A classe Todo representa a estrutura de dados fundamental, localizada na camada de domínio para garantir desacoplamento de frameworks específicos.

Kotlin

```
data class Todo(
    val id: String = "",            // Identificador único (UUID do Firestore)
    val title: String = "",         // Título da tarefa
    val description: String? = null, // Detalhes opcionais
    val isDone: Boolean = false,    // Status de conclusão
    val userId: String = ""         // Chave estrangeira (Vínculo com Auth)
)
```

Estrutura no Banco de Dados (Firestore)

No Cloud Firestore, os dados são persistidos na coleção raiz todos. Cada documento é um objeto JSON independente.

- **Isolamento de Dados:** A segurança e privacidade são garantidas pelo campo userId. Embora todas as tarefas residam na mesma coleção, as consultas são estritamente filtradas pelo ID do usuário autenticado (.whereEqualTo("userId", userId)).

2. Implementação da Persistência

A camada de persistência foi arquitetada seguindo o padrão **Repository Pattern**, servindo como uma ponte entre a lógica de negócios (ViewModels) e a fonte de dados externa (Firebase).

Componentes da Arquitetura

1. **Interface TodoRepository:** Define o contrato de operações (CRUD) de forma agnóstica. Isso permite que a implementação do banco de dados seja substituída (ex: por Room ou Realm) sem afetar o restante do aplicativo.
2. **Implementação FirestoreRepositoryImpl:** Classe concreta que gerencia as chamadas ao SDK do Firebase.
    - **Assincronismo:** Utiliza **Kotlin Coroutines** e funções suspend para realizar operações de I/O (Leitura/Escrita) fora da Thread Principal, prevenindo travamentos na interface (ANR).
    - **Mapeamento:** Converte automaticamente os documentos do Firestore para objetos Kotlin (toObjects(Todo::class.java)).

Estratégia de Sincronização e Ciclo de Vida

Para garantir que a interface do usuário (UI) esteja sempre consistente, adotou-se uma estratégia baseada no Ciclo de Vida da Activity:

- **Leitura Sob Demanda (On-Resume):** Diferente de uma conexão via Socket aberta permanentemente, o aplicativo otimiza recursos buscando dados atualizados sempre que a tela ganha foco. Isso foi implementado na ListScreen utilizando um DisposableEffect que observa o evento ON_RESUME.
- **Escrita Otimista:** Na alteração de status (checkbox), a UI é atualizada instantaneamente para o usuário, enquanto a sincronização com o banco ocorre em segundo plano.

3. Melhorias Futuras (Roadmap) Para a evolução do produto (Versão 2.0), as seguintes melhorias técnicas e funcionais são recomendadas:

A. Arquitetura e Performance (Offline-First)

Atualmente, o app depende de conexão ativa. A principal melhoria seria a implementação do **Room Database** como fonte de verdade local (Cache).

- **Funcionamento:** O app leria/escreveria sempre no banco local (rápido e offline). O WorkManager sincronizaria os dados com o Firebase quando a conexão fosse restabelecida.

B. Reatividade em Tempo Real

Migrar as chamadas de "One-Shot" (busca única) para **Kotlin Flows** com addSnapshotListener.

- **Benefício:** Permitiria que, se o usuário alterasse uma tarefa na Web ou em outro dispositivo, o aplicativo atualizasse a lista instantaneamente sem a necessidade de recarregar a tela.