Quiz App Integrado ao Firebase

JOÃO LUCAS (UI/UX): TELAS, NAVEGAÇÃO.

GUILHERME (BACKEND): INTEGRAÇÃO COM FIREBASE (LOGIN, BANCO, RANKING).

KAUAN (LÓGICA LOCAL): BANCO LOCAL (ROOM), SINCRONIZAÇÃO E CONTROLE DO QUIZ.

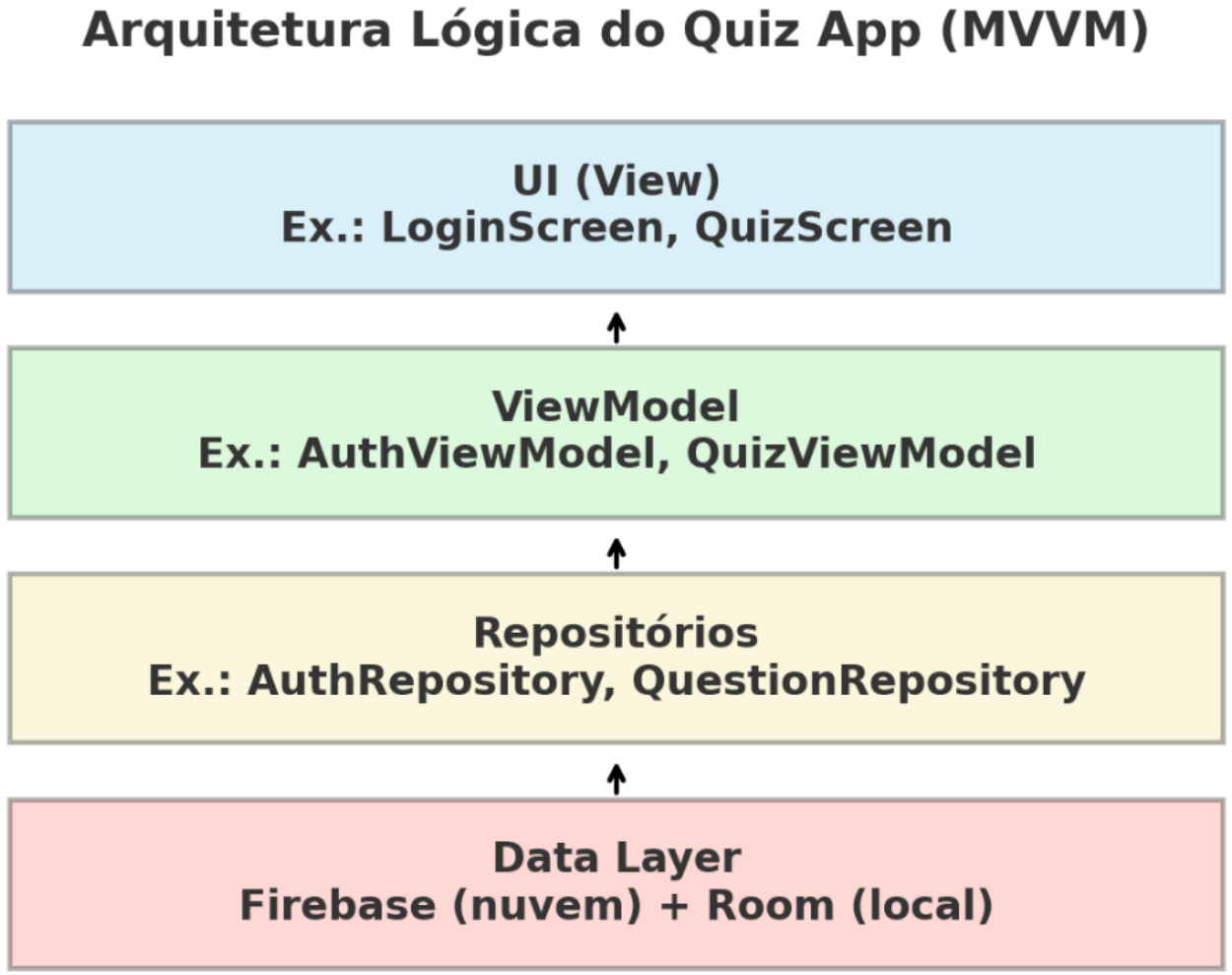
UFU

Arquitetura MVVM e Injeção de Dependência com Hilt

Injeção de Dependência com Hilt: Utilizamos Hilt para gerenciar as dependências entre as camadas (View, ViewModel, Repository).

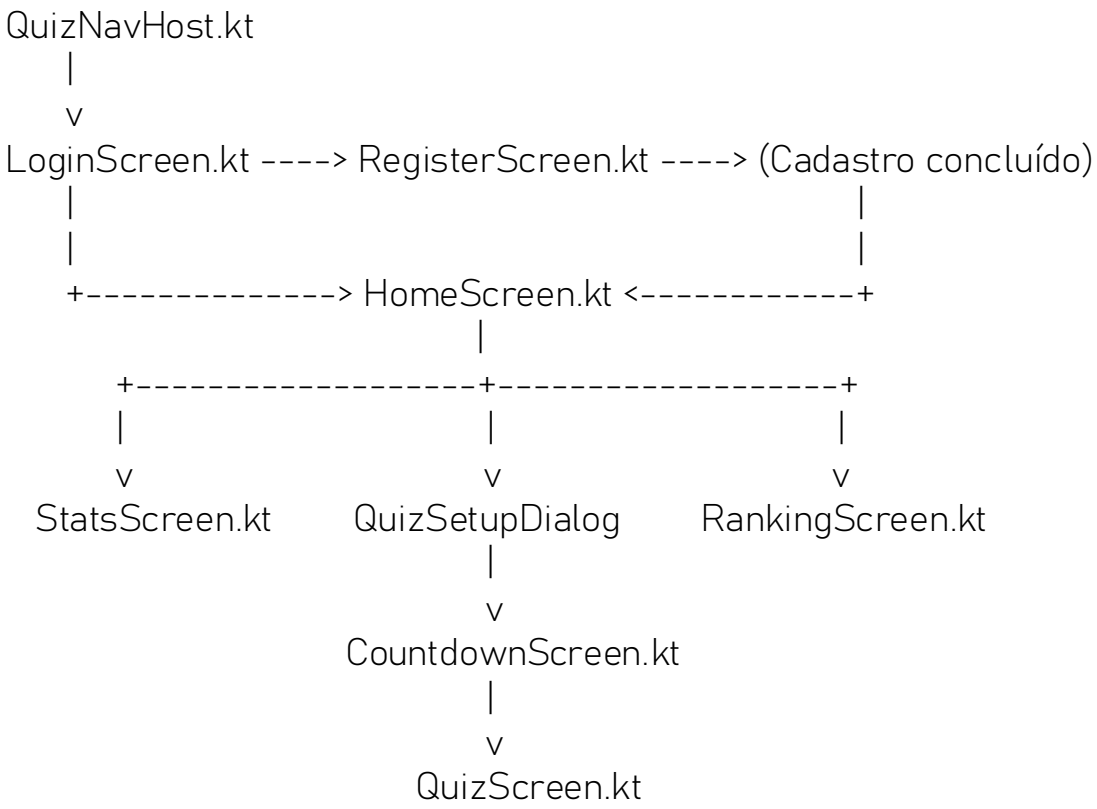
Benefícios: Código mais limpo, desacoplado e muito mais fácil de testar.

```
// ViewModel recebendo o repositório via Hilt
@HiltViewModel
class AuthViewModel @Inject constructor(
private val repository: AuthRepository
) : ViewModel() {
 // ... lógica do ViewModel ...
}
```



Interface e Fluxo de Navegação com Jetpack Compose

Interface 100% construída com Jetpack Compose, de forma declarativa.
Design moderno seguindo as práticas de Material Design.
Criação de componentes de UI reutilizáveis para consistência visual.




```
// Exemplo do QuizNavHost.kt
composable("login") {
    LoginScreen(
        onLoginSuccess = {
navController.navigate("home") },
        onRegisterClick = {
navController.navigate("register") }
    )
}
```

Modelo de Dados no Firebase Realtime Database

1. Perguntas:

As perguntas são organizadas em uma estrutura JSON aninhada.

Benefício: Essa organização permite que o app filtre e carregue quizzes de forma dinâmica e escalável.

Hierarquia:
Tema > Dificuldade > ID da Pergunta.

2. Users:

Cada usuário possui um nó com seu uid do Firebase Auth.

recentGames alimenta a tela de Histórico. statsByTheme permite o funcionamento do Dashboard.

O objeto stats centraliza todo o desempenho do jogador.

- **Armazenamento local (Room):** garante acesso offline às perguntas do quiz.
- **Banco remoto (Firebase):** fonte de verdade dos dados.
- **Controle de versão:** só atualiza se houver versão mais nova no servidor.
- **Sincronização automática:** baixa e salva localmente quando necessário.
- **Benefício:** app funciona offline e mantém dados consistentes.

Funcionalidades-Chave: Autenticação e Ranking

Autenticação de Usuários:

Acesso individualizado com o Firebase Authentication.

```
FirebaseAuth.getInstance
()
.signInWithEmailAndPass
word(email, senha)
.addOnSuccessListener {
... }
.addOnFailureListener { e
-> Log.e(...) }
```

Ranking em Tempo Real:

O desempenho é salvo no Realtime Database para gerar um ranking competitivo.

```
// Inserir pontuação no
ranking val rankingRef =
database.getReference("r
anking")
rankingRef.child(playerId
).setValue(playerData)
```

Desafios e Soluções: Depuração com Logcat

Desafio: Garantir a comunicação estável e identificar erros na troca de dados com o Firebase.

Solução: Uso intensivo do **Logcat** com tags personalizadas para monitorar as respostas da rede e o estado do app em tempo real.

Desafio: Garantir boas práticas de código como, por exemplo, a injeção de dependências.

Solução: Uso do Framewrok Hilt.

Uso de LLMs no Desenvolvimento

Ferramenta: Gemini 2.5 PRO

Exemplo de Prompt:
Você deve se comportar como um desenvolvedor Android especialista em Kotlin e que está ensinando alunos de graduação a criar um aplicarivo de um quiz que contará com Login, Autenticação, Tela para jogar, Dashboard de estatísticas do jogador e tudo será salvo localmente e em nuvem do firebase.

Estratégia: A LLM foi usada para acelerar a criação da arquitetura base (Room, Repository, Hilt), permitindo que o time focasse na lógica de negócio.

Opinião do Grupo:
Ferramenta excelente para gerar código padrão e aprender boas práticas, mas todo código gerado exigiu revisão e adaptação manual.