Relatório do Projeto

1. Decisões de Arquitetura e Tecnologia

Para atender aos requisitos de um aplicativo moderno, funcional e escalável, o grupo tomou uma série de decisões estratégicas de arquitetura e tecnologia, detalhadas a seguir.

a. Arquitetura MVVM (Model-View-ViewModel)

Adotamos a arquitetura MVVM, recomendada oficialmente pelo Google, para garantir uma separação clara de responsabilidades.

- **View (UI Layer):** Implementada com Jetpack Compose, é responsável apenas por exibir o estado da UI e capturar as interações do usuário.
- **ViewModel (State Layer):** Atua como o cérebro de cada tela, contendo a lógica de UI, o estado (`StateFlow`) e se comunicando com a camada de dados.
- **Model (Data Layer):** Composta pelos Repositórios, fontes de dados locais (Room, SharedPreferences) e remotas (Firestore).
    - **Justificativa:** Essa separação tornou o código mais testável, fácil de manter e permitiu que os membros da equipe trabalhassem em camadas diferentes simultaneamente com menos conflitos.

b. Jetpack Compose para a UI

A interface do usuário foi construída inteiramente com Jetpack Compose.

- **Justificativa:** Optamos por Compose por ser o kit de ferramentas moderno para desenvolvimento de UI no Android. Ele nos permitiu criar uma interface reativa e dinâmica com menos código, além de facilitar a criação de componentes reutilizáveis, como o `QuizCategoryCard` e o `TopBarProfile`.

c. Hilt para Injeção de Dependência

Para gerenciar as dependências do projeto (como fornecer instâncias do Firebase ou do banco de dados para os repositórios), escolhemos o Hilt.

- **Justificativa:** Hilt foi preferido em relação a outras opções como o Koin por sua segurança em tempo de compilação. Em um projeto colaborativo, a capacidade do Hilt de detectar erros de injeção de dependência antes da execução do app foi crucial para evitar bugs e garantir a integração contínua do código.

d. Cloud Firestore como Banco de Dados Remoto

Para o armazenamento de dados na nuvem, optamos pelo Cloud Firestore em vez do Realtime Database.

- **Justificativa:** A estrutura de coleções e documentos do Firestore é mais organizada e escalável para o nosso caso de uso (perfis de usuário, quizzes, histórico). Além disso, suas capacidades de consulta avançada foram essenciais para implementar funcionalidades como o ranking (`orderBy`) e o histórico ordenado por data.

e. Persistência Local Dupla: Room e SharedPreferences

Para atender ao requisito de funcionalidade offline e desempenho, utilizamos duas estratégias de armazenamento local:

- **SharedPreferences:** Usado para armazenar dados simples e de acesso rápido, como o ID do usuário logado, gerenciando a sessão do app.
- **Room Database:** Usado para armazenar dados mais complexos e estruturados, como o histórico de quizzes (`quiz_history`), permitindo que o usuário consulte seu desempenho mesmo sem conexão com a internet.

2. Papéis e Divisão de Tarefas

O projeto foi desenvolvido em equipe, com uma divisão de tarefas clara para maximizar a produtividade e o foco de cada membro.

- **João Gabriel Santos Rodrigues - Arquiteto de UI/UX:**
  - **Responsabilidades:** Focou na camada de **View**. Foi responsável por traduzir os designs em componentes Jetpack Compose funcionais e reutilizáveis (`QuizScreen`, `HomeScreen`, `QuizCategoryCard`, etc.). Cuidou da estilização, da experiência do usuário e da conexão da UI com os `ViewModels`.
- **João Guilherme Araújo Viana - Especialista em Backend (Firebase):**
  - **Responsabilidades:** Focou na integração com os serviços do **Firebase**. Configurou o projeto no console, implementou o Firebase Authentication para login e cadastro, e modelou a estrutura de dados no Cloud Firestore (coleções `users`, `quizzes`, `quiz_history`). Também foi responsável por escrever e testar as Regras de Segurança para proteger os dados.
- **Matheus Gualter Silva Resende - Gerente de Dados e Lógica de Negócio:**
  - **Responsabilidades:** Focou na camada de **Model (Data Layer)**. Implementou o banco de dados local com Room, o gerenciador de sessão com SharedPreferences (`PrefsManager`), e criou a camada de Repositório (`AuthRepository`, `QuizRepository`, `UserRepository`). Sua principal função foi criar a lógica que abstrai as fontes de dados, decidindo quando buscar informações do Firebase ou do cache local.

3. Principais Dificuldades Enfrentadas

Durante o desenvolvimento, enfrentamos alguns desafios técnicos que foram cruciais para o nosso aprendizado.

a. Configuração do Gradle e Conflitos de Versão

A dificuldade inicial mais significativa foi a configuração correta das dependências no Gradle, especialmente com o Firebase. O erro persistente "Failed to resolve" nos ensinou na prática a importância de usar o **Firebase BOM (Bill of Materials)** para garantir a compatibilidade entre as bibliotecas. A depuração desse problema envolveu a análise cuidadosa dos arquivos `build.gradle.kts` e do catálogo de versões `libs.versions.toml`.

b. Gerenciamento de Estado Assíncrono

Lidar com dados que vêm de fontes assíncronas (como o Firestore) e exibi-los em uma UI reativa com Jetpack Compose foi um desafio. A adoção do padrão `StateFlow` nos `ViewModels` para conter o estado da tela e a sua coleta com `collectAsState` nos Composables se mostrou a solução mais robusta, garantindo que a UI sempre refletisse o estado mais recente dos dados.

c. Regras de Segurança do Firestore

Inicialmente, as operações de escrita no Firestore falhavam silenciosamente. A depuração via Logcat nos levou a descobrir que as Regras de Segurança padrão (em modo de produção) bloqueiam todas as leituras e escritas. Aprender a sintaxe das regras e a escrever condições seguras (ex: `allow write: if request.auth.uid == userId;`) foi um passo fundamental para garantir a segurança e a funcionalidade do aplicativo.