Relatório de Desenvolvimento: Aplicativo Quiz Online

Projeto: Desenvolvimento de um aplicativo de quiz para Android utilizando Kotlin e Firebase.

Equipe:

Gabriel Silva Tassara - 12311BSI218

Kevin Martins de Souza Barbosa - 12311BSI216

Marcos Paulo Oliveira Gomes - 12311BSI231

Data: 18 de agosto de 2025

1. Introdução

Este relatório apresenta o processo de desenvolvimento do nosso aplicativo de quiz para Android. O objetivo era criar um app funcional e com uma interface moderna, que permitisse a cada usuário ter seu próprio login e acompanhar seu desempenho. Para isso, usamos o Firebase para cuidar da parte de autenticação e do armazenamento dos dados, e focamos em garantir que o app funcionasse bem, mesmo offline.

2. Decisões Técnicas

Para construir o aplicativo, escolhemos algumas ferramentas e padrões que são muito usados no mercado hoje. A ideia era não só cumprir os requisitos, mas também aprender a forma correta de estruturar um projeto Android.

- **Arquitetura MVVM (ModelView-ViewModel):** Decidimos usar o padrão MVVM para deixar o código mais organizado. Com ele, separamos tudo em três partes: a **View** (as telas que o usuário vê), o **ViewModel** (que cuida da lógica da tela) e o **Model** (que busca e guarda os dados). Na prática, isso significou que as Activities ficaram mais simples, e a lógica de negócio ficou mais fácil de testar e dar manutenção.
- **Cloud Firestore como Banco de Dados Principal:** Entre as opções do Firebase, escolhemos o **Cloud Firestore** porque ele é mais flexível para organizar os dados. A

forma como ele trabalha com coleções e documentos facilitou muito a criação de funcionalidades como os quizzes, o ranking e o histórico do usuário, que precisavam de consultas mais específicas.

- **Room para Cache e Funcionalidade Offline:** Um requisito importante era que o app funcionasse sem internet. Para resolver isso, usamos a biblioteca **Room**. A lógica é simples: na primeira vez que o usuário abre o app, baixamos os quizzes do Firestore e salvamos uma cópia no banco de dados local do celular. Nas próximas vezes, o app carrega tudo instantaneamente do aparelho, o que deixa a experiência muito mais rápida.
- **Kotlin e Coroutines:** Usamos **Kotlin** por ser a linguagem padrão para Android. Para as tarefas que dependem de rede, como buscar dados do Firebase, adotamos as **Coroutines**. Elas ajudaram a organizar o código assíncrono de uma forma muito mais limpa, o que deixou o código mais legível e fácil de depurar.

3. Divisão de Papéis

Para organizar o trabalho, dividimos as responsabilidades de acordo com as principais áreas do projeto.

- **Gabriel Tassara - Desenvolvedor de UI/UX:**

- Responsável por toda a camada de interface do aplicativo.
- Criou os layouts em XML para as telas principais (Login, Tela Inicial, Quiz, Histórico, etc.).
- Definiu o tema visual do app, aplicando as diretrizes do Material Design para cores e fontes.
- Conectou as telas com a lógica de negócio usando ViewBinding, garantindo que os dados fossem exibidos corretamente, e fez o polimento final da experiência do usuário.

- **Marcos Paulo - Arquiteto de Backend (Firebase):**

- Cuidou de toda a infraestrutura na nuvem.
- Configurou o projeto no Firebase e o integrou ao Android Studio.
- Implementou o sistema de autenticação com e-mail e senha.
- Estruturou o Cloud Firestore para armazenar os quizzes e os dados dos usuários, e desenvolveu as funções para baixar questões e salvar os resultados das partidas.

- **Kevin - Desenvolvedor de Banco de Dados Local (Room):**

- Focado na funcionalidade offline do aplicativo.
- Configurou a biblioteca Room e definiu as tabelas (entidades) para guardar os dados localmente.
- Implementou a lógica de cache, salvando as questões baixadas do Firebase e o histórico de desempenho do usuário no banco de dados do dispositivo.
- Desenvolveu a lógica de sincronização para atualizar os dados locais quando necessário.

4. Principais Dificuldades Enfrentadas

Durante o desenvolvimento, passamos por alguns problemas que nos ensinaram bastante na prática.

- **Implementação de Cache Offline (Cache-First):** Fazer o app funcionar offline foi um dos maiores desafios.
  - **Desafio:** No começo, o app era lento para abrir porque sempre esperava a internet para carregar os quizzes. Outro problema foi que, ao tentar salvar o histórico localmente, os registros ficavam duplicados porque não tínhamos um ID único para cada um.
  - **Solução:** Mudamos a lógica para **"cache-first"**: o app carrega os dados direto do **Room** e só depois busca atualizações na internet. Para resolver a duplicação, passamos a usar o **ID do documento do Firestore como chave primária (@PrimaryKey)** também no Room. Assim, garantimos que cada registro era único tanto online quanto offline.
- **Configuração do Firebase e Gestão de Dependências:** Tivemos vários erros de compilação que demoramos a entender.
  - **Desafio:** As regras de segurança padrão do Firestore bloqueavam o acesso aos dados, e o nosso arquivo build.gradle.kts estava com uma confusão de dependências do Firebase que causava conflitos.
  - **Solução:** Primeiro, aprendemos a escrever as **regras de segurança** do Firestore para permitir que um usuário logado acesse apenas os seus próprios dados. Depois, limpamos o arquivo de dependências e passamos a usar a **Firebase**

BOM (Bill of Materials), que gerencia as versões de todas as bibliotecas do Firebase para que elas sejam sempre compatíveis entre si.

- **Consistência de Dados em Tempo Real:** Um bug que nos incomodou foi que a pontuação no Dashboard não atualizava depois que o usuário terminava um quiz.
    - **Desafio:** A tela só pedia os dados do usuário uma vez e não "sabia" quando eles mudavam.
    - **Solução:** A virada de chave foi usar um **listener em tempo real (addSnapshotListener)** do Firestore. Com ele, o Firebase nos avisa sempre que um dado muda. Usamos um **Kotlin Flow** para que o ViewModel recebesse essa atualização e a UI fosse modificada automaticamente.
- **Adaptação da Interface a Conteúdo Dinâmico:** Percebemos que, em algumas perguntas, o texto era tão grande que era cortado na tela.
    - **Desafio:** Os campos de texto tinham um tamanho fixo, o que não funcionava para um conteúdo que não conhecíamos.
    - **Solução:** A solução foi simples: mudamos a altura dos TextViews para wrap_content, para que eles crescessem conforme o texto. Além disso, colocamos a tela inteira do quiz dentro de um **ScrollView**, para que o usuário pudesse rolar a tela se o conteúdo fosse muito grande.

5. Conclusão

Concluímos o projeto com um aplicativo que atende a todos os requisitos funcionais. O mais importante foi a experiência de aplicar o que aprendemos na faculdade em ferramentas de mercado, como Firebase e a arquitetura MVVM. Foi um aprendizado prático que mostrou como a teoria se conecta com o desenvolvimento real de um aplicativo.

Os maiores desafios, como resolver os erros de permissão do Firestore e fazer o cache com Room funcionar, foram os que mais nos ensinaram. Cada problema nos obrigou a pesquisar e entender as ferramentas a fundo. No final, o projeto não só resultou em um aplicativo funcional, mas também nos deu uma experiência prática para futuros trabalhos.