Trabalho 4 – Quiz

Integrantes:

- João Lucas Gonçalves Teixeira
- Kauan Felipe Desterro Carvalho
- Guilherme Siqueira Botelho

---

1. Introdução

Inicialmente, convém destacar que neste relatório nós documentamos, de maneira detalhada e coesa, o processo de desenvolvimento do aplicativo **“Quiz App”**, concebido como um sistema de quiz nativo voltado para a plataforma Android. Desse modo, tendo em vista os objetivos traçados desde o princípio, buscamos construir uma aplicação não apenas funcional, mas também completa e robusta, a qual englobasse desde a autenticação individualizada dos usuários até a jogabilidade dinâmica, além do acompanhamento contínuo de estatísticas. Ademais, visando à integridade das informações, garantimos a persistência de dados tanto localmente no dispositivo quanto na nuvem. Portanto, o documento a seguir descreve, em linhas gerais e também específicas, as ferramentas que utilizamos, as decisões arquitetônicas que estabelecemos, os desafios que enfrentamos ao longo da trajetória e, consequentemente, as soluções que elaboramos para superá-los, bem como a nossa dinâmica colaborativa.

2. Metodologia e Ferramentas Adotadas

Em primeiro lugar, cumpre ressaltar que desenvolvemos o projeto utilizando a linguagem **Kotlin**, adotada como pilar central de programação, enquanto a interface gráfica foi construída de forma declarativa mediante o **Jetpack Compose**, toolkit moderno da Google para UI. Outrossim, visando à manutenção de um código limpo, escalável e testável, estruturamos a arquitetura em camadas distintas (UI, ViewModel, Repositório e Fontes de Dados).

No tocante à persistência, estabelecemos uma abordagem híbrida: de um lado, a biblioteca **Room** assegurou o armazenamento local, garantindo não apenas desempenho no acesso aos dados, mas também funcionalidade offline; de outro, os serviços do **Firebase** desempenharam papel fundamental, sendo utilizados o Authentication, para a gestão dos usuários, e o Realtime Database, para o armazenamento e sincronização imediata das perguntas e dos perfis dos jogadores. Além disso, a navegação entre telas foi regida pela biblioteca **Navigation Compose**, o que reforçou a consistência do projeto.

Convém salientar que adotamos a metodologia **UI-First**, isto é, construímos primeiramente as interfaces com dados mockados, possibilitando, antes de tudo, a concentração na experiência do usuário. Somente após consolidar tal etapa, partimos para a implementação da lógica de negócios mais complexa, o que, portanto, assegurou equilíbrio entre estética e funcionalidade.

3. Desenvolvimento e Decisões de Arquitetura

De antemão, uma das decisões mais relevantes que tomamos incidiu sobre a gestão de dependências. Visto que a criação manual de instâncias como bancos de dados, DAOs e repositórios para os ViewModels poderia gerar acoplamento excessivo e dificultar a manutenção, optamos pela utilização do **Hilt**, framework de injeção de dependência recomendado pela Google. Dessa forma, o Hilt, baseado sobre o Dagger, nos possibilitou automatizar a criação de objetos mediante anotações. Consequentemente, anotamos classes de UI com `@AndroidEntryPoint` e ViewModels com `@HiltViewModel`, ao passo que requisitamos suas dependências nos construtores via `@Inject`.

Adicionalmente, convém destacar que definimos, em um módulo centralizado (`@Module`), as dependências não reconhecidas nativamente pelo Hilt, como instâncias do Room e do Firebase. Assim, garantimos que o ciclo de vida e o fornecimento dos objetos fossem integralmente administrados pelo framework.

No que se refere à sincronização de dados, definimos uma estratégia dual. Para os perfis de usuários, suscetíveis a modificações mesmo offline, implementamos uma lógica bidirecional baseada em **timestamp** (lastUpdated). Desse modo, no momento do login, o sistema comparava a versão local com a versão na nuvem, adotando como fonte da verdade aquela mais recente. Já no tocante às perguntas do quiz, definimos o **Firebase** como única fonte da verdade, sendo empregada uma chave de versão (questions_version) que, ao ser divergente da versão armazenada localmente, desencadeava um processo de atualização integral. Portanto, asseguramos aos usuários acesso contínuo às questões mais recentes, inclusive em ambiente offline.

4. Principais Desafios e Soluções Encontradas

No decorrer do desenvolvimento, enfrentamos diversos desafios. Primeiramente, a configuração inicial do Hilt ocasionou erros de compilação relacionados a conflitos entre class loaders de plugins do Hilt e do KSP. Como consequência, a resolução exigiu ajustes minuciosos nos arquivos **build.gradle.kts**, de modo a harmonizar a declaração de ambos os plugins em escopo uniforme.

Paralelamente, outro obstáculo significativo consistiu nos denominados **“erros silenciosos”**: situações em que o aplicativo não apresentava falhas críticas, mas determinadas funcionalidades, como o carregamento de perguntas ou a exibição do ranking, simplesmente não eram exibidas. Superamos esse problema mediante o uso sistemático do **Logcat**, aliado a logs personalizados implementados em pontos estratégicos da lógica, como no **QuestionRepository** e nos ViewModels. Dessa forma, conseguimos

mapear o fluxo de dados em tempo real, identificando com precisão os pontos de falha, por exemplo, consultas ao Firebase que retornavam listas vazias devido a regras de segurança ou inconsistências estruturais.

Além disso, a interação com o Firebase nos trouxe desafios adicionais, como o erro de **argument constructor**, o qual ocorria ao tentar desserializar informações da nuvem para data classes do Kotlin. Para contornar tal dificuldade, garantimos que todos os campos dos modelos possuíssem valores padrão, permitindo a instanciação correta pelo SDK. Do mesmo modo, as queries de ordenação do ranking apenas se tornaram funcionais após a adição de regras de indexação (`.indexOn`) no Realtime Database, em consonância com exigências de otimização.

5. Colaboração e Papel dos Membros

No que se refere à colaboração, elaboramos o projeto em conjunto, cada um de nós desempenhando papéis específicos, como planejamento arquitetural, desenvolvimento da interface em Compose e implementação da lógica de persistência com Room e Firebase.

Para além disso, é relevante sublinhar que integramos a Inteligência Artificial **Gemini** como uma espécie de **"mentor sênior de programação"**, desempenhando papel decisivo na resolução de impasses técnicos. O fluxo consistia em definir objetivos, submeter trechos de código e, sobretudo, analisar erros diagnosticados via Logcat. Nesse sentido, sempre que surgia uma exceção, submetíamos a mensagem à IA, a qual nos fornecia não apenas explicações acessíveis, mas também soluções práticas acompanhadas do raciocínio técnico. Assim, estabelecemos um ciclo iterativo de **"executar → identificar erro → consultar IA → aplicar correção"**, o qual acelerou consideravelmente o avanço do projeto.

6. Conclusão

Diante de todo o exposto, concluímos que o projeto atingiu plenamente seus objetivos, consolidando-se em um aplicativo de quiz funcional, sustentado por uma arquitetura moderna e resiliente. Os principais aprendizados se concentraram na adoção do **Hilt** para injeção de dependências, na construção de interfaces reativas mediante o Jetpack Compose e na implementação de sofisticada lógica de sincronização entre base local e serviço em nuvem.

Portanto, como perspectivas futuras, recomendamos a ampliação do cache offline das perguntas e a implementação definitiva da lógica de UI para o ranking global, o que, sem dúvida, potencializará ainda mais a experiência do usuário. Assim, reafirmamos que a realização do projeto não apenas consolidou nossas competências técnicas, mas também aprimorou nossas habilidades de trabalho em equipe e resolução de problemas complexos.