UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Desenvolvimento Mobile

Francisco Davi Rodrigues Xavier

Entrega 3 - Relatório

QUIXADÁ
2021.1

Sumário

1 Comunicação e Persistência

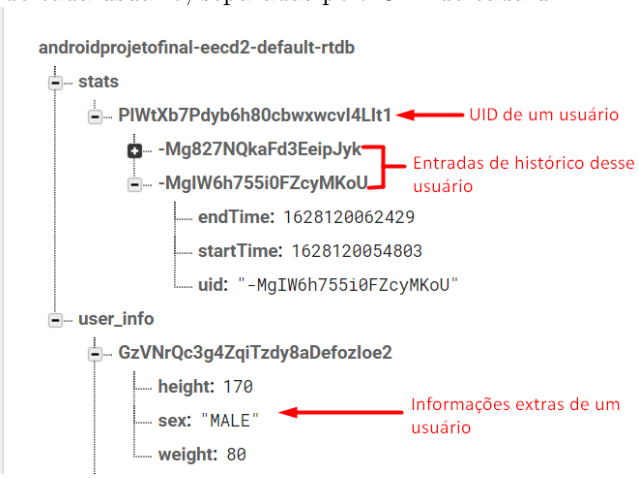
O back-end utilizado para o desenvolvimento do aplicativo foi o Firebase com realtime database para persistência de dados. A comunicação em si com o Firebase foi separada em uma camada de repository, encapsulando as chamadas.

Visto que o Firebase utiliza a API de Tasks para suas chamadas, foi utilizada a biblioteca *kotlinx-coroutines-play-services*, que adiciona uma função de extensão para que as chamadas que retornam uma Task possam ser síncronas em vez de callbacks, ou seja, permitindo o uso das mesmas em co-rotinas do Kotlin. Desse modo, todas as chamadas de escrita para o Firebase foram implementadas utilizando co-rotinas, onde as chamadas para suspend functions são iniciadas das activities ou fragments utilizando-se do seu escopo de execução e têm seu contexto trocado para um dispatcher de IO na camada de viewmodel.

Para chamadas síncronas, ou seja, chamadas de get, foram feitas por meio de uma implementação customizada da classe LiveData, que recebe uma Query do realtime database e gerencia um listener próprio para a mesma, sendo responsável por construir os dados corretamente para o retorno no callback de observe, deixando, assim, mais simples o gerenciamento de listeners para as queries do Firebase.

1.1 Estrutura do Banco

Na estrutura do realtime database devem existir dois nós iniciais, um para guardar informações adicionais para cada usuário e outro para guardar as entradas de histórico de cada usuário, separadas pelo UID de cada um.



1.1.1 Regras de Acesso

Um usuário só deve poder acessar as entradas de histórico e as informações que estejam relacionadas ao seu UID, o mesmo também deve estar logado. Desse modo, foram definidas as seguintes regras de acesso no Firebase:

```
{
  "rules": {
    "user_info": {
      "$uid": {
        ".write": "auth != null && auth.uid == $uid",
        ".read": "auth != null && auth.uid == $uid"
      }
    },
    "stats": {
      "$uid": {
        ".indexOn": ["endTime"],
        ".write": "auth != null && auth.uid == $uid",
        ".read": "auth != null && auth.uid == $uid"
      }
    }
  }
}
```

2 Tarefas em Segundo Plano

2.1 Usuário - UserViewModel

- Login, *login* - Realiza o login do usuário.
- Logoff, *logoff* - Realiza o logoff do usuário.
- Histórico Atual, *setCurrentStatUid* - Seta a entrada de exercício atual do usuário. Chamado quando uma entrada nova é criada para aquele usuário.
- Cadastro, *signUp* - Cadastra um novo usuário no back-end do app.
- Esqueci minha senha, *forgotPassword* - Envia um e-mail de verificação de senha para o usuário.

2.2 Histórico

- Criar histórico, *createStat* - Cria uma nova entrada no histórico com UID e uma timestamp de criação. Chamado quando o fab de play no fragment de home é tocado.
- Finalizar históricos pendentes, *finishStats* - Finaliza todos as entradas do histórico que ainda não tem data final, deve ser chamado no momento de

criação de uma nova entrada, desse modo garantindo que nunca vai ter duas entradas ativas ao mesmo tempo.

- Remover histórico, *removeStat* - Remove uma entrada do histórico por seu UID.
- Atualizar histórico, *updateStat* - Atualiza uma entrada do histórico. Os valores de UID, timestamp de início e timestamp de finalização não devem poder ser alterados.