

Dokumentácia k semestrálnej práci z predmetu *VAMZ*

RACELOG

**vypracoval:** Branislav Blažek

**študijná skupina:** 5ZYI21

**cvičiaci:** doc. Ing. Patrik Hrkút, PhD.

**Termín cvičenia:** streda blok 1-2 V Žiline dňa 11.06.2024

1. Popis a analýza riešeného problému

RaceLog je aplikácia na zaznamenávanie času na pretekoch. Jej výhodou by mala byť online dostupnosť dát a jednoduché zapojenie.

Existuje pomerne veľa profesionálnych časomier ktoré majú o dosť viac prepracované svoje mobilné aplikácie. Ich nevýhodou (v našom prípade) je spomínaná komplexnosť a priamo úmerne aj ich cena. Mojim cieľom bolo vytvoriť použiteľnú časomieru na menšie, menej významné preteky za ktoré nezaplatí organizátor trojciferné sumy na jedny preteky. Samozrejme moja nie je presná na tisíciny sekúnd ale tieto údaje nie sú pre nás podstatné. Ide hlavne zmerať čas približne na sekundy, vedieť rozloženie pretekárov na jednotlivých checkpointoch a mať výsledky v reálnom čase.

Najpoužívanejšie u nás sú:

• https://www.sportsofttiming.sk/

• https://www.vysledkovyservis.sk/

• https://www.raceresult.com/sk

• https://mobilnacasomiera.sk/

Neuvádzam priamo odkaz aplikácií na Obchod Play ale na webové stránky časomier z dôvodu, že tieto aplikácie nie sú verejne publikované a spravidla jednotlivé firmy majú svoje mobilné zariadenia, kde majú tieto aplikácie nainštalované „cez kábel“.

Teda firma na preteky požičiava svoje mobilné zariadenia s nahratou aplikáciou, čím sazdvíhajú výdavky

Aplikácia je napísaná v Kotline, primárne na použitie na Android zariadeniach, ktoré podporujú API od verzie 26 vyššie. Aplikácia ukladá dáta na Firebase databázu. Nie je potrebné sa do nej nijak prihlasovať. Databáza je nateraz voľne prístupná verejnosti (cez aplikáciu).

1. Návrh riešenia problému

Prípady použitia:

1. Sledovanie pretekov

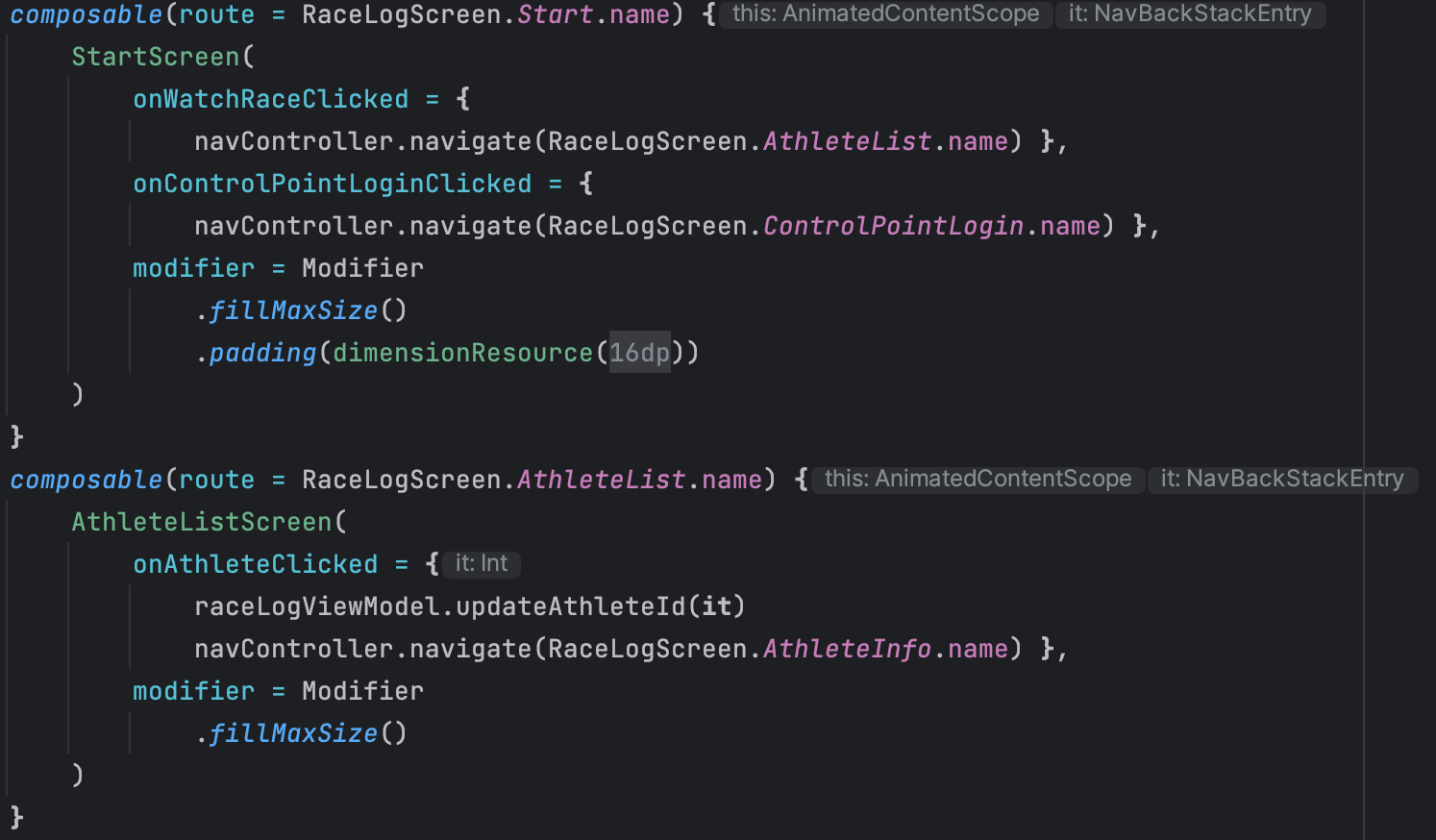
* Používateľ chce sledovať vývoj preteku, klikne na Sledovať preteky.
* V zozname vie filtrovať pretekárov podľa mena, priezviska alebo BIB čísla (pretekárske číslo).
* Klikom na Prehľad sa zobrazí detailné informácie o pretekárovi.
* Tu je možné vidieť viacero stavov:
  + stav = či pretekár dobehol, beží alebo je vyradený
  + čas = čas pretekára zaznamenaný na poslednej kontrole
  + poradie = poradie v rámci absolútneho počtu bežcov
  + Krajina, vek, tím = doplnkové informácie

1. Zapisovanie pretekárov
   * Okno dostupné len pre organizátorov behu.
   * Potrebné poznať prihlasovacie údaje pre konkrétnu stanicu
   * Po prihlásení je možné nahlasovať prebeh pretekárov. Základná funkčnosť je napísanie čísla pretekára a nahratie do databázy.
   * V prípade iného nahlásenia je možnosť kliknúť na trojbodku.
   * Tu je možné diskvalifikovať pretekára alebo ho inak vyradiť.
   * Všetky zmeny sa ihneď prejavia na výpise.
2. Popis implementácie

3.1. Navigácia

Pohyb medzi obrazovkami je umožnený vďaka Jetpack Navigation. Je definovaných 5 ID v enume. Vďaka týmto ID sa knižnica vie orientovať ktorú obrazovku má vykresliť. Každá obrazovka má definovanú akciu, ktorá sa zavolá po potrebe navigovať do inej časti aplikácie. Niekde to je volané klikom na tlačidlo, inde úspešným prihlásením sa.

Navyše celé toto je zabalené v Scaffolde, ktorý má implementovaný TopBar. Vďaka nemu užívateľ vie na akej stránke sa práve nachádza (pre každé ID je definovaný názov stránky) a vie sa klikom na šípku späť vrátiť o jednu úroveň vyššie. Správanie navigácie je podobné ako pri webových stránkach.



Obr. 1: Ukážka vetvenia, ktorú obrazovku zobraziť na základe ID

3.2. ViewModel

Na uchovanie stavu aplikácia aj po zmene aktivity sa využil ViewModel. Každá stránka má definovaný svoj ViewModel. Ten sa skladá z PageUiState súboru, ktorý obsahuje data class a v nej jednotlivé stavy. Tieto dáta potom spracúva súbor PageViewModel. Tento súbor ponúka metódy na zmenu stavu.

Okrem ViewModels na správu stavu ešte jednotlivé obrazovky využívajú aj ViewModel pre komunikáciu s databázou (popísané v ďalšej časti).

A screenshot of a computer program

Description automatically generated

Obr 2. Ukážka UiState pre prvú obrazovku

A computer screen shot of a program code

Description automatically generated

Obr 3. Ukážka ViewModel, definícia metód na zmenu stavu

A screen shot of a computer program

Description automatically generated

Obr 4. Volanie ViewModel v konkrétnej obrazovke

A screen shot of a computer program

Description automatically generated

Obr 5. Ukážka volania metódy po spracovaní kliku od užívateľa

3.3. Firebase Firestore

Na ukladanie dát som nezvolil riešenie ponúknuté na cvičení ale vyskúšal som Firebase databázu. Na samotné využívanie bolo najprv potrebné nastaviť konfiguráciu vo webovej Google konzole. Následne som stiahol do projektu súbor na certifikáciu komunikácie mojej aplikácie s databázou a pridal závislosti.

Samotná inicializácia databázy je v súbore MainActivity.kt kde trebalo pod volanie aplikácia pridat riadok:

FirebaseApp.initializeApp(this)

Následne som v nastaveniach databázy vytvoril dokumenty a nahodil testovacie dáta. Po tomto bolo potrebné vytvoriť data class s presne tými istými kľúčami a ich hodnotami, ako je aj v databáze. To je potrebné kvôli prevodu databázových dát do Kotlin objektu.

Na prácu s databázou sa využívajú funkcie, ktoré sú volané v korutinách vo viewModel. Telo funkcie tvorí:

* Získanie inštancie databázy.

 vykonáva sa jednoducho prostredníctvom príkazu

val db = FirebaseFirestore.getInstance()

* Príprava objektu dát ktorý obsahuje dáta z databázy.

val athletes = *mutableListOf*<Athlete>()

* Query príkaz.
  + - Triviálny query môže vyzerať takto:

db.collection("athletes").get().await()

Tento príkaz vezme zadanú kolekciu (musí existovať v databáze), vytiahne z nej všetky dáta a počká, dokým sa všetky načítajú.

* + - Rozšírený query sa dá napísať aj takto:

querySnapshot = db.collection("raceLogs")  
 .whereEqualTo("athleteBib", raceLog.athleteBib)  
 .whereEqualTo("controlPointId", raceLog.controlPointId)  
 .get()  
 .await()

Vykonáva obdobne aj query čo vyššie, ale zužuje množinu výsledkov podľa zadaných podmienok. Tento príkaz vracia len jeden objekt.

* + - Pre prípad skladania query sa dá využiť kód:

val querySnapshot = db.collection("raceLogs")  
 .*let* **{** query **->** if (athleteBib != null) {  
 query.whereEqualTo("athleteBib", athleteBib)  
 } else {  
 query  
 }  
 **}** .*let* **{** query **->** if (controlPointId != null) {  
 query.whereEqualTo("controlPointId", controlPointId)  
 } else {  
 query  
 }  
 **}** .get()  
 .await()

Tu sa očakáva, že vstupné parametre nemusia byť zadané (teda nastavené na null hodnotu). V takom prípade nechceme porovnávať hodnotu s null, tak preto sme pridali podmienku. Ta zaručí, že ak nemám hodnotu porovnania, porovnanie sa nevykoná.

* + - Posledná ukážka je na vytvorenie záznamu v tabuľke

db.collection("raceLogs").add(raceLog).await()

Tu je zmena oproti triviálnej query, že nevolám metódu .get() ale .add() s parametrom. Parameter obsahuje triedu RaceLog, ktorej hodnoty sú zadané.

* Premapovanie databazových objektov do Kotlin objektov.

document.toObject(ControlPoint::class.*java*)

Tento kód vezme document, ktorý mi vrátila databáza a snaží sa ho prekonvertovať na Kotlin objekt. Tu je veľmi dôležité, aby to ako objekt vyzerá v databáze bol rovnako prepísaný ako objekt v Kotline na ktorý sa mapuje.

To znamená, že ak tabuľka\* obsahuje stĺpce s názvami:

Id, name, lastName, age

a typmi

Int, String, String, Int

musí mať tieto hodnoty rovnaké aj Kotlin objekt. Ani o hodnotu viac ani menej.

* Zachytenie prípadnej chyby.
* Vrátenie dát.

1. Zoznam použitých zdrojov

Android developer kurz od Google

Android developer príručka

Prednášky

Obrázky v aplikácií stiahnuté z fotobanky https://unsplash.com