MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Nik Harmanec

Interaktivní hra na platformu Android

Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: RNDr. Jan Kofroň, Ph.D.

Studijní program: Informatika

Studijní obor: Softwarové inženýrství

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V …..... dne..... podpis

Název práce: Interaktivní hra na platformu Android

Autor: Nik Harmanec

Katedra / Ústav: Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: RNDr. Jan Kofroň, Ph.D., Katedra distribuovaných a spolehlivých systémů

Abstrakt: Práce se zabývá tvorbou mobilní aplikace na platformě Android. Jedná se o interaktivní online hru odehrávající se v reálném světě. Uživatelé na mapě uvidí závody ve svém okolí. Ve chvíli kdy se přiblíží ke startu budou ho moci odstartovat a úkolem bude se, jakýmokoli způsobem, co nejrychleji dostat do cíle. Výsledky se budou zaznamenávat a u každého závodu bude seznam nejlepších časů s jejich autory. Aplikace sestává ze dvou částí. Z backendu, starajícího se o ukládání a zprostředkovávání dat a z frontendu, kterým je samotná mobilní aplikace a kterou má uživatel na svém telefonu.

Klíčová slova: hra, rozšířená realita, Android

Title: Interactive Game for Android

Author: Nik Harmanec

Department: Department of Distributed and Dependable Systems

Supervisor: RNDr. Jan Kofroň, Ph.D., Department of Distributed and Dependable Systems

Abstract: The thesis focuses on creating a mobile application on the Android platform. It is an interactive online game which takes place in the real world. Users will see races around them in the map. When they get close enought to the start they will be able to start the race. The goal will be to get to the finish line as quickly as possible. The results will be recorded and a list of the best times with their authors will be displayed for each race. The application consists of two parts. From the backend that takes care of storing and providing data and from the frontend which is the mobile application itself and which the users have on their phones.

Keywords: game, augmented reality, Android

Na tomto místě bych rád poděkoval svému vedoucímu RNDr. Janu Kofroňovi, Ph.D za jeho čas a ochotu při psaní této práce. Také bych rád poděkoval své rodině, zejména svému bratrovi a své přítelkyni, za neutuchající podporu.

Obsah

1. Úvod **1**

2. Analýza **2**

2.1. Platforma Android 2

2.2. Struktura 3

2.3. Technologie 5

2.4. Uživatelské rozhraní 11

2.2. Podobné aplikace 12

3. Architektura **13**

3.1. Databáze 13

3.2. API volání 15

4. Implementace **18**

4.1. Backend 18

4.2. Frontend 19

5 Závěr **30**

5.1. Srovnání výsledku se zadáním 30

5.2. Možná rozšíření 30

6. Seznam použité literatury **32**

7. Přílohy **35**

7.1. Uživatelská dokumentace 35

7.2. Struktura databáze 42

7.3. Dokumentace API volání 44

7.4. Obsah přiloženého CD 50

1. Úvod

1.1. Motivace

Každý má rád hry, každý má v dnešní době chytrý mobilní telefon. Cesta do školy, do práce, na dovolenou většinou není moc zábavná, proč si ji tedy nespestřit jednoduchou hrou, ve které je možné soutěžit se svými kamarády, kolegy i lidmi z celého světa. Většina mobilních her nevyžaduje mnoho pohybu, ba naopak. Oproti tomu tato aplikace má potenciál motivovat lidi k pohybu venku. Rozšířená realita je velmi populární, uživatelsky atraktivní, ale v současnosti mnoho takových her není.

Rozšířená realita je propojení reálného světa se světem virtuálním. Má široké uplatnění napříč obory. Dá se použít v architektuře, kde si architekt může zobrazit objekt přímo do reálného světa, v internetových obchodech, kdy si zákazník může vyzkoušet virtuální produkty v reálném prostředí, ale třeba i ve vědě a vzdělávání, k vizualizaci různých experimentů a dat. V neposlední řadě se používá v zábavním průmyslu a ve hrách. Kreativitě se meze nekladou, hráči můžou v reálném světe střílet virtuální zombíky, skládat puzzle nebo hrát geocachingové hry.

1.2. Cíl práce

Cílem práce je vytvořit onlinu hru pro mobilní telefony. Hra bude používat rozšířenou realitu a odehrávat se v reálném světě. Cílem hry bude dostat se co nejrychleji z konkrétního bodu na mapě do nějakého jiného bodu. Na mapě uživatel uvidí jednotlivé závody v okolí, přijde ke startu a poté se musí co nejrychleji, jakýmkoliv způsobem, dostat do cíle závodu. Uživatelé si budou moci vytvořit profil a jednotlivé výsledky se budou zaznamenávat, a tedy je uvidí všichni uživatelé. U jednotlivých závodů budou žebříčky s nejrychlejšími závodníky a jejich časy. Hra bude obsahovat jak předem vygenerované závody, tak závody, které vytvoří sami uživatelé.

2. Analýza

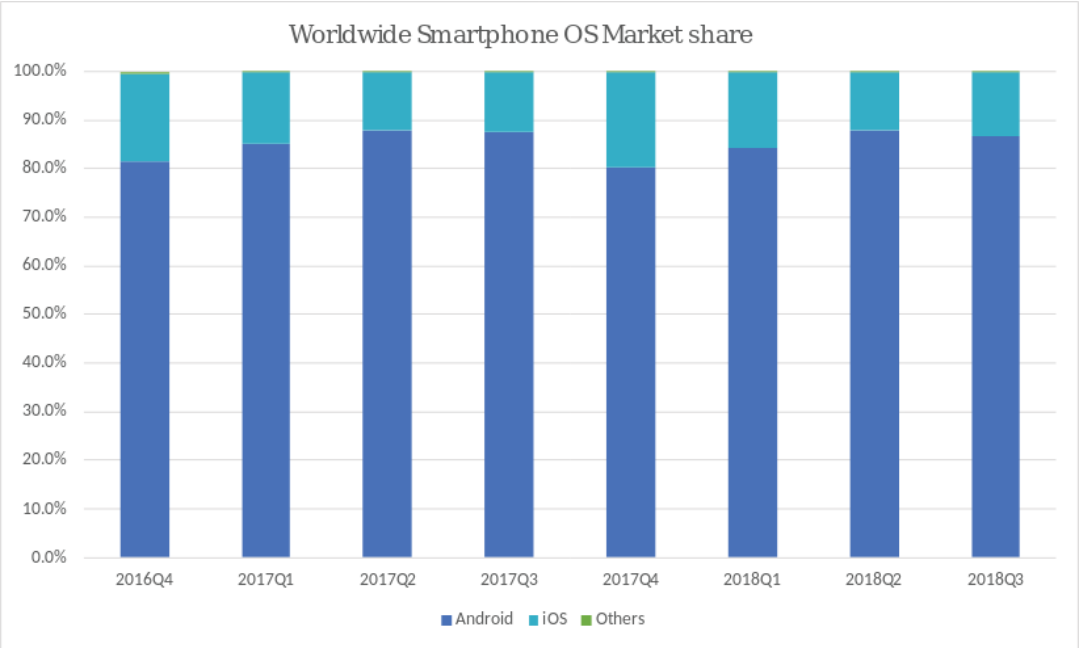
2.1. Platforma Android

Dnešní mobilní trh ovládají v podstatě dva hráči. Firma Google s platformou Android a Apple se svými iPhony a iOS. Byl zde pokus i ze strany Microsoftu se systémem Windows phone, ten ovšem v roce 2018 oznámil, že s vývojem končí a doporučil přechod právě na výše zmíněný Android nebo iOS.

Jako první odstartoval revoluci s chytrými telefony Apple, když v roce 2007 představil první iPhone. Další firmy na sebe nenechaly dlouho čekat a začaly vycházet další dotykové telefony. Nejdříve nebyl žádný jednotný systém a jednotlivé značky si vyvíjely pro vlastní telefony vlastní systémy. V roce 2008 ovšem přišel Google se svým řešením, tehdy ještě opensource, Androidem. Díky své otevřenosti nabýval rychle na popularitě, v roce 2009 už ho podporovalo více než dvacet zařízení a v roce 2010 se stal nejvíce zastoupeným mobilním systémem na trhu. V roce 2012 už měl nadpoloviční většinu a v roce 2013 už jasně dominoval s 80% viz obrázek 1 [1].

Oproti Applu, který od počátku držel svůj ekosystém zavřen, byl Android vývojářům otevřen, první Android SDK byly dokonce opensource. Nejprve se na Android vyvíjelo v Eclipse, později se Google rozhodl přejít k intelliJ IDEA a používat název Android Studio. Opět narozdíl od Applu, kde se aplikace na iOS dají vyvíjet pouze na macu, Android Studio se dá spustit jak na Windows a Linuxu, tak na macOS.

Android je operační systém postavený na modifikovaném Linuxovém jádře a dalších open source technologiích. Je uzpůsoben pro dotykové zařízení zejména pro mobilní telefony a tablety, ale také pro televize, automobily, chytré hodinky a další zařízení. Systém se vývojářům stará o vše od správy paměti, komunkace se senzory, až po zobrazování notifikací nebo vykreslování animací.

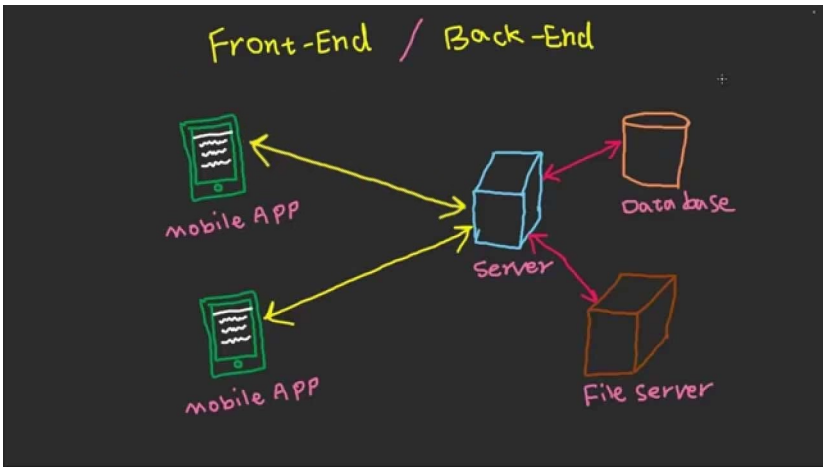


1. Podíl jednotlivých platforem na trhu. *Zdroj: https://www.idc.com/promo/smartphone-market-share*

2.2. Struktura

Jedná se o online hru, ve které hráči sdílí data s ostatními hráči. Závody jsou společné pro všechny. Je tedy potřeba, aby všichni hráči měli na mapě k dispozici stejné závody a aby hráči viděli výsledky ostatních hráčů. Toho nelze, nebo jen velmi obtížně, dosáhnout jenom tak, že hráči budou mít všechny data uloženy ve svých zařízeních a budou si je předávat peer-to-peer. Zároveň by to byl poměrně velký objem dat a aplikace by se tak velmi rychle rozrostla a potenciálně by mohla zabrat všechno místo v telefonu. Aplikace by měla uložená i data, které uživatel nepotřebuje a ani je nevidí, což nechceme.

Je tedy potřeba mít k aplikaci ještě nějaký společný server, na který se budou moci jednotlivá zařízení připojit, přečíst si data, která potřebují a zapsat ta data, který sdílí s ostatními. Tomuto serveru budeme říkat backend a aplikaci, kterou uživatelé vidí budeme říkat frontend. Viz obrázek 2.



2. Frontend / Backend diagram. *Zdroj: https://www.youtube.com/watch?v=nMtgFZSdtwk (upraveno)*

Uživatele potřebujeme podle něčeho identifikovat, aby se všechny jejich výsledky zobrazovali pod stejnou identitou. Toho by se dalo docílit třeba použítím unikátního čísla telefonu, to má ovšem tu nevýhodu, že pokud si uživatel pořídí nový telefon, nebude už schopen se dostat do stejného účtu. Proto je lepší použít přihlášení a vytvoření uživatelského účtu. Uživatel si zároveň může vytvořit vlastní unikátní přezdívku, podle které bude snadno identifikovatelný.

Přihlašovací systém dnes nabízí řada firem. Výhoda těchto přihlášení je hlavně pro uživatele, který nemusí mít na X službách X různých účtů, stačí se v aplikaci vždy jedním kliknutím přihlásit přes například Facebook. Ovšem ne každý má účet na Facebooku a tedy by byla část uživatelů od aplikace odříznuta. To by mohlo vyřešit přilášení přes Google, kde účet musí mít každý, kdo používá Android telefon. Nevýhoda těchto přihlášení je závislost na daných službách a malá kontrola nad daty uživatele. Jejich API se můžou v průběhu času měnit a bude tedy nutné aplikaci adaptovat, jinak se přihlašování rozbije. Z těchto důvodů bylo zvoleno vlastní řešení přihlášení, pomocí přezdívky, emailu a hesla.

2.3. Technologie

2.3.1 Backend

Backendová část aplikace má tedy za úkol ukládat data a prostřednictvím API je předávat aplikaci. Data jsou více méně statická a mění se a přidávají pouze když přijde volání z frontendu.

Ukládání dat

Data se dají ukládat v různých strukturách různými způsoby, každý z nich má nějaké výhody a nevýhody. Na strukturu, v které budeme data ukládat jsou následující nároky:

- **Persistence**

  Data musí zůstávat v takovém stavu, v jakém je uložíme. Nesmí se samovolně mazat ani upravovat.

- **Rychlost**

  Je potřeba, aby zejména čtení ze struktury, bylo co nejrychlejší a uživatel tedy nemusel na každý dotaz dlouze čekat.

- **Škálovatelnost**

  Dalším důležitým kritériem je škálovatelnost. Struktura musí být schopná pracovat i s velkým objemem dat.

- **Rozšiřitelnost**

  Ne úplně nejzásadnější, ale určitě dobrá vlastnost struktury je rozšiřitelnost, v případě, že bude rozhodnuto, že se aplikaci bude dále rozšiřovat.

Dnes v aplikacích asi nejrozšířenější způsob je použití databáze. Výhoda databáze je poměrně snadné nasazení a údržba. Pokud se databáze vhodně navrhne, je i poměrně rychlá. Databáze je jednoduše rozšiřitelná, stačí přidat další tabulky. Nevýhodou

může být nutnost vlastního serveru, kde databáze běží. Při nevhodné struktuře databáze můžou některé dotazy trvat poměrně dlouho.

Variantou, která by odbourala problém vlastního serveru, je mít data v cloudu. Jelikož vyvíjíme aplikaci pro telefon, nabízí se Firebase Realtime Database [2]. Její nespornou výhodou je provázanost s platformou Android (ale i iOS). Nevýhodou by byla totální závyslost na Googlu, který Firebase vyvíjí a vlastní. A při větších objemech dat a větší frekvenci dotazů je tato varianta poměrně finančně náročná.

Další možností je vytvořit si nějaký vlastní systém ukládání dat například do souborů na server. Výhodou by byla možnost veliké optimalizace na konkrétní řešení a mohlo by jít dosáhnout rychlých výsledků. Toto řešení by ovšem bylo velmi pracné (s nejistým výsledkem) a pravděpodobně nad rámec této práce.

Pro potřeby aplikace byla jako nejvhodnější vybrána databáze. Splňuje všechny kritéria a je poměrně snadno propojitelná s další částí backendu.

Komunikační protokol

Nad daty je potřeba vytvořit ještě vrstvu, která bude zpracovávat příchozí dotazy, najde správná data a vrátí je zpět. Obě strany přitom musí používat nějaký, stejný, protokol, aby si rozuměly. Jak bylo zmíněno výše, uživatelé se budou do aplikace přihlašovat, a tedy budeme chtít, aby se uživatelé v rámci přenosu dat nějak autentifikovali. Nabízí se použít nějaký stavový protokol, kterým se uživatel jednou přihláší a pak si bude držet toto spojení, přes které bude komunikovat. Takovým protokolem je například SMTP. Toto řešení není ovšem úplně vhodné do mobilní aplikace, protože potřebuje udržovat spojení. Při závodění budou uživatelé typicky připojení na mobilní data a pokud bude uživatel závodit třeba někde v přírodě, nemusí být signál úplně silný a spojení může vypadávat. Bude tedy lepší použít nějaký bezestavový protokol a přihlašování vyřešit jinak. Takový protokol, který se dnes používá na většinu přenosů na internetu a Android ho umí zpracovávat je HTTP (nebo jeho, o šifrování, rozšířená varianta HTTPS).

Nad protokolem je ještě potřeba definovat rozhraní, kterému budou rozumět obě strany. Možnosti rozhraní jsou dnes v podstatě dvě. Representational state transfer (REST) [3] a Simple Object Access Protocol (SOAP) [4]. REST přenáší data ve formátu JSON, zatímco SOAP je přenáší ve formátu XML. XML, díky své zdlouhavé syntaxi, je náročnější na parsování a na přenos dat a proto, pro telefonní aplikaci, kde uživatelům typicky záleží na spotřebě dat, je vhodnější úspornější JSON.

REST nabízí 4 základní metody volání:

- **GET**

  který se používá k získávání dat. Měl by být read-only, tedy volání nedostává žádná vstupní data a není nic zapisováno do databáze, pouze se z ní čte.

- **POST**

  POST volání zapisují nová data do databáze.

- **PUT**

  Pomocí volání PUT se data v databázi upravují.

- **DELETE**

  Jak název napovídá, DELETE volání data z databáze maže.

Vidíme, že volání nám krásně pokrývají to, co od backendu budeme chtít. Pomocí GET dostaneme závody, informace o uživateli atd., pomocí POST uložíme do databáze nový výsledek nebo vytvoříme nový závod, pomocí PUT můžeme měnit uživatelovi preference a pomocí DELETE třeba smazat účet.

Poslední nevyřešenou částí je tedy autentifikace uživatele. Jelikož je přenos bezestavový, nebude nám stačit uživatele jednou identifikovat, ale budeme muset při každém volání posílat nějakou formu identifikace. Nejpřímočařejším řešením by bylo si uložit uživatelovo přihlášovací jméno a heslo a posílat ho s každým dotazem. To ovšem není úplně bezpečné, heslo chceme použít pouze na první identifikaci a

poté si vytvořit přihlašovací token, který bude platný po omezenou dobu a sloužit jako uživatelova autentifikace.

Token tedy budeme posílat s každým dotazem. Nemůžeme ho ovšem posílat jako data, to by potom všechna GET volání musela být změněna na POST. URL volání také není úplně ideální místo, protože je lehce čitelné. Naštěstí HTTP protokol nabízí „schovat“ některé malé údaje do hlavičky volání, což bude ideální místo pro náš token.

Backend bude tedy zpracovávat RESTové volání a podle něj číst a zapisovat do databáze. Takové řešení už existují, například populární Spring framework [5] v jazyce Java. Jeho výhodou je jeho robustnost a rozšiřitelnost. Jelikož je napsán v jazyce Java, vyžaduje větší náročnost na hardware, nejspíše tedy vlastní server.

Dalším možným řešením je napsat backend nad JavaScriptovým NodeJS. JSON je nativní JavaScriptový objekt a tedy by implementace byla poměrně přímočará. Oproti Javě nemá takové nároky na hardware, nicméně JavaScript je poměrně pomalý a ne tak robustní.

Kompromisem mezi Springem a NodeJS může být PHP. Pro jeho běh není potřeba nijak výkoný hardware a může běžet klidně na sdíleném hostingu, který je v dnešní době cenově dostupný. Pro práci s JSON formátem má nativní knihovny a požadavky REST rozhraní také zvládne. Pro potřeby aplikace bude tedy dostatečné.

Poslední částí backendu, která není rozhodnutá je databáze. I zde je na trhu mnoho variant. Jednou z nich je Oracle [6]. Jedná se o robustní databázový systém, s celou řadou pokročilých funkcí, vysokým výkonem a snadnou škálovatelností. Databáze ovšem vyžaduje výkoný hardware a je zpoplatněna.

Otevřenou open-source variantou je třeba PostgreSQL [7]. Poslední dobou více a více populární databáze vyvíjená pro unixové systémy. Skýtá celou řadu pokročilých funkcí, ale opět vyžaduje vlastní server.

MySQL je tedy opět kompromisem, který je k dostání k sdílenému hostingu a je velmi dobře propojitelná s PHP. Není tak rychlá, jako Postgres nebo Oracle, ani nemá tolik pokročilých funkcí, ale pro potřeby této aplikace bude dostačující.

2.3.2. Frontend

Frontend je mobilní aplikace pro platformu Android, kterou má uživatel na svém telefonu a v které se hra odehrává. Přes backend získává data, tak aby měli všichni uživatelé k dispozici stejné závody a sdílí své výsledky tak, aby je v žebříčcích viděli všichni uživatelé.

Mobilní aplikace jdou psát dvěma způsoby. Buď nativně, nebo pomocí frameworku. Oba přístupy mají své výhody a nevýhody. Nativní přístup má výhodu, že aplikace bude mít přístup ke všem systémovým API, poběží rychleji a bude mít responzivnější uživatelské rozhraní. Nevýhodou je svázanost s danou platformou a tedy aplikace nebude nijak přenositelná.

U frameworku existují dva druhy přístupu. Buď se jedná v podstatě o webovou aplikaci, která běží ve WebView (př. Ionic [8]) nebo se aplikace kompiluje do nativního kódu (př. ReactNative [9], Flutter [10]). První přístup má výhodu, že aplikaci stačí psát jednou pro web i pro mobilní aplikaci. Ovšem nemůže využívat pokročilejší systémové API a běží o poznání pomaleji. Obecnou výhodou frameworků je, že je aplikace přenositelná mezi více systémy (Android, iOS, ..), nevýhodou pak je, nemožnost používat systémové funkce specifické pro danou platformu.

Vzhledem k tomu, že vyvíjíme aplikaci na platformu Android, byl výběr technologií pro frontend poměrně přímočarý. Aplikace je napsána nativně v jazyce Java. Alternativou, nebo spíše rozšířením, by bylo použití Kotlinu [11], což je jakási nadstavba Javy, která má vylepšit syntax. Zkracuje některé zápisy, nemá nutnost psaní středníků atp. Vyvíjí ho firma Jet Brains a Google ho momentálně upřednostňuje pro vývoj Android aplikací. Nicméně veškerý kód v Kotlinu je kompilován zpět do Javy.

Od roku 2008, kdy vyšel první Android, vyšlo už několik verzí. Aktuální používaná verze je 9.0 (Pie). Podporovat všechny verze je v současné době nesmyslné. Jednak by musely být aplikace v podstatě dvě, protože staří verze nepodporují některou novou funkcionalitu a hlavně starší verze Androidu už v podstatě nikdo nepoužívá. Google, v Android studiu, uvádí, že v podstatě 100% zařízení je Android 4.0 a výš. Nicméně Android od verze 4.0 do verze 9.0 ušel poměrně značnou cestu a proto aplikace podporuje zařízení od verze 4.4 (KitKat) a výšší. Zde Google uvádí zastoupení zhruba 95,3% Android zařízení.

Aplikace ke svému fungování potřebuje připojení na internet, ke komunikaci se servrem a provázání s ostatními uživateli a GPS, k zaznamenávání a sledování polohy během závodu. Proto jsou podporována pouze zařízení, které tyto dvě funkcionality mají.

V aplikaci bylo potřeba vyřešit několik funkcionalit. Hlavní z nich jsou mapy, které jsou stavebním kamenem aplikace. Google nabízí v základu Androidu propojení s Google maps. Jejich velkou výhodou je široká nabídka funkcí [12] a jejich úzká provázanost s Androidem. Nevýhodou může být jejich zpoplatnění, při překročení určitého počtu requestů. Alternativou by mohly být například Openstreetmaps [13], které jsou v některých místech detailnější. Ty ovšem nenabízí tolik funkcí a jejich propojení s Androidem je řádově náročnější.

Další funkčnost, kterou bylo potřeba vyřešit bylo zpracovávání JSON souborů, které chodí z backendu. Android respektive Java má nativní knihovnu pro práci s nimi. Nicméně by bylo pro každou možnou odpověď potřeba připravit vlastní parsování. To je poměrně neefektivní a nepraktické na případné rozšiřování aplikace. Proto byla vybrána knihovna gson [14], přímo od Googlu, která JSON namapuje přímo na objekt. To umožňuje velkou generalizaci a jednoduchou rozšiřitelnost.

Obdobně jako v backendové části, je i v aplikaci potřeba někde uchovávat lokální data. Ideálně, aby data, která už z backendu přišla a uživatel je často využívá zůstala lokálně a nemusela se pokaždé znovu tahat z backendu. Zároveň ale

nechceme, aby zabíraly hodně prostoru a aplikace zůstala co nejmenší. Nabízí se si ukládat data do vlastních souborů, což by ovšem bylo poměrně náročné na implementaci a pravděpodobně dost pomalé. Lepší variantou je, stejně jako na backendu, použít databázi. Android opět nabízí nativní podporu pro SQLite [15], která je naprosto vyhovující potřebám aplikace.

2.4. Uživatelské rozhraní

Často opomíjená, ale velmi důležitá část softwarového díla je uživatelské rozhraní. Aplikace může umět sebelepší věci, ale pokud jí nebudou uživatelé umět nebo moct používat, bude to k ničemu. Od uživatelského rozhraní chceme, aby bylo přehledné a jednoduché, aby se v něm uživatelé vyznali a aby nejdůležitější věci byly na první pohled dobře vidět. Chceme, aby prostředí bylo rychlé, aby uživatel nemusel čekat déle než musí a intuitivní, aby uživatel hned věděl, kam má klikat. Ideálně, aby prostředí bylo i hezké, aby uživatel měl radost z jeho používání.

Google od Androidu 5.0 vydaném v roce 2014 používá tzv. Material design [16], kterým doporučuje, aby se vývojáři řídili. Systémové UI se jím řídí a všechny předinstalované aplikace od Googlu také. Je zde snaha, aby aplikace byly svým způsobem podobné, aby používaly podobné prvky a aby se uživatel v nové aplikaci rychle zorientoval. Zároveň Google dáva k dispozici celou řadu komponent v material design stylu. Aplikace se snaží řídit alespoň základními pravidly tohoto stylu.

Jednotlivé obrazovky jsou na Androidu reprezentované třídami, které dědí od třídy Activity. Ke každé activitě pak patří ještě xml layout soubor definující uživatelské rozhraní. Prvky v tomto rozhraní dědí od třídy View. Android jako takový nabízí širokou paletu prvků k použití a dají se použít i prvky z externích knihoven. Jednu z těchto externích knihoven aplikace používá hdodenhof/CircleImageView [17], pomocí které zobrazuje uživatelské avatary jako kruhy.

2.5 Podobné aplikace

Rozšířená realita je stále populárnější a her odehrávajících se v reálném světe přibývá. Pravděpodobně nejznámější je Pokémon go [18]. Hra se odehrává v reálném světe, ve kterém se hráč pohybuje a loví pokémony. Na mapě jsou tzv. Pokéstopy, ke kterým když hráč přijde dostatečně blízko dostane odměnu. Hráči proti sobě mohou bojovat v zabírání tzv. Gymů, které jsou též rozmístěné v reálném světě. Populární stvoření, zejména před dvěma lety, kdy hra vyšla, lovili miliony lidí po celém světě a hra se rychle stala fenoménem, který odstartoval boom s hrami s rozšířenou realitou.

O něco novější hrou je Angry birds AR: Isle of Pigs [19], pokračování populární série Angry birds. Tentokrát se ovšem níčení zlých prasátek naštvanými ptáky odehrává v reálném světě. Hráč namíří fotoaparát svého zařízení do volného prostoru, kde se mu vykreslí hrad s prasátky, na který z praku střílí naštvané ptáky.

Další poměrně podobnou aplikací jsou různé run trackery, které lidé používají k měření času při běhání, jízdě na kole atp. Uživatelé se snaží dosáhnout co nejlepšího výkonu, který pak můžou sdílet se svými kamarády. Nejsou zde ovšem pevně dané trasy, ani žádné žebříčky nejrychlejších uživatelů. Aplikace jsou primárně určeny k běhu, v naší aplikaci se závodí volným stylem.

3. Architektura

3.1. Databáze

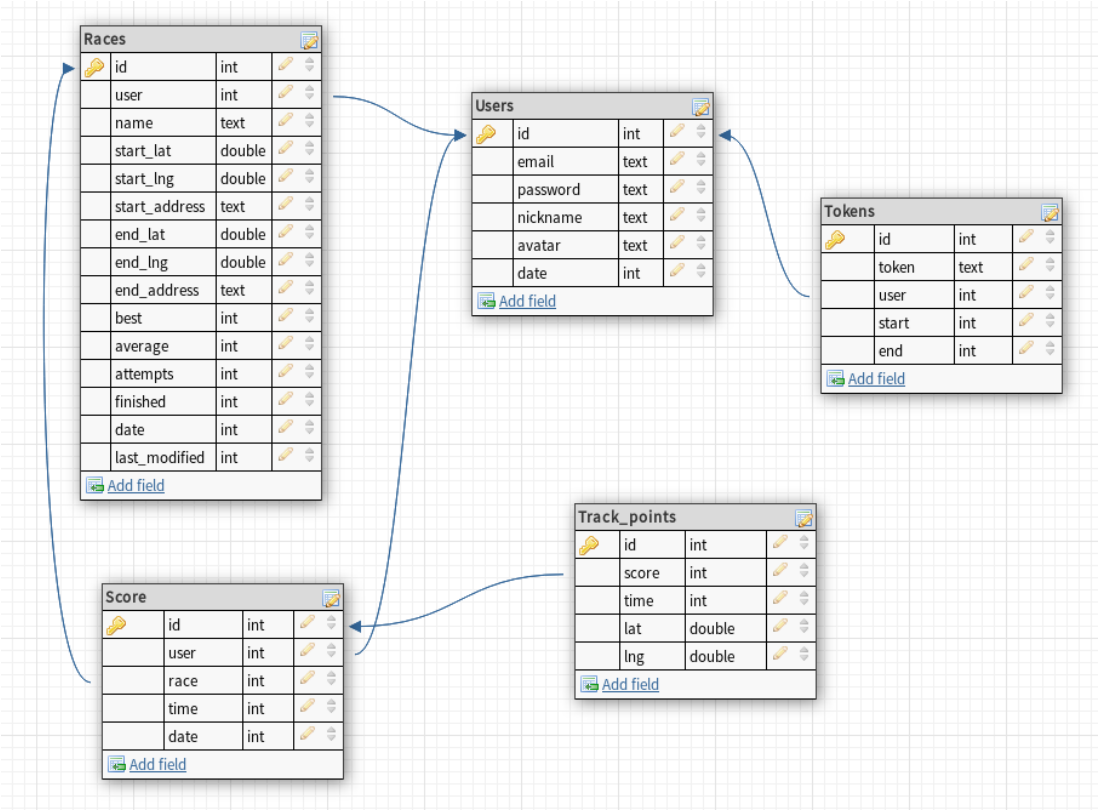
Databázi je vhodné navrhnout tak, aby spolu související objekty bylo možné jednoduchým dotazem z tabulky dostat. Zároveň chceme, aby byla databáze normalizovaná do normální formy [20], tedy: všechny atributy tabulky musí být atomické (1NF), každý neklíčový atribut musí být závislý na primárním klíči (2NF) a neobsahují tranzitivní závislosti (3NF). To nám všechny tabulky kromě *races* splňují. Tam to porušují sloupce s adresami startu a cíle, které jsou závislé na zeměpisné šířce a výšce startu respektive cíle. Nicméně nechceme, aby závody začínaly a končily na stejných místech a adresy se budou používat vždy se zbytkem závodu, tedy toto porušení nebude tolik vadit.

Dále to porušují i sloupce s nejlepším dosazeným časem, průměrným časem, počtem všech pokusů a počtem uživatelů, kteří závod dokončili. Tyto data jsou v tabulce proto, aby se nemusely při každém dotazu znovu přepočítávat. Možným vylepšením by bylo přesunout tyto atributy do nové tabulky. Tyto údaje jsou průběžně upravovány a tedy by stačilo modifikovat menší tabulku. Ovšem čteny jsou data se zbytkem závodu a čteních bude řádově víc než zápisů a tedy byly ponechány v jedné tabulce.

Databáze obsahuje následující tabulky (obrázek 3):

- **Races** – obsahující seznam závodů. Ke každému závodu jsou uloženy tyto informace:
  - id; id uživatele, který závod vytvořil; název závodu; startovní zeměpisná šířka a délka; startovní adresa; cílová zeměpisná šířka a délka; cílová adresa; nejlepší dosažený čas; průměrný čas všech pokusů; počet pokusů; počet uživatelů, kteří závod dokončili; datum vytvoření závodu a datum poslední modifikace
- **Score** - obsahující seznam pokusů. Ke každému pokusu jsou uloženy tyto informace:

    - id; id uživatele, kterému pokus patří; id závodu, ke kterému pokus patří; dosažený čas a datum, kdy pokus proběhl
- **Track_points** – obahující seznam bodů v čase k pokusu. Ke každému bodu jsou uloženy tyto informace:
    - id; id pokusu, ke kterému bod patří; čas, ve kterém byl uživatel v daném bodě a zeměpisná šířka a výška bodu
- **Users** – obsahující seznam uživatelů. Ke každému uživateli jsou uloženy tyto informace:
    - id, email, zašifrované heslo, přezdívka, avatar, datum registrace
- **Tokens** – obsahující seznam přihlašovacích tokenů. Ke každému tokenu jsou uloženy tyto informace:
    - id; unikátní token; id uživatele, kterému token patří; začátek a konec platnosti tokenu



3. Diagram databáze Vytvořeno na www.dbdesigner.net

Kompletní struktura tabulek je v příloze (7.2.).

3.2. API volání

API volání jsou navržená tak, aby splňovala požadavky aplikace a zároveň splňovala pravidla REST rozhraní. Dotazy, které pouze čtou, jsou volána pomocí metody GET. Dotazy, které zapisují nová data jsou volána pomocí metody POST. A dotazy, které data upravují jsou volány pomocí metody PUT. Součástí každého dotazu je hlavička, ve které (kromě dotazů na registraci a přihlášení) je pole *token*, které by mělo obsahovat přihlašovací token uživatele.

Jak již bylo zmíněno výše, odpovědi i vstup jsou ve formátu JSON. Odpověď má vždy pole *success*, které nabývá hodnot *true* nebo *false*, čímž volající rychle zjistí, jestli bylo volání úspěšné. V případě, že volání úspěšné nebylo a pole *success* je *false*, pak jsou zde ještě další dvě pole. *Error*, které značí číslo chybové hlášky a *message*, ve které je chybová hláška.

API má následující volání:

GET /races?location=<lat>,<lng>|<lat>,<lng>&known=id,date|..

Volání, které pro danou lokaci vrací seznam závodů a aktualizaci případných závodů, které už jsou v aplikaci uloženy lokálně.

GET /attempts/<id závodu>?last=<date>

Volání, které vrací seznam uživatelových pokusů pro daný závod, novějších než dané datum.

GET /race/<id závodu>

Volání, které vrací žebříček deseti nejlepších závodníků a jejich časů k danému závodu. Pokud uživatel není v první desítce, vrací i jeho výsledek s pořadím, na kterém se umístil.

GET /user/<id uživatele>

Volaní, které vrací informace o daném uživatelovi. Jeho přezdívku a avatar.

GET /userextended/<id uživatele>

Volání, které vrací detailnější informace k danému uživateli. Vrátí jeho přezdívku a avatar, seznam všech jeho pokusů a všech jím vytvořených závodů.

POST /user

Volání, které vytvoří nový účet. Nejprve zkontroluje, že zaslané údaje jsou validní. Přezdívka má alespoň 4 znaky, sestává pouze z písmen a číslic a nepoužívá jí jiný uživatel, email je validní a nepoužívá ho jiný uživatel a heslo má alespoň 6 znaků. Zároveň rovnou vytvoří přihlašovací token, kterým je uživatel hned přihlášen.

POST /login

Volání, které přihlásí uživatele. Nejprve zkontroluje, že účet s daným emailem nebo přezdívkou existuje a poté zkontroluje platnost hesla. Vytvoří pro uživatele přihlašovací token a uživatele přihlásí.

POST /check_login

Volání, které ověří přihlášení uživatele. Zkontroluje platnost tokenu a ověří, ke kterému uživateli patří. Toto volání by mělo být zavoláno vždy při novém spuštění aplikace.

POST /logout

Volání, které odhlásí uživatele. Změní konec platnosti tokenu na datum volání.

POST /race/<id závodu>

Volání, které zaznamená uživatelův pokus k danému závodu. Uloží jak uživatelův čas, tak jednotlivé body v závodu, ke každému uloží zeměpisnou šířku a výšku a čas ve který v něm uživatel byl. Zároveň upraví v tabulce races celkový průměrný čas, navýší počet uskutečněných pokusů. Pokud uživatel absolvoval tento závod poprvé, navýší i počet uživatelů, kteří závod dokončili. A pokud uživatel zaznamenal rekordní čas, uloží i ten.

POST /newrace

Volání, které vytvoří nový závod. Nejprve zkontroluje platnost odeslaných údajů, že start a cíl jsou od sebe vzdálené alespoň 50 metrů a že jméno závodu má alespoň 5 znaků. Informace o vytvořeném závodu poté vrátí.

PUT /avatar

Volání, které změní avatar pro daného uživatele.

Detailní popis API, včetně formátu vstupu a výstupu je v příloze (7.3.).

4. Implementace

3.1. Backend

Jak již bylo zmíněno výše, backend je naprogramován v jazyce PHP. Je implementován pomocí Front controller paternu [21], který je v PHP poměrně běžný. Pomocí konfiguračního souboru *.htaccess* jsou všechny volání přesměrována na *index.php*. Zde se inicializuje spojení s databází, přečte se přihlašovací token z hlavičky a naparsuje se adresa volání. Poté podle metody volání naincluduje odpovídající soubor pro zpracování, tedy například pro metodu POST naincluduje soubor *post.php*.

V těchto souborech se ověří, jestli se jedná o validní volání a poté deleguje na odpovídající soubor. Tedy pokud budeme volat například GET /search/race/<id závodu>, pak bude volání includovat soubor */get/search/race.php*. Tedy pro každé volání máme právě jeden soubor, který ho zpracovává. U všech těchto souborů musíme ještě ohlídat, aby je nebylo možné otevírat přímo, ale pouze skrz *index.php*.

V souborech zpracovávajících volání se výsledek uloží jako pole do proměnné *result*. Když se pak probublá zpátky do *indexu*, ten na závěr převede obsah proměnné do formátu JSON a vrátí.

Ve složce *objects* je několik společných tříd, které jsou používány napříč celým projektem. Je zde třída *Login*, která pomocí přihlašovacího tokenu ověřuje zda je uživatel přihlášen. Dále je zde třída *User*, která reprezentuje uživatele a vrací k němu odpovídající údaje.

3.2. Frontend

Aplikace je rozdělena do balíčků (package) podle funkcionality. Funkce a obrazovky související s mapou jsou v balíčku *map*. Je zde activita s mapou, třídy a rozhraní starající se o zobrazení závodů na mapě a activita vyhledávání závodů, která je dostupná z mapy.

Funkce související se závody jsou v balíčku *race*. Zde je activita detailu závodu a s ní související pomocné třídy. Je zde i activita na vytváření nového závodu.

V balíčku *profile* jsou funkce související s uživatelským profilem. Je zde activita profilu, activita detailních seznamů dostupných z profilu a activita pro výběr nového avataru. Balíček obsahuje i všechny pomocné třídy k těmto activitám.

Dalším balíčkem je balíček *start*, který obsahuje activity související se startem aplikace, přihlášením a registrací.

Pak projekt obsahuje ještě dva balíčky, které jsou společné pro celý projekt. Prvním z nich je balíček *mapper*, který obsahuje třídy a rozhraní mapující volání z backendu. Druhým je balíček *common*, který obsahuje pomocné třídy, které se používají napříč projektem. Obsahuje třídu pro práci s databází, třídu pro práci s GPS, třídu obsluhující menu aplikace, třídu, která provádí volání na backend a třídu, která parsuje volání z backendu. Dále třídu, která se stará o zobrazování avatarů, třídu která formátuje vzdálenosti a čas a třídu, která se stará o přihlášení uživatele.

Aplikace používá několik externích knihoven:

- **Google Play services**

  Knihovna od Googlu nutná ke správnému fungování Google map, kterým zajišťuje přesnější uživatelovu lokaci.

- **gson** [14]

  Knihovna zajišťující parsování a mapování JSON objektů na třídu a zpět. Jedná se o open-source distribuovaný pod Apache, Version 2.0 licencí.

- **CircleImageView** [17]

  Knihovna pro zobrazování obrázků jako koleček. Aplikace ji používá na zobrazování avatarů. Knihovna je open-source distribuovaná pod Apache, Version 2.0 licencí.

- **FloatingSearchView** [28]

  Knihovna, která je použitá na obrazovce s mapou na vyhledávací pole. Jedná se o open-source distribuovaný pod Apache, Version 2.0 licencí.

Nejdříve si popíšeme některou obecnou funkcionalitu, která je společná napříč aplikací a využívá jí většina activit, poté si popíšeme, jak fungují samostatné obrazovky.

Volání backend API

Pro volání backendového API je společná třída APICall. Třída je potomkem AsyncTask. Na systému Android v hlavním vlákně běží UI, které uživatel vidí a všechny ostatní akce by měly běžet v novém vlákně, zejména ty, které můžou trvat delší dobu, jako komunikace přes internet, dlouhý výpočet atd.

Backend vrací odpovědi ve formátu JSON, v aplikaci je proto vytvořena třída JSONParser. Třída dostane odpověd, zkontroluje, že pole *success* je true, pole odstraní a poté jí pomocí knihovny gson [14] namapuje na odpovídající třídu z balíčku *mapper*.

Celé volání tedy funguje tak, že se vytvoří třída APICall, která v konstruktoru dostane které backendové API má zavolat, metodu volání (GET, POST, ..) a třídu objektu, na který se má namapovat odpověď. Navíc se může funkcí setCallback, nastavit třída, která se má zavolat po získání odpovědi. Dál se může nastavit funkcí setFlag, nějaký číselný ukazatel, který se zpropaguje až do callback volání po získání odpovědi. U volání, kde se přenáší větší objemy dat a trvají delší dobu, je zde ještě funkce setProgressHandler, která během volání posílá průběžný stav.

Volání se spustí zavoláním funkce execute na třídě APICall, do které se můžou poslat JSONObject, který má být poslán na backend.

Třídy na které se mapují odpovědi implementují interface Mapper s jedinou funkcí saveData, kde je jako argument callback třída, která se má zavolat po uložení informací. Samostatná implementace by pak měla vytvořit nové vlákno a v něm uložit získaná data do lokální databáze nebo preferencí a posléze zavolat callback třídu.

Lokální databáze

Aplikace si již získaná data z backendu drží v lokální databázi. Databáze je SQLite, protože je do platformy Android implementována [15].

Databázi obsluhuje třída DatabaseHandler. Je implementovaná jako singleton. Při prvním vytvoření inicializuje tabulky a poté pomocí funkce getDatabase dostaneme v aplikaci databázi, nad kterou se můžeme dotazovat.

Databáze obsahuje tabulky races, score, track_points, které odpovídají tabulkám na backendu. Pak obsahuje tabulku users:

Název sloupce	**Typ**	**Popis**
id	INTEGER	Unikátní id
nickname	TEXT	Jméno uživatele
avatar	TEXT	Uživatelův avatar

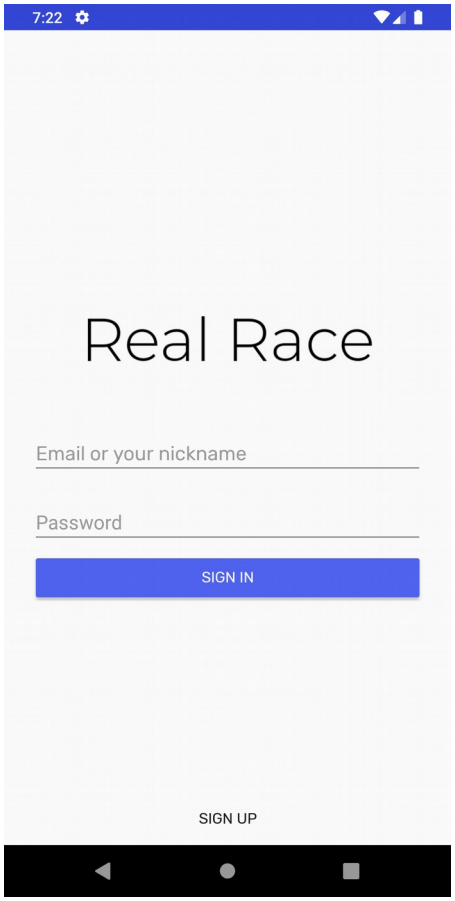
Přihlášení a registrace

Při prvním zapnutí aplikace se uživatel musí přihlásit, nebo si vytvořit účet. O správu přihlášení se stará třída AccountManager. Je implementovaná jako singleton. Při spuštění aplikace se třída inicializuje a zavolá se na ní metoda login. Ta vezme token, který je uložen v preference [22] a zavolá backendové API (/check_login), aby se ověřila platnost tokenu. Pokud je token platný, třída si drží uživatelovo id a lze ho z

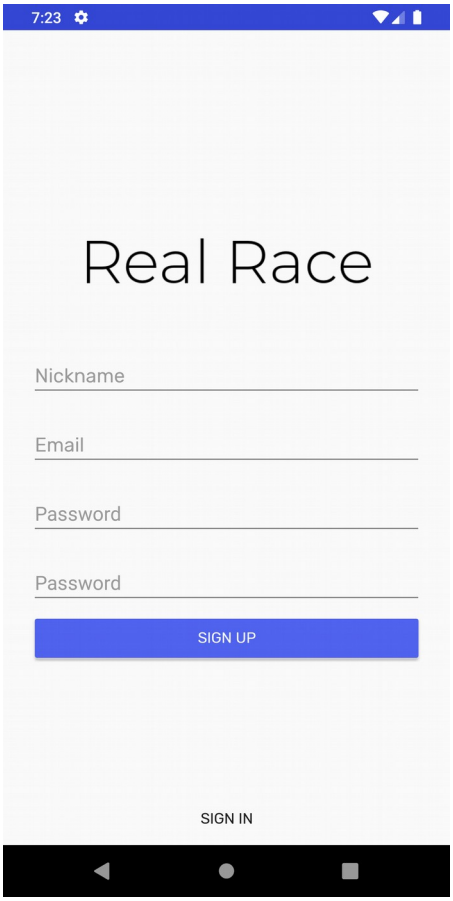
ní získat metodou getUserId. Pokud token není platný, přesměruje uživatele na přihlášení.

Když uživatel vyplní přihlašovací údaje a stiskne přihlašovací tlačítko, ověří se vyplněné údaje a zašle se dotaz na backend (/login). Obdobně funguje obrazovka s registrací. Když uživatel vyplní registrační údaje a stiskne tlačítko registrovat, zkontrolují se údaje a zavolá se backend (/register). Pokud je registrace úspěšná, uživateli se vytvoří účet a je rovnou přihlášen.

Před odesláním dochází ke kontrole platnosti údajů. Zkontroluje se, že uživatel zadal platnou emailovou adresu, že nickname obsahuje pouze povolené znaky (a-Z, 0-9) a je alespoň 4 znaky dlouhý, že heslo má alespoň 6 znaků a že se zadaná hesla shodují.



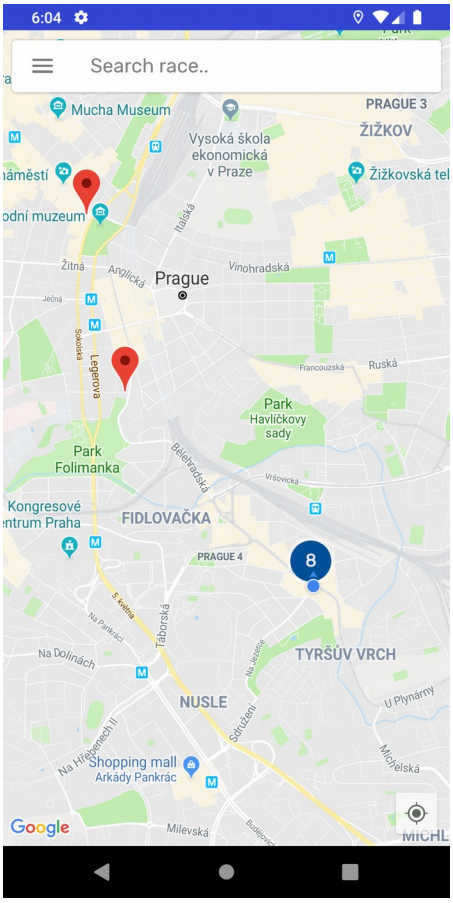
4. Přihlášení



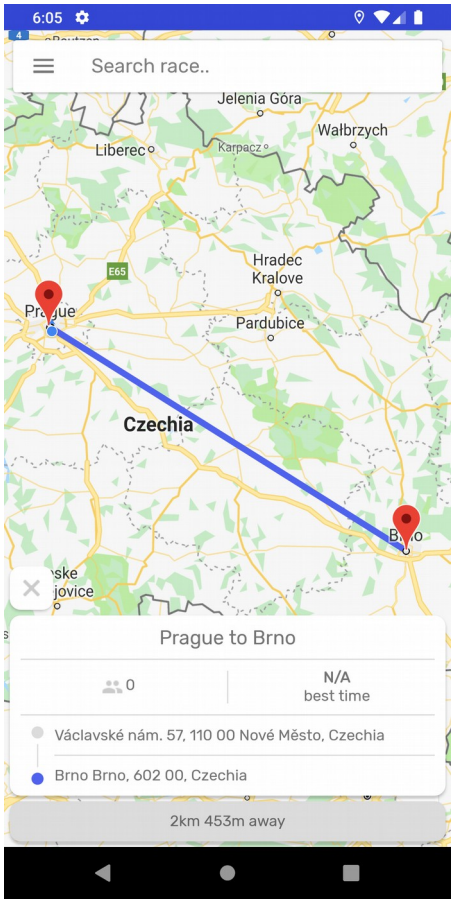
5. Registrace

Mapa

Po přihlášení se uživateli zobrazí mapa se závody v okolí. Jak již bylo zmíněno výše, používají se Google maps. O získávání a zobrazování závodů se stará třída RacesHandler. Třída má nastavený listener, pokud se změní poloha na mapě a posílá dotaz na backend pro závody v právě zobrazené lokaci (/races). Aby byla mapa přehlednější, zobrazuje závody pomocí ClusterManageru [23], který závody, které jsou blízko u sebe, shlukuje do clusterů.



6. Mapa se závody



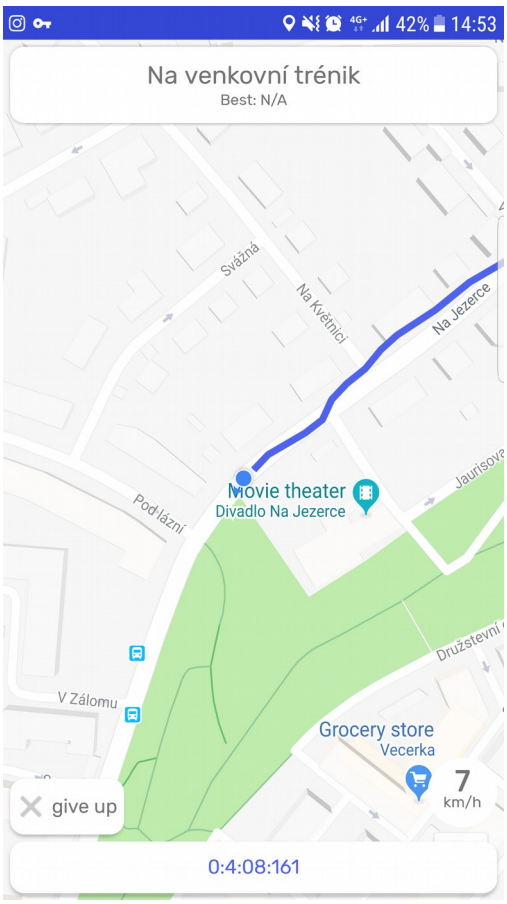
7. Náhled závodu

Závod

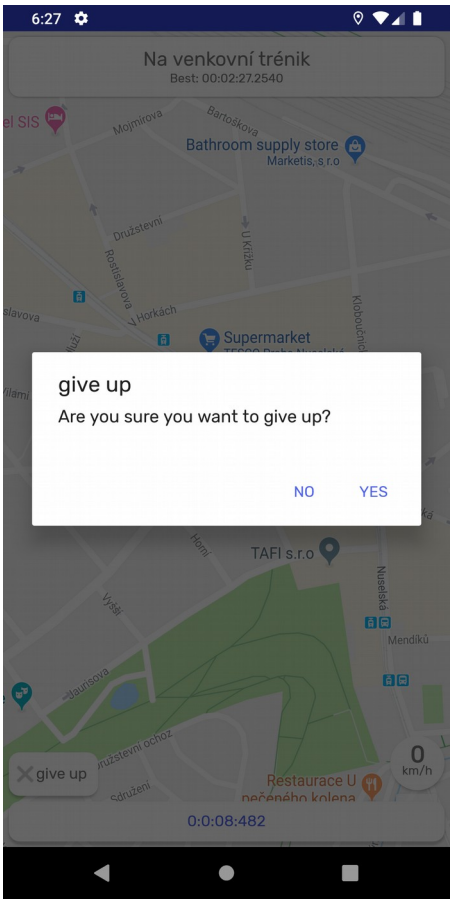
Na získávání aktuální polohy uživatele je zde třída GPSTracker, která implementuje onLocationChangedListener. Pokud je rozkliknutý detail závodu, kontroluje vzdálenost od startu. Pokud je dostatečně blízká, dovolí uživateli závod začít. Když uživatel zahájí závod, začne třída GPSTracker vykreslovat uraženou cesta do mapy. Zároveň, ze vzdálenosti kterou uživatel právě urazil, počítá každou vteřinu rychlost v daném okamžiku. Vezme poslední 3 naměřené vzdálenosti a z nich vypočítá průměr.

Tím se snaží aspoň částečně zamezit zobrazování nesmyslných rychlostí, pokud zrovna na chvíli vypadne GPS signál a uražená vzdálenost je pak skokově větší.

Čas se měří pomocí třídy Timer, kterou dostane třída GPSTracker pomocí metody setTimer. Zároveň si třída GPSTracker během závodu zaznamenává body, kudy uživatel závodil a čas, ve kterém tam byl. Při každé změně lokace kontroluje vzdálenost od cíle a pokud je uživatel dostatečně blízko, závod ukončí. Pošle se volání na backend (/race/<id závodu>), kde se výsledek uloží. Uložený výsledek se vrátí z backendového volání a uloží se i do lokální databáze.



8. Průběh závodu



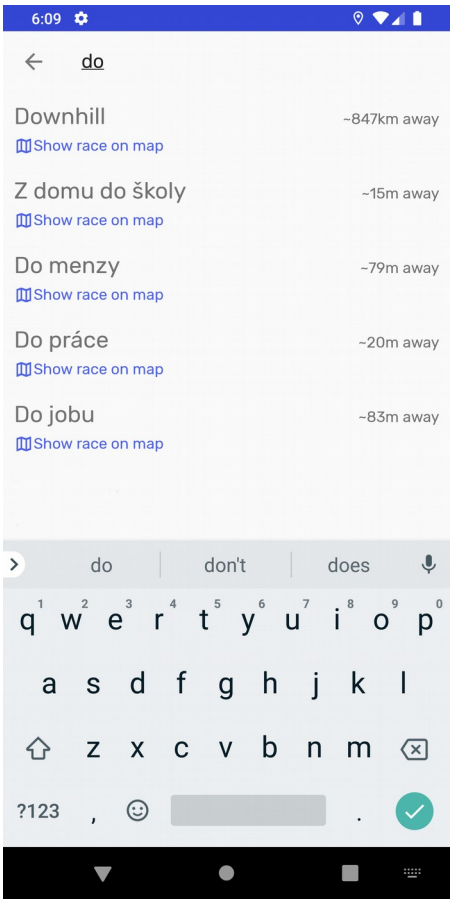
9. Předčasné ukončení závodu

Vyhledávání závodu

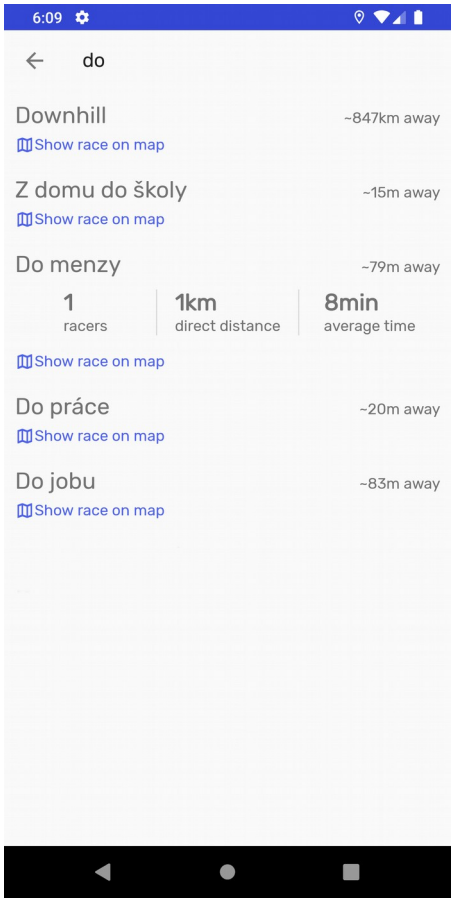
Na poli „Search race..“ je nastavený *onFocusChangeListener*, který pokud je aktivován otevře novou activitu s vyhledáváním závodů. Když se activita otevře, nastaví se focus do vyhledávacího pole, aby mohl uživatel hned psát a přechod byl

plynulý. Na poli je change listener, který čeká až uživatel něco napíše. Ve chvíli, kdy něco napíše, zavolá se metoda *loadSearch*, která nejprve zobrazí odpovídající závody z lokální databáze a poté zavolá backend (/search/race) pro další závody. Backend se volá pouze v případě, když nové volání není součastí předchozího volání. Když se vrátí výsledek z backendu, uloží se nově získané závody do lokální databáze a aktualizují se výsledky vyhledávání.

Závody se zobrazují v seznamu pomocí RecyclerView [24]. Adapter dostane list závodů, který zobrazí. K zobrazení aktuální vzdálenosti využívá třídu GPSTracker, ze které pomocí funkce getLocation, dostane uživatelovu aktuální polohu. Po kliknutí na tlačítko „Show race on map“ otevře activitu s mapu, které pošle ID závodu, který má zobrazit.



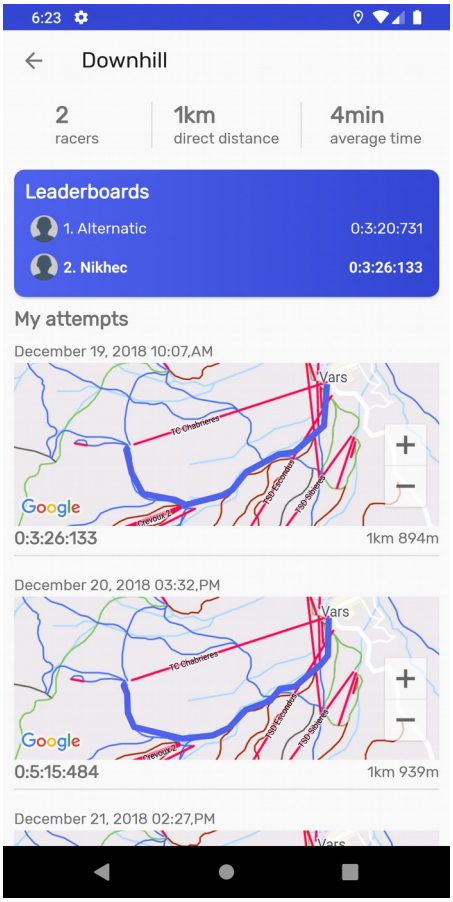
10. Vyhledávání závodu



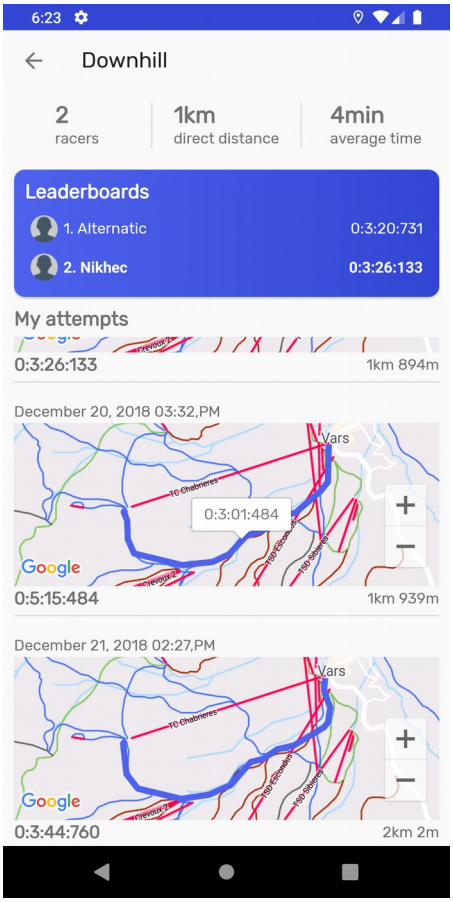
11. Detail vyhledaného závodu

Detail Závod

Obrazovka s detailem závodu zobrazí data relevantní k danému závodu. Nejprve se zobrazí informace, které jsou již uložené v lokální databázi a pak se zároveň pošlou požadavky na backend na aktualizaci informací (/race, /attempts). Uživatelovi pokusy jsou zobrazeny pomocí *RecyclerView*, kde pro každý pokus *adapter* zobrazí mapu, do které pomocí bodů v průběhu závodu vykreslí trasu, kterou uživatel závod absolvoval. Zároveň do každého bodu přidá „neviditelný“ marker, který po kliknutí zobrazí čas v daném bodě.



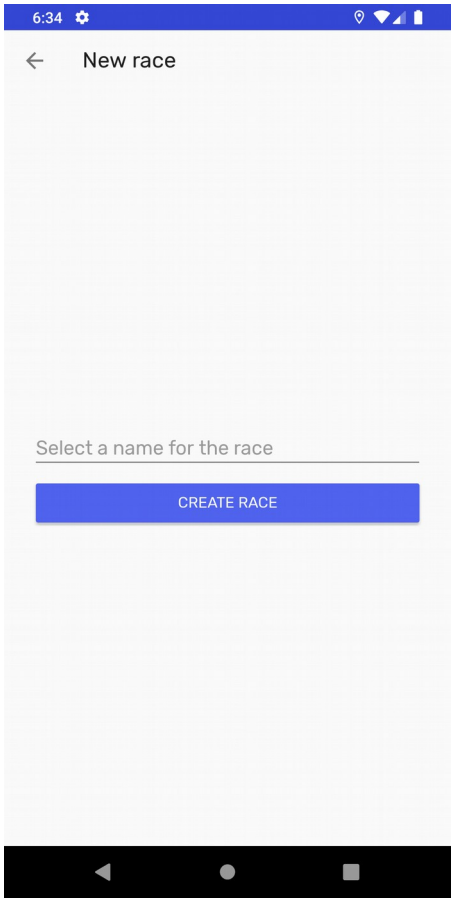
12. Detail závodu



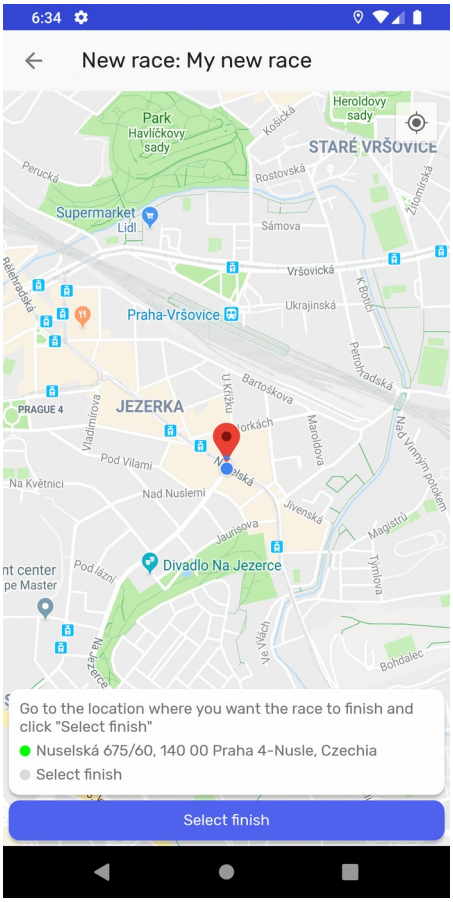
13. Detail bodu v mapě

Nový závod

Při vytváření závodu, se data která uživatel zadal ukládají nejdříve pouze lokálně, dokud uživatel vytváření závodu nedokončí. Při výběru startu a cíle se z třídy GPSTracker pomocí funkce getLocation dostane aktuální poloha, která se uloží. Přepíší se instrukce aby odpovídaly danému kroku a pomocí třídy Geocoder [25] se získá adresa pro dané místo. Když uživatel vyplní všechny údaje, odešlou se na backend (/race), v odpovědi získaný nový závod se uloží do lokální databáze a otevře se activita s mapou, které se pošle ID závodu, který se má zobrazit.



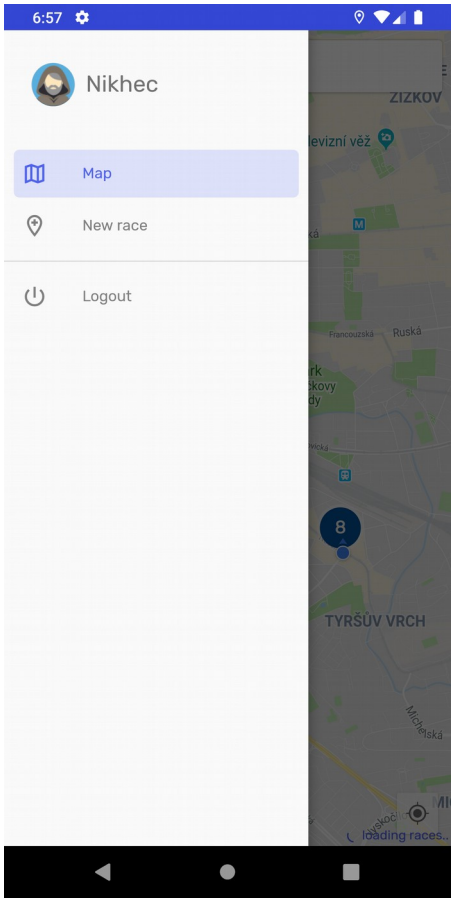
14. Nový závod: krok 1



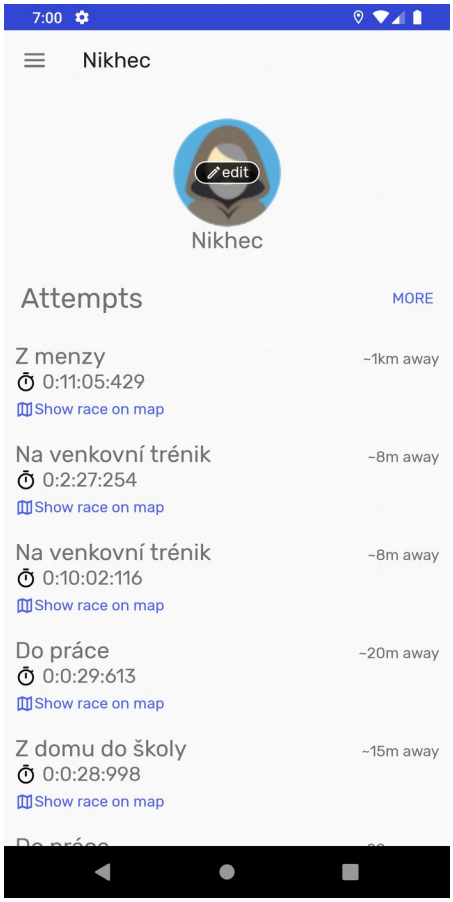
15. Nový závod: krok 2

Profil

Na obrazovce profilu se nejprve zkontroluje, že uživatel s daným id existuje v lokální databázi, pak se pro něj zobrazí data z lokální databáze a pošle se dotaz na backend (/ userextended). Když se dotaz vrátí, získaná data se uloží do lokální databáze a aktualizují se zobrazená data. Avatar se zobrazuje pomocí třídy *AvatarManager*, která dostává ID uživatele a *view* do kterého má avatar vykreslit.



16. Menu aplikace

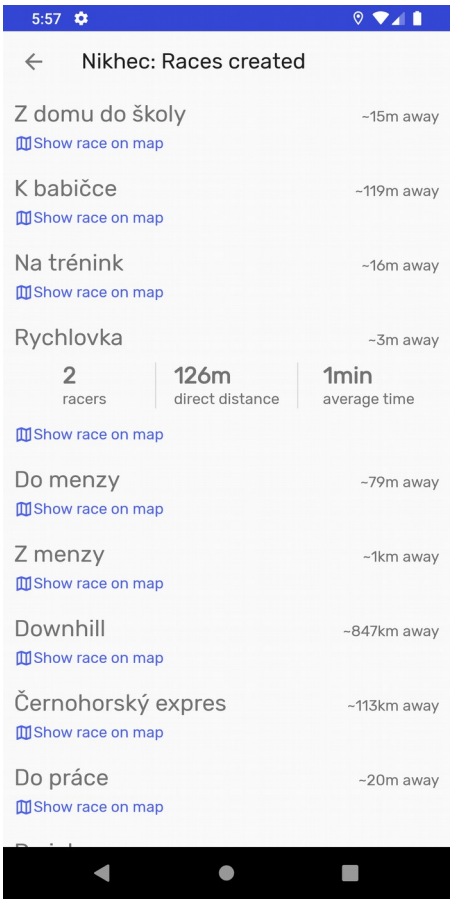


17. Uživatelský profil

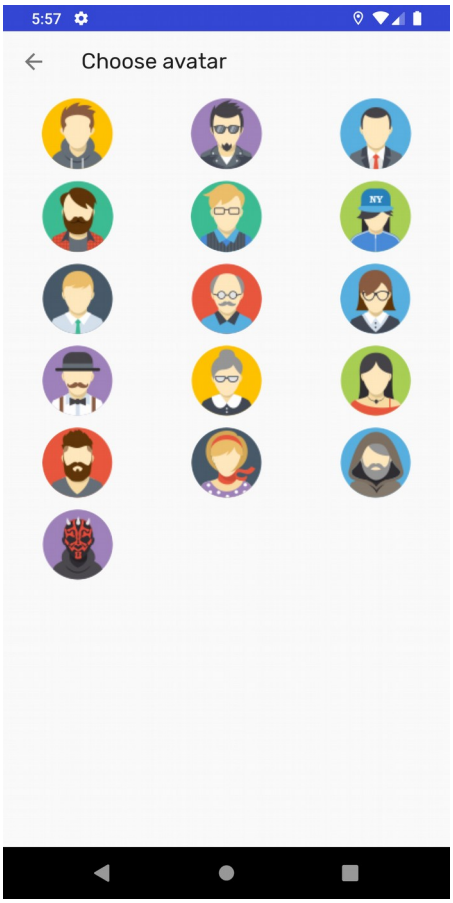
Když se vrátí dotaz z backendu, vedle nadpisů seznamů se zobrazí tlačítko „MORE“, kterým se otevře nová activita s kompletním seznamem pokusů respektive vytvořených závodů. Tlačítko se zobrazuje až po vrácení dotazu z backendu, protože, při prvním zobrazení cizího profilu by nebylo co zobrazovat.

Na stránce profilu se zároveň zkontroluje, jestli je uživatel na vlastním profilu, pokud ano, zobrazí se v avataru tlačíto „edit“, kterým si může avatar změnit. Změna opět

otevře novou obrazovku, kde se zobrazí seznam avatarů uložených v *resources* v poli *avatars*. [26].



18. Kompletní seznam



19. Výběr nového avataru

5. Závěr

5.1. Srování výsledku se zadáním

Hlavní cíle práce se podařilo splnit. Použité technologie se ukázaly jako dostatečné. Návrh aplikace se osvědčil a aplikace běží poměrně plynule i na slabších zařízeních. Uživatelé si v aplikaci vytvoří účet a poté můžou v reálném světě závodit. Na mapě mají zobrazené již vytvořené závody a můžou vytvářet i závody vlastní. U každého závodu je žebříček nejlepších časů a jejich autorů.

5.2. Možná rozšíření

Prvním rozšířením pro získání nových uživatelů se nabízí rozšíření na další platformy, hlavně iOS. Backendová část by zůstala stejná, akorát aplikace by se musela naprogramovat víceméně znovu. Nativně se aplikace na iOS programují v jazyce Swift, což je v podstatě nadstavba Objective-C [27]. Google nabízí Google maps API i pro iOS, aplikace by tedy mohla fungovat i vypadat velmi podobně.

Další možností rozšíření by mohlo být přidání další funkcionality aplikace. Aby se uživatelé cítili více angažování, bylo by zajímavé zavést větší propojení mezi uživateli. Mohli by se označit za „přátele“ a mohli by se vzájemně sledovat a závodit proti sobě. Zajímavým vylepšením by mohlo být sdílení aktuální polohy polohy, tak, že by se uživatelé vzájemně viděli online v mapě a mohli třeba závodit v reálném čase.

Větší kompetativnost by mohl přinést nějaký obecnější žebříček. Nyní jsou závody a žebříčky u nich na sobě nezávislé. Kdyby se za umístění v jednotlivých závodech získavaly body do celkového žebříčku, mohlo by to hře přinést nový rozměr.

Zpestření závodů by se dalo dosáhnout přidáním například checkpointů, kterými musí uživatel během závodu projít. Do checkpointů by se daly přidat třeba i další rozšíření, jako že by uživatel na daném místě musel vyřešit nějakou úlohu typu „kolik je na náměstí soch?“ a bez správné odpovědi by nemohl pokračovat dál. To by hru rozšířilo o novou dimenzi, kdy by nešlo jenom o rychlost pohybu, ale i o rychlost řešení daných úloh.

Dalo by se zaměřit i na uživatelskou přívětivost, zjednodušit uživatelům přihlášení třeba pomocí Googlu nebo Facebooku, zasílat uživatelům notifikace o nových závodech v jejich okolí a tak podobně.

6. Seznam použité literatury

[1] *Wikipedie: Otevřená encyklopedie: Android (operační systém)* [online]. c2019 [citováno 16. 02. 2019]. Dostupný z WWW: https://cs.wikipedia.org/w/index.php?title=Android_(opera%C4%8Dn%C3%AD_syst%C3%A9m)&oldid=17233811

[2] *Firebase: Firebase realtime database documentation* [online]. c2019 [citováno 05. 03. 2019]. Dostupný z WWW: https://firebase.google.com/docs/database/

[3] *Wikipedia: Otevřená encyklopedie: Representation state tranfer* [online]. c2019 [citováno 05. 03. 2019]. Dostupný z WWW: https://en.wikipedia.org/wiki/Representational_state_transfer

[4] *Wikipedia: Otevřená encyklopedie: SOAP* [online]. c2019 [citováno 05. 03. 2019]. Dostupný z WWW: https://en.wikipedia.org/wiki/SOAP

[5] *Pivotal: Spring documentation* [online]. c2019 [citováno 05. 03. 2019]. Dostupný z WWW: https://spring.io/

[6] *Oracle: Database from Oracle* [online]. c2019 [citováno 05. 03. 2019]. Dostupný z WWW: https://www.oracle.com/database/

[7] *Postrgre SQL: Postrgre SQL documentation* [online]. c2019 [citováno 05. 03. 2019]. Dostupný z WWW: https://www.postgresql.org/

[8] *Ionic: Ionic – Cross-Platform Mobile App Development* [online]. c2019 [citováno 18. 03. 2019]. Dostupný z WWW: https://ionicframework.com/

[9] *Facebook: React Native · A framework for building native apps using React* [online]. c2019 [citováno 18. 03. 2019]. Dostupný z WWW: https://facebook.github.io/react-native/

[10] *Google: Flutter - Beautiful native apps in record time* [online]. c2019 [citováno 18. 03. 2019]. Dostupný z WWW: https://flutter.dev/

[11] *Jet Brains: Kotlin Programming Language* [online]. c2019 [citováno 18. 03. 2019]. Dostupný z WWW: https://kotlinlang.org/

[12] *Google: Maps SDK for Android* [online]. c2019 [citováno 18. 03. 2019]. Dostupný z WWW: https://developers.google.com/maps/documentation/android-sdk/intro

[13] *OpenStreetMap: OpenStreetMap* [online]. c2019 [citováno 18. 03. 2019]. Dostupný z WWW: https://www.openstreetmap.org/about

[14] *Google: gson: A Java serialization/deserialization library to convert Java Objects into JSON and back* [online]. c2019 [citováno 15. 04. 2019]. Dostupný z WWW: https://github.com/google/gson

[15] *Google: android.database | Android Developers* [online]. c2019 [citováno 15. 04. 2019]. Dostupný z WWW: https://developer.android.com/reference/android/database/package-summary

[16] *Google: Material design* [online]. c2019 [citováno 21. 04. 2019]. Dostupný z WWW: https://material.io/

[17] *Dodenhof Henning: A circular ImageView for Android* [online]. c2019 [citováno 21. 04. 2019]. Dostupný z WWW: https://github.com/hdodenhof/CircleImageView

[18] *The Pokemon Company International: Pokémon Go* [online]. c2019 [citováno 02. 05. 2019]. Dostupný z WWW: https://www.pokemongo.com/en-us/

[19] *Rovio: Angry Birds AR: Isle of Pigs* [online]. c2019 [citováno 02. 05. 2019]. Dostupný z WWW: https://www.angrybirds.com/angry-birds-ar-isle-pigs/

[20] Wikipedie: Otevřená encyklopedie: Normalizace databáze [online]. c2019 [citováno 15. 04. 2019]. Dostupný z WWW: https://cs.wikipedia.org/w/index.php?title=Normalizace_datab%C3%A1ze&oldid=17225228

[21] Wikipedie: Otevřená encyklopedie: Front controller [online]. c2019 [citováno 15. 04. 2019]. Dostupný z WWW: https://en.wikipedia.org/wiki/Front_controller

[22] *Google: Preference | Android Developers* [online]. c2019 [citováno 15. 04. 2019]. Dostupný z WWW: https://developer.android.com/reference/android/preference/Preference

[23] *Google: ClusterManager documentation* [online]. c2019 [citováno 15. 04. 2019]. Dostupný z WWW: http://googlemaps.github.io/android-maps-utils/javadoc/com/google/maps/android/clustering/ClusterManager.html

[24] *Google: RecyclerView | Android Developers* [online]. c2019 [citováno 21. 04. 2019]. Dostupný z WWW: https://developer.android.com/reference/androidx/recyclerview/widget/RecyclerView.html

[25] *Google: Geocoder | Android Developers* [online]. c2019 [citováno 21. 04. 2019]. Dostupný z WWW: https://developer.android.com/reference/android/location/Geocoder

[26] *Bondarchuk Dmitriy: Free avatars flat icons* [online]. c2019 [citováno 21. 04. 2019]. Dostupný z WWW: https://www.behance.net/gallery/47035405/Free-avatars-flat-icons

[27] *Apple: Start Developing iOS Apps* [online]. c2019 [citováno 02. 05. 2019]. Dostupný z WWW: https://developer.apple.com/library/archive/referencelibrary/GettingStarted/DevelopiOSAppsSwift/

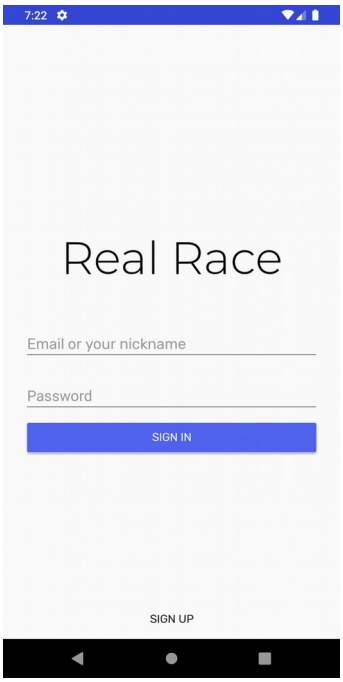
[28] *Cheshin Ari: FloatingSearchView* [online]. c2019 [citováno 14. 05. 2019]. Dostupný z WWW: https://github.com/arimorty/floatingsearchview

7. Přílohy

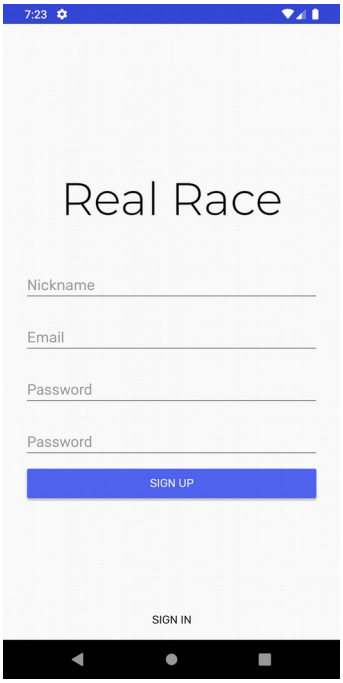
7.1. Uživatelská dokumentace

Start aplikace a přihlášení

Při prvním spuštění aplikace uživatele přivítá přihlašovací obrazovka. Pro používání aplikace je potřeba se přihlásit nebo si vytvořit nový účet. Pro vytvoření účtu uživatel vyplní unikátní přezdívku (dlouhou alespoň 4 znaky), unikátní email, kterého je vlastníkem a heslo (obsahující alespoň 6 znaků). Po registraci, je stejně jako po přihlášení, uživatel automaticky přihlášen a může začít používat aplikaci. Při dalším spuštění už je uživatel přihlášen automaticky.



Přihlášení



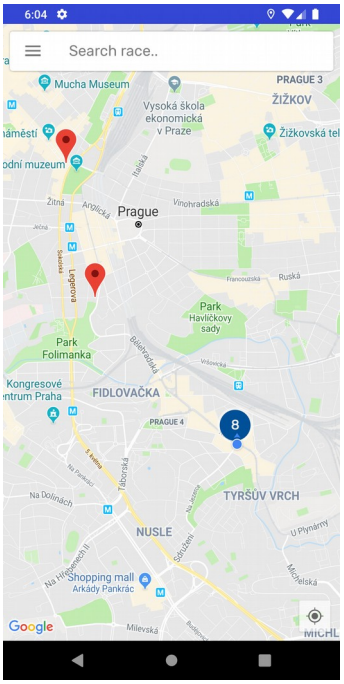
Registrace

Mapa a závody

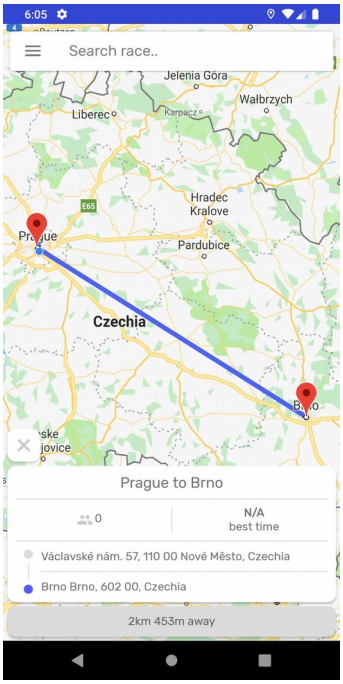
Po přihlášení je uživatel přenesen na hlavní obrazovku aplikace – mapu. Na mapě jsou vidět dostupné závody, pro právě zobrazenou oblast. Pokud uživatel posune mapu na jinou oblast, donačtou se závody i pro ní. Pokud je v oblasti více závodů u sebe, shlukují se do clusterů. Po přiblížení se zobrazí závody samostatně.

Na jednotlivé závody je možné kliknout a zobrazí se o nich základní údaje. Na mapě se ukáže start a cíl a mezi nimi trať vzdušnou čarou. V dolní části obrazovky se pak zobrazí název závodu, počet uživatelů, kteří závod dokončili, nejlepší dosažený čas a adresa startu a cíle. Pod informacemi je tlačítko na započetí

závodu. To je ovšem aktivní pouze, pokud je uživatel dostatečně blízko startu. Pokud ne, tak se v něm ukazuje aktuální vzdálenost od startu. Kliknutím na informace se otevře podrobnější stránka závodu. Náhled závodu se dá zavřít křížkem vlevo nad informacemi nebo systémovým tlačítkem zpět.

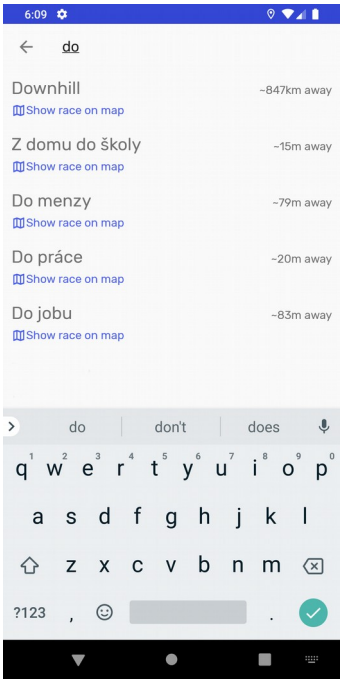


Mapa se závody

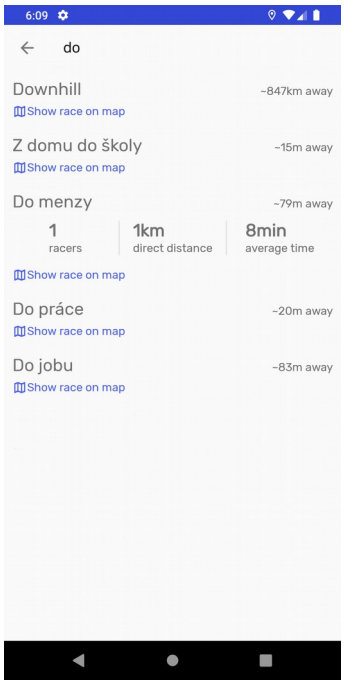


Náhled závodu

Závody je možné vyhledávat kliknutím do pole „Search race..“ v horní části obrazovky. Vyhledává se podle jména. Jak uživatel začne psát hledaný výraz, zobrazí se odpovídající závody, seřazené od největšího po nejmenší počet závodníků, kteří závod dokončili. Pro každý závod se zobrazí jeho jméno a aktuální vzdálenost od začátku. Po kliknutí na jméno závodu se zobrazí ještě počet závodníků, kteří závod dokončili, vzdálenost od startu do cíle vzdušnou čarou a průměrný čas všech pokusů. Pod informacemi je tlačítko „Show race on map“, na které když uživatel klikne, tak ho přenese na mapu a zobrazí náhled daného závodu.



Vyhledávání závodu

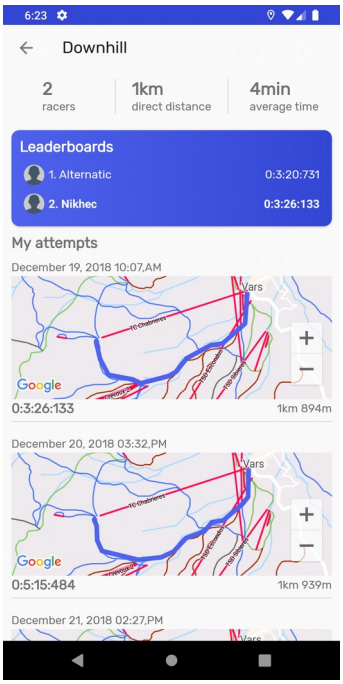


Detail vyhledávaného závodu

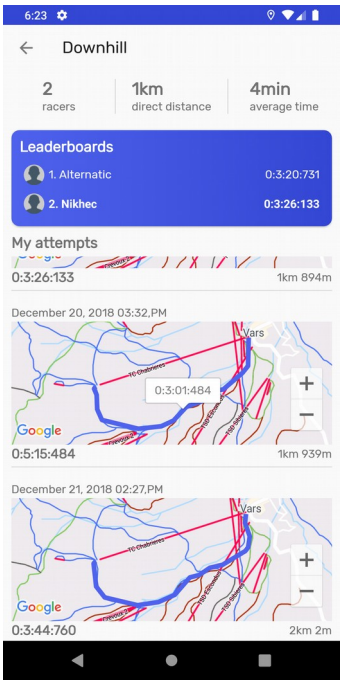
Detail závodu

Na obrazovce detailu závodu je v horní části jméno závodu, pod ním počet závodníků, kteří závod dokončili, vzdálenost ze startu do cíle vzdušnou čarou a průměrný čas všech dosavadních pokusů. V modrém rámečku je žebříček s nejlepšími časy pro daný závod. V něm je zobrazeno deset nejlepších závodníků s jejich časy. Pokud není přihlášený uživatel v nejlepší desítce, a závod dokončil, je zde zobrazen také, na konci seznamu se svým pořadovým číslem. Zároveň je přihlášený uživatel v žebříčku zvýrazněn tučně.

Pod žebříčkem je seznam uživatelových pokusů, seřazených od nejnovějšího po nejstarší. Pro každý pokus je zobrazeno datum, kdy uživatel pokus absolvoval, pod ním je mapa s vykreslenou trasou, kudy uživatel závodil. Pod mapou je dosažený čas a celková vzdálenost, kterou urazil. V mapě se dá kliknout na bod v trase a ukáže se čas, jaký uživatel měl v daném bodě. Pokud uživatel závod ještě neabsolvoval, bude seznam prázdný.



Detail závodu



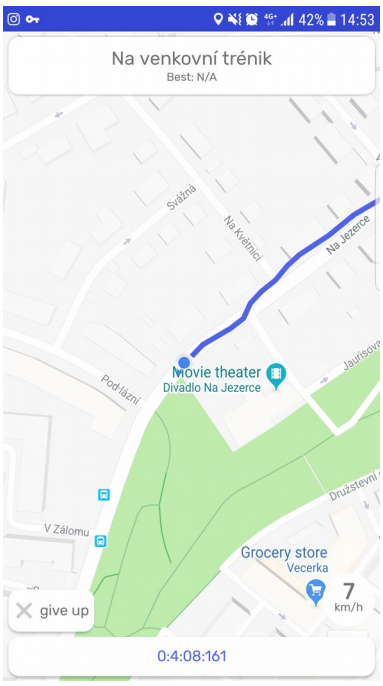
Detail bodu v závodě

Průběh závodu

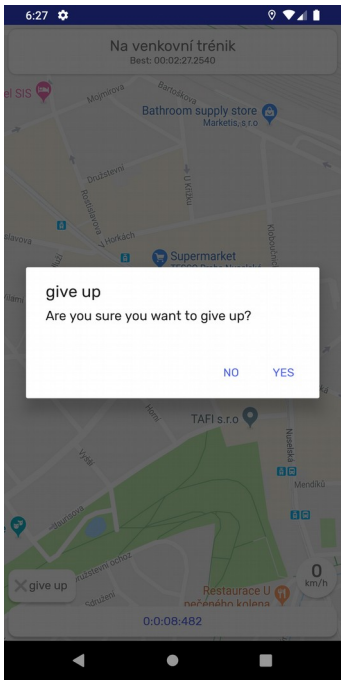
Když uživatel na mapě vybere závod, kterého by se chtěl zůčastnit, kliknutím zobrazí jeho náhled. Musí se dostat dostatečně blízko startu a pod detailem se zobrazí tlačítkou „Begin race“. Zmáčknutím tlačítka závod začíná.

V horní části obrazovky je pak vidět název závodu a dosavadní nejlepší čas. Na mapě se zobrazuje cíl závodu, aktuální poloha uživatele a postupně se vykresluje trasa, kterou uživatel již urazil. V dolní části je časomíra, aktuální rychlost a tlačítko „give up“, kterým se závod ukončí.

Ve chvíli, kdy se uživatel dostatečně přiblíží k cíli, je zastavena časomíra a zaznamená se výsledek. O úspěšném zaznamenání výsledku informuje vyskakovací hláška. Dokud uživatel neodklikne dokončení závodu, je na mapě vidět trasa, kterou uživatel za závod urazil.



Průběh závodu



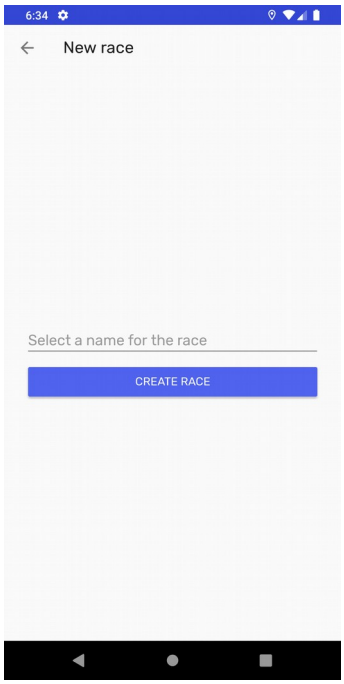
Předčasné ukončení závodu

Nový závod

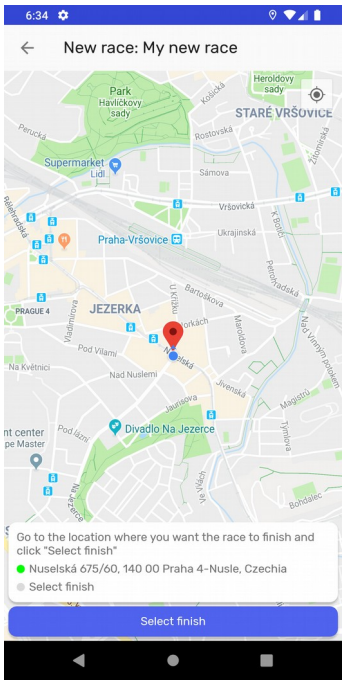
Uživatelé mají možnost vytvořit vlastní závod. Můžou tak učinit, když v levém menu kliknou na položku „New race“. Na úvodní obrazovce vyplní jméno pro nový závod a kliknou na tlačítko „Create race“. Po celou dobu vytváření závodu je pak jméno vidět v horní části obrazovky.

Na další obrazovce je vidět mapa, na které je vidět uživatelova aktuální poloha. V dolní části jsou potom zobrazeny instrukce k vytvoření závodu. Nejprve uživatel dorazí na místo, kde chce, aby byl start závodu a stiskne tlačítko „Select start“. Na mapě se pak zobrazí marker, kde bude závod začínat. Zároveň se v instrukcích zobrazí adresa startu (pokud je k dispozici). Poté se uživatel přemístí na místo, kde chce aby byl cíl závodu a stiskne „Select finish“. Opět se zobrazí marker, kde bude závod končit a v instukcích se zobrazí adresa cíle. Mezi začátkem a koncem se zobrazí trasa vzdušnou čarou. Začátek a cíl mezi sebou musí být vzdálené alespoň 50 metrů.

Když uživatel vyplnit všechny tyto údaje, zobrazí se tlačítko „Create race“, po jehož zmáčknutí se vytvoří nový závod a uživatel je přenesen zpět na mapu, kde se zobrazí detail nově vytvořeného závodu.



Nový závod: krok 1



Nový závod: krok 2

Uživatelský profil

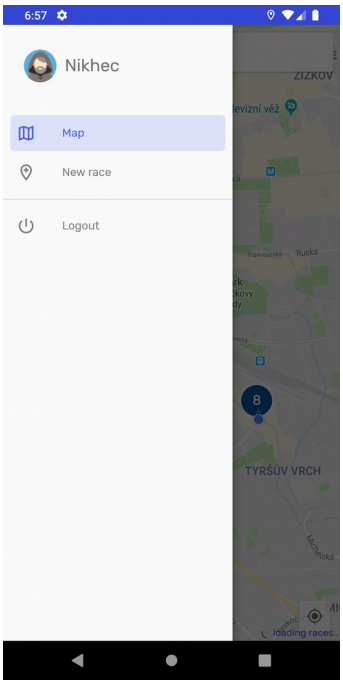
Každý uživatel má svůj uživatelský profil. Na vlastní profil se lze dostat z postranního menu kliknutím na svůj avatar nebo přezdívku. Na cizí profil se lze dostat z detailu závodu po kliknutí na uživatelův avatar nebo přezdívku v žebříčku nejlepších závodníků.

Na profilu je seznam nejčerstvějších pokusů, které daný uživatel absolvoval a seznam nejnovějších závodů, které uživatel vytvořil. Pro každý ze seznamů se zobrazí nejnovějších 10 položek. Po načtení profilu se dá kliknout vedle nadpisu jednotlivých seznamů na tlačítko „MORE“, které zobrazí všechny položky.

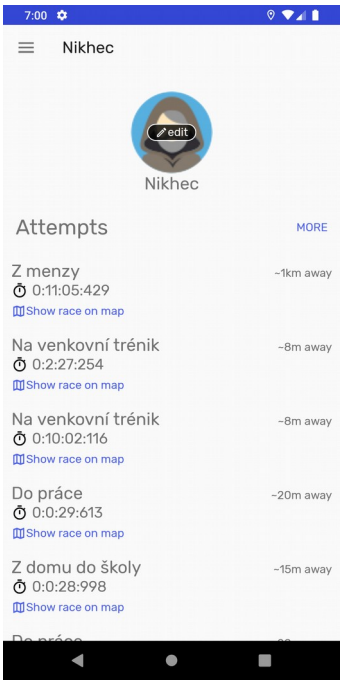
Seznam pokusů zobrazuje jméno závodu, ke kterému pokus patří, aktuální vzdálenost od daného závodu, čas, kterého uživatel dosáhl a tlačítko „Show race on map“, které zobrazí závod na mapě.

Seznam vytvořených závodů zobrazuje jméno závodu, aktuální vzdálenost a tlačítko „Show race on map“, které zobrazí závod v mapě. Po kliknutí na název závodu se zobrazí další informace: počet uživatelů, kteří závod dokončili, délka závodu vzdušnou čarou a průměrný čas všech pokusů.

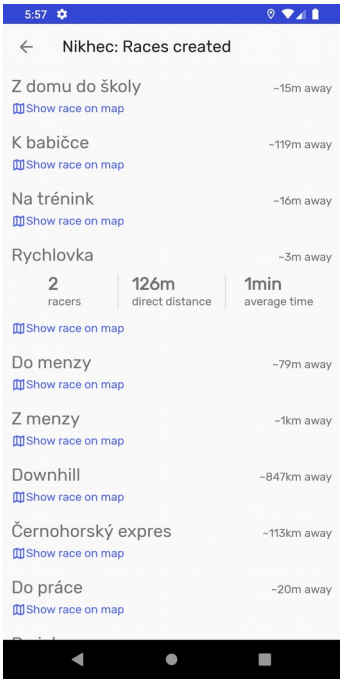
Pokud je uživatel na vlastním profilu, může si kliknutím na avatar vybrat nový obrázek, který chce mít jako nový avatar.



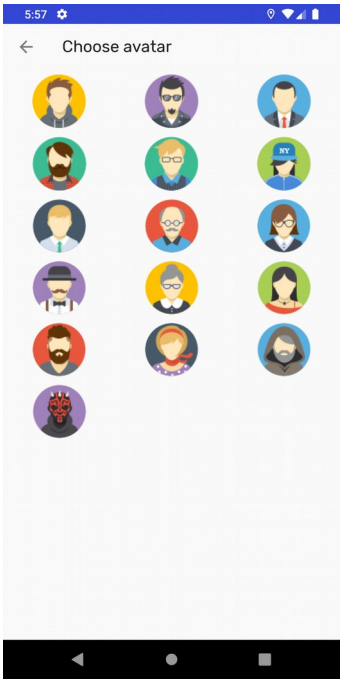
Menu aplikace



Uživatelský profil



Kompletní seznam



Výběr nového avataru

7.2. Struktura databáze

(pk_ - primární klíč, fk_ - cizí klíč)

veškeré hodnoty, které souvisí s datem jsou ukládány ve formátu Unix timestamp - počtu sekund od 1.1.1970 00:00:00

Races

seznam závodů

Název sloupce	**Typ**	**Popis**
pk_id	int	unikátní id závodu
fk_user	int	id uživatele, který závod vytořil
name	varchar	jméno závodu
start_lat	double	GPS zeměpisná šířka začátku závodu
start_lng	double	GPS zeměpisná délka začátku závodu
start_address	varchar	adresa začátku závodu
end_lat	double	GPS zeměpisná šířka konce závodu
end_lng	double	GPS zeměpisná délka konce závodu
end_address	varchar	adresa konce závodu
best	int	nejlepší dosažený čás v milisekundách
average	int	průměrný čas všech pokusů v milisekundách
attempts	int	celkový počet pokusů
finished	int	počet uživatelů, kteří závod dokončili
date	int	datum vytvoření závodu
last_modified	int	datum poslední modifikace

Score

seznam pokusů

Název sloupce	Typ	Popis
pk_id	int	unikátní id pokusu
fk_user	int	id uživatele, který absolvoval tento pokus
fk_race	int	id závodu, který byl absolvován
time	int	dosažený čas v milisekundách
date	int	datum pokusu

Track_points

seznam bodů v čase k pokusu

Název sloupce	Typ	Popis
pk_id	int	unikátní id bodu v závodu
fk_score	int	id pokusu, ke kterému se bod vztahuje
time	int	čas v závodě, ke kterému se bod vztahuje
lat	double	GPS zeměpisná šířka bodu
lng	double	GPS zeměpisná výška bodu

Users

seznam uživatelů

Název sloupce	Typ	Popis
pk_id	int	unikátní id uživatele
email	varchar	přihlašovací email uživatele
password	varchar	zašifrované heslo uživatele
nickname	varchar	uživatelské jméno
avatar	varchar	cesta k uživatelovu avataru
date	int	datum založení uživatelského účtu

Tokens

seznam přihlašovacích tokenů

Název sloupce	Typ	Popis
pk_id	int	unikátní id tokenu
token	string	zašifrovaný token
fk_user	int	id uživatele, kterému token patří
start	int	datum, od kterého token platí
end	int	datum, do kterého token platí

7.3. Dokumentace API volání

GET /races?location=<lat>,<lng>|<lat>,<lng>&known=id,date|..

Vstupní parametry: zeměpisná šířka a výška levého horního rohu mapy a zeměpisná šířka a výška pravého dolního rohu mapy. Seznam id závodů, které už jsou stažené v aplikaci a datum jejich poslední aktualizace.

Odpověd: seznam závodů, které jsou v dané oblasti mapy. A seznam případných aktualizací pro závody, které už jsou uložené v aplikaci.

```
{
    „success“: true,
    „races“: [
        [„id“: <id závodu>,
        „user“: <id uživatele, který závod vytvořil>,
        „name“: <název závodu>,
        „start_lat“: <zeměpisná šířka začátku závodu>,
        „start_lng“: <zeměpisná délka začátku závodu>,
        „start_address“: <adresa začátku závodu>,
        „end_lat“: <zeměpisná šířka konce závodu>,
        „end_lng“: <zeměpisná délka konce závodu>,
        „end_address“: <adresa konce závodu>,
        „best“: <nejlepší dosažený čas>,
        „average“: <průměrný čas všech pokusů>,
        „finished“: <počet uživatelů, kteří dokončili závod>,
        „date“: <datum vytvoření závodu>],
        ..
    ]
```

}

GET /attempts/<id závodu>?last=<date>

Vstupní parametry: id závodu, ke kterému mají být pokusy. Datum od kterého mají být pokusy.

Odpověď: seznam uživatelových pokusů a k nim odpovídajícím bodům.

```
{
    „success“: true,
    „race“: <id závodu>,
    „user“: <id uživatele>,
    „attempts“: [
        [„score“: <id pokusu>,
        „date“: <datum pokusu>,
        „time“: <dosažený čas>,
        „track“: [
            [„time“: <čas bodu>,
            „lat“: <zeměpisná šířka bodu>,
            „lng“: <zeměpisná výška bodu>]
            ..
            ]
        ]
        ..
    ]
}
```

GET /race/<id závodu>

Vstupní parametry: id závodu, pro ktetý chceme výsledky.

Odpověď: Seznam deseti nejlepších závodníků s jejich časy a čas a umístění daného uživatele.

```
{
    „success“: true,
    „scores“: [
        [ „user“: [
            „id“: <id uživatele>,
```

„nickname“: <přezdívka uživatele>,
„avatar“: <avatar uživatele>,
],
„position“: <umístění uživatele>,
„time“: <uživatelův nejlepší čas pro daný závod>],
..
],
„average“: <průměrný čas>
}

GET /user/<id uživatele>

Vstupní parametry: id uživatele

Odpověď: informace k danému uživateli.

{
„success“: true,
„id“: <id uživatele>,
„nickname“: <jméno uživatele>,
„avatar“: <uživatelův avatar>
}

POST /user

Vstupní parametry: registrační údaje

{
„email“: <email uživatele>,
„password“: <heslo uživatele>,
„nickname“: <přezdívka uživatele>,
}

Odpověď: Vytvoří nový učet s danými parametry.

{
„success“: true,
„token“: <přihlašovací token>,
„id“: <id uživatele>,
„nickname“: <přezdívka uživatele>,
„avatar“: <avatar uživatele>,

„end“: <datum vypršení platnosti tokenu>

}

POST /login

Vstupní parametry: přihlašovací údaje

{

„email“: <email nebo přezdívka uživatele>,

„password“: <heslo uživatele>

}

Odpověď: Ověří zadané údaje a uživatele přihlásí.

{

„success“: true,

„token“: <přihlašovací token>,

„id“: <id uživatele>,

„nickname“: <přezdívka uživatele>,

„avatar“: <avatar uživatele>,

„end“: <datum vypršení platnosti tokenu>

}

POST /check_login

Vstupní parametry: přihlašovací token

{

„token“: <přihlašovací token>

}

Odpověď: Ověří platnost přihlašovacího tokenu.

{

„success“: true,

„login“: <true / false>,

„user“: <id uživatele>

}

POST /race/<id závodu>

Vstupní parametry: id závodu, dosažený čas a body průběhu závodu.

{

„time“: <dosažený čas>,
„track“: [
[„time“: <čas v bodě>
„lat“: <zeměpisná šířka pro daný bod>,
„lng“: <zeměpisná délka pro daný bod>
]
..
]
}

Odpověď: Zaznamená čas a uloží pro něj mezičasy.

{
„success“: true,
„race“: <id závodu>,
„score_id“: <id pokusu>,
„average“: <nový průměrný čas>,
„finished“: <nový počet uživatelů, kteří závod dokončili>,
„best“: <nový nejlepší čas>,
„time“: <uživatelův čas>,
„date“: <aktuální datum>,
„track“: [
[„time“: <čas v bodě>
„lat“: <zeměpisná šířka pro daný bod>,
„lng“: <zeměpisná délka pro daný bod>
]
..
]
}

POST /newrace

Vstupní parametry: Parametry nového závodu.

{
„name“: <jméno závodu>,
„start_lat“: <zeměpisná šířka začátku závodu>,
„start_lng“: <zeměpisná délka začátku závodu>,

„start_address“: <adresa začátku závodu>,

„end_lat“: <zeměpisná šířka konce závodu>,

„end_lng“: <zeměpisná délka konce závodu>,

„end_address“: <adresa konce závodu>

}

Odpověď: Vytvoří nový závod podle daných parametrů.

{

„success“: true,

„id“: <id závodu>,

„user“: <id uživatele, který závod vytvořil>,

„name“: <název závodu>,

„start_lat“: <zeměpisná šířka začátku závodu>,

„start_lng“: <zeměpisná délka začátku závodu>,

„start_address“: <adresa začátku závodu>,

„end_lat“: <zeměpisná šířka konce závodu>,

„end_lng“: <zeměpisná délka konce závodu>,

„end_address“: <adresa konce závodu>,

„best“: 0,

„average“: 0,

„finished“: 0,

„date“: <aktuální datum>

}

PUT /avatar

Vstupní parametry: nový avatar.

{

„avatar“: <avatar>

}

Odpověď: změní uživatelův avatar.

{

„success“: true,

„user“: <id uživatele>,

„avatar“: <nový avatar>

}

7.4. Obsah přiloženého CD

- Bakalarska_prace.pdf
- Zdrojove kody
    | - Frontend
    | - Backend