VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

MOBILNÍ APLIKACE PRO SLEDOVÁNÍ A SDÍLENÍ POLOHY CYKLISTŮ NA MAPĚ

MOBILE APPLICATION FOR TRACKING AND SHARING CYCLISTS LOCATION ON MAP

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. RICHARD SCHLÉGER**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JIŘÍ HYNEK, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2021/2022

Zadání diplomové práce

Student: **Schléger Richard, Bc.**
Program: Informační technologie a umělá inteligence
Specializace: Informační systémy a databáze
Název: **Mobilní aplikace pro sledování a sdílení polohy cyklistů na mapě**
Mobile Application for Tracking and Sharing Cyclists Location on Map
Kategorie: Informační systémy
Zadání:

1. Prozkoumejte oblast rekreační cyklistiky a problematiku sdílení polohy cyklistů.
2. Prostudujte existující technologie (mobilní zařízení, chytré hodinky) a aplikace určené pro sledování a sdílení polohy cyklistů, záznam a vyhodnocování jízd.
3. Seznamte se s principem tvorby multiplatformních aplikací pro mobilní zařízení a chytré hodinky. Prostudujte technologie určené pro tento účel.
4. Analyzujte požadavky cyklistů na sledování jejich polohy, vyhodnocování jízd a sdílení těchto informací s dalšími uživateli (detekce a hlášení pádu, automatické sdílení polohy v reálném čase). Porovnejte tyto požadavky s možnostmi existujících aplikací a vyhodnoťte jejich nedostatky.
5. Navrhněte multiplatformní mobilní aplikaci řešící požadavky z bodu 3.
6. Navržené řešení implementujte.
7. Proveďte uživatelské testování aplikace při reálném používání a vyhodnoťte výsledky. Nastiňte možné rozšíření.

Literatura:

- Johnson, J.: *Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Guidelines*. Morgan Kaufmann Publishers/Elsevier, 2010, ISBN: 9780123750303.
- Google Developers: *Get started with Wear OS* [online]. 2021 [cit. 2021-10-09]. Dostupné z: https://developer.android.com/training/wearables
- Google Developers: *Firebase Documentation* [online]. 2021 [cit. 2021-10-09]. Dostupné z: https://firebase.google.com/docs

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 4.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Hynek Jiří, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 18. května 2022
Datum schválení: 25. října 2021

Abstrakt

Táto práca sa zaoberá analýzou, návrhom a implementáciou mobilnej aplikácie a aplikácie pre inteligentné hodinky slúžiacej na sledovanie a zdieľanie polohy športovcov, špeciálne cyklistov. Hlavným cieľom aplikácie je zvýšiť úroveň bezpečnosti športovcov počas aktivity. Táto úloha je zabezpečená pomocou zdieľania polohy a taktiež pomocou automatickej detekcie pádu a následnej notifikácií. Výsledná aplikácie je určená pre mobilné operačné systémy Android a iOS a rovnako pre inteligentné hodinky so sytémom Wear OS. Teoretická časť práce sa zaoberá analýzou aktuálnych trendov v oblasti cyklistiky, tvorby mobilných aplikácií a aplikácií pre inteligentné hodinky. Realizačnú časť tvorí návrh aplikácie, jej následná implementácia a testovanie.

Abstract

This thesis deals with analysis, design and implementation of mobile application and application for smartwatches for tracking and sharing location of athlets, especially cyclists. Main goal of this application is to increase level of safety of athletes during the activity. This task is accomplished by location sharing and also by automatic fall detection and following notification. Final application is intended for mobile operating system Android and iOS a also for smartwatches with Wear OS. Theoretical part is dealing with analysis of actual trends in cycling, mobile applications and smartwatch applications development. Implementation part consists of design of application, implementation itself and testing.

Kľúčové slová

mobilná aplikácia, Android, iOS, React Native, detekcia pádu, zdieľanie polohy, cyklistika

Keywords

mobile application, Android, iOS, React Native, fall detection, location sharing, cycling

Citácia

SCHLÉGER, Richard. *Mobilní aplikace pro sledovaní a sdílení polohy cyklistů na mapě.* Brno, 2022. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jiří Hynek, Ph.D.

Mobilní aplikace pro sledovaní a sdílení polohy cyklistů na mapě

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením pána Ing. Jiřího Hyneka Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Richard Schléger
11. mája 2022

Poďakovanie

Ďakujem vedúcemu tejto práce Ing. Jiřímu Hynekovi Ph.D. za odbornú pomoc, vedenie a ďalšie cenné rady pri riešení mojej diplomovej práce.

Obsah

1 Úvod **3**

2 Cyklistika **4**
2.1 Druhy cyklistiky 4
2.2 Monitorovanie polohy cyklistov 5
2.3 Využívané zariadenia v oblasti cyklistiky 6
2.4 Existujúce aplikácie 7

3 Mobilné aplikácie **10**
3.1 Natívne mobilné aplikácie 11
3.2 Webové mobilné aplikácie 16
3.3 Hybridné mobilné aplikácie 20
3.4 Komunikácia mobilnej aplikácie so serverom 23

4 Inteligentné hodinky **26**
4.1 Apple WatchOS 27
4.2 Google Wear OS 28
4.3 Komunikácia so serverom 29

5 Analýza **30**
5.1 Cieľová skupina používateľov 30
5.2 Potreby používateľov 30
5.3 Súčasný stav 31
5.4 Požiadavky 32

6 Návrh **33**
6.1 Návrh architektúry 33
6.2 Návrh databázy 34
6.3 Návrh aplikačného rozhrania 37
6.4 Návrh grafického používateľského rozhrania 41

7 Implementácia **44**
7.1 Backend 44
7.2 Mobilná aplikácia 47
7.3 Aplikácia pre Wear OS 53

8 Testovanie **55**
8.1 Testovanie serverovej časti aplikácie 55
8.2 Testovanie mobilnej aplikácie 55

8.3 Výsledky testovania 57

9 Záver **58**

Literatúra **60**

A Snímky obrazovky vytvorenej aplikácie **65**

B Obsah priloženého pamäťového média **71**

Kapitola 1

Úvod

Mobilné aplikácie patria v dnešnej dobe ku každodennej a neodmysliteľnej súčasti dňa pre väčšinu z nás. Uľahčujú alebo spríjemňujú nám život, umožňujú nám komunikáciu s priateľmi, ale taktiež môžu prispievať k našej bezpečnosti.

V nedávnej minulosti prenikli mobilné aplikácie aj do sveta športu a tým pádom aj do cyklistiky. Existuje veľké množstvo aplikácií, ktoré slúžia k zaznamenávaniu, vyhodnoteniu alebo zdieľaniu jazdy medzi svojimi priateľmi. Viaceré z týchto aplikácií umožňujú v určitej miere zdieľanie svojej aktuálnej polohy počas jazdy, čo môže do veľkej miery prispieť k zvýšenej bezpečnosti cyklistov. Avšak vo veľa prípadoch je toto zdieľanie zložité nastaviť, treba ho explicitne zapínať pred každou jazdou a sledujúci má možnosť vidieť väčšinou iba jedného cyklistu súčasne. Taktiež žiadna z najpoužívanejších cyklistických aplikácií nepodporuje pri zdieľaní polohy automatickú detekciu pádu a prípadné upozornenie na túto skutočnosť, čo by v niektorých extrémnych prípadoch mohlo cyklistovi zachrániť život.

Táto práca sa zaoberá vytvorením mobilnej aplikácie pre Android a iOS, ktorá bude riešením vyššie uvedených problémov súčasných aplikácií a bude vo veľkej miere zvyšovať bezpečnosť cyklistov. Vyvíjaná aplikácia nemá za cieľ nahradiť aktuálne využívané mobilné aplikácie v oblasti cyklistiky, ale skôr rozšíriť ich funkčnosť a spolupracovať s nimi. Aplikácia bude slúžiť všetkým cyklistom a ich rodinným príslušníkom alebo priateľom, ktorý chcú zdieľať alebo sledovať aktuálnu polohu počas jazdy. Bude obsahovať aj jednoduchú detekciu pádu, ktorá v prípade pádu tento jav detekuje a do pár sekúnd odošle upozornenie. Taktiež bude spolupracovať s aplikáciou Strava, aktuálne najpoužívanejšou aplikáciou na zaznamenávanie športových aktivít, z ktorej bude možné stiahnuť naplánované trasy a rovnako do nej nahrávať vykonané aktivity. Ďalším cieľom práce je vytvorenie aplikácie pre inteligentné hodinky, ktoré sa v poslednej dobe čoraz častejšie využívajú ako v bežnom živote, tak aj v cyklistike. Aplikácia pre hodinky bude zameraná hlavne na odosielanie aktuálnej polohy, detekciu pádu a upozornenie na zistený pád, keďže sledovanie polohy iných cyklistov kvôli veľkosti displeja hodiniek nedáva veľký zmysel.

V prvej kapitole si priblížime samotnú oblasť cyklistiky, definujeme si základné pojmy a pozrieme sa na najpoužívanejšie aplikácie a zariadenia. V kapitole číslo 3 sa zoznámime s rôznymi prístupmi a technológiami na tvorbu mobilných aplikácií. Kapitola 4 sa zaoberá inteligentnými hodinkami, ich typmi, využívanými operačnými systémami a tvorbou aplikácií pre tieto zariadenia. Kapitola 5 analyzuje súčasný stav, požiadavky a problémy, ktoré je potrebné riešiť. V kapitole 6 je popísaný návrh vyvíjanej aplikácie, jej architektúra, vývojové diagramy a návrh používateľského rozhrania. Následná implementácia navrhnutého riešenia je popísaná v kapitole 7. Posledná kapitola číslo 8 je zameraná na testovanie vytvoreného produktu a zhodnotenie dosiahnutých výsledkov.

Kapitola 2

Cyklistika

Cyklistika patrí medzi najviac vykonávané športy v súčasnosti. Podľa štúdie z roku 2015 [43], 42% všetkých domácností na svete vlastní aspoň jeden bicykel. To znamená, že na svete je viac ako 580 miliónov bicyklov a tým pádom aj cyklistov. Ako môžeme vidieť na výročnej správe za rok 2020 [51] z najpoužívanejšej športovej aplikácie Strava, za minulý rok sa počet zaznamenaných aktivít na bicykli zvýšil 1,8 násobne. To predstavuje za rok 2020 viac ako 500 miliónov zaznamenaných jázd s celkovou dĺžkou vyše 13 miliárd kilometrov.

V rámci tejto práce budeme cyklistikou myslieť iba vonkajšiu cyklistiku, keďže existuje aj viacero druhov cyklistiky, ktorá sa môže vykonávať vo vnútorných priestoroch, ako napríklad dráhová cyklistika, jazda na trenažéroch a iné.

2.1 Druhy cyklistiky

Cyklistiku môžeme deliť podľa viacerých kritérií. Prvé delenie cyklistiky je podľa toho, na čo sa bicykel hlavne využíva. V tomto prípade ide buď o športovú cyklistiku, kedy nie je cieľom niekam doraziť, alebo ide o využívanie bicykla na dochádzanie, kedy sa chceme niekam prepraviť.

Ďalším kritériom je delenie podľa terénu, v akom sa cyklistika vykonáva. Podľa tohto môžeme tento šport rozdeliť na tri základné kategórie, a to cestnú cyklistiku, cyklokros a horskú cyklistiku. Na každú z týchto kategórií sa teraz pozrieme bližšie z hľadiska výslednej aplikácie.

2.1.1 Cestná cyklistika

Cestná cyklistika je najrozšírenejšou formou cyklistiky [22], pri ktorej sa cyklista pohybuje iba po spevnených cestách. Z tohto hľadiska sa pri nej nevyskytujú žiadne prudké zmeny smeru, výšky alebo orientácie. To ju robí ideálnou z hľadiska detekcie pádu, kedy sa nemusíme obávať falošných detekcií spôsobených napríklad skokom na bicykli. Takisto je pri cestnej cyklistike jednoduchšie získať presnú polohu, keďže sa cyklista pohybuje po spevnených cestách, ktoré sú vo väčšine prípadov dobre pokryté signálom GPS.

2.1.2 Horská cyklistika

Horská cyklistika je v mnohom úplným opakom cestnej cyklistiky. Je to odvetvie, v ktorom sa jazdí mimo asfaltových ciest, často v ťažkom teréne. Získavanie presnej polohy pri horskej cyklistike nie je také ideálne ako pri cestnej, keďže cyklista väčšinu trasy prejde po

nespevnených cestách, ktoré sa môžu nachádzať napríklad v lesoch, kde je pokrytie signálom GPS všeobecne slabšie ako na otvorenom priestranstve. Taktiež sa tu môžu vyskytovať prudké zmeny smeru, výšky alebo orientácie, ktoré môžu spôsobovať pri detekovaní pádu nechcené falošné detekcie.

2.1.3 Cyklokros

Cyklokros je kombináciou prvých dvoch možností. Nejazdí sa len po spevnených cestách, ale ani v zložitom teréne. Z tohto pohľadu je získavanie polohy jednoduchšie ako pri cyklistike horskej, ale nie také dobré ako pri cestnej. Rovnako je to aj s detekciou pádov, kedy môže prísť k ojedinelým falošným detekciám.

2.2 Monitorovanie polohy cyklistov

Monitorovanie polohy cyklistov zohráva dôležitú úlohu najmä v oblasti športovej cyklistiky. Prvou oblasťou, do ktorej môže informácia o aktuálnej polohe prispieť, je zvýšená bezpečnosť cyklistu. Druhou oblasťou, kde je poloha cyklistu využívaná, je ukladanie informácií o tejto polohe pre následné spracovanie, analýzu a vyhodnocovanie danej jazdy.

2.2.1 Monitorovanie z hľadiska bezpečnosti

Určite je vhodné, ak iné osoby, ktoré sa nenachádzajú s cyklistom na určitej jazde, vedia zistiť, kde sa tento cyklista nachádza, či sa drží pôvodnej trasy a kedy približne dorazí do svojho cieľa. Aj samotný športovec sa môže cítiť omnoho bezpečnejšie, keďže vie, že jeho blízka osoba na neho dáva pozor, ak sa dostane do problémov. Nemusí sa teda báť vyraziť hlavne na dlhšie trasy, aj ak pôjde sám.

Ak sa cyklista ocitne v problémoch, môže jednoducho poslať správu, že potrebuje pomoc a nemusí riešiť a zisťovať, kde sa nachádza a ako svoju polohu poslať pomoci. Rovnako, ak aj cyklista nemá možnosť poslať správu, jeho blízka osoba môže vidieť, či športovec napreduje, alebo nastali nejaké nečakané udalosti.

2.2.2 Monitorovanie z hľadiska tréningu a následného spracovania

Ak sa cyklista chce na bicykli zlepšovať, už dávno to nie je len o tom najazdiť čo najväčší počet kilometrov. V súčastnosti začínajú byť čoraz populárnejšie štruktúrované tréningy, pri ktorých je veľmi dôležitý už počiatočný výber trasy tak, aby nám priniesla želaný tréning v niektorej oblasti.

Ďalej je taktiež možné využiť, a vo veľkej miere sa aj skutočne využívajú, informácie o prekonanej alebo plánovanej nadmorskej výške. Cyklista tak vie, koľko kopcov ho bude čakať a aké budú náročné. Taktiež vie túto informáciu využiť aj počas tréningu, kedy nemusí mať ani naplánovanú trasu a iba vie, koľko výškových metrov má nastúpať.

Neposlednou výhodou využívania GPS polohy pri tréningu je jej presnosť. Športovec tak presne vie, akú vzdialenosť prekonal a na základe toho vieme presne vypočítať napríklad priemernú rýchlosť.

2.3 Využívané zariadenia v oblasti cyklistiky

V oblasti cyklistiky sa na zaznamenávanie jázd využívajú tri druhy zariadení. Prvým typom je mobilný telefón s nainštalovanou príslušnou aplikáciou, druhým typom sú inteligentné hodinky, ktoré taktiež obsahujú nejakú vhodnú aplikáciu a tretím druhom zariadenia je špecializovaný GPS cyklistický počítač. Každé zariadenie má svoje výhody a nevýhody, ktoré si v krátkosti priblížime v nasledujúcich podkapitolách.

2.3.1 Mobilný telefón

Inteligentný mobilný telefón (tzv. *smartphone*), má využitie v množstve oblastí a rovnako tak aj v cyklistike. Po nainštalovaní vhodnej mobilnej aplikácie sa telefón môže zmeniť na zariadenie schopné monitorovať polohu cyklistu, poskytovať mu informácie počas jazdy, ale rovnako tieto informácie poskytovať aj osobám sledujúcim polohu cyklistu. Pre využitie mobilného telefónu na poskytovanie informácií počas jazdy na bicykli je ale potrebné ho nejakým spôsobom upevniť na bicykel. Výhodou mobilného telefónu je to, že v dnešnej dobe ho vlastní takmer ktokoľvek, takže nie je potrebná kúpa nového špeciálneho zariadenia. Výhodou je taktiež všestrannosť použitia, keďže využitie v cyklistike je iba jednou z okrajových možností využitia smartfónu. Neposlednou výhodou je množstvo rôznych aplikácií, kde má používateľ na výber podľa toho, čo bude presne požadovať.

2.3.2 Inteligentné hodinky

Inteligentné hodiny sa v poslednej dobe tešia v oblasti športu veľkej obľube. Už od výroby väčšinou podporujú množstvo športov a taktiež niektoré z nich ponúkajú možnosť doinštalovať ďalšie aplikácie, ktoré budú spĺňať požadované vlastnosti. Ak by sme chceli, aby inteligentné hodinky slúžili na zdieľanie aktuálnej polohy, potrebujeme zaistiť pripojenie hodiniek k internetu pre prenos dát o polohe. Prvým riešením sú hodinky s priamym pripojením k internetu, ktoré nevyžadujú prítomnosť mobilného telefónu. Druhým riešením sú hodinky pripojené cez Bluetooth k mobilnému telefónu, ktorý bude komunikovať s internetom a bude slúžiť ako medzikrok v komunikácii. Výhodou je, že tento telefón nemusí byť viditeľný a môže byť bezpečne odložený vo vrecku alebo ruksaku. Ostatné výhody sú podobné ako pri mobilnom telefóne, teda to, že ich vlastní veľké množstvo ľudí, sú všestranne použiteľné a existuje viacero vhodných aplikácií. Poslednou výhodou oproti mobilnému telefónu je, že nie je nutné dokupovať žiadne príslušenstvo pre uchytenie zariadenia, keďže hodinky sú bezpečne upevnené na ruke cyklistu.

2.3.3 GPS cyklistický počítač

Posledným používaným zariadením v cyklistike je špecializovaný GPS cyklistický počítač. Je to zariadenie, ktoré sa uchytí priamo na riadidlá bicykla a zobrazuje informácie o aktuálnej jazde. Toto zariadenie taktiež zaznamenáva pozíciu cyklistu a na konci jazdy je možné celú históriu polohy stiahnuť a použiť na ďalšie spracovanie. Výhodou takéhoto zariadenia je plná špecializácia na cyklistiku, dobrá výdrž batérie a cyklista má celý čas pred sebou všetky potrebné informácie. Nevýhodou môže byť nemožnosť nainštalovať akékoľvek ďalšie aplikácie, tým pádom nemožnosť nijako upraviť funkčnosť zariadenia.

2.4 Existujúce aplikácie

V nasledujúcej časti si priblížime niekoľko najznámejších a najpoužívanejších mobilných aplikácií v oblasti cyklistiky. Uvedieme si ich hlavné výhody, poprípade nevýhody a priblížime si možnosti zdieľania aktuálnej polohy v rámci danej aplikácie.

2.4.1 Strava

Medzi cyklistami sa často hovorí: „Čo nie je na Strave, to sa nestalo.“ Toto vyjadrenie iba potvrdzuje popularitu tejto aplikácie medzi športovcami, zvlášť cyklistami. Obľúbenosť tejto aplikácie dokazujú aj čísla z každoročného reportu z roku 2020. Za tento rok mala aplikácia Strava viac ako 73 miliónov používateľov, ktorí dokopy zaznamenali viac ako 17 miliárd kilometrov v takmer 1,2 miliardy aktivitách [51]. To robí túto aplikáciu jednou z najobľúbenejších mobilných aplikácií v oblasti športu.

Prvá verzia aplikácie Strava bola vydaná v roku 2009 [12] a od tej doby táto aplikácie ponúka čoraz väčšie množstvo užitočných funkcií nielen pre cyklistov, ale pre športovcov vo všeobecnosti. Aktuálne podporuje 32 rôznych typov aktivít, od cyklistiky, behu, viacero zimných športov až po surfovanie alebo jogu, pričom väčšina funkcionality je zameraná a dostupná pri troch základných typoch aktivity, a to všetky druhy cyklistiky, behov a plávania [38]. Výhodou je taktiež existencia verzie aplikácie určená pre inteligentné hodinky.

Čo sa týka zdieľania aktuálnej polohy počas aktivity, Strava ponúka funkciu nazývanú *Beacon*. Táto funkcia pracuje na princípe vygenerovania unikátnej URL, ktorú môže športovec zdieľať počas výjazdu maximálne s tromi ďalšími ľuďmi, nazývanými bezpečnostné kontakty. Tento bezpečnostný kontakt dostane textovú správu, ktorá obsahuje odkaz na zobrazenie aktivity v reálnom čase. Taktiež môžu vidieť históriu polohy športovca, čas začiatku aktivity, trvanie, vzdialenosť, ale aj zostávajúcu batériu. Počas zdieľania sa poloha športovca aktualizuje približne každých 15 sekúnd v závislosti od pokrytia sieťami v danej oblasti [39]. Nevýhodou tejto funkcie je, že neobsahuje žiadnu formu detekcie pádu, maximálny počet bezpečnostných kontaktov je obmedzený na tri a na jednej obrazovke nie je možné sledovať viacerých športovcov naraz.

2.4.2 Komoot

Komoot patrí medzi najpoužívanejšie aplikácie v cyklistike hlavne pred samotnou jazdou, teda v čase plánovania trasy. Táto aplikácia obsahuje jednoduchý plánovač trás, odporúčania od komunity, hlasovú navigáciu alebo množstvo inšpiratívneho obsahu od ostatných používateľov. Práve pre tieto vlastnosti sa aplikácia Komoot stala voľbou pre viac ako 18 miliónov ľudí po celom svete [30].

Najsilnejšou funkciou celej aplikácie je práve plánovanie trás. Športovec si pri plánovaní môže vybrať z ôsmich druhov športu, medzi ktoré patria hlavne rôzne druhy cyklistiky, turistika alebo beh [33]. Na základe zvoleného druhu športu potom aplikácia navrhne ideálnu trasu, ktorú si môže používateľ následne ľubovolne upravovať. Veľkou výhodou aplikácie Komoot sú aj doplňujúce informácie o naplánovanej trase, ako napríklad typy ciest alebo povrch, po ktorom je trasa naplánovaná [32]. Rovnako je v tejto aplikácií možnosť zaznamenať samotnú aktivitu. Potom je možné túto aktivitu označiť ako jeden z množstva rôznych druhov športu, ako všetky typy cyklistiky, beh, turistika, ale aj beh na lyžiach, snowboarding alebo lezenie [33]. Rovnako ako aplikácia Strava, aj aplikácie Komoot je dostupná nielen na mobilné zariadenie, ale aj na inteligentné hodinky.

V apríli 2021 pribudla aj v tejto aplikácií možnosť zdieľania aktuálnej polohy. Pre jej používanie je potrebné mať zakúpenú prémiovú licenciu Komoot Premium. Táto funkcia funguje veľmi podobne ako v aplikácií Strava. Vygeneruje sa unikátny odkaz na sledovanie konkrétnej aktivity, ktorý potom športovec môže zdieľať v rámci samotnej aplikácie alebo aj mimo nej. Sledujúci potom môže vidieť aktuálnu polohu športovca, prejdenú a plánovanú dĺžku trasy, trvanie aktivity, zostávajúcu batériu a navyše oproti aplikácií Strava aj odhadovaný čas príjazdu do cieľa [31]. Bohužiaľ, rovnako ako pri aplikácií Strava ani aplikácia Komoot neobsahuje v žiadnej forme detekciu pádu a taktiež nie je možne naraz sledovať viacerých športovcov.

2.4.3 Ride with GPS

Ride With GPS je ďalšou z veľmi obľúbených aplikácií medzi cyklistami. Na rozdiel od predchádzajúcich aplikácií Ride With GPS nepodporuje žiadne iné typy aktivít okrem cyklistiky. Celá aplikácia je teda ladená a zameraná tak, aby vyhovovala potrebám cyklistov v čo najväčšom rozsahu. Aplikácia Ride With GPS ponúka viaceré funkcie od naplánovania konkrétnej trasy, cez navigáciu počas tejto jazdy až po vytvárania a zdieľanie reportov z jázd, ktoré môžu slúžiť ako inšpirácia pre ostatných [46]. Túto aplikáciu je možné využívať ako na mobilnom zariadení, tak aj v inteligentných hodinkách.

Jednou z hlavných funkcií tejto aplikácie je práve vytváranie samotných trás, ktoré môže prebiehať dvomi spôsobmi. Prvým spôsob je využitie interaktívnej mapy, na ktorej používateľ môže jednoducho pridávaním bodov vytvoriť požadovanú trasu. Druhou metódou je nájdenie vhodnej trasy na základe požadovaných parametrov trasy. Používateľ zadá približnú lokáciu, požadovanú vzdialenosť trasy a aplikácia mu ponúkne na výber z trás, ktoré spĺňajú tieto požiadavky [46].

Rovnako ako ostatné spomínané aplikácia aj Ride With GPS poskytuje určitú formu zdieľania aktuálnej polohy. Táto funkcia sa tu nazýva Live Logging a je súčasťou predplatného úrovne Basic a Premium, nie je teda dostupná v bezplatnej verzii aplikácie. Pri tejto funkcii je možné nastaviť napríklad obnovovaciu frekvenciu polohy od 30 sekúnd do 10 minút a rovnako je možné nastaviť, kto bude mať prístup k polohe cyklistu. Toto je možné obmedziť na verejné, kedy hocikto môže navštíviť profil cyklistu a vidieť jeho polohu, iba priatelia, kedy hocikto zo zoznamu priateľov tohto cyklistu môže vidieť jeho polohu, a súkromné, kedy iba samotný profil cyklistu môže vidieť aktuálnu polohu. Poslednú možnosť je možné využiť napríklad v prípade, ak niekto z rodiny tohto cyklistu je prihlásený pod jeho účtom napríklad doma na počítači. Zaujímavou funkčnosťou zdieľania polohy je zdieľanie fotografií v reálnom čase. Taktiež je možné pridať reakciu na fotografiu alebo príspevok, ktorú športovec uvidí už počas výjazdu. Nevýhodou je opäť možnosť sledovať iba jedného cyklistu na obrazovke naraz. Taktiež Ride With GPS neobsahuje žiadnu formu detekcie pádu a následnej notifikácie ostatných používateľov [47].

2.4.4 Map My Ride

Ďalšou populárnou aplikáciou s viac ako 5 miliónom stiahnutiami na Google Play je aplikácia Map My Ride od spoločnosti Under Armour. Popri verzii pre cyklistov existujú aj ďalšie verzie zamerané na chôdzu, beh alebo fitness. Map My Ride obsahuje nástroje na analýzu vykonaných aktivít, ich zdieľanie s ostatnými používateľmi, ale aj na plánovanie trás. Rovnako ako pri predchádzajúcich aplikáciách, aj Map My Ride existuje vo verzii určenej pre inteligentné hodinky. Vo väčšej miere je však aplikácia zameraná na zlepšovanie

sa na bicykli, takže ponúka možnosti hlasového trénera počas jazdy aj viaceré pokročilé metriky a vizualizácie po samotnej aktivite [37].

V prémiovej verzii tejto aplikácie je taktiež funkcia nazývaná *Live Tracking*, ktorá poskytuje možnosť zdieľania aktuálnej polohy so všetkými kontaktmi v rámci aplikácie. Výhodou je, že používatelia, ktorí nechcú zdieľať polohu, ale chcú iba sledovať niektorý zo svojich kontaktov, nemusia vlastniť prémiovú verziu aplikácie. Taktiež je možné na jednej obrazovke sledovať viacero svojich kontaktov naraz. Z hľadiska bezpečnosti ale stále nie je prítomná žiadna detekcia pádu a následná notifikácia kontaktov [45].

2.4.5 Google Maps

Google Maps patrí rozhodne medzi najpoužívanejšie aplikácie v oblasti mapovania, navigácie a zdieľania polohy, nielen v oblasti športu. Táto aplikácia ponúka možnosti tvorenia trás prispôsobené podľa toho, či plánujeme kráčať, ísť na bicykli alebo autom. Bohužiaľ, práve plánovanie trasy pre bicykel nefunguje na území Slovenskej ani Českej republiky, čo do veľkej miery obmedzuje použitie Google Maps pri cyklistike.

V oblasti zdieľania polohy ale Google Maps ponúka viaceré zaujímavé funkcie. Prvou je klasické zdieľanie aktuálnej polohy, pri ktorom môžeme vidieť len polohu sledovaného, bez nejakej histórie polohy alebo naplánovanej trasy. Nedávno ale pribudla druhá možnosť zdieľania, a to zdieľať priebeh cesty. Táto možnosť sa sprístupní potom, ako v aplikácii spustíme navigovanie do určitého cieľa. Táto možnosť funguje ako pri chôdzi, tak pri jazde autom a v určitých oblastiach je už podporovaná aj pri cyklistike. Ak využijeme zdieľanie priebehu cesty, sledujúci môže vidieť históriu našej polohy, plánovanú trasu a aj odhadovaný čas príchodu. Obe tieto funkcie sú dostupné ako na zariadeniach so systémom Android, tak na zariadeniach iPhone alebo iPad od Apple [27]. Aplikácia Google Maps je tiež dostupná v inteligentných hodinkách, aj keď s obmedzenou funkčnosťou.

2.4.6 Find My

Poslednou relevantnou aplikáciou je aplikácia od firmy Apple s názvom Find My. Táto aplikácia v sebe spája predtým dve samostatné aplikácie, a to Find My iPhone a Find My Friends. Práve druhá aplikácia slúžila na zdieľanie svojej aktuálnej polohy s priateľmi a teraz je súčasťou spomínanej aplikácie Find My [2].

Aj keď táto aplikácia nebola vyvinutá za účelom zdieľania polohy pri vykonávaní športu, je možné ju takýmto spôsobom využiť. Zaujímavou funkciou v aplikácii Find My je možnosť nastaviť upozornenia, keď sledovaný človek opustí nejakú lokáciu, alebo príde do určitej lokácie. To môže byť dobre využiteľné aj v cyklistike, kedy sledujúci dostane upozornenie, keď vyrazíte z domu na jazdu a tiež, keď sa z nej vrátite naspäť domov. Rovnako je možné sledovať naraz viacero osôb alebo zdieľať polohu s viacerými osobami [2]. Keďže však aplikácia nie je určená primárne na použitie v športe, nie je možné pri sledovanom vidieť žiadne informácie o aktuálnom výjazde, históriu lokácie alebo predpokladaný príchod. Z toho istého dôvodu aplikácia neobsahuje ani žiadny detektor pádu.

2.4.7 Zhrnutie

Ako je možné vidieť z analýzy aktuálne používaných aplikácií v oblasti cyklistiky, žiadna aplikácie nepodporuje sledovanie viacerých športovcov naraz. Väčšina z nich je zameraná hlavne na spätnú analýzu jazdy a na zlepšovanie kondície a výkonnosti športovca. Rovnako tieto aplikácie neobsahujú žiadnu formu detekcie pádu, ktorá môže v hraničných prípadoch veľmi napomôcť bezpečnosti cyklistu.

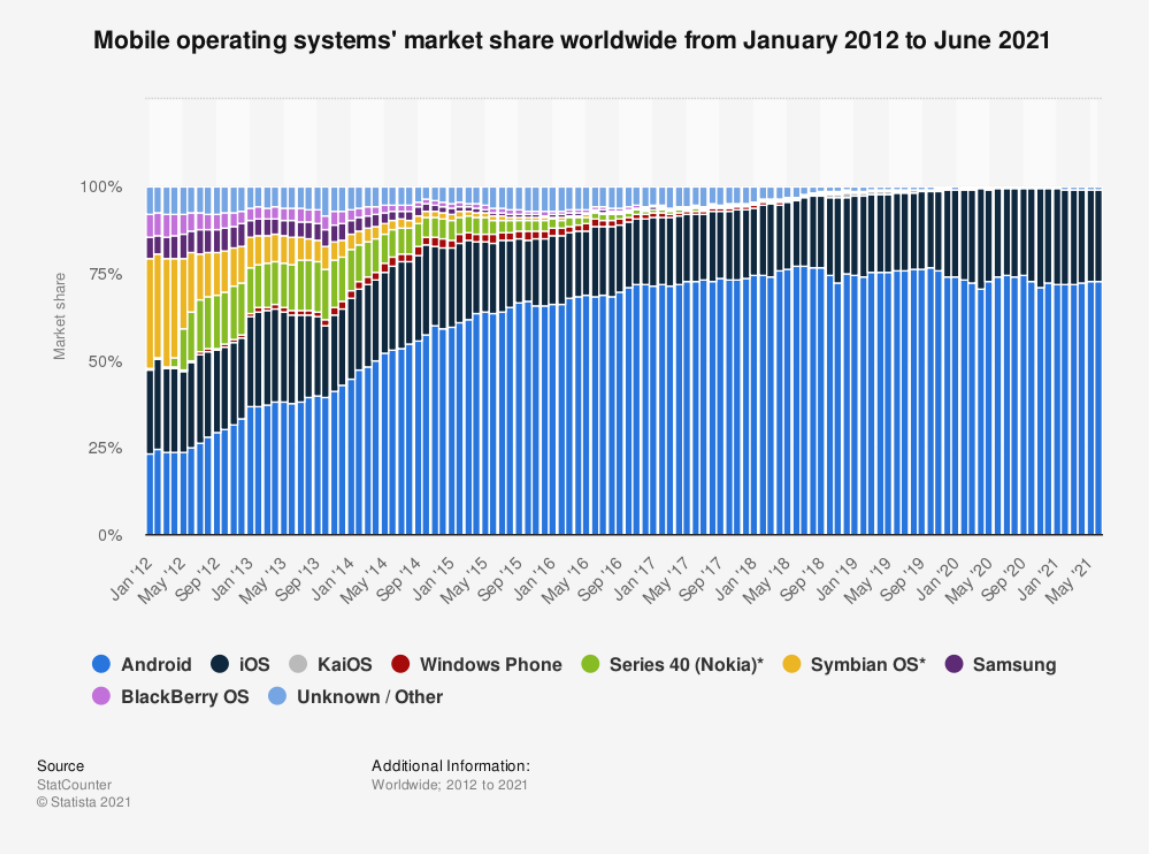
Kapitola 3

Mobilné aplikácie

V súčasnej dobe je trendom používanie inštalovaných mobilných aplikácií v mobilnom telefóne, ktoré nám uľahčujú každodenný život. Používatelia si už zvykli a nevadí im, ak si potrebujú nainštalovať viaceré mobilné aplikácie, z ktorých je každá zameraná a špecifikovaná na niečo iné.

Pri vývoji mobilných aplikácií sa dá v súčasnosti postupovať tromi spôsobmi [56]. Prvým je vývoj natívnej mobilnej aplikácie určenej pre jeden konkrétny operačný systém. Ako môžeme vidieť na obrázku 3.1, trh s mobilnými zariadeniami zastupujú takmer stopercentne dvaja hráči, a to operačný systém Android od spoločnosti Google a operačný systém iOS od spoločnosti Apple [50]. Keďže tieto systémy sa v mnohom líšia, je potrebné vyvinúť jednu samostatnú mobilnú aplikáciu pre každý z týchto systémov. Aj samotný vývoj je v mnohom odlišný. Pre operačný systém iOS sú aplikácie vyvíjané hlavne v programovacom jazyku C, alebo Objective-C, naopak pre vývoj na systéme Android sa používa hlavne programovací jazyk Java a v posledných rokoch sa k nemu pridáva aj jazyk Kotlin [9]. Už z tohto môžeme vidieť, že takýto vývoj je dosť náročný, ako časovo, tak aj finančne, keďže sú potrební viacerí experti v jednotlivých systémoch. Druhou cestou je vývoj mobilnej webovej aplikácie, čo je v podstate klasická webová aplikácia, ktorá je prispôsobená na zobrazenie aj na menších obrazovkách mobilných telefónov. Poslednou možnosťou je vývoj takzvaných hybridných mobilných aplikácií, ktoré sú jednoduchšie na počiatočný vývoj, hlavne z dôvodu, že veľká časť kódu je zdieľaná pre oba systémy. Tento kód je potom preložený a beží ako klasická natívna aplikácia na koncovom zariadení.

V nasledujúcich podkapitolách si priblížime spomenuté prístupy vývoja, ich výhody, nevýhody a uvedieme si príklad technológie, ktorá daný prístup používa.



Obr. 3.1: Podiel jednotlivých mobilných operačných systémov od roku 2012 do 2021. Prevzaté z [50].

3.1 Natívne mobilné aplikácie

Natívna mobilná aplikácia je to, čo si bežný používateľ predstaví, keď sa povie mobilná aplikácia. Je to samostatný program, ktorý používateľ musí stiahnuť a nainštalovať do svojho zariadenia. Tento program je určený pre beh na konkrétnom operačnom systéme, s ktorým počas svojho behu komunikuje a využíva prístup k rozhraniam poskytovaným týmto systémom. Takáto aplikácia je teda postavená na sade nástrojov a knižníc ponúkaných priamo tvorcom daného operačného systému. V prípade Apple sa jedná o iOS SDK a v prípade Google je to Android SDK.

Prvou veľkou výhodou natívnej aplikácie je všeobecne vyššia rýchlosť a vyšší výkon finálnej aplikácie v porovnaní s iným prístupom. Ďalšou výhodou je možnosť využiť všetky natívne poskytované funkcie telefónu, ako je napríklad prístup k fotoaparátu, polohe zariadenia alebo k dátam z rôznych senzorov nachádzajúcich sa na zariadení. Nevýhodami sú už spomínaná vyššia finančná a časová náročnosť vývoja, ako aj nutnosť pravidelne aktualizovať túto aplikáciu. Aj napriek spomenutým nevýhodám ponúkajú natívne mobilné aplikácie vo veľa prípadoch najlepšiu použiteľnosť, funkcie a používateľskú skúsenosť [56].

3.1.1 Android

Android je mobilný operačný systém, ktorý je postavený na operačnom systéme Linux a rozširuje ho. Prvotne bol vyvíjaný start-upom rovnakého mena, Android. Neskôr, v roku 2005, ho odkúpila spoločnosť Google, ktorá od tohto roku spravuje a vyvíja tento operačný systém. Ten bol na začiatku voľne prístupný pod licenciou *Apache Licence*, teda každý si mohol stiahnuť voľne celý zdrojový kód tohto systému. To znamená, že výrobcovia mobilných zariadení sú schopní a je im umožnené poupraviť si Android tak, aby plne vyhovoval ich požiadavkám. Tento model robí Android veľmi zaujímavým z hľadiska vývoja [35].

Systém

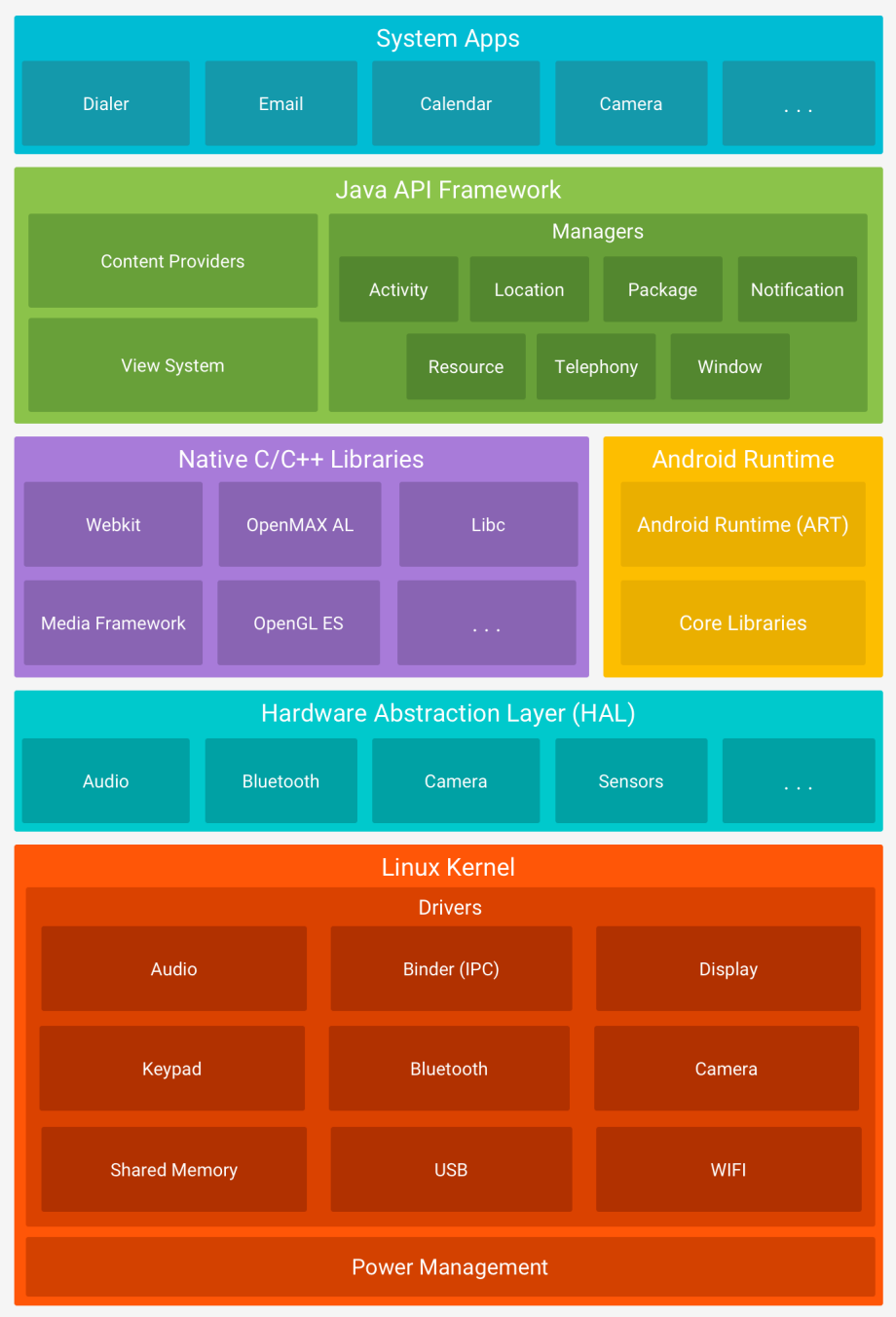
Ako sme mohli vidieť na obrázku 3.1, operačný systém Android používa celosvetovo takmer 75% majiteľov mobilných telefónov. Je to hlavne z dôvodu, že tento systém je používaný na zariadeniach rôznych značiek, nie je teda obmedzený iba na jedného konkrétneho výrobcu, ako je to v prípade iOS od Apple. Z tohto dôvodu je Android veľmi zaujímavou cieľovou platformou pre vývoj mobilnej aplikácie. Tieto aplikácie sú potom dostupné na stiahnutie buď v spoločnom obchode pre všetky zariadenia so službami od Google, teda *Google Play Store*, alebo v obchodoch s aplikáciami od samotného výrobcu, ako napríklad *Galaxy Store* od Samsungu alebo *AppGallery* od výrobcu Huawei.

Ako sme už spomenuli v úvode, Android je operačný systém, ktorého základ tvorí linuxové jadro. Toto jadro obsahuje potrebné hardvérové ovládače, ale rovnako poskytuje možnosť pre výrobcov vytvárať vlastné ovládače, čo im uľahčuje práve využitie dobre známeho linuxového jadra. Nad linuxovým jadrom beží hardvérová abstraktná vrstva (*Hardware Abstraction Layer, HVL*), ktorá tvorí rozhranie medzi hardvérom a vyššími vrstvami softvéru. Táto vrstva pozostáva z množstva knižníc, z ktorých každá implementuje rozhranie pre niektorý hardvérový komponent, ako napríklad fotoaparát. Nad touto vrstvou sa vyskytuje sada knižníc napísaných v jazyku C a C++, ktoré obsluhujú natívne funkcie zariadenia. Popri týchto knižniciach sa na tejto vrstve nachádza aj behové prostredie Androidu (*Android Runtime*). Od verzie Androidu 5 každá aplikácia beží v samostatnom procese a inštancii behového prostredia. Toto prostredie sa teda stará o beh výsledných aplikácií. Vyššou vrstvou je potom aplikačné rozhranie Javy (*Java API Framework*), ktorá poskytuje celú funkcionalitu systému pre komunikáciu s výslednou aplikáciou. Toto rozhranie vytvára stavebné bloky pre tvorbu aplikácií na systéme Android a obsahuje [28]:

- *Content Provider*, ktorý umožňuje aplikáciám prístup k dátam z inej aplikácie,
- *View System*, ktorý sa využíva na tvorbu používateľského rozhrania,
- *Activity Manager*, starajúci sa o životný cyklus aplikácie,
- *Location Manager*, umožňujúci prístup k polohe zariadenia,
- *Package Manager*, vďaka ktorému sa aplikácie môžu dozvedieť o iných nainštalovaných aplikáciách v zariadení,
- *Notification Manager*, ktorý umožňuje všetkým aplikáciám zobrazenie notifikácie v stavovom riadku,
- *Resource Manager*, poskytujúci prístup k nekódovým zdrojom, ako napr. lokalizované reťazce alebo obrázky,

- *Telephony Manager*, starajúci sa o telefónne služby,
- *Window Manager*, zodpovedný za zobrazovanie okien.

Celú schému architektúry systému Android je možné vidieť na obrázku 3.2.



Obr. 3.2: Schéma architektúry systému Android. Najnižšia vrstva obsahuje známe jadro Linuxu. Nad ním sa nachádza hardvérová abstraktná vrstva, ktorá obsahuje ovládače pre hardvérové prvky zariadenia. Nad hardvérovou vrstvou sa nachádza sada natívnych knižníc systému Android. Nad nimi je už len aplikačné rozhranie jazyka Java a samotná aplikácia. Obrázok prevzatý z [28].

Vývoj

Android je v súčasnosti vyvíjaný najmä v dvoch hlavných programovacích jazykoch, a to Java a Kotlin. Obidva tieto jazyky sú kompilované do takzvaného *bytekódu*, ktorý je následne spustený na virtuálnom stroji Javy *(Java Virtual Machine, JVM)*. Java bola vyvinutá už v roku 1991 ako časť tzv. *Green Project-u*, ktorého cieľom bolo priniesť novú vlnu v oblasti výpočetnej techniky [8]. Je to objektovo orientovaný programovací jazyk, ktorého úmyslom bolo nahradiť jazyk C++. Veľkou výhodou tohto jazyka je práve beh na JVM, ktorý zabezpečuje úplnú platformovú nezávislosť, ktorú dokáže ponúknuť máloktorý iný programovací jazyk. Na druhej strane jej nevýhodou je, že aj na vykonanie jednoduchej úlohy je potrebné značné množstvo riadkov kódu. Aj toto sa snaží riešiť programovací jazyk Kotlin, ktorý bol vyvinutý nedávno, v roku 2010, spoločnosťou JetBrains. Je to pragmatický programovací jazyk, ktorý v sebe kombinuje objektovú orientáciu s funkcionálnym programovaním. Aj z tohto dôvodu Kotlin priťahuje čoraz väčšie množstvo vývojárov [44]. Porovnanie vytvorenia jednoduchej triedy v oboch jazykoch môžeme vidieť na ukážkach kódu 3.1 a 3.2 nižšie, ktoré boli prevzaté z [21].

```
class Book {
    private String title;
    private Author author;

    public String getTitle(){
        return title;
    }
    public void setTitle(String title){
        this.title = title;
    }

    public Author getAuthor(){
        return author;
    }
    public void setAuthor(Author author){
        this.author = author;
    }
}
```

Výpis 3.1: Príklad jednoduchej triedy v jazyku Java [21]

```
data class Book(var title: String, var aut: Author)
```

Výpis 3.2: Príklad jednoduchej triedy v jazyku Kotlin [21]

Pre vývoj aplikácií na systém Android spoločnosť Google vytvorila program nazývaný Android Studio. Ten poskytuje sadu nástrojov pre rýchly a efektívny vývoj aplikácií buď v jazyku Java alebo Kotlin. V tomto nástroji je možné vytvárať aplikácie pre všetky druhy zariadení využívajúcich systém Android, ako mobilné telefóny, tablety, televízory alebo inteligentné hodinky. Rovnako obsahuje emulátory týchto zariadení vhodné k testovaniu aplikácie [25].

3.1.2 iOS

Mobilný operačný systém iOS je vyvíjaný firmou Apple Inc. pre jeho vlastné zariadenia, ako iPhone, iPad a iné Apple mobilné zariadenia. Prvýkrát bol iOS predstavený s prvým zariadením iPhone v roku 2007. Na rozdiel od systému Android, iOS je vyvíjaný ako proprietárny softvér, teda nie je ako celok voľne dostupný, aj keď niektoré jeho časti sú open-source. To

však nemusí byť problém, keďže tento operačný systém Apple používa výhradne na svojich vlastných zariadeniach.

Systém

Z obrázku 3.1 môžeme vidieť, že iOS je po systéme Android druhý najpoužívanejší operačný systém s vyše 25% podielom na trhu mobilných operačných systémov. Preto je logické, že väčšina vývojárov cieli svoje aplikácie práve na tieto dve platformy, ktoré ovládajú trh.

Samotný systém iOS sa skladá zo štyroch vrstiev. Prvou a najnižšou je *Core OS*, na ktorej sú postavené všetky technológie iOS. Nad ňou sa nachádza vrstva nazývaná *Core Services*, ktorá obsahuje množstvo frameworkov využiteľných práve pri vývoji aplikácií a poskytujúcich lepšiu funkcionalitu samotného systému. Nad touto vrstvou beží vrstva *Media*, ktorá umožňuje systému všetky grafické, audio a video technológie. Poslednou vrstvou je *Cocoa Touch*, tiež známa ako vrstva aplikácií. Táto vrstva vytvára rozhranie pre používateľa na prácu s operačným systémom iOS [6]. Schému tohto systému môžeme vidieť na obrázku 3.3.

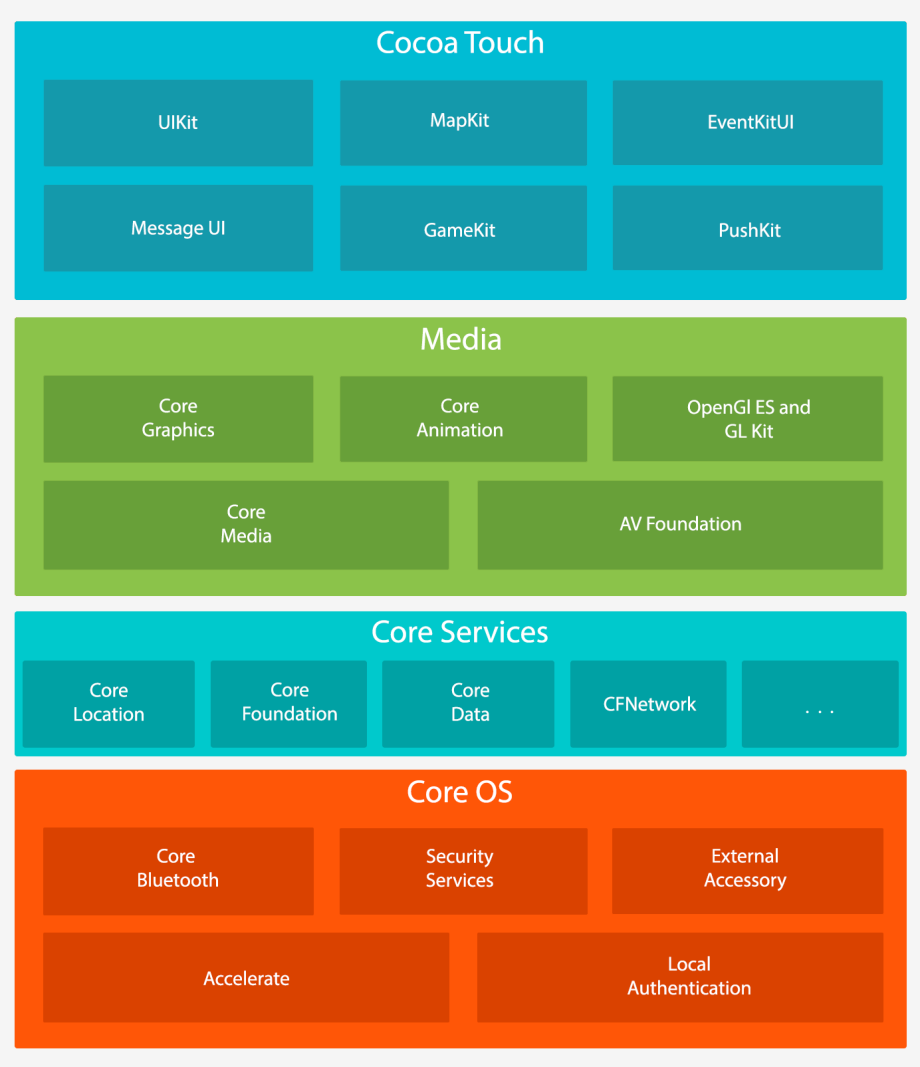
Vývoj

Aplikácie na systém iOS sú v súčasnosti vyvíjané v jazyku Swift, ktorý bol vytvorený firmou Apple ako náhrada za pôvodne používaný jazyk Objective-C. Tento jazyk bol prvýkrát oznámený v roku 2014 a od tej doby sa stal jedným z najrýchlejšie rastúcich programovacích jazykov v histórií. Je to objektovo orientovaný jazyk, ktorý bol vyvinutý s cieľom umožniť jednoduchý vývoj mobilných alebo desktopových aplikácií. Rovnako kladie veľký dôraz na bezpečnosť a rýchlosť výsledného kódu [1]. Príklad triedy v jazyku Swift môžeme vidieť na ukážke kódu 3.3.

```
1 class Book {
2     var title: String;
3     var author: Author;
4 }
```

Výpis 3.3: Príklad jednoduchej triedy v jazyku Swift

Vývoj mobilných aplikácií pre systém iOS je možný iba na operačnom systéme OSX, ktorý je taktiež vyvinutý spoločnosťou Apple Inc. V tomto systéme je pre vývoj mobilnej aplikácie pripravené vývojové prostredie XCode, ktoré obsahuje sadu nástrojov pre vývoj, preklad a testovanie aplikácie na iOS. Tiež obsahuje emulátor virtuálneho zariadenia iPhone, ktorý môže byť použitý počas testovania. Vývoj aplikácie práve pre iOS môže byť o niečo jednoduchší z hľadiska toho, že platforma iOS obsahuje relatívne malé množstvo verzií, ktoré sú všetky priamo podporované spoločnosťou Apple [55].



Obr. 3.3: Schéma architektúry systému iOS. Obrázok inšpirovaný z [48].

3.2 Webové mobilné aplikácie

Mobilný web je jednoducho povedané klasický web, na ktorý sa pristúpi pomocou mobilného zariadenia. Ľudia si v poslednej dobe zvykli na prístup k dátam na internete pomocou prehliadača v mobilnom telefóne, preto má aj vývoj takejto aplikácie zmysel. Každá webová stránka, na ktorú sa pristúpi pomocou mobilného zariadenia je mobilný web, či už bola na to stánka prispôsobená, alebo nie [40].

To, čo bolo problémom pri natívnej tvorbe aplikácií, teda nutnosť tvoriť samostatnú aplikáciu pre každú cieľovú platformu, neplatí pre webovú aplikáciu. Pri nej pre nás len dôležité, aby bol na cieľovej platforme nainštalovaný prehliadač, cez ktorý bude možné webovú aplikáciu zobraziť [9]. Ďalšou výhodou je jednoduchší a rýchlejší vývoj, ktorý je umožnený vďaka využívaniu známych a jednoduchých technológií. Rovnako výhodou je aj jednoduchšia údržba výslednej aplikácie, keďže je potrebné udržiavať iba jednu verziu, ktorá využíva iba známe a jednoduché technológie. Nevýhodou mohla byť nemožnosť pristupovať

k natívnej funkcionalite samotných zariadení, to sa ale dá riešiť pomocou frameworkov ako je napríklad *PhoneGap* [4].

Nevýhodami na druhej strane budú vlastnosti, v ktorých vynikali natívne aplikácie. V prvom rade je to výkon výslednej vyvinutej aplikácie. Ďalšou nevýhodou je oveľa obtiažnejšie testovanie a ladenie aplikácie na mobilných zariadeniach. Aj keď je možné využiť rovnaké emulátory ako pri natívnych aplikáciách, tie neposkytujú presné výsledky, hlavne v oblastiach výsledného výkonu. Nevýhodou je tiež ťažšia propagácia a speňaženie aplikácie kvôli absencii obchodu, ako tomu bolo v prípade natívnych aplikácií. Poslednou a jednou z najväčších nevýhod je, že aj pri použití frameworkov, ako je napríklad *PhoneGap* , stále nie je možné pristúpiť ku všetkým natívnym funkciám zariadenia. Ak naša aplikácia potrebuje pre svoje správne fungovanie niektorú z týchto funkcií, je táto prekážka neprekonateľná [4].

Pri samotnom vývoji aplikácie sa potom postupuje rovnako, ako pri tvorbe klasickej webovej aplikácie. Používajú sa technológie ako HTML, CSS alebo JavaScript, ktoré slúžia na zobrazenie finálnej aplikácie používateľovi. Táto časť aplikácie sa potom nazýva *frontend*. Takmer pri každej webovej aplikácii je nutné vyvinúť aj tzv. *backend*, teda časť aplikácie, ktorá zvyčajne beží na nejakom serveri a slúži na vykonávanie zložitejšej logiky kódu, ukladanie alebo získavanie informácií z databáz atď. Pre komunikáciu medzi týmito dvoma časťami aplikácie sa potom najčastejšie využíva určitá forma aplikačného rozhrania (*Application Programming Interface, API*), napríklad *REST API*.

3.2.1 React

React je populárna JavaScriptová knižnica používaná na tvorbu používateľských rozhraní vo webových aplikáciách. Bola vyvinutá spoločnosťou *Facebook* v roku 2013 a prvotne bola vnímaná pomerne skepticky, hlavne kvôli konvenciám, ktoré sú v knižnici React pomerne unikátne [5].

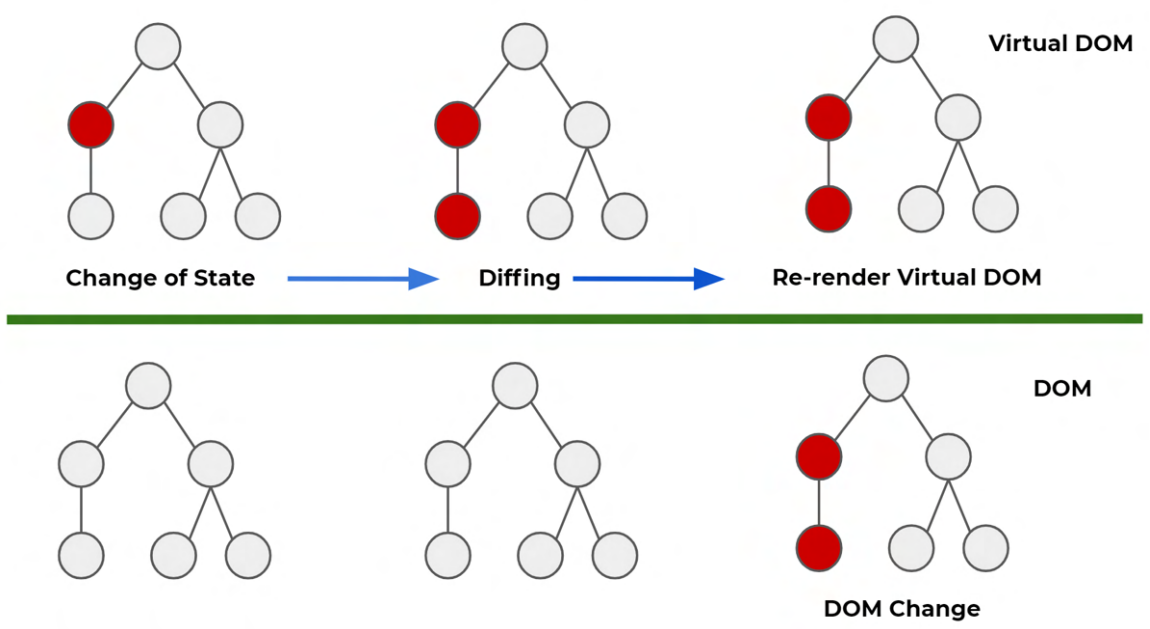
Aplikácie v knižnici React sú najčastejšie tvorené pomocou JSX, čo je syntaktické rozšírenie klasického JavaScriptu. Nie je nutné JSX rozšírenie používať, ale umožní nám vlastnosti, ktoré nám uľahčia tvorbu výsledného kódu. Toto rozšírenie totiž umožňuje tvorbu HTML elementov priamo v JavaScriptovom kóde. JSX môže na prvý pohľad pripomínať klasický značkovací jazyk, má však plnú silu jazyka JavaScript. React pri použití rozšírenia JSX pracuje na princípe, že logika aplikácie je veľmi úzko spätá s logikou reakcie na vstupy používateľov alebo zmenu stavu aplikácie. Nie je teda potrebné oddeľovať značkovací jazyk a logiku aplikácie do samostatných súborov [20]. Príklad využitia rozšírenia JSX je možné vidieť na príklade kódu 3.4.

```
1 const user = 'John';
2
3 const App = () => {
4     return <h1>Hello, {user}!</h1>
5 };
```

Výpis 3.4: Príklad využitia rozšírenia JSX

Kľúčovým prvkom vo vývoji aplikácie v knižnici React je princíp vytvárania komponentov. Komponenty umožňujú programátorom rozdeliť používateľské rozhranie do nezávislých, znovu použiteľných častí kódu, nad ktorými môžeme premýšľať izolovane bez vplyvu na iné komponenty. V knižnici React môžeme uvažovať o dvoch základných typoch komponentov, a to funkčné a triedne. Každý z týchto typov má svoje výhody a nevýhody a je len na programátorovi, ktorý prístup preferuje, keďže čo sa funkčnosti týka, sú si ekvivalentné.

Ďalšou vlastnosťou, prečo sa React stal populárnym, je vytváranie virtuálnej kópie DOM (*Document Object Model*), vďaka ktorej je následná manipulácia s jednotlivými objektmi jednoduchšia a hlavne rýchlejšia. Práve toto je jeden z najdôležitejších konceptov celej knižnice. Tento koncept funguje na princípe, že pri zmene dát nie je potrebné nanovo prekresliť celú stránku (celý DOM), namiesto toho sa aktualizuje virtuálny DOM v pamäti a vypočíta sa najmenšia nutná časť reálneho DOM, ktorú je potrebné aktualizovať [23]. Tento princíp vo veľkej miere pozitívne ovplyvňuje rýchlosť finálnej aplikácie. Priebeh aktualizácie DOM na základe zmeny vo virtuálnom DOM je zobrazený na obrázku 3.4.



Obr. 3.4: Proces aktualizácie DOM na základe zmeny vo virtuálnom DOM. Na začiatku sa detekuje zmena nejaké prvku vo virtuálnom DOM. Následne sa vypočíta, ktorá časť virtuálneho DOM je touto zmenou ovplyvnená. Nakoniec sa prekreslí iba zasiahnutá časť virtuálneho aj skutočného DOM. Obrázok prevzatý z [10].

Posledným konceptom, ktorý si pri knižnici React priblížime, je stav komponentov (*State*). Stav je rovnako ako *props*, ktorý je jediným parametrom pri vytváraní komponentu, čistý objekt jazyka JavaScript. Rozdiel medzi stavom a *props* je v tom, že stav je spravovaný iba interne v rámci komponentu. Meniť stav je možné výhradne pomocou špecializovanej funkcie na to určenej, ktorá sa v tomto prípade nazýva *setState()*. Táto funkcia je asynchrónna, čo znamená, že zmena sa nemusí prejaviť okamžite. Pristúpiť k hodnote stavu je potom možné pomocou *this.state*, keďže *state* je v triednom komponente uložený ako objekt konštruktora. Rovnako je možné využívať stav komponentu pri funkčných komponentoch, kde k tomuto účelu slúži špeciálny hook *useState* [18]. Stavy sa používajú na uchovávanie informácií v komponente, ktoré sa môžu počas jeho životného cyklu meniť. Pri zmene týchto dát sa potom automaticky zavolá prekreslenie komponentu obsahujúceho zmenené dáta. Príklad využitia stavu komponentu v triednom a funkčnom komponente môžeme vidieť na ukážke kódu 3.5.

```
1 class Welcome extends React.Component {
2   constructor(props) {
3     super(props);
```

```
    this.state = {name: "John"};
  }

  render() {
    return <h1>Hello, {this.state.name}</h1>;
  }
}

const Welcome = (props) => {
  const [name, setName] = useState("John");
  return <h1>Hello, {name}</h1>;
}
```

Výpis 3.5: Príklad využitia stavu komponentu v triednom a funkčnom komponente

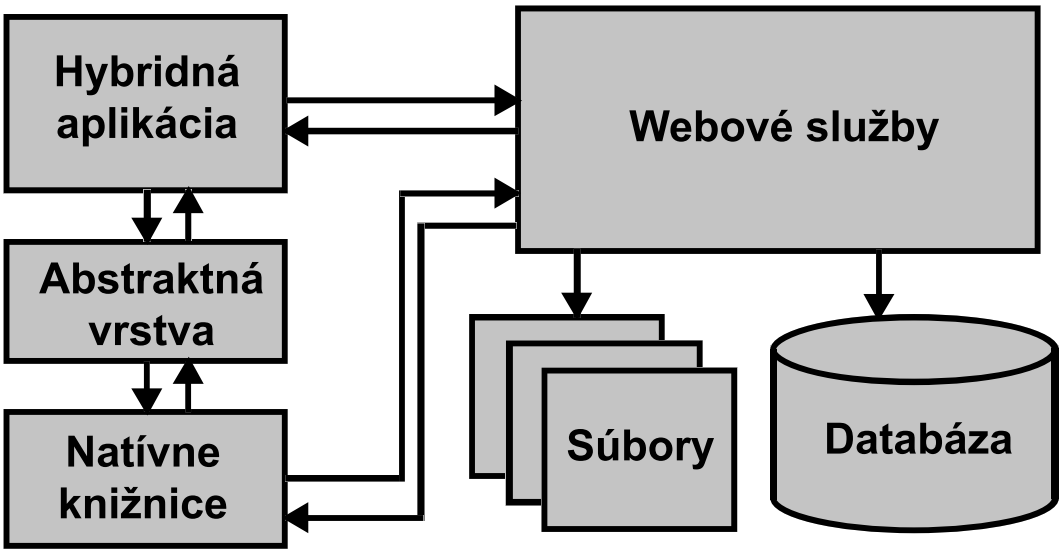
3.2.2 Progressive Web Application

Špeciálnym prípadom webovej aplikácie je PWA (Progressive Web Application). Jedná sa o webovú aplikáciu, ktorá je vyvíjaná s využitím špecifických postupov, ktoré umožňujú využívanie ako webových, tak aj natívnych funkcií zariadenia. Výhodou takejto aplikácie je jej veľmi jednoduché získanie, kedy nie je potrebné hľadať aplikáciu v obchode. Stačí navštíviť webovú stránku, z ktorej pomocou jedného kliknutia dokážeme vytvoriť odkaz na PWA na domovskej obrazovke zariadenia. Po pridaní na domovskú obrazovku používateľ môže PWA využívať ako natívnu aplikáciu zariadenia, ktorá pracuje rýchlejšie ako jej webový ekvivalent a rovnako môže fungovať aj bez pripojenia na internet [52].

Progresívne webové aplikácie naberajú na popularite vo viacerých oblastiach. Táto popularita vychádza z určitých charakteristík, ktorý musia byť splnené pri každej PWA.

- **Progresívnosť** – aplikácia musí pracovať pre všetkých používateľov na všetkých prehliadačoch a musí využívať benefity zariadenia aj prehliadača.
- **Responzívnosť** – používateľské rozhranie PWA sa musí prispôsobiť všetkým veľkostiam a typom zariadení.
- **Odkaz na domovskej obrazovke** – PWA je v zariadení identifikovaná ako klasická aplikácia.
- **Natívnosť** – PWA spustená z domovskej obrazovky sa musí javiť ako natívna aplikácia.
- **Nezávislosť na pripojení** – možnosť aplikácie pracovať bez pripojenia na internet. PWA výpadok pripojenia nemôže brať ako chybu a musí sa s touto situáciou vedieť vysporiadať.
- **Aktuálnosť** – ihneď po pripojení k internetu aktualizovať obsah aplikácie.
- **Bezpečnosť** – nutnosť implementovať HTTPS a SSL certifikáty.
- **Push notifikácie** – PWA dokáže zobrazovať správy zo serveru vo forme push notifikácií [52].

Výhody tvorby progresívnej webovej aplikácie sa prekrývajú s výhodami tvorby klasickej webovej aplikácie. Je to teda hlavne jednoduchosť vývoja, rýchlosť a nižšia finančná náročnosť vývoja a taktiež jednoduchšia aktualizácia aplikácie. PWA musí oproti webovej



Obr. 3.5: Schéma hybridnej mobilnej aplikácie. Obrázok inšpirovaný z [7].

aplikácií navyše čeliť určitým výzvam. Prvou nevýhodou je podporovanie PWA naprieč rôznymi prehliadačmi, kedy niektoré plne podporujú PWA, iné čiastočne a niektoré zatiaľ vôbec. Nevýhodu oproti natívnej aplikácií je opäť nemožnosť využitia všetkých vlastnosti a možností zariadenia, na ktorom PWA beží [52].

3.3 Hybridné mobilné aplikácie

Hybridný vývoj, ako už názov napovedá, kombinuje predchádzajúce dva prístupy a snaží sa tým priniesť to najlepšie z oboch svetov. Hybridná mobilná aplikácia sa vyvíja rovnako ako webová aplikácia, teda s použitím jazykov HTML, CSS a JavaScript. Následne aplikácia takéhoto typu beží v natívnom kontajneri mobilného zariadenia. Väčšina hybridných aplikácií používa nástroje na zobrazenie webového obsahu (*UIWebView* na iOS a *WebView* na platforme Android), pomocou ktorých je prezentované HTML a JavaScriptové súbory v režime na celú obrazovku, rovnako ako klasické natívne aplikácie [49]. Výnimku tvoria aplikácie vytvorené pomocou frameworku React Native, ktorý si bližšie predstavíme nižšie. Na rozdiel od webovej aplikácie má hybridná prístup k natívnym funkciám telefónu cez špeciálnu vrstvu abstrakcie. Táto vrstva sprístupňuje natívne funkcie ako špeciálne JavaScript aplikačné rozhranie. Ďalším rozdielom medzi webovou a hybridnou aplikáciou je, že hybridnú aplikáciu je potrebné stiahnuť a nainštalovať, rovnako ako natívnu aplikáciu [7]. Schému takejto aplikácie môžeme vidieť na obrázku 3.5.

Výhody hybridného prístupu sa v podstate zhodujú s výhodami tvorby mobilných webových aplikácií. Vývoj hybridnej aplikácie je jednoduchší ako vývoj natívnej aplikácie, hlavne kvôli využitiu známych a rozšírených technológií a možnosti vytvoriť aplikáciu pre rôzne platformy z jedného zdrojového kódu. S týmto súvisí aj rýchlosť a cena samotného vývoja, ktorá bude nižšia ako pri natívnych aplikáciách. Existujú však aj určité výhody oproti webovej aplikácii, ako napríklad možnosť behu aplikácie aj bez pripojenia na internet alebo

distribúcia pomocou oficiálnych obchodov s aplikáciami [17]. Ako môžeme vidieť, naozaj sa jedná o most medzi natívnym a webovým vývojom a prináša nám výhody z obidvoch prístupov.

Čo sa týka nevýhod, prvá je opäť spoločná s webovou aplikáciou, teda výkon výslednej aplikácie. To je spôsobené hlavne behom výslednej aplikácie v nástroji na zobrazovanie webového obsahu, ale aj špecifikáciami jednotlivých platforiem. Druhou nevýhodou je nemožnosť pristupovať k špecifickým vlastnostiam platformy. V extrémnych prípadoch to môže spôsobiť, že pri tvorbe hybridnej aplikácie sa budeme musieť vzdať najlepšej kľúčovej vlastnosti danej platformy. Nie je teda možné tvoriť inovatívne a prispôsobené aplikácie pre každú platformu. Poslednou nevýhodou je obtiažne hľadanie chýb vo výslednej aplikácii práve kvôli tomu, že aplikácia zostavená z jedného zdrojového kódu bude bežať na viacerých platformách [34].

Teraz si predstavíme niekoľko technológií na tvorbu hybridných mobilných aplikácií. Postupne sa pozrieme na *Flutter* – open-source rozhranie na tvorbu moderných natívnych aplikácií, ďalej *Ionic*, ktorý je zameraný na tvorbu hybridných aplikácií pomocou webových technológií a nakoniec si priblížime *React Native*, ktorý je postavený na webovej knižnici React a využíva sa na tvorbu hybridných aplikácií, ktoré využívajú natívne funkcie zariadenia, na ktorom aplikácia beží.

3.3.1 Flutter

Flutter je prenositeľný UI[1] framework, vytvorený pomerne nedávno, v roku 2015 spoločnosťou Google. Tento framework slúži na tvorbu moderných natívnych reaktívnych aplikácií pre mobilné operačné systémy iOS a Android. Jedná sa o *open-source* projekt, teda celý jeho zdrojový kód je voľne prístupný na internete, v prípade Flutteru prostredníctvom služby GitHub.[2] Na vývoj sa používa programovací jazyk Dart, čo je moderný objektovo orientovaný programovací jazyk, ktorý sa kompiluje do natívneho ARM kódu. Jeho príklad je možné vidieť na ukážke kódu 3.6. Na zobrazovanie obsahu Flutter využíva vykresľovací nástroj Skia 2D, ktorý pracuje a podporuje viaceré typy hardvéru a platforiem, ako napríklad Chrome OS, Android, Firefox OS a ďalšie [42]. Pre iOS Flutter využíva vykresľovací nástroj priamo od Apple.

```
1 class Book {
2     String title;
3     Author author;
4 }
```

Výpis 3.6: Príklad jednoduchej triedy v jazyku Dart

Ako už bolo spomenuté, Flutter využíva na vytváranie používateľského rozhrania programovací jazyk Dart, čím sa odstraňuje nutnosť využitia samostatného jazyka na vizuálnu stránku aplikácie. Toto používateľské rozhranie je vytvárané tak, aby odpovedalo stavu aplikácie, inými slovami je deklaratívne. Keď sa teda stav aplikácie zmení, rozhranie sa prekreslí. Toto vykresľovanie prebieha dostatočne rýchlo, aby na podporovaných zariadeniach stíhalo vykresliť 60 snímkov za sekundu a najnovšie dokonca aj 120 snímkov za sekundu, ak to zariadenie potrebuje a podporuje. Tým je zabezpečený plynulý beh aplikácie [42].

[1] Používateľské rozhranie (*User Interface, UI*)
[2] https://github.com/flutter/flutter

3.3.2 Ionic

Ionic je open-source framework pre tvorbu hybridných mobilných aplikácií s pomocou webových technológií ako HTML, CSS alebo JavaScript, ktorý bol vytvorený v roku 2013. Ionic navyše poskytuje komponenty, vďaka ktorým je možné vytvárať mobilnú aplikáciu, ktorá dokáže využívať natívne funkcie zariadenia. Medzi tieto funkcie patria napríklad modálne okná, podpora gest, vyskakovacie okná a mnoho ďalších. Ďalšie natívne funkcie, ako napríklad prístup k polohe zariadenia alebo fotoaparátu už sú prístupné cez Apache Cordova, ktorý v tomto prípade úzko spolupracuje s Ionicom.

Framework Ionic je postavený na technológii AngularJS, ktorá je považovaná za jednu z najlepšie otestovaných a najviac používaných JavaScript technológií. Toto nám umožňuje pri vývoji aplikácií v Ionic využívať všetky známe silné stránky Angularu. V minulosti mohlo byť zložité navrhnúť architektúru hybridnej mobilnej aplikácie. To je ale možné vyriešiť pomocou technológie Angular a jej možnosti vytvoriť tzv. jednostránkovú aplikáciu (*Single Page Application, SPA*) [59].

3.3.3 React Native

React Native je JavaScriptový framework pre tvorbu skutočne natívne vykresľovaných mobilných aplikácií na platformách iOS a Android. Ako už jeho názov napovedá, je postavený na JavaScriptovej knižnici React od spoločnosti Facebook, ktorú sme si predstavili v kapitole 3.2.1. To mu dovoľuje využiť všetky výhody knižnice React, ako sú napríklad jazyk JSX alebo virtuálny DOM. Vďaka React Native je vývoj mobilných aplikácií umožnený aj webovým vývojárom, ktorí z pohodlia JavaScriptového frameworku dokážu vytvoriť skutočne natívne mobilné aplikácie [16].

Aj napriek tomu, React Native nepoužíva žiadny WebView ako iné hybridné frameworky pre hybridný vývoj mobilných aplikácií, stále ho môžeme radiť do tejto kategórie. Je to vďaka tomu, že ako alternatívu WebView využíva vykresľovanie pomocou natívnych UI komponentov. Toto mu sprístupňuje takzvaný *React Native Bridge*, ktorý vyvolá aplikačné rozhranie Object-C na platforme iOS, respektíve aplikačné rozhranie Java na platforme Android [16]. Tento prístup umožňuje frameworku React Native prístup k natívnym funkciám telefónu, ako je napríklad fotoaparát alebo aktuálna poloha zariadenia.

Výhody

Prvou veľkou výhodou oproti väčšine nástrojov pre tvorbu hybridných mobilných aplikácií je už spomínané vykresľovanie pomocou natívnych komponentov. Prekresľovací cyklus potom funguje presne rovnako, ako v knižnici React, teda po zmene stavu komponentu sa tento nanovo prekreslí. Jediný rozdiel je v tom, že knižnica React využije na prekreslenie technológie HTML a CSS, zatiaľ čo React Native využije natívne vykresľovacie komponenty. Ďalšou výhodou je pomerne vysoký výkon vytvorenej aplikácie, vďaka využívaniu samostatného vlákna pre vykresľovanie UI komponentov [16]. Ďalšou nespornou výhodou je jednoduchší a tým pádom aj rýchlejší a lacnejší vývoj. Toto je zapríčinené práve využitím jazyka JavaScript, ktorý má v porovnaní s inými jazykmi pomerne rýchlu krivku učenia. Rovnako je výhodou, že väčšina vytvoreného kódu môže byť zdieľaná pre obe cieľové platformy, teda iOS a Andriod.

Vývoj

Pre vývoj aplikácie pomocou frameworku React Native je ako prvé potrebné nainštalovať *Node.js*. Node.js je prostredie jazyka JavaScript, ktoré ale na rozdiel od klasického JavaScriptu beží na strane servera. Táto technológia je postavená na behovom prostredí od spoločnosti Google, ktoré sa volá *V8*, a rovnako ako Node.js je implementované pomocou jazykov C a C++ a zameriava sa na vysoký výkon a nízku pamäťovú náročnosť [53]. Po nainštalovaní Node.js môžeme použiť jeho správcu balíčkov (*Node Package Manager, NPM*) pre nainštalovanie samotného frameworku React Native [14]. Následne môžeme inicializovať projekt pomocou rozhrania príkazového riadku (*React Native CLI*). Potom pre spustenie projektu využijeme opäť rozhranie príkazového riadku, kde špecifikujeme, pre akú platformu chceme projekt spustiť.

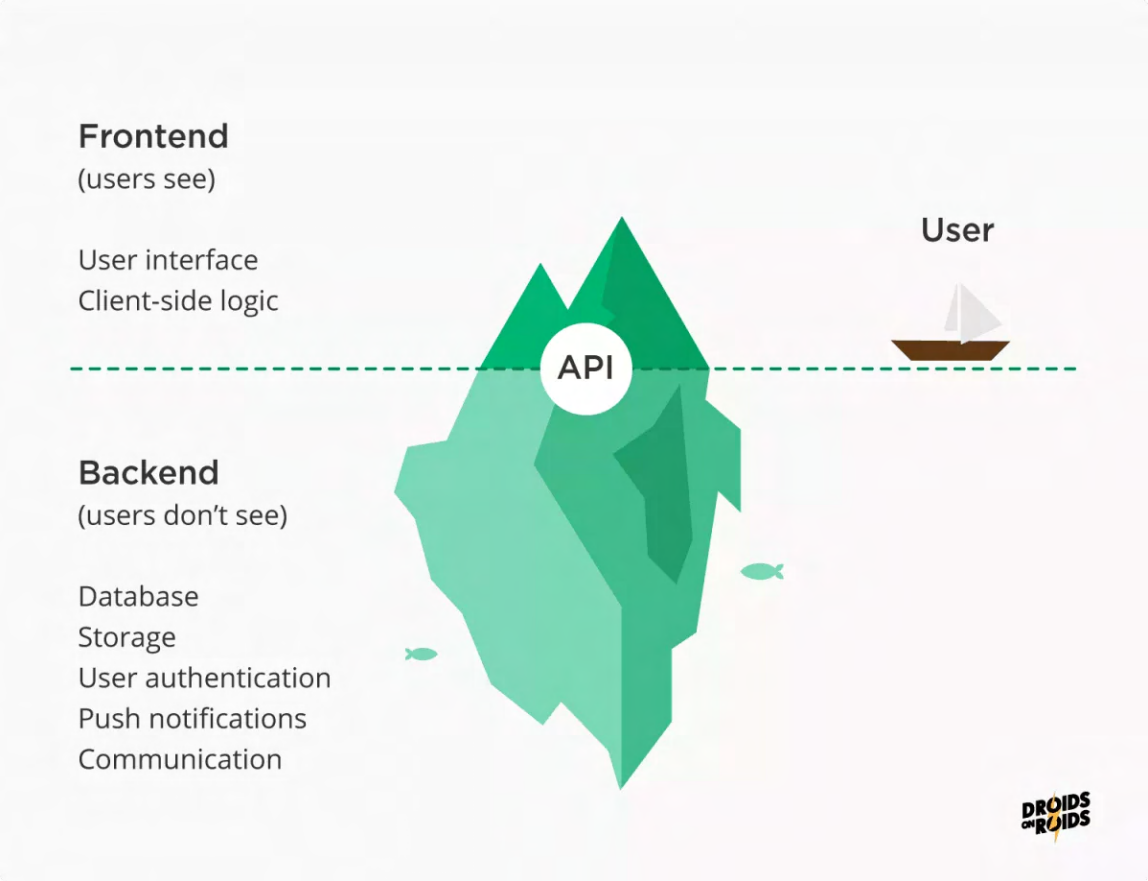
React Native rozširuje knižnicu React a ponúka množstvo zabudovaných komponentov. Tie by sme opäť mohli rozdeliť do niekoľkých kategórií. Prvou sú základné komponenty, ktoré slúžia ako úplný základ pre každú aplikáciu vytvorenú v React Native. Patria sem [19]:

- **View** – základný komponent vytvárajúci kontajner, ktorý slúži podobne ako *div* v jazyku HTML.
- **Text** – každý text, ktorý chceme zobraziť sa musí nachádzať v komponente Text
- **Image** – komponent slúžiaci na zobrazenie obrázku
- **TextInput** – komponent pomocou ktorého môžeme získať textový vstup od používateľa
- **ScrollView** – funguje podobne ako klasický View, je v ňom navyše možné rolovať
- **StyleSheet** – poskytuje vrstvu abstrakcie podobne ako CSS štýly

Ďalšou skupinou komponentov sú takzvané prvky používateľského rozhrania. Patria sem tlačidlá a prepínače. Ďalšie komponenty slúžia na zobrazovanie objektov v zoznamoch. Patria sem opäť dva komponenty, a to *FlatList* a *SectionList*. Tieto komponenty sa líšia od generickejšieho ScrollView tým, že vykresľujú iba obsah, ktorý je aktuálne viditeľný. Ďalšie dve kategórie sú každá špecifická pre jednu platformu, buď Android alebo iOS. Posledná skupina už len zoskupuje všetky komponenty, ktoré nespadali do predchádzajúcich kategórií. Patria sem prvky ako oznamovacie dialógy, *Dimensions* pre prístup k rozmerom zariadenia alebo prístup k stavovému riadku aplikácie [19].

3.4 Komunikácia mobilnej aplikácie so serverom

Ako sme už spomínali v predchádzajúcich kapitolách, všetky mobilné aplikácie, ktoré nejaký spôsobom pracujú s dátami svojich používateľov, potrebujú pre svoje fungovanie takzvaný *backend*. Backend si môžeme predstaviť ako druhú samostatnú aplikáciu, ktorá nebeží na zariadení klienta, ale na špeciálnom vzdialenom zariadení, serveri. Serverová časť aplikácie slúži práve na ukladanie, spracovanie a inú prácu s dátami používateľov. Taktiež tu prebieha autentifikácia, ale napríklad aj generovanie notifikácií. Aplikácia nachádzajúca sa na serveri nie je navrhnutá na priamu prácu s ľuďmi, ale pre komunikáciu s aplikáciou bežiacou na zariadení klienta. Pre lepšiu ilustráciu rozdelenia aplikácie na frontend a backend môže poslúžiť obrázok 3.6. Ako môžeme vidieť na tomto obrázku, na komunikáciu medzi týmito



Obr. 3.6: Obrázok ilustrujúci rozdelenie mobilnej aplikácie. Obrázok prevzatý z [60].

dvoma časťami slúži takzvané *API*, teda aplikačné rozhranie. Rovnako je možné si na obrázku všimnúť, že používateľ vidí a pracuje len s malou časťou celkovej aplikácie. Väčšia a zložitejšia časť je pred používateľom ukrytá a nachádza sa na serveroch.

Ešte pred samotnou tvorbou mobilnej aplikácie je potrebné určiť, či nami zamýšľaná aplikácia bude vyžadovať backend alebo nie. Vo väčšine prípadov bude odpoveď áno. V prípade backendu sa nemusí vždy jednať a veľkú komplexnú aplikáciu, v niektorých prípadoch môže stačiť iba jednoduchá služba. Ak nami zamýšľaná aplikácia bude vyžadovať pre svoje fungovanie pripojenie k internetu je pravdepodobné, že bude vyžadovať aj backend. Rovnako, ak aplikácia bude potrebovať prihlasovanie používateľov a ich autentifikáciu, synchronizáciu určitých dát naprieč viacerými klientskými aplikáciami, alebo je nutné ukladať dáta o používateľoch pre ich neskoršie spracovanie, je nutné tieto funkcie vykonávať na strane servera. Existujú ale aj aplikácie, ktoré na svoje fungovanie backend nepotrebujú. Spravidla sa jedná o aplikácie, ktoré dokážu po inštalácii fungovať bez pripojenia na internet. Takéto aplikácie nepožadujú od používateľov žiadne informácie o sebe, iba vykonávajú jednoduchú činnosť, ktorú je možné celú vykonať na zariadení klienta. Môže sa jednať o aplikácie ako kalkulačka, fotoaparát, určité hry alebo záznamníky [60].

3.4.1 Aplikačné rozhranie

Aplikačné rozhranie slúži ako most pre komunikáciu medzi množstvom frontendtových aplikácií bežiacich na zariadeniach klientov a jednou backendovou aplikáciou bežiacou na vzdialenom serveri. API v sebe zahŕňa množstvo definícií a protokolov, pomocou ktorých dokáže túto komunikáciu zrealizovať. Toto rozhranie funguje na princípe, kedy mobilná aplikácia posiela žiadosti na získanie alebo modifikáciu dát na vopred špecifikované koncové body, na ktorých už čaká backendová aplikácia, ktorá tieto žiadosti zachytí a následne spracuje. V poslednej dobe stavajú moderné aplikačné rozhrania na 4 princípoch, ktoré ich robia veľmi cennými a užitočnými. Medzi tieto princípy patria [41]:

- **Využívanie štandardov** – rozhranie je postavené na štandardoch, ktoré sú verejne známe, dobre chápané a dostupné.
- **Viac produkt ako kód** – samotné rozhranie je v poslednej dobe chápané viac ako samostatný produkt, než len časť kódu. Často je tento produkt určený pre špecifickú skupinu používateľov.
- **Silná bezpečnosť a riadenie** – vďaka vysokej miere štandardizácie dokážu dodržať silnú bezpečnosť, úroveň riadenia ale aj vysokú úroveň dokumentácie.
- **Životný cyklus** – rovnako ako každý softvérový produkt, aj moderné aplikačné rozhranie má svoj vlastný životný cyklus. Tento cyklus pozostáva z návrhu, testovania, zostavovania, riadenia a verzovania.

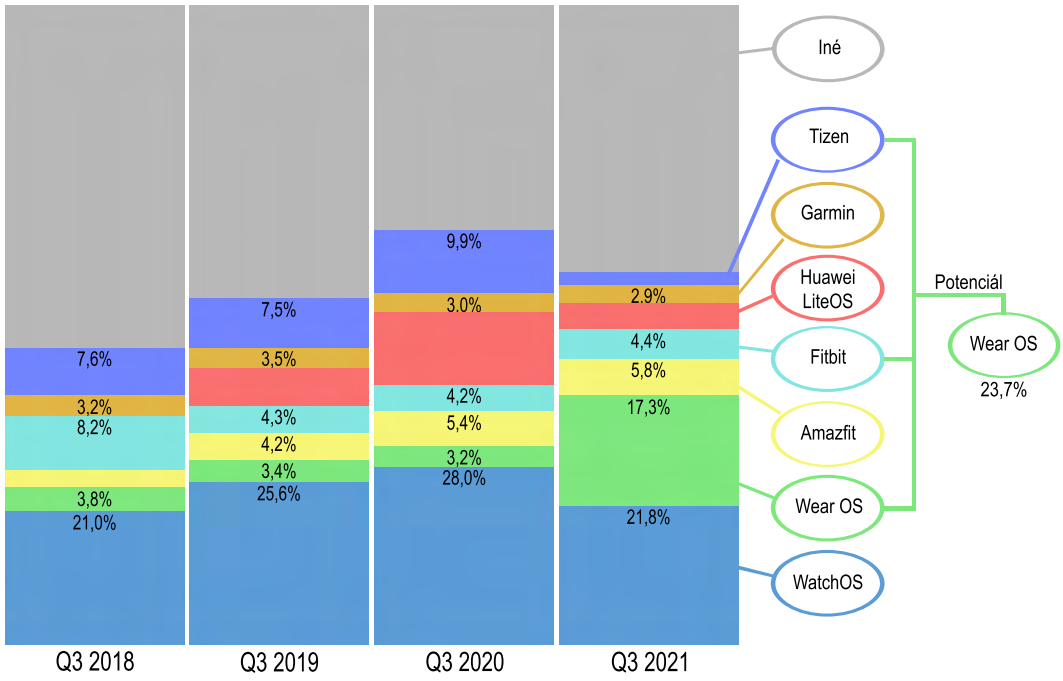
REST API

Jednou z najznámejších a najpoužívanejších architektúr aplikačného rozhrania je REST (*REpresentational State Transfer*) [57]. Ako každá iná architektúra obsahuje aj REST sadu princípov a obmedzení, ktoré ho charakterizujú. Prvým princípom je jednotnosť rozhrania, ktorá by mala prinášať zjednodušenie celkovej architektúry systému. Druhým princípom je razantné oddelenie klienta a servera. To napomáha k jednoduchému samostatnému vývoju týchto dvoch častí. Ďalším kľúčovým princípom architektúry REST je bezstavovosť. To znamená, že každá požiadavka je samostatná jednotka, ktorá musí obsahovať všetky potrebné informácie a dáta pre jej splnenie. Server nemôže na splnenie aktuálnej požiadavky využívať znalosti z predchádzajúcich prijatých požiadaviek. Štvrtým princípom je možnosť ukladať výsledky dotazov do rýchlej vyrovnávacej pamäte. Tento princíp umožňuje a prikazuje odpovedi, aby bola explicitne označená ako uložiteľná alebo neuložiteľná do pamäti. Ďalším princípom je vrstvený systém. To znamená, že klient by mal vidieť iba najbližšiu vrstvu, s ktorou priamo komunikuje. Posledným princípom, ktorý je už ale označený ako voliteľný je kód na vyžiadanie (*Code on Demand)* [61]. Tento princíp hovorí, že server môže rozšíriť funkcionalitu klienta za behu tým, že mu odošle časť kódu, ktorý klient môže následne vykonať. Tento kód môže byť napríklad vo forme Java Appletov alebo JavaScriptu.

Kapitola 4

Inteligentné hodinky

Táto kapitola sa zameriava na tvorbu aplikácií pre inteligentné hodinky. Budú predstavené dva najpoužívanejšie operačné systémy na trhu s inteligentnými hodinkami podľa prieskumu spoločnosti Counterpoint [36]. Najskôr si priblížime samotné systémy a potom sa pozrieme na detaily vývoja pre danú platformu. Výsledky spomínaného prieskumu je možné vidieť na obrázku 4.1.



Obr. 4.1: Výsledok prieskumu spoločnosti Counterpoint týkajúci sa vývoja podielov jednotlivých operačných systémov inteligentných hodiniek na trhu. Obrázok inšpirovaný z [36].

4.1 Apple WatchOS

Podľa spomínaného prieskumu sa na prvej priečke najpoužívanejšieho operačného systému nachádza WatchOS od spoločnosti Apple. Tento operačný systém má stále takmer štvrtinové zastúpenie trhu, aj keď oproti minulému roku stratil takmer 7% aj napriek tomu, že celkový počet zákazníkov používajúcich tento operačný systém sa zvýšil.

4.1.1 Systém

Apple WatchOS bol predstavený v apríli 2015 a je postavený na operačnom systéme iOS, ktorý sa nachádza na zariadeniach od spoločnosti Apple ako napríklad iPhone. Tieto dva operačné systémy sú si veľmi podobné a ponúkajú viacero spoločných funkcií. Rozdielom je len zameranie systému, kde WatchOS je špeciálne prispôsobený na používanie na hodinkách, ako poskytovanými funkciami (napríklad zdravotnými), tak ovládaním [13]. Od roku 2015 je tento operačný systém pravidelne aktualizovaný a sú pridávané nové zaujímavé funkcie. V súčasnosti je aktuálnou už ôsma verzia systému, ktorá napríklad priniesla podporu ovládania inteligentnej domácnosti alebo novú aplikáciu pre fotografie [3].

4.1.2 Vývoj

Rovnako ako pre vývoj aplikácie na zariadenia so systémom iOS, aj pre vývoj aplikácie na WatchOS je potrebné zariadenie od Apple, ako napríklad Mac. Bez počítača od Apple nie je možné výslednú aplikáciu preložiť a zostaviť. Celý takýto projekt je možné rozdeliť na tri časti [13]:

- Watch extension – obsahuje kód a zdroje pre aplikáciu pre hodinky. Tento kód a zdroje v prvej verzii operačného systému bežali v pripojenom zariadení iPhone. V súčasnosti sa Watch extension nachádza priamo v hodinkách spolu s Watch app.
- Watch app – samotná aplikácia v hodinkách. Obsahuje menšie zdroje nachádzajúce sa priamo v hodinkách a odkazuje sa na watch extension.
- iOS app – hlavná aplikácia v zariadení iPhone, ktorá zabaľuje watch app a watch extension do jednej komplexnej aplikácie.

Aplikácia pre hodinky Apple je postavená na takzvanom WatchKit. WatchKit je framework obsahujúci množstvo potrebných tried pre podporu funkcionality. WatchKit sa automaticky postará o komunikáciu medzi spárovaným a pripojeným zariadením iPhone a samotnými hodinkami. Celá táto komunikácia prebieha cez rozhranie Bluetooth [13].

V prvej verzii tohto operačného systému aplikácia pre hodinky Apple fungovala ako používateľské rozhranie na zobrazovanie údajov a zachytávanie používateľských vstupov. Na to, aby sme vedeli vytvoriť aplikáciu pre Apple WatchOS 1, museli by sme najskôr vytvoriť aplikáciu pre iOS, ktorá by zastrešovala a obsahovala aplikáciu pre hodinky. Samotná aplikácia bola teda takmer nepoužiteľná bez WatchKit extension, ktorý bežal v pripojenom iPhone. Kedykoľvek používateľ spustil aplikáciu na Apple Watch, pripojené zariadenie iPhone spustilo na pozadí WatchKit extension pre túto aplikáciu. Každá interakcia používateľa s aplikáciou na hodinkách bola presmerovaná na WatchKit extension, ktorý ju spracoval [13].

V novších verziách WatchOS sa WatchKit extension presunul z pripojeného zariadenia iPhone do samotných hodiniek. Toto bolo umožnené vďaka neustálemu vývoju a zvyšovaniu

výkonu samotných hodiniek. Tento presun umožňuje omnoho rýchlejšie načítanie aplikácie a taktiež nám poskytuje plynulejšie používateľské rozhranie [13].

WatchKit aplikácia sa skladá z dvoch samostatných častí. Prvou sú statické súbory, ktoré sú potrebné pre správne zobrazovanie používateľského rozhrania na hodinkách. Druhou časťou je takzvaný storyboard, ktorý v podstate nie je nič viac ako vizuálna reprezentácia aplikácie. Storyboard sa využíva na návrh aplikácie pridávaním rôznych pohľadov a ich ovládačov [13].

Na vývoj sa používa rovnaký nástroj od firmy Apple ako na vývoj aplikácie pre iOS – XCode. Ako sme už popísali vyššie, vývoj aplikácie pre hodinky veľmi často prebieha súčasne s vývojom mobilnej aplikácie a v mnohých prípadoch iba rozširuje jej funkcionalitu, aj keď v novších verziách WatchOS je možné vytvoriť samostatnú aplikáciu pre hodinky. Rovnako ako v prípade vývoja pre iOS sa používa jazyk Swift, ktorý bol popísaný v kapitole 3.1.2.

4.2 Google Wear OS

Na druhej priečke v prieskume využívaných operačných systémov v inteligentných hodinkách [36] sa umiestnil operačný systém Wear OS od spoločnosti Google. Tento operačný systém zaznamenal najväčší medziročný nárast, kedy jeho využitie stúplo o takmer 15%. Zaujímavým je taktiež potenciál tohto operačného systému. Keďže sa jedná o systém, ktorý môžu používať viacerí výrobcovia, je potenciál jeho využiteľnosti na úrovni až takmer 24%. Práve tieto vlastnosti robia Wear OS najzaujímavejším operačným systémom z hľadiska vývoja aplikácie pre inteligentné hodinky v súčasnosti.

4.2.1 Systém

Základom operačného systému Wear OS je upravený operačný systém Android, ktorý je určený pre mobilné zariadenia. Wear OS tento systém rozširuje a prináša určite špecifiká využitia v inteligentných hodinkách [15]. Tento systém bol prvýkrát oznámený v marci 2014 pod starším názvom Android Wear. V roku 2018 bol názov systému zmenený na aktuálne Wear OS [58].

Hlavný rozdiel oproti vyššie popísanému WatchOS od Apple je práve v množstve podporovaných zariadení. Kým WatchOS je nasadený len vo vlastných hodinkách firmy Apple s obdĺžnikovým displejom, Wear OS môže byť a je využívaný v zariadeniach s okrúhlym, štvorcovým alebo obdĺžnikovým displejom. To prináša pri vývoji ďalšie výzvy, ktoré je nutné vyriešiť. Ďalšou kľúčovou vlastnosťou tohto systému je možnosť prepojenia s mobilným zariadením s oboma najpoužívanejšími operačnými systémami, Android aj iOS [29].

4.2.2 Vývoj

Vývoj aplikácie pre Wear OS je opäť veľmi podobný vývoju klasickej mobilnej aplikácie pre systém Android. Rovnako ako novšie verzie vyššie popísaného WatchOS, aj Wear OS od Google ponúka možnosť vytvoriť samostatnú aplikáciu, ktorá bude celá fungovať priamo v hodinkách používateľa.

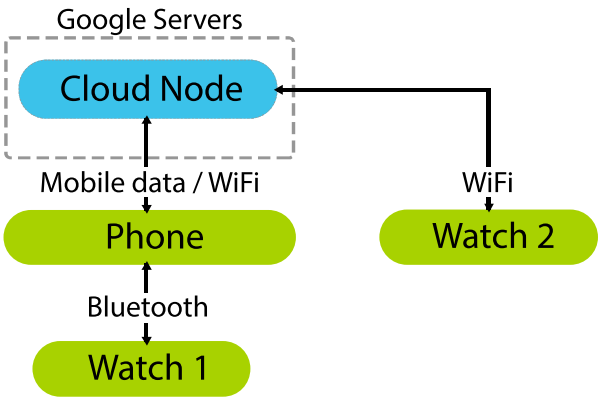
Vývoj takejto aplikácie by sme mohli rozdeliť na dve samostatné časti. Prvou by bola tvorba používateľského rozhrania, tj. rôznych pohľadov a obrazoviek, ktoré by slúžili na následnú prezentáciu dát používateľovi. Pre Wear OS je táto časť aplikácie tvorená pomocou značkovacieho jazyka XML. Takto je možné vytvoriť rôzne kontajnery, textové bloky alebo napríklad obrázky, pričom tieto sú identifikované jedinečným identifikátorom, podobne ako

tomu je v jazyku HTML. Druhá časť aplikácie sa stará o vytváranie alebo získavanie údajov, ktoré chceme používateľovi prezentovať. Tu sa vieme odkazovať na používateľské rozhranie pomocou vopred definovaných identifikátorov a prideľovať im dynamický obsah. Táto časť môže byť vyvíjaný v jazykoch Java alebo Kotlin, rovnako ako mobilná aplikácia pre systém Android. Oba tieto jazyky môžete vidieť popísané podrobnejšie v kapitole 3.1.1. Pre správne fungovanie akejkoľvek aplikácie pre Wear OS je nutné vyvinúť obe časti aplikácie [29].

Pre vývoj aplikácie pre operačný systém Wear OS je využívaný nástroj Android Studio. Tento nástroj sa taktiež využíva pre tvorbu mobilných aplikácií pre systém Android. Android Studio obsahuje viacero emulátorov, ktoré je možné využiť pri vývoji aplikácie. Vďaka tomu je možné aplikáciu otestovať na viacerých druhoch zariadení s rôznymi špecifikáciami displeja.

4.3 Komunikácia so serverom

Pre komunikáciu aplikácie nachádzajúcej sa v inteligentných hodinách so serverom platia rovnaké princípy, ako pre komunikáciu klasickej mobilnej aplikácie. Samotná komunikácia je teda typu dotaz – odpoveď. Aplikácia v takomto type komunikácie generuje dotazy na získanie dát alebo požiadavky na zmenu dát. Server spracuje daný dotaz alebo požiadavku, upraví alebo získa dáta a tieto dáta odošle naspäť do aplikácie. Pre odosielanie a prijímanie požiadaviek je ale potrebné internetové pripojenie, cez ktoré sa tieto požiadavky prenášajú. Problémom je, že nie všetky typy inteligentných hodiniek majú priamy prístup k internetu. Ak hodinky nepodporujú priamy prístup k internetu, je potrebné využiť internetové pripojenie pripojeného mobilného telefónu. Našťastie pri tvorbe aplikácií pre hodinky nemusíme rozlišovať tieto dva typy, pretože výber komunikačného kanálu prebieha automaticky. Ak je k dispozícii mobilné internetové pripojenie na pripojenom mobilnom telefóne, vždy sa prednostne používa toto pripojenie. Ak toto k dispozícii nie je, pokúsi sa zariadenie na odoslanie využiť priame pripojenie k internetu, ak je toto dostupné [26]. Keďže ako sme spomenuli, pripojenie k internetu sa rieši automaticky, programátor pri tvorbe aplikácie iba jednoducho odošle požiadavku a nemusí riešiť nižšie úrovne prenosu. Schému komunikácie je možné vidieť na obrázku 4.2.



Obr. 4.2: Schéma komunikácie aplikácie na inteligentných hodinkách so serverom. Obrázok inšpirovaný z [26].

Kapitola 5

Analýza

Táto kapitola sa zaoberá analýzou cieľovej skupiny používateľov, ich potrieb a požiadaviek. Analýza bola vypracovaná na základe osobnej skúsenosti s tématikou cyklistiky a taktiež na základe výsledkov rozhovorov s viacerými cyklistami. Následne sú v kapitole popísané aktuálne problémy, ktoré je potrebné vyriešiť a z nich plynúce základné požiadavky kladené na výslednú aplikáciu.

5.1 Cieľová skupina používateľov

Ako už z názvu celej práce vyplýva, cieľovou skupinou výsledného riešenia sú hlavne cyklisti, či už športoví, alebo len tí, ktorí využívajú bicykel na dochádzanie. Táto cieľová skupina nie je nijako obmedzovaná vekom alebo pohlavím. Taktiež nechceme, aby používatelia aplikácie potrebovali špeciálne technické zručnosti na ovládanie aplikácie v mobilnom telefóne alebo v inteligentných hodinkách. Preto musí byť kladený dôraz na jednoduchosť výsledného riešenia. Rovnako bolo počas analýzy zistené, že takáto aplikácia by sa uplatnila nielen v cyklistike, ale aj v iných odvetviach športu, ako je napríklad beh, či turistika. Preto do širšej skupiny používateľov môžeme zaradiť okrem cyklistov aj športovcov vykonávajúcich exteriérové aktivity vo všeobecnosti.

5.2 Potreby používateľov

Analýza potrieb používateľov prebiehala formou osobnej skúsenosti s oblasťou cyklistiky a taktiež osobnými rozhovormi s osobami, ktoré sa cyklistike venujú. Jednalo sa ako o 12 mužov a žien vo vekovom rozmedzí od 17 do 55 rokov. Vo väčšine prípadov sa jednalo o športových cyklistov, jedna osoba z analýzy ale využíva bicykel iba na každodenné dochádzanie do zamestnania. Priemerne tieto osoby počas cyklistickej sezóny na bicykli strávia od 3 do 6 hodín týždenne. Väčšina opýtaných športovcov má skúsenosti aspoň s niektorou z aplikácií popísaných v kapitole 2.4, avšak pri niektorých z týchto aplikácií ani nevedeli, že možnosť zdieľania polohy ponúkajú.

Prvou zistenou potrebou vyplývajúcou aj zo zamerania výslednej aplikácie je možnosť zdieľať aktuálnu polohu so svojimi spolujazdcami alebo rodinou či známymi doma. Je potrebné, aby sa poloha pravidelne aktualizovala a bolo ju možné zdieľať aj z oblastí so slabším pokrytím internetom. Takisto cyklisti upozorňovali na to, že je nutné zaistiť vhodnú dobu obnovovania lokácie, ktorá bude kompromisom medzi presnosťou polohy a vybíjaním batérie príliš častým obnovovaním. Rovnako bolo zistené, že by bolo vhodné neobmedzovať

maximálny počet osôb, s ktorými je možné polohu zdieľať. Taktiež z opačnej strany by osoby sledujúce polohu cyklistov uprednostnili možnosť naraz na mape sledovať viacerých cyklistov naraz, napríklad v prípade hromadného výjazdu.

Ďalšou potrebou súvisiacou so zdieľaním polohy je možnosť výberu konkrétnej trasy, po ktorej sa cyklista plánuje vybrať. Cyklisti chcú týmto zaistiť, aby osoby sledujúce jeho pohyb mali prehľad aj o smere a pláne jazdy. Preto je nutné, aby mal používateľ možnosť buď synchronizovať trasy s inou používanou aplikáciu v oblasti cyklistiky, alebo priamo nahrať súbor s trasou.

Väčšina opýtaných cyklistov vyslovila potrebu na automatické ukladanie prejdenej trasy po jej dokončení do aplikácie Strava, čo je jedna z najpoužívanejších aplikácií v cyklistike. Táto požiadavka je pochopiteľná, pretože aplikácia Strava slúži hlavne na následnú analýzu vykonanej aktivity. Ak táto funkcionalita bude podporovaná, nebude nutné mať počas aktivity spustené viaceré aplikácie naraz.

Ďalšou veľmi dôležitou potrebou používateľov, ktorá bola už niekoľkokrát spomenutá, je aspoň základná detekcia pádu a následná notifikácia priateľov v aplikácii. Táto detekcia by mala vedieť rozlíšiť pád napríklad od jazdy na horskom bicykli v nerovnom teréne. Ak aj aplikácia detekuje pád, mala by ponúknuť používateľovi určitý čas pred odoslaním notifikácie o páde, počas ktorého bude mať používateľ možnosť túto notifikáciu zrušiť. Ďalej niektorí cyklisti navrhli, že by bolo vhodné o páde informovať nielen priateľov v aplikácii, ale aj iných používateľov aplikácie, ktorí sa aktuálne nachádzajú v blízkosti zisteného pádu.

Čo sa týka vzhľadu aplikácie, cyklisti prejavili záujem najmä o jednoduchosť cieľového riešenia, aby bolo možné aplikáciu ľahko používať aj počas aktivity. Ďalej pri sledovaní polohy iného športovca by malo byť možné vidieť ako jeho aktuálnu polohu, tak aj históriu pohybu a prípadnú naplánovanú trasu. Spolu s informáciami na mape by mali byť na obrazovke viditeľné aj základné informácie o tejto aktivite. Iné potreby týkajúce sa vzhľadu aplikácie neboli prezentované.

5.3 Súčasný stav

V súčasnosti najpoužívanejšie a najpopulárnejšie aplikácie v oblasti cyklistiky sú poväčšine zamerané na analýzu aktivity, následnú prácu s ňou a jej uloženie a zdieľanie medzi ostatnými používateľmi. Podrobnejšie sú spomínané aplikácie popísané v kapitole 2.4. Máloktorá z nich však poskytuje podporu športovcovi počas vykonávania samotnej aktivity. Ak aj obsahuje aplikácia nejakú formu zdieľania polohy, je ju nutné explicitne pred každým výjazdom zapínať a určovať, s kým chce športovec polohu zdieľať. Okrem toho je nastavenie zdieľania polohy neraz ukryté hlboko v nastaveniach a nie je také jednoduché, ako len zapnúť nejakú funkciu. Navyše v žiadnej zo spomínaných aplikácií nie je možné naraz na jednej obrazovke sledovať polohu viacerých ako len jedného cyklistu alebo športovca. Ďalej môže byť problémom komplikovanosť aktuálnych riešení aplikácií. V mnohých prípadoch obsahujú množstvo funkcií, o ktoré cyklista, ktorý nechce následne analyzovať alebo zdieľať svoje jazdy, nemusí mať záujem.

Najväčším vnímaným problémom súčasných aplikácií je absencia akejkoľvek formy detekcie pádu. Je síce možné na toto využívať inú, samostatnú aplikáciu, ale tá zase nebude poskytovať integráciu do cyklistickej aplikácie, takže napríklad nebude možné upozorniť na zistený pád priateľov cyklistu. Rovnaký problém je vo využívaní automatickej detekcie pádu, ktorá sa nachádza vo viacerých inteligentných hodinkách. Je síce možné poslať upozornenie na pád vopred vybraným ľuďom, tí sa ale väčšinou nenachádzajú v blízkosti športovca a preto mu nemajú ako pomôcť.

5.4 Požiadavky

Z vyššie popísanej analýzy potrieb cyklistov, súčasného stavu a problémov existujúcich riešení vyplýva na nami tvorenú aplikáciu niekoľko základných požiadaviek, ktoré je potrebné vyriešiť.

1. Prvou požiadavkou je zjednodušenie samotného procesu zdieľania aktuálnej polohy športovca. Cieľom našej aplikácie teda bude vytvoriť čo najjednoduchšiu aplikáciu, ktorá bude poskytovať možnosť zdieľať, alebo na druhej strane sledovať polohu športovcov. Rovnako bude cieľom implementovať možnosť sledovania viacerých športovcov naraz, čo môže byť v niektorých situáciách, ako napríklad hromadný výjazd, požadované.

2. Druhou ešte dôležitejšou požiadavkou je zapracovanie istej formy detekcie pádu do vytvorenej aplikácie. Podpora detekcie pádu priamo v aplikácii prinesie viaceré výhody, ako napríklad možnosť v prípade pádu notifikovať nielen priateľov cyklistu, ale aj iných používateľov aplikácie nachádzajúcich sa v blízkosti cyklistu, ktorého pád bol zistený.

3. Ďalšou požiadavkou vyplývajúcou z analýzy je jednoduchosť obsluhy takejto aplikácie. Ak by ju chcel napríklad využívať používateľ, ktorý na bicykli niekde dochádza a chce sa počas cesty cítiť bezpečnejšie, nechce pred každou cestou stráviť niekoľko minút nastavovaním aplikácie. Rovnako by malo byť možné ovládať základné prvky aplikácie počas jazdy na bicykli a preto musí byť kladený dôraz na čo najväčšiu jednoduchosť výsledného riešenia.

4. Poslednou požiadavkou, ktorá vyplýva ako z vykonanej analýzy, tak aj z povahy vytváranej aplikácie, je možné napojenie na už existujúce služby. Keďže cieľom nebude vytvoriť konkurenčnú aplikáciu pre doteraz využívané aplikácie, ale rozšíriť ich funkčnosť, bolo by vhodné umožniť používateľom prepojiť si nami tvorenú aplikáciu s doteraz využívanou službou. V takomto prípade by používateľovi stačilo využívať počas výjazdu iba nami vyvíjanú aplikáciu, z ktorej by sa po dokončení výjazdu údaje synchronizovali do inej služby, ktorá bude poskytovať následnú analýzu trasy alebo jej zdieľanie.

Kapitola 6

Návrh

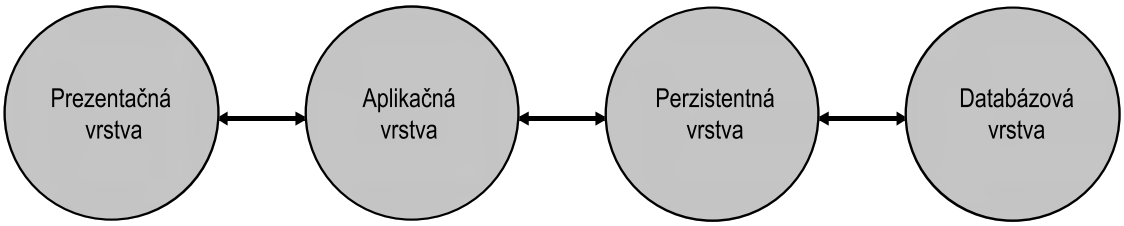
V tejto kapitole sú postupne opísané všetky vykonané návrhy potrebné pre správne fungovanie aplikácie. Prvým popísaným je návrh celkovej architektúry systému, nasleduje popis dátovej vrstvy a ukladaných dát, následne pokračujeme návrhom aplikačného rozhrania pre komunikáciu medzi mobilnou aplikáciou a serverom a posledným vytvoreným návrhom je grafický návrh aplikácie.

6.1 Návrh architektúry

Architektúru správne fungujúcej mobilnej aplikácie by sme mohli rozdeliť na tri samostatné časti. Prvou časťou je samotná mobilná aplikácia, takže *frontend*. Druhou časťou je aplikácia bežiaca na serveri, ktorá sa stará o ukladanie a spracovanie dát, takže *backend*. Poslednou časťou je zabezpečenie komunikácie medzi týmito časťami, na ktoré je využité aplikačné rozhranie.

Pre architektúru serverovej časti aplikácie som sa rozhodol využiť viacvrstvovú architektúru. Tú by sme vedeli v našom konkrétnom prípade rozdeliť na štyri samostatné vrstvy. Medzi tie patria:

- Prezentačná vrstva – najvyššia vrstva, ktorá sa stará o spracovanie HTTP požiadaviek prichádzajúcich cez aplikačné rozhranie.
- Aplikačná vrstva – vrstva obsahujúca celú aplikačnú logiku, hlavne služby vykonávajúce pokročilé spracovanie dát.
- Perzistentná vrstva – vrstva, ktorá obsahuje logiku ukladania dát. Táto vrstva reprezentuje most medzi objektmi aplikačnej logiky a samotnou databázou.
- Databázová vrstva – najnižšia vrstva architektúry, ktorú reprezentujú v našom prípade dve databázy, ktoré sú podrobnejšie popísané v kapitole 6.2.



Obr. 6.1: Schéma architektúry serverovej časti aplikácie. Obrázok inšpirovaný z [24].

Na prezentáciu dát používateľovi bude primárne slúžiť mobilná aplikácia, ktorá tvorí frontend našej aplikácie. Táto mobilná aplikácia bude pozostávať z viacerých samostatných modulov, z ktorých každý bude riešiť istú časť požiadaviek na aplikáciu. Komunikovať bude mobilná aplikácia so serverom pomocou HTTP dotazov odosielaných na aplikačné rozhranie.

Inú, menšiu časť prezentačnej logiky bude riešiť aj aplikácia pre inteligentné hodinky. Tá bude pozostávať z jednoduchého používateľského rozhrania a funkcií, ktoré budú obsluhovať prácu s týmto rozhraním.

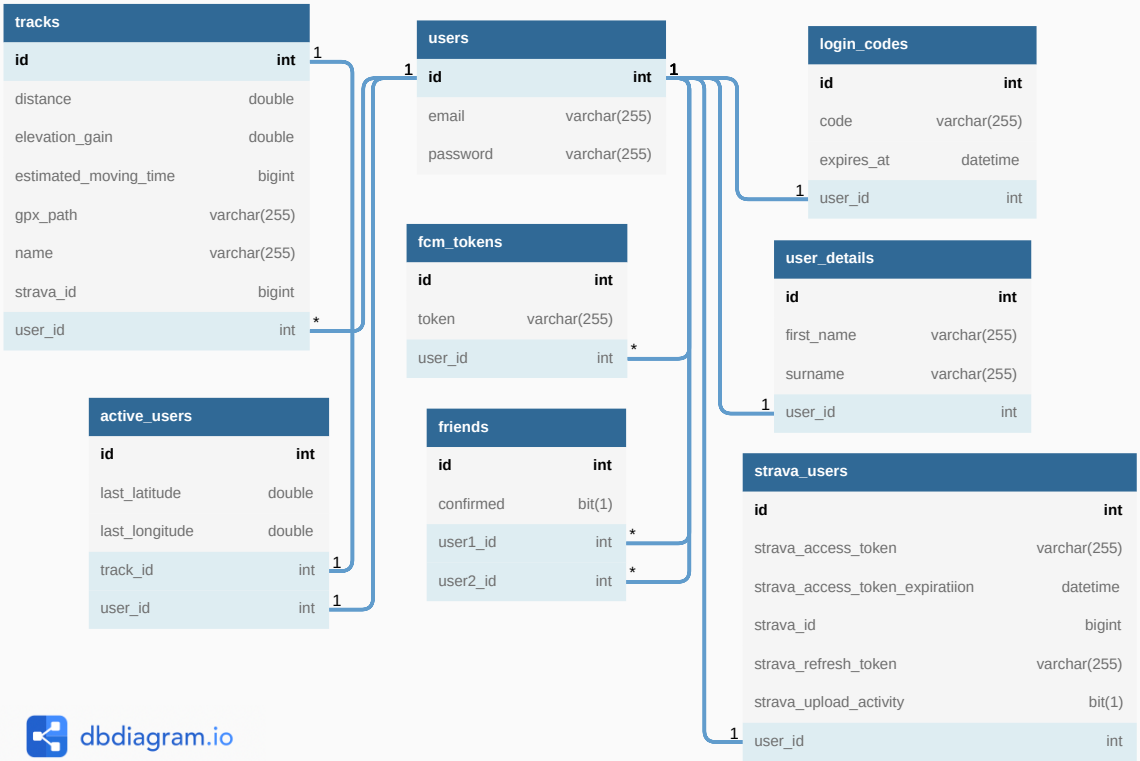
Poslednou časťou návrhu architektúry je návrh aplikačného rozhrania, ktoré bude slúžiť na komunikáciu medzi frontendom a backendom aplikácie. Pre toto rozhranie som sa rozhodol použiť jednu z najpoužívanejších architektúr aplikačných rozhraní, REST API. Podrobnejšie je navrhnuté aplikačné rozhranie popísané v kapitole 6.3.

6.2 Návrh databázy

Dátová vrstva pre našu aplikáciu sa mierne líši od klasických databáz, pretože bolo potrebné vyriešiť úlohu ukladania geografických lokácií. Z tohto dôvodu bolo potrebné rozdelenie databázy na dve samostatné časti. Prvou je klasická relačná databáza, ktorá obsahuje základné údaje o používateľoch a trasách, ktoré sa pravidelne nemenia v čase. Druhou časťou je NoSQL databáza, ktorá slúži na ukladanie reťazcov geografických lokácií. Obidve časti si teraz popíšeme podrobnejšie.

6.2.1 Relačná databáza

Ako sme už spomenuli vyššie, relačná databáza bude obsahovať všetky osobné údaje používateľov, informácie o jazdách, aktuálnej aktivite používateľov, prepojení používateľov na aplikáciu Strava, vygenerovaných prihlasovacích kódoch pre inteligentné hodinky a taktiež o FCM tokenoch slúžiacich pre identifikáciu zariadení používateľov. Na tieto účely bude slúžiť sedem vytvorených tabuliek, ktoré si teraz popíšeme.



Obr. 6.2: Schéma relačnej databázy. Databáza obsahuje tabuľky používateľov, používateľských detailov, informácií o pripojení k aplikácii Strava, aktívnych používateľov, priateľov, trás, prihlasovacích kódoch a FCM tokenov.

Tabuľka používateľov

Jedná sa o základnú tabuľku s názvom `users`, ktorá obsahuje iba 3 jednoduché atribúty. Prvým je identifikátor, druhým je email používateľa slúžiaci na prihlásenie do aplikácie a tretím atribútom je heslo. Pre vyššiu bezpečnosť sa heslo neukladá v otvorenej podobe a na jeho zašifrovanie bude použitá hašovacia funkcia Bcrypt.

Tabuľka používateľských detailov

Druhou tabuľku, ktorá obsahuje detaily používateľa, je tabuľka `user_details`. V tabuľke sa budú nachádzať údaje ako sú meno a priezvisko, ktoré sa budú zobrazovať v aplikácii ale aj identifikátor používateľa, user_id, ktorý slúži na prepojenie tabuliek používateľov a používateľských detailov.

Tabuľka detailov o aplikácii Strava

Ďalšia tabuľka slúži na ukladanie detailov o prepojení používateľa na aplikáciu Strava a nazýva sa `strava_users`. Bude obsahovať všetky potrebné prístupové tokeny a identifikátory pre správne prepojenie používateľa s touto aplikáciou.

Tabuľka aktívnych používateľov

Štvrtá tabuľka obsahuje informácie o aktuálne aktívnych používateľoch. V tejto tabuľke sa záznam vytvorí v momente, keď používateľ začne nahrávanie trasy v aplikácii. Záznam sa naopak odstráni hneď, ako používateľ ukončí zaznamenávanie trasy. Táto tabuľka sa nazýva `active_users`. Riadok tabuľky bude opäť identifikovaný pomocou automaticky generovaného identifikátora a bude obsahovať údaje o poslednej známej geografickej šírke a dĺžke. Ďalej bude riadok obsahovať atribúty slúžiace na prepojenie na iné tabuľky, konkrétne na tabuľku naplánovaných trás a tabuľku používateľov.

Tabuľka priateľov

Ďalšia tabuľka, `user_friends`, slúži na uchovávanie informácií o priateľstvách medzi používateľmi. Tabuľka bude obsahovať dva identifikátory používateľov a príznak určujúci, či je priateľstvo potvrdené.

Tabuľka trás

Šiesta tabuľka, ktorá slúži na ukladanie detailov o naplánovaných trasách, sa nazýva `tracks`. Ako vo všetkých tabuľkách, aj tu je riadok identifikovaný automaticky generovaným identifikátorom. Ďalej bude obsahovať vlastnosti naplánovanej trasy, ako jej vzdialenosť v metroch, množstvo výškových metrov, odhadovaný čas trvania, ak je tento dostupný, názov naplánovanej trasy a prípadný identifikátor tejto trasy v aplikácii Strava, ak bola trasa importovaná z nej. Ďalej bude tabuľka obsahovať stĺpec s cestou k uloženému GPX súboru, ktorý obsahuje celú postupnosť bodov určujúcich trasu.

Tabuľka prihlasovacích kódov

Táto tabuľka bude slúžiť na ukladanie vygenerovaných prihlasovacích kódov slúžiacich pre prihlásenie na inteligentných hodinkách. Tabuľka bude obsahovať štyri stĺpce. Prvým bude automaticky generovaný identifikátor, ďalej samotný kód, dátum a čas expirácie tohto kódu a cudzí kľúč odkazujúci na používateľa do tabuľky `users`.

Tabuľka FCM tokenov

Posledná tabuľka obsahuje FCM tokeny slúžiace na identifikáciu zariadení používateľov. Táto tabuľka je veľmi jednoduchá a obsahuje len tri atribúty. Prvým je opäť automaticky generovaný identifikátor, ďalej nasleduje samotný token a posledným je identifikátor používateľa, ktorému zariadenie identifikované týmto tokenom patrí.

6.2.2 NoSQL databáza

NoSQL databáza bude slúžiť na ukladanie veľkého množstva údajov, konkrétne bodov geografických lokácií. Každý takýto bod bude jednoznačne určený identifikátorom trasy a časovým razítkom získania tejto lokácie. Ďalej tento bod obsahuje zemepisnú šírku, zemepisnú dĺžku a nadmorskú výšku získanej lokácie. Body trasy pre každého používateľa budú do databázy automaticky pridávané každých 10 sekúnd počas aktívneho pohybu športovca. Ak bude zaznamenané, že sa športovec aktuálne nepohybuje, nové body trasy nebudú odosielané do momentu, kým sa znovu nepohne. Týmto sa taktiež obmedzuje veľkosť výslednej databázy. Po skončení zaznamenávania trasy sa všetky body tejto trasy vymažú z NoSQL

databázy, čo poslúži na ukladanie čo najmenšieho množstva dát. Ešte pred vymazaním sa tieto body môžu uložiť do GPX súboru v prípade, že si používateľ bude chcieť túto trasu uložiť do naplánovaných trás.

6.3 Návrh aplikačného rozhrania

Návrh aplikačného rozhrania je veľmi dôležitou súčasťou celkového návrhu. Keďže aplikačné rozhranie zabezpečuje komunikáciu medzi mobilnou aplikáciou a serverom, jeho návrh v podstate definuje finálne funkcie aplikácie. Tento návrh môžeme rozdeliť na dve časti, ktoré sú ale úzko prepojené. Prvou je návrh požiadaviek, ktoré budú meniť stav systému a upravovať alebo pridávať nové dáta do databázy. Druhou časťou je definícia dotazov, ktoré budú slúžiť na získavanie údajov z databázy a nebudú meniť jej stav.

6.3.1 Požiadavky

V aktuálne naplánovanej verzii systému sa nachádza 19 požiadaviek na pridanie, úpravu alebo odstránenie dát z databáz. Každú požiadavku si teraz popíšeme podrobnejšie.

PUT /user/start

Prvá požiadavka slúži na začatie novej, nenaplánovanej trasy. Pri zasielaní tejto požiadavky je ešte v jej tele potrebné odoslať model pozície. Ten obsahuje aktuálnu geografickú šírku, geografickú dĺžku, nadmorskú výšku a časové razítko. Táto požiadavka, rovnako ako všetky nasledujúce požiadavky týkajúce sa aktuálne prihláseného používateľa, nemusí obsahovať jeho identifikátor, ktorý je získaný automaticky z priloženého access tokenu. Po úspešnom vykonaní tejto požiadavky sa v databáze, konkrétne v tabuľke `active_users` vytvorí záznam pre odosielateľa požiadavky a taktiež sa uloží aktuálna lokácia ako prvý bod novej postupnosti polohy.

PUT /user/start/route_id

Táto požiadavka je veľmi podobná tej predchádzajúcej. Jediný rozdiel je v tom, že táto požiadavka slúži na začatie niektorej z vopred naplánovaných trás. Rovnako je v jej tele potrebné odoslať model aktuálnej lokácie. Pri vkladaní záznamu do tabuľky `active_users` sa okrem ostatných informácií uloží aj identifikátor tejto trasy. Do NoSQL databázy sa opäť uloží aktuálna lokácia používateľa z tela požiadavky ako prvý bod postupnosti.

PUT /user/end

Požiadavka odoslaná na túto adresu bude slúžiť na ukončenie aktuálne spustenej trasy bez jej následného uloženia medzi naplánované trasy. V tele tejto požiadavky sa tentokrát nemusí nachádzať žiadny model. Po úspešnom spracovaní požiadavky sa z relačnej databázy odstráni záznam o aktivite odosielateľa a z NoSQL databázy sa odstráni celá postupnosť reprezentujúca históriu polohy tohto používateľa.

POST /user/save

POST požiadavka na adresu `/user/save` je určená pre ukončenie aktuálne spustenej trasy a jej následné uloženie ako naplánovanú trasu. Táto požiadavka musí vo svojom tele obsahovať model pre uloženie trasy. Tento model bude obsahovať iba jednu položku, a to

požadovaný názov pre túto novú trasu. Vykonanie tejto požiadavky prebieha takmer rovnako ako predchádzajúca požiadavka. Najskôr sa vymaže záznam o aktivite používateľa a následne sa prejdená trasa uloží do GPX súboru a záznam o trase do tabuľky `tracks`. Na konci sa rovnako ako pri predošlej požiadavke z NoSQL databázy odstránia všetky body pre tohto používateľa.

POST /user/location

Táto požiadavka slúži pre ukladanie aktuálnej polohy cyklistu. Požiadavka na túto adresu bude automaticky a periodicky odosielaná z mobilnej aplikácie cyklistu. Vo svojom tele musí obsahovať rovnaký model aktuálnej polohy ako v prípade začiatku jazdy, teda model obsahujúci zemepisnú šírku, zemepisnú dĺžku, nadmorskú výšku a časové razítko. Pri spracovaní tejto požiadavky sa táto lokácia uloží na koniec postupnosti v NoSQL databáze a rovnako sa táto lokácia uloží do tabuľky `active_users` ako posledná známa poloha.

POST /user/location/cached

Veľmi podobná požiadavka tej predchádzajúcej s tým rozdielom, že nebude slúžiť na odoslanie jednej aktuálnej polohy používateľa, ale zoznam viacerých lokácií, ktoré sa uložili v zariadení počas doby, kým nebol dostupný internet. Ostatné spracovanie tejto požiadavky bude prebiehať rovnako ako ukladanie jednej polohy.

POST /user/profile

Požiadavka na zmenu údajov v používateľskom profile aktuálne prihláseného používateľa, ktorý požiadavku odoslal. V tele tejto požiadavky sa musí nachádzať model profilu, ktorý obsahuje meno, priezvisko a prípadné informácie o pripojení používateľa k aplikácii Strava. Pri spracovaní sa tieto informácie uložia do tabuliek `users`, prípadne `strava_users`.

POST /user/fall

Jedna z najdôležitejších požiadaviek v celom systéme. Bude slúžiť na zaslanie informácie z mobilnej aplikácie cyklistu v prípade, že bol automaticky detekovaný pád. Táto požiadavka vo svojom tele obsahuje rovnaký model, ako požiadavka na aktualizáciu poslednej známej polohy. Spracovanie tejto požiadavky začne získaním FCM tokenov všetkých priateľov cyklistu, ktorého pád bol detekovaný. Následne budú získané FCM tokeny používateľov v okolí detekovaného pádu. Po získaní týchto tokenov sa vygeneruje nová notifikácia, ktorá bude odoslaná do aplikácií na základe získaných FCM tokenov. Notifikácia bude obsahovať práve polohu zisteného pádu a meno a priezvisko cyklistu.

POST /user/strava

Táto požiadavka bude slúžiť na uloženie informácií o pripojení k službe Strava. V tele tejto požiadavky musí byť model, ktorý obsahuje identifikátor používateľa v aplikácii Strava, prístupový token do aplikácie Strava, platnosť tohto tokenu, refresh token a informáciu o tom, či sa majú jazdy po ich dokončení automaticky ukladať aj do aplikácie Strava.

POST /user/uploadFile

Veľmi zaujímavá požiadavka, ktorá bude slúžiť na nahratie súboru GPX s trasou do aplikácie. Ak sa podarí priložený súbor overiť a spracovať, bude tento súbor uložený a bude vytvorený záznam v tabuľke `tracks` s informáciami o tejto trase.

POST /authenticate

Jednoduchá požiadavka na prihlásenie používateľa do systému pomocou emailu a hesla. Požiadavka teda musí v tele obsahovať tieto dva údaje. Ako odpoveď na úspešné prihlásenie bude naspäť odoslaný access token slúžiaci na autorizáciu všetkých ostatných požiadaviek a dotazov.

POST /google/authenticate

Táto požiadavka bude rovnako ako predchádzajúca slúžiť na prihlásenie. Na rozdiel od predchádzajúcej požiadavky však na prihlásenie bude využitý Google účet, konkrétne získaný token, ktorý bude využitý na prihlásenie do systému. Rovnako ako v predchádzajúcom prípade bude návratom prístupový access token slúžiaci na autorizáciu ostatných požiadaviek.

POST /register

Ďalšou požiadavkou je požiadavka na pridanie nového používateľa do aplikácie. Táto požiadavka bude vo svojom tele obsahovať model registrácie. Tento model sa skladá z emailu, hesla, mena a priezviska osoby, ktorá sa pokúša zaregistrovať do aplikácie. Pri spracovaní sa vytvoria nové záznamy v tabuľkách `users` a `user_details`. Po tomto sa osoba bude schopná prihlásiť do aplikácie pomocou údajov zadaných pri registrácií.

POST /fcmtoken

Požiadavka slúžiaca na uloženie fcm tokenu zariadenia, z ktorého sa používateľ prihlási do aplikácie. Tieto tokeny slúžia pre identifikáciu samotných zariadení, ktoré sú využívané pre používanie aplikácie. Na túto adresu bude požiadavka automaticky odoslaná vždy po prihlásení do aplikácie.

POST /verifytoken

Ďalšia naplánovaná požiadavka bude slúžiť na overenie prihlasovacieho kódu odoslaného z inteligentných hodiniek. V tele požiadavky bude samotný prihlasovací kód. Ak je tento kód platný, bude používateľovi vrátený prístupový token rovnako, ako pri ostatných požiadavkách slúžiacich na prihlásenie.

POST /friend/request/send/user_id

Táto požiadavka slúži na odoslanie novej žiadosti o priateľstvo inému používateľovi. V tele požiadavky nie sú potrebné žiadne informácie. Po úspešnom spracovaní požiadavky bude vytvorené nové priateľstvo medzi odosielateľom požiadavky a používateľom s identifikátorom v URL požiadavky.

POST /friend/request/confirm/friend_id

Jednoduchá požiadavka slúžiaca na potvrdenie priateľstva s identifikátorom zadaným v URL adrese. V tele požiadavky nie sú potrené žiadne ďalšie atribúty.

POST /friend/request/reject/friend_id

Veľmi podobná požiadavka predchádzajúcej s tým rozdielom, že táto bude slúžiť na odmietnutie požiadavky na priateľstvo so zadaným id.

POST /friend/remove/friend_id

Posledná požiadavka z oblasti správy priateľov, ktorá bude slúžiť na zrušenie už existujúceho priateľstva. Po úspešnom spracovaní bude priateľstvo s identifikátorom zadaným v URL adrese zrušené.

6.3.2 Dotazy

Systém v aktuálne naplánovanej verzii bude poskytovať 9 koncových bodov na získavanie informácií z databáz. Tieto body plne pokrývajú zamýšľanú funkčnosť výsledného systému.

GET /user/profile

Prvým z dotazov je dotaz na získanie detailov o profile aktuálne prihláseného používateľa. Ako pri ostatných požiadavkách a dotazoch týkajúcich sa aktuálne prihláseného používateľa, ani tu nie je potrebné zadávať jeho identifikátor. Ten sa získa automaticky na serveri pomocou priradeného autorizačného tokenu. Tento dotaz vráti model obsahujúci meno a priezvisko používateľa, informáciu o tom, či je pripojený ku službe Strava a v prípade že je pripojený, či chce do tejto aplikácie automaticky nahrávať vykonané výjazdy.

GET /user/friends/pending

Dotaz slúžiaci na získanie všetkých žiadostí o priateľstvo týkajúcich sa aktuálne prihláseného používateľa. Po úspešnom spracovaní dotazu bude používateľovi vrátený zoznam modelov priateľov. Každý takýto model pozostáva z mena, priezviska a prípadnej informácie o poslednej známej polohe používateľa v prípade, že je aktuálne aktívny.

GET /user/friends/confirmed

Tento dotaz bude využívaný pre získanie zoznamu priateľov aktuálne prihláseného používateľa. Dotaz vráti zoznam obsahujúci modely priateľov. Tento model je rovnaký ako pri dotaze na žiadosti o priateľstvo.

GET /user/friends/active

Ďalší naplánovaný dotaz je v mnohom podobný predchádzajúcemu dotazu. Jediný rozdiel je v tom, že tento dotaz vráti zoznam len aktuálne aktívnych priateľov. Rovnako ako pri predchádzajúcich dvoch dotazoch je vrátený zoznam modelov priateľov obsahujúcich ich meno, priezvisko a poslednú známu polohu.

GET /user/find/text

Tento dotaz sa bude využívať pred vytváraním nových priateľstiev a bude slúžiť na nájdenie používateľov, ktorých meno, priezvisko alebo email obsahuje zadaný text. Po spracovaní tohto dotazu bude používateľovi vrátený opäť zoznam modelov priateľov, pretože tento model bude obsahovať všetky potrebné informácie.

GET /user/detail

Dotaz slúžiaci na získanie aktuálne spustenej jazdy pre odosielateľa dotazu. Bude sa využívať pri spustení aplikácie pre prípad, že aplikácia bude ukončená bez ukončenia zdieľania polohy. Ako odpoveď na tento doraz bude vrátený model obsahujúci meno a priezvisko používateľa, poslednú známu geografickú šírku a výšku, zoznam histórie lokácií, po ktorých sa používateľ pohyboval spolu s informáciami o tejto doteraz prejdenej trase, ako dĺžka, čas a prevýšenie a nakoniec prípadne aj zoznam geografických bodov spustenej naplánovanej trasy.

GET /user/detail/id

Takmer identický dotaz tomu predchádzajúcemu s tým rozdielom, že nebude vrátený detail aktuálne prihláseného používateľa ale používateľa s identifikátorom zadaným v URL dotazu. Model odpovede bude úplne identický predchádzajúcemu dotazu.

GET /refreshtoken

Tento dotaz bude slúžiť na získanie nového prístupového tokenu po tom, ako vyprší platnosť pôvodného prístupového tokenu. Po odoslaní dotazu na túto adresu s pôvodným expirovaným tokenom bude navrátený nový prístupový token.

GET /generatecode

Dotaz slúžiaci na vygenerovanie prihlasovacieho kódu, ktorý môže byť využitý na prihlásenie v inteligentných hodinkách. Ak už odosielateľ požiadavky má vygenerovaný prihlasovací kód, bude mu vrátený tento. Okrem samotného kódu bude používateľovi vrátený aj dátum a čas expirácie tohto kódu.

6.4 Návrh grafického používateľského rozhrania

Grafický návrh tvorí neoddeliteľnú súčasť tvorby akejkoľvek mobilnej aplikácie. Pri návrhu grafického používateľského rozhrania treba brať do úvahy nielen samotný vzhľad, ale aj rozmiestnenie prvkov na obrazovke pre jednoduchšie ovládanie. Aby sa aplikácia mohla stať populárnou a používanou, musí teda nielen dobre vyzerať, ale musí byť aj jednoduchá a intuitívna na ovládanie.

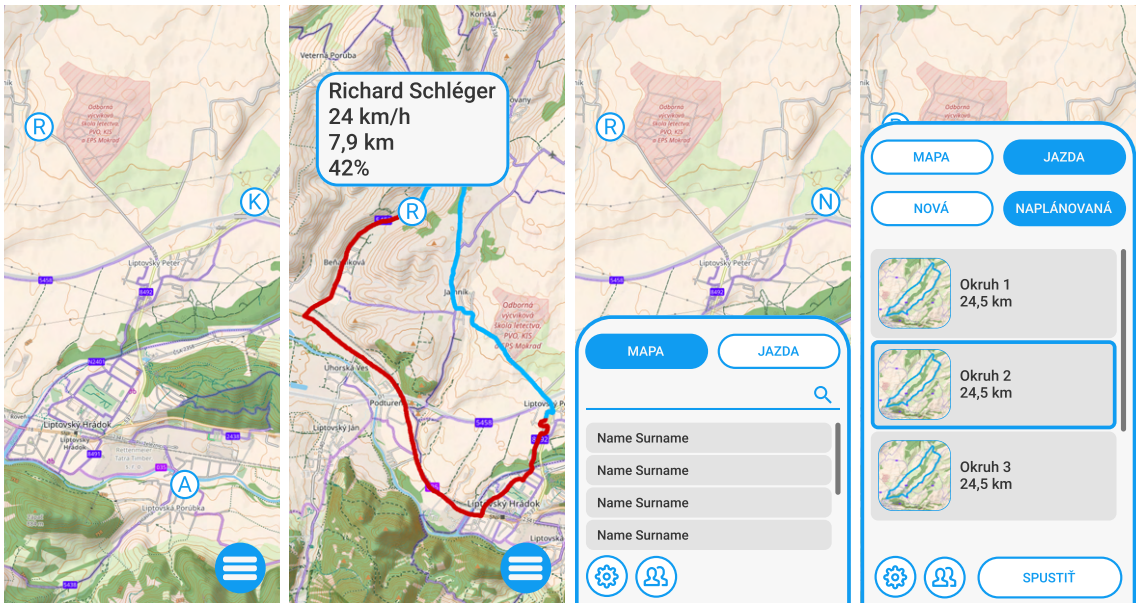
Zvlášť v oblasti športu je dôležité myslieť hlavne na ovládanie aplikácie počas aktivity. Konkrétne pri cyklistike to znamená, že aplikácia by sa mala dať ovládať jednou rukou, musí byť čo najjednoduchšia a najprehľadnejšia, aby cyklista dokázal využiť jej funkčnosť počas jazdy na bicykli. Práve z tohto dôvodu sú všetky ovládacie prvky aplikácie umiestnené v jej spodnej časti tak, aby boli čo najjednoduchšie dosiahnuteľné pri ovládaní telefónu jednou rukou. Toto rozmiestnenie je možné vidieť na obrázku 6.3.

Samotná grafická stránka aplikácie je taktiež veľmi dôležitá. Väčšinu používateľov je možné prilákať alebo stratiť práve vďaka grafickej stránke aplikácie. V súčasnosti je trendom tvorba minimalistických a jednoduchých aplikácií. Tomu je podriadený aj grafický návrh celej mobilnej aplikácie. Aplikácia sa bude skladať z dvoch základných častí. Prvou bude sledovanie iných cyklistov na mape a detailov jázd týchto cyklistov. Druhou časťou bude záznam jazdy, ktorú môžu priatelia cyklistu sledovať.

Sledovaniu aktívnych priateľov na aktuálnych výjazdoch bude venovaná hneď úvodná obrazovka aplikácie po prihlásení. Na tejto obrazovke nebudú zobrazené žiadne rušivé elementy, iba jednoduchá mapa s polohami všetkých aktívnych cyklistov medzi priateľmi. Táto mapa sa bude pravidelne obnovovať a aktualizovať polohy cyklistov. Po kliknutí na niektorého z cyklistov sa zobrazia detailnejšie informácie o aktuálnej jazde. Okrem týchto informácií sa na mape vykreslí história polohy cyklistu od začiatku výjazdu. V prípade, že sa nachádza na naplánovanej trase, sa taktiež zobrazí plánovaná trasa, po ktorej by mal cyklista pokračovať. Okrem tejto mapy sa na obrazovke bude nachádzať iba jedno tlačidlo, pomocou ktorého bude možné zobraziť jednoduché menu. V menu určenom na sledovanie polohy bude možné vyhľadávať medzi priateľmi pre jednoduchšie nájdenie konkrétneho používateľa. Z tohto menu sa bude rovnako možné dostať do nastavení, správy priateľov alebo do druhej časti aplikácie venovanej jazde.

Po prepnutí používateľa do časti „Jazda“ z menu aplikácie bude môcť používateľ vybrať, či chce spustiť novú jazdu, alebo si vybrať niektorú z naplánovaných. Pri spustení novej jazdy nie je potrebné zadávať ani vyberať žiadne informácie. Pri výbere naplánovanej trasy sa zobrazí zoznam všetkých naplánovaných trás používateľa, spolu s ich detailmi. Pri priebehu jazdy bude môcť používateľ vidieť svoju polohu na mape, históriu polohy a taktiež detailnejšie informácie o aktuálnej jazde, ako napríklad trvanie, vzdialenosť alebo priemernú rýchlosť.

Poslednými časťami aplikácie budú nastavenia a správa priateľov. V nastaveniach bude používateľ vedieť napríklad upraviť informácie vo svojom profile, pripojiť sa k aplikácii Strava, alebo nahrať novú naplánovanú trasu. Správa priateľov bude obsahovať vyhľadávanie používateľov s možnosťou zaslania žiadosti o priateľstvo, ale napríklad aj zoznam priateľov, či žiadostí o priateľstvo čakajúcich na schválenie.



Obr. 6.3: Grafický návrh aplikácie. Prvý obrázok zľava ukazuje prostredie sledovania iných cyklistov na mape. Druhý obrázok zobrazuje detail jedného konkrétneho cyklistu s informáciami o jeho jazde. Tretí obrázok ukazuje otvorenie vysúvacieho menu zo spodnej časti obrazovky s vyhľadávaním priateľov na mape. Posledný obrázok zobrazuje výber jednej z naplánovaných trás pred začiatkom zaznamenávania.

Kapitola 7

Implementácia

V nasledujúcej kapitole si popíšeme základy samotnej implementácie ako mobilnej aplikácie, tak aj backendu, ktorý je pre takýto druh aplikácie nevyhnutný.

7.1 Backend

Ako už bolo spomenuté v kapitole 3.4, backend je časť aplikácie, ktorá beží na serveri a vykonáva podpornú činnosť pre správne fungovanie či už ako v našom prípade mobilnej aplikácie, ale rovnako aj webovej alebo klasickej desktopovej aplikácie.

Pre jeho implementáciu som sa rozhodol využiť programovací jazyk Java spolu s frameworkom Spring[1]. Java je v čase písania tejto práce (apríl 2022) tretím najpopulárnejším programovacím jazykom [54] a poskytuje množstvo knižníc, ktoré uľahčujú tvorbu backendových častí aplikácií. Rovnako je výsledný systém jednoducho nasaditeľný na ľubovoľný stroj s ľubovoľným operačným systémom. Jedinou podmienkou je, aby bola v systéme nainštalovaná Java. Viac o programovacom jazyku Java je napísané v kapitole 3.1.1.

Nami implementovaný backend sa skladá z niekoľkých vrstiev, ktoré sú medzi sebou prepojené a spolupracujú. Teraz sa bližšie pozrieme na každú z týchto vrstiev a priblížime si ich fungovanie.

7.1.1 Databázová vrstva a entity

Základom celého backendu našej mobilnej aplikácie je databázová vrstva. Tá slúži na ukladanie všetkých údajov napríklad o používateľoch alebo ich jazdách a aktuálnej polohe. Nami implementovaná databázová vrstva sa skladá z dvoch samostatných databáz, z ktorých každá je vhodná a určená na prácu s iným typom dát.

Prvou databázov je relačná databáza bežiaca na databázovom serveri MySQL. Táto databáza slúži na ukladanie jednotlivých entít, ako napríklad používateľov, a je vhodná pre toto využitie práve z dôvodu, že entity majú medzi sebou viacero väzieb a tie sú reprezentované práve reláciami medzi tabuľkami v databáze. Pre správne vytvorenie databázy spolu s databázovým používateľom, ktorý bude využitý na prácu s touto databázou, bol vytvorený skript `init.sql`. Následné prvotné vytvorenie tabuliek na ukladanie dát prebehne automaticky s prvým spustením backendu podľa tried v backende, ktoré sú anotované ako entity. Toto vygenerovanie prebehne automaticky pomocou takzvaného ORM alebo objektovo-relačného mapovania. Tým sa pre každú entitu vytvorí v relačnej databáze prí-

[1] https://spring.io/

slušná tabuľka aj s potrebnými stĺpcami. Pomocou rovnakého princípu sú tiež automaticky vytvorené aj vzťahy medzi tabuľkami. Všetky nastavenia spojené s pripojením backendu k tejto databáze sa nachádzajú v súbore `application.properties`.

Druhá využitá databáza je navrhnutá pre prácu s časovými radami. Svojimi charakteristikami je ale rovnako vhodná aj pre prácu s postupnosťou geografických lokácií, ktoré spolu reprezentujú aktuálnu trasu používateľa. Konkrétne sa jedná o databázu InfluxDB. Táto databáza funguje na inom princípe ako MySQL databáza popísaná vyššie. Nie je potrebné vytvárať databázu, databázového používateľa a tabuľky, ale je nutné vytvoriť organizáciu, v rámci tejto organizácie vytvoriť používateľa a takzvaný „bucket“, do ktorého budú merania ukladané. Pre prístup k databáze sa potom nevyužíva prihlasovacie meno a heslo používateľa, ako v prípade MySQL, ale token, ktorý má každý používateľ vygenerovaný a unikátny. Ďalej meranie bude v našom prípade reprezentovať práve postupnosť GPS lokácií, kde každá postupnosť je identifikovaná pomocou id používateľa, ktorému tieto lokácie patria, a je zoradená podľa času získania jednotlivých lokácií. Všetky nastavenia pre úspešné pripojenie backendu k databáze InfluxDB sa nachádzajú v súbore `influx2.properties`.

Poslednou časťou súvisiacou s ukladaním dát používateľov je ukladanie ich naplánovaných trás. Tieto trasy si môže používateľ buď stiahnuť pomocou pripojenia k aplikácii Strava alebo manuálne nahrať súbor vo formáte GPX. Tieto trasy sú potom každá samostatne uložená vo formáte GPX, čo je formát podobný XML a je určený práve pre prácu s GPS záznamami ciest, trás a podobne. V relačnej databáze sa potom ukladá iba cesta ku GPX súboru a niektoré základné charakteristiky tejto trasy.

7.1.2 Vrstva repozitárov

Na to, aby bolo možné ukladať, meniť alebo získavať dáta z databázovej vrstvy je potrebná sada objektov, ktoré budú dotazy nad databázou vykonávať. Tieto objekty sa nazývajú repozitáre, alebo niekedy tiež DAO (Data Access Object). Pre každú entitu je tak vytvorený jeden repozitár, ktorý slúži práve na prácu s príslušnou tabuľkou v databáze. Ako sme už spomínali vyššie, pre jednoduchšiu tvorbu repozitárov, ale aj backendu celkovo, som sa rozhodol využiť framework Spring, spolu s jeho rozšíreniami ako napríklad Spring Boot. Aj vďaka využitiu tohto frameworku je väčšina objektov repozitárov veľmi podobná a je vytvorená ako rozšírenie generického repozitára `CrudRepository`, pre ktorý je nutné špecifikovať entitu a dátový typ primárneho kľúča. Ak je potrebné vytvoriť špecifickejší dotaz nad tabuľkou v databáze, framework Spring Boot je schopný vytvoriť potrebnú metódu automaticky iba na základe jej názvu v rozhraní. Každé takéto rozhranie je potom označené anotáciou `@Repository`, ktorou dávame najavo, že sa má jednať o repozitár.

Zložitejším na implementáciu bol repozitár pre prácu s geografickými lokáciami a príslušnou entitou `CurrentTrackPoint`. Keďže tento repozitár musí pracovať nad NoSQL databázou InfluxDB, bolo potrebné vytvoriť vlastnú implementáciu repozitára. Pri tejto implementácii boli využité API objekty z knižnice `com.influx.client`[2]. Pomocou týchto objektov v kombinácii s dotazovacím jazykom Flux[3] bola vytvorená trieda `TrackPointRepositoryImpl`, ktorá implementuje rozhranie `TrackPointRepository`. Pomocou našej implementácie repozitára je tak možné získať všetky body trasy pre konkrétneho používateľa, získať jeho poslednú zaznamenanú geografickú lokáciu, ale aj uložiť nový geografický bod trasy a zmazať všetky body trasy po ukončení aktivity.

[2] https://github.com/influxdata/influxdb-client-java
[3] https://www.influxdata.com/products/flux/

7.1.3 Vrstva služieb

Ďalšou vyššou vrstvou v hierarchii je vrstva služieb. Tieto služby využívajú repozitáre na prístup k databázam. Služby ďalej buď odošlú nové alebo upravené dáta do databázy, alebo získajú určité dáta, prípadne nad nimi vykonajú požadované transformácie a odošlú ich opäť na vyššiu vrstvu. Každá služba je vo frameworku Spring Boot označená pomocou anotácie `@Service`. Každá takáto služba potom môže využívať viacero repozitárov pre prístup k rôznym tabuľkám v jednej, alebo aj viacerých databázach. Repozitáre sú automaticky inicializované pomocou princípu Constructor Injection, čo nám ako vývojárom opäť uľahčí vývoj. Jednotlivé metódy v rámci služby sú potom anotované pomocou anotácie `@Transactional`, ktorá nám zaručí priebeh celej metódy ako jednej transakcie. Je vhodnejšie využiť túto anotáciu práve pri metódach v službách oproti metódam v repozitároch, a to práve z dôvodu, že jedna služba dokáže pracovať s viacerými repozitármi.

Komunikácia so službou Strava

Zaujímavou službou z hľadiska implementácie je `StravaService`, ktorá nespolupracuje so žiadnou databázou, ale komunikuje s otvoreným aplikačným rozhraním aplikácie Strava. Táto služba sa stará napríklad o aktualizácie prístupových tokenov používateľa, získanie naplánovaných trás používateľa v aplikácií Strava, nahranie novej aktivity do tejto aplikácie po ukončení jazdy alebo odpojenie.

Existujú neoficiálne implementácie klientov pre komunikáciu s otvoreným aplikačným rozhraním Strava, nie sú však udržiavané a obsahujú viacero zraniteľností. Z tohto dôvodu som sa rozhodol implementovať dotazy na aplikačné rozhranie manuálne s využitím `HttpClient` z knižnice `apache`. Každý dotaz na toto rozhranie musí byť zabezpečený pomocou prístupového tokenu, ktorý používateľ získa po pripojení k Strave. Token má potom platnosť 24 hodín a po tejto dobe musí byť prístupový token obnovený pomocou druhého tokenu, nazývaného „refresh token“. Na začiatku každého dotazu sa teda skontroluje platnosť tokenu a v prípade nutnosti sa získa nový prístupový token. Následne sa môže pristúpiť k odoslaniu samotného dotazu a prípadnému namapovaniu odpovede na nami implementované modely.

7.1.4 Kontroléry a modely

Najvyššou vrstvou v nami implementovanom backende je vrstva kontrolérov. Tie zabezpečujú komunikáciu medzi samotnou mobilnou, poprípade inou aplikáciou, a serverom. Kontrolér počúva dotazy na určitej URL adrese cez aplikačné rozhranie. Po prijatí dotazu ďalej komunikuje a spolupracuje s nižšou vrstvou služieb, pomocou ktorej modifikuje alebo získava dáta a tie posiela naspäť odosielateľovi požiadavky. Služby sú rovnako ako repozitáre inicializované automaticky pomocou Constructor Injection.

Vo frameworku Spring Boot je každý kontrolér označený anotáciou `@RestController`. Ďalšou používanou anotáciou pri kontroléroch v Spring Boote je `@RequestMapping`, pomocou ktorej vieme bližšie špecifikovať adresu URL, na ktorej bude daný kontrolér čakať na dotazy. Podobná anotácia sa potom používa aj pri jednotlivých metódach, rozdielom však je, že pomocou tejto anotácie okrem adresy špecifikujeme aj metódu požiadavky. Konkrétne sa jedná o anotácie `@GetMapping` pre metódu GET, `@PostMapping` pre metódu POST, `@PutMapping` pre metódu PUT a `@DeleteMapping` pre metódu DELETE.

V nami implementovanom backende sa aktuálne nachádzajú tri kontroléry, z ktorých jeden je určený na prácu s neautentifikovanými požiadavkami, ako prihlásenie pomocou

emailu a hesla, cez účet Google alebo registrácia nového používateľa. Druhý kontrolér sa stará o prácu s priateľstvami medzi používateľmi aplikácie. Jedná sa o obsluhu požiadaviek ako vytvorenie novej žiadosti o priateľstvo, potvrdenie alebo zrušenie tejto žiadosti alebo zrušenie celého priateľstva. Posledný kontrolér obsluhuje všetky ostatné požiadavky. Vo väčšine týchto požiadaviek sa používateľ identifikuje na základe tokenu, ktorým je požiadavka autentifikovaná. Môže sa jednať o začiatok alebo koniec trasy, odoslanie aktuálnej polohy, zobrazenie alebo úprava profilu a viacero ďalších požiadaviek. Aplikačné rozhranie je detailnejšie popísané v kapitole 6.3.

Nevyhnutnou časťou backendu pre komunikáciu cez aplikačné rozhranie je sada objektov nazývaných modely. Každý takýto model pozostáva z privátnych vlastností (*properties*). Pre prístup a nastavovanie týchto vlastností model obsahuje pre každú z nich metódy nazývané `getter` a `setter`. Poslednou časťou je viacero možných konštruktérov na vytvorenie modelu. Tieto modely potom slúžia na správne namapovanie tela požiadavky na objekt v backende alebo naopak na odoslanie požadovaných dát z backendu cez aplikačné rozhranie. V niektorých prípadoch je možné na toto použiť priamo entity, málokedy však potrebujeme všetky údaje danej entity a aj z hľadiska bezpečnosti je lepšie tieto objekty od seba oddeliť. Kontroléry tak pracujú so samostatnou sadou objektov, modelmi, tie sa potom mapujú na entity a s týmito entitami ďalej pracujú nižšie vrstvy.

7.1.5 Zabezpečenie

Celá serverová časť aplikácie je zabezpečená pomocou prístupového tokenu JWT. Tento token má opäť platnosť 24 hodín a po jeho vypršaní môže byť expirovaný token využitý na vygenerovanie nového platného prístupového tokenu. Všetky potrebné nastavenia týkajúce sa zabezpečenia, ako napríklad platnosť tokenu, kontrola používateľského emailu a hesla alebo generovanie samotného prístupového tokenu sa nachádzajú v balíčku `configs`. Následne sú pomocou kontroléru sprístupnené koncové body slúžiace na prihlásenie pomocou emailu a hesla alebo cez účet Google, získanie nového prístupového tokenu v prípade expirácie pôvodného alebo aj registráciu používateľa v prípade, že nechce využiť prihlásenie cez účet Google. Po úspešnom prihlásení je používateľovi odoslaný práve JWT prístupový token, ktorý musí od tohto momentu posielať v hlavičke každej z požiadaviek na backend.

Zaujímavou časťou zabezpečenia je práve prihlásenie do aplikácie pomocou overenia totožnosti používateľa v inej službe. V našej aplikácii je aktuálne možné využiť na prihlásenie okrem emailu a hesla prihlásenie pomocou účtu Google. V budúcnosti by bolo možné rozšíriť túto možnosť aj o iných overovateľov totožnosti. Overenie totožnosti v spoločnosti Google pracuje na princípe získania idTokenu a následného overenia tohto tokenu z backendu aplikácie. Proces získavania idTokenu je popísaný neskôr v kapitole 7.2.1. Po odoslaní idTokenu na náš backend, je potrebné tento token v spoločnosti Google overiť. Na túto činnosť som implementoval samostatnú službu nazvanú `GoogleAuthenticateService`. Tá obsahuje jedinú metódu, ktorá sa pokúsi získaný idToken overiť na strane Google. Ak sa toto podarí, používateľ je buď prihlásený do už existujúceho účtu, alebo je mu vytvorený nový účet v našej aplikácii zviazaný s jeho emailom v účte Google.

7.2 Mobilná aplikácia

Pre vývoj samotnej mobilnej aplikácie som sa rozhodol využiť cestu hybridnej mobilnej aplikácie, a to hlavne z dôvodu jednoduchšieho a rýchlejšieho vývoja celej aplikácie a možnosti využiť zdieľať jeden zdrojový kód pre aplikáciu na systém Android aj iOS. Konkrétne

som využil JavaScriptový framework React Native, ktorý nám všetky tieto veci umožňuje. Rovnako pre nás bola dôležitá možnosť mať prístup k natívnym funkciám zariadenia, ako napríklad prístup k polohe alebo k senzorom ako akcelerometer či gyroskop, čo taktiež React Native spĺňa. Tento framework je do väčších detailov popísaný v kapitole 3.3.3.

Naša mobilná aplikácia sa skladá z viacerých častí, modulov, z ktorých každý si teraz popíšeme a vysvetlíme jeho základnú funkčnosť a účel.

7.2.1 Prihlásenie a registrácia

Prihlásenie je prvá vec, ktorú používateľ uvidí, keď prvýkrát otvorí aplikáciu. Preto táto stránka musí urobiť dobrý prvý dojem, teda musí vyzerať moderne, jednoducho a prehľadne, aby neodradila potenciálneho nového používateľa. Prihlasovací formulár tvoria dva vstupy, jeden na email, jeden na heslo a následne ešte odkaz na registračný formulár v prípade, že používateľ ešte nemá vytvorený účet v našej aplikácii. Po odoslaní správnych prihlasovacích údajov bude z backendu navrátený prístupový token. Tento token si uložíme v mobilnom telefóne a používateľ je presmerovaný na hlavnú obrazovku aplikácie, slúžiacu na sledovanie polohy priateľov na mape.

Okrem prihlásenia pomocou emailu a hesla aplikácia poskytuje možnosť prihlásiť sa pomocou účtu Google. Na toto prihlásenie som využil balíček `react-native-google-signin`. Tento balíček nám umožňuje po správnej konfigurácii využiť získanie takzvaného idTokenu zo strany Google pre používateľa, ktorý sa pokúša prihlásiť do aplikácie. Po úspešnom získaní idTokenu je tento odoslaný na náš backend, kde je overený a v prípade, že overenie prebehne úspešne, bude používateľ prihlásený do už existujúceho účtu s rovnakým emailom, alebo mu bude vytvorený nový účet a rovnako ako pri klasickom prihlásení bude presmerovaný na hlavnú obrazovku so sledovaním priateľov Celú prihlasovaciu obrazovku je možné vidieť na obrázku A.1.

Registračný formulár je tiež veľmi jednoduchý a obsahuje vstupné polia pre meno, priezvisko, email a heslo. Ak používateľ zadá email, ktorý sa už v aplikácií viaže s existujúcim účtom, bude na túto skutočnosť upozornený. V prípade zadania a odoslania platných údajov bude používateľovi vytvorený účet a bude presmerovaný na prihlásenie. V aktuálnej implementácii neexistuje žiadne overenie emailovej adresy používateľa. Pred spustením aplikácie do prevádzky bude potrebné toto overenie doimplemetnovať, aby sme predišli zneužívaniu cudzích emailových adries na registráciu do aplikácie. Registračný formulár je zobrazený na obrázku A.2.

7.2.2 Sledovanie priateľov na mape

Modul sledovania aktívnych používateľov spomedzi priateľov by sme mohli označiť za základný prvok celej aplikácie. Používateľovi je na celú obrazovku zobrazená mapa, na ktorej sú vyznačené najaktuálnejšie pozície všetkých jeho aktívnych priateľov. Po kliknutí na niektorého z priateľov či už na mape, alebo v zozname aktívnych priateľov v menu, sa zobrazí detail aktuálnej jazdy tohto používateľa. V detaile je na mape možné vidieť históriu pohybu používateľa, ako aj prípadnú naplánovanú trasu. V hornej časti obrazovky je krátky súhrn informácií o tomto používateľovi a jeho jazde, ako meno a priezvisko, priemerná rýchlosť, prejdená vzdialenosť a prípadne koľko percent z naplánovanej trasy má už používateľ za sebou.

Na zobrazovanie mapy, ako aj všetkých značiek a trás som sa rozhodol použiť knižnicu `react-native-maps`. Táto knižnica využíva mapy na základe platformy, na ktorej výsledná aplikácia beží. Pre systém Android sa teda využívajú mapové podklady od spoločnosti Go-

ogle, kým na platforme iOS zase mapy od Apple. Zobrazovanie vlastných značiek, ako aj trás je potom zabezpečené pomocou komponentov tejto knižnice. Rovnako táto knižnica poskytuje užitočnú funkciu zameranú na správne priblíženie mapy. Pomocou metódy hlavného komponentu `MapView` vieme automaticky nastaviť priblíženie a polohu mapy tak, aby sa na ňu zmestili všetky aktuálne značky umiestnené na tejto mape. Túto funkciu využívame automaticky po prepnutí medzi modulmi aplikácie, ako aj po kliknutí na tlačidlo centrovať zobrazené na obrazovke.

S modulom určeným na sledovanie polohy priateľov je spojená aj jedna konkrétna časť menu, označená ako „Mapa“. Samotné menu som implementoval ako vysúvacie zo spodnej časti obrazovky. Zobraziť ho môžeme pomocou tlačidla zobrazeného na obrazovke. Na zatvorenie menu som sa rozhodol využiť natívny krok späť zariadenia, ktorý dokážeme odchytiť pomocou React hooku `useBackHandler` z knižnice `@react-native-community/hooks`. V menu spojenom so sledovaním priateľov môže používateľ vidieť zoznam všetkých aktuálne aktívnych priateľov a jednoducho v ňom vyhľadávať pomocou textového vstupu. Vyhľadávanie prebieha automaticky, teda po zadaní znaku do vstupného poľa sa zoznam prefiltruje a zobrazení zostanú iba tí priatelia, ktorých meno obsahuje zadaný text. Po kliknutí na priateľa v zozname sa opäť zobrazí jeho detail.

7.2.3 Správa priateľov

Ďalšou dôležitou časťou aplikácie je správa priateľov. V tomto module môže používateľ vidieť všetkých aktuálnych priateľov, ako aj čakajúce žiadosti o priateľstvá. Tieto dva zoznamy sú implementované v komponente `ScrollView`, aby sa v nich dalo jednoducho listovať v prípade, že bude mať používateľ viacerých priateľov. V časti žiadostí o priateľstvo sú tiež dva tlačidlá, pomocou ktorých je možné potvrdiť alebo zamietnuť priateľstvo. Po potvrdení je implementovaná možnosť priateľstvo kedykoľvek zrušiť z oboch strán.

Súčasťou tohto modulu je tiež časť určená pre pridanie nového priateľa. Obsahuje jeden textový vstup, kde môže používateľ nájsť iných používateľov aplikácie na základe ich mena, priezviska alebo emailu. Tu som už nezvolil cestu automatického vyhľadávania po zadaní znaku z dôvodu, že pri väčšom množstve používateľov by toto generovalo obrovské množstvo zbytočných požiadaviek na backend. Teda až po potvrdení vyhľadávania sa používateľovi zobrazí zoznam iných používateľov aplikácie vyhovujúci zadaným kritériám vyhľadávania a používateľ môže odoslať žiadosť o priateľstvo niektorému z nájdených používateľov. Celý tento modul sa zobrazuje ako súčasť vysúvacieho menu v spodnej časti obrazovky. Modul správy priateľov je možné vidieť na obrázku A.7.

7.2.4 Nastavenie používateľského účtu v aplikácií

V tejto časti aplikácie sa nachádzajú nastavenia používateľského účtu v aplikácii. Je tu možné napríklad upraviť meno a priezvisko používateľa, nahrať GPX súbor medzi naplánované trasy, prepojiť účet s aplikáciou Strava alebo vygenerovať prihlasovací kód pre inteligentné hodinky. Obrazovku s nastaveniami aplikácie je možné vidieť na obrázku A.8.

Nahratie nového GPX súboru

Po kliknutí na tlačidlo nahrať GPX súbor sa zobrazí natívny výber súborov zo zariadenia používateľa. Aktuálne je možné nahrať jedine súbory vo formáte GPX, iné súbory aktuálne nie sú podporované. Po výbere súboru je tento odoslaný na nami implementovaný backend, kde je tento súbor skontrolovaný, vypočíta sa dĺžka trasy v tomto súbore a prípadne pre-

výšenie a celý súbor spolu s týmito informáciami sa uloží. Používateľovi sa zobrazí text informujúci o úspešnom uložení súboru a bude môcť túto trasu vidieť v zozname naplánovaných trás pri spúšťaní jazdy.

Pripojenie k aplikácii Strava

Na prepojenie s aplikáciou Strava som využil knižnicu `react-native-app-auth`[4], ktorá slúži ako most pre prihlásenie z aplikácie vytvorenej v React Native k iným aplikáciám. Pred samotným prepojením bolo nutné v službe Strava zaregistrovať novú aplikáciu, z ktorej sa budeme chcieť dotazovať na aplikačné rozhranie Strava. Po zaregistrovaní bolo našej aplikácii pridelené klientské id a heslo, pomocou ktorých bude možné prepojiť používateľa našej aplikácie so Stravou. Aktuálne má naša aplikácia obmedzený počet dotazov na 100 požiadaviek každých 15 minút a 1000 požiadaviek denne. V prípade potreby bude možné tieto limity v budúcnosti navýšiť. Po správnom nastavení tejto časti služby Strava a aj knižnice `react-native-app-auth` je možné sa jednoducho pripojiť k aplikácii Strava. Po kliknutí na tlačidlo „Pripojiť k STRAVA“ je otvorená stránka tejto aplikácie v predvolenom prehliadači zariadenia, kde je používateľ vyzvaný, aby sa prihlásil. Po prihlásení do Stravy sa zobrazí stránka s povoleniami potrebnými pre prepojenie našej aplikácie. Tu si vie používateľ napríklad vybrať, či chce záznamy z našej aplikácie automaticky nahrávať aj do aplikácie Strava. Po výbere požadovaných povolení a potvrdení je z otvoreného aplikačného rozhrania Stravy odoslaný používateľovi prístupový token, ktorý je nutné využívať pri všetkých požiadavkách na toto aplikačné rozhranie. Rovnako je vrátený aj refresh token, pomocou ktorého sa dá získať nový prístupový token po expirácií pôvodného. Tieto tokeny sú následne odoslané z našej mobilnej aplikácie na backend, kde sú uložené do databázy a celá následná komunikácia s otvoreným aplikačným rozhraním služby Strava prebieha výhrade z backendu našej aplikácie pre vyššiu úroveň zabezpečenia. Používateľ má samozrejme možnosť kedykoľvek sa od služby Strava odpojiť.

Poslednými dvoma položkami v nastaveniach sú odhlásenie z aplikácie a vygenerovanie prihlasovacieho kódu pre hodinky. Odhlásenie je veľmi jednoduché, kedy sa len vymaže prístupový token uložený v pamäti telefónu a používateľ bude presmerovaný naspäť na prihlásenie. Vygenerovanie kódu pre hodinky prebieha na serveri, následne sa tento kód odošle do našej aplikácie a je zobrazený používateľovi. Každý vygenerovaný prihlasovací kód je unikátny a má platnosť 3 minúty. Príklad zobrazenia prihlasovacieho kódu je možné vidieť na obrázku A.9.

7.2.5 Zdieľanie aktuálnej polohy

Ďalším kľúčovým modulom mnou implementovanej aplikácie je zdieľanie aktuálnej polohy počas jazdy na bicykli, alebo počas ľubovoľnej inej aktivity. Prvým krokom pri zdieľaní polohy je výber novej alebo naplánovanej trasy pred začiatkom jazdy. Ak si používateľ vyberie novú jazdu, sprístupní sa mu tlačidlo spustiť, pomocou ktorého začne zdieľať svoju aktuálnu polohu po celú dobu, kým toto zdieľanie nevypne. Druhou možnosťou je výber niektorej z naplánovaných trás. Ak používateľ vyberie túto možnosť a spustí zdieľanie polohy, jeho priatelia budú môcť okrem aktuálnej polohy a histórie tejto polohy vidieť aj celú naplánovanú trasu, po ktorej sa používateľ chce pohybovať.

Počas zdieľania aktuálnej polohy používateľ vidí informácie o aktuálnej jazde, ako aktuálnu a priemernú rýchlosť, prejdenú vzdialenosť a trvanie. Rovnako na mape vidí svoju

[4] https://github.com/FormidableLabs/react-native-app-auth

aktuálnu polohu a históriu svojej polohy, prípadne môže vidieť svoju naplánovanú trasu, ak si niektorú vybral. To, že používateľ zdieľa svoju polohu, mu neznemožňuje využívanie iných modulov aplikácie. To znamená, že počas svojej jazdy môže používateľ prejsť do časti sledovania priateľov alebo upraviť zoznam svojich priateľov.

So zdieľaním polohy je spojené získavanie aktuálnej lokácie zo zariadenia. Keďže je predpoklad, že používateľ spustí zdieľanie polohy a odloží mobilný telefón, je nutné zabezpečiť získavanie polohy aj počas uzamknutého zariadenia alebo keď je aplikácia spustená len na pozadí. Prvotnou myšlienkou bolo na získavanie lokácie na oboch operačných systémoch využiť knižnicu `react-native-location`[5]. Nakoniec bolo možné túto knižnicu využiť iba pre získavanie lokácie zo zariadení so systémom iOS, pretože staršie verzie systému Android nepodporovali získavanie lokácie z aplikácie na pozadí. Preto bolo nutné implementovať vlastný natívny modul na získavanie lokácie zo zariadenia so systémom Android. Tento modul funguje na princípe zobrazenia trvalej notifikácie po dobu spustenia aplikácie. Táto notifikácia zabezpečuje, že aplikácia aj napriek tomu, že beží na pozadí, môže získať aktuálnu polohu. Obnovovacia doba lokácie je nastavená na 5 sekúnd. Tento čas sa zdá ako vhodný kompromis medzi príliš častým obnovovaním lokácie s veľkou spotrebou energie a dlhým časom obnovovania s nízkou presnosťou.

7.2.6 Detekcia pádu

Posledným implementovaným modulom je detektor pádu. Práve tento modul bol najnáročnejší na implementáciu a bolo potrebné vyriešiť viacero rôznych problémov, kým celý detektor pracoval správne.

Prvou časťou, ktorú bolo treba pre detektor pádu vyriešiť, bol prístup k senzorom zariadenia. Konkrétne som sa rozhodol využiť dva senzory, a to akcelerometer a gyroskop. Poslednou časťou, ktorá sa zapája do detektora je aktuálna poloha používateľa, ktorej získavanie bolo popísané vyššie. Na prístup k senzorom som využil knižnicu `react-native-sensors`[6], pomocou ktorej vieme jednoducho monitorovať dáta zo senzorov v požadovanej frekvencii. Ako vhodnú obnovovaciu frekvenciu som určil obnovenie každých 25 milisekúnd. To nám zaručuje zachytenie všetkých výkyvov hodnôt oboch senzorov.

Druhým problémom bolo získanie parametrov rozhodovacej funkcie, ktorá bude rozhodovať, či sa jedná o pád, alebo nie. Bolo teda potrebné nájsť vhodnú dátovú sadu, ktorá bude slúžiť na natrénovanie klasifikačného modelu, ktorý by sme následne používali v aplikácii. Ja som sa rozhodol využiť dátovú sadu určenú pre vývoj detekcie pádu založenú na senzoroch zariadenia z roku 2019 [11]. Táto sada pozostáva z niekoľkých druhov pádu a takisto obsahuje viacero každodenných aktivít, ktoré slúžia práve na určenie hranice rozhodovacej funkcie detekcie pádu. Sada obsahuje pre každý pád alebo aktivitu postupnosť hodnôt z viacerých senzorov, z ktorých sú pre nás najzaujímavejšie práve akcelerometer a gyroskop.

Pre upravenie dátovej sady do tvaru, s ktorým bude pracovať klasifikátor, som implementoval v jazyku Python skript `parse_dataset.py`, ktorý prejde všetky postupnosti pádov a každodenných aktivít a vyberie z nich najvyššie hodnoty súčinu hodnôt z akcelerometra a takisto aj gyroskopu. Ku každému takémuto záznamu s maximálnymi hodnotami ešte skript pridá triedu, ktorá určuje, či záznam reprezentuje pád, alebo nie. S takto upravenou dátovou sadou bolo možné pristúpiť k samotnému trénovaniu vhodného klasifikačného modelu.

[5] https://github.com/timfpark/react-native-location
[6] https://react-native-sensors.github.io/

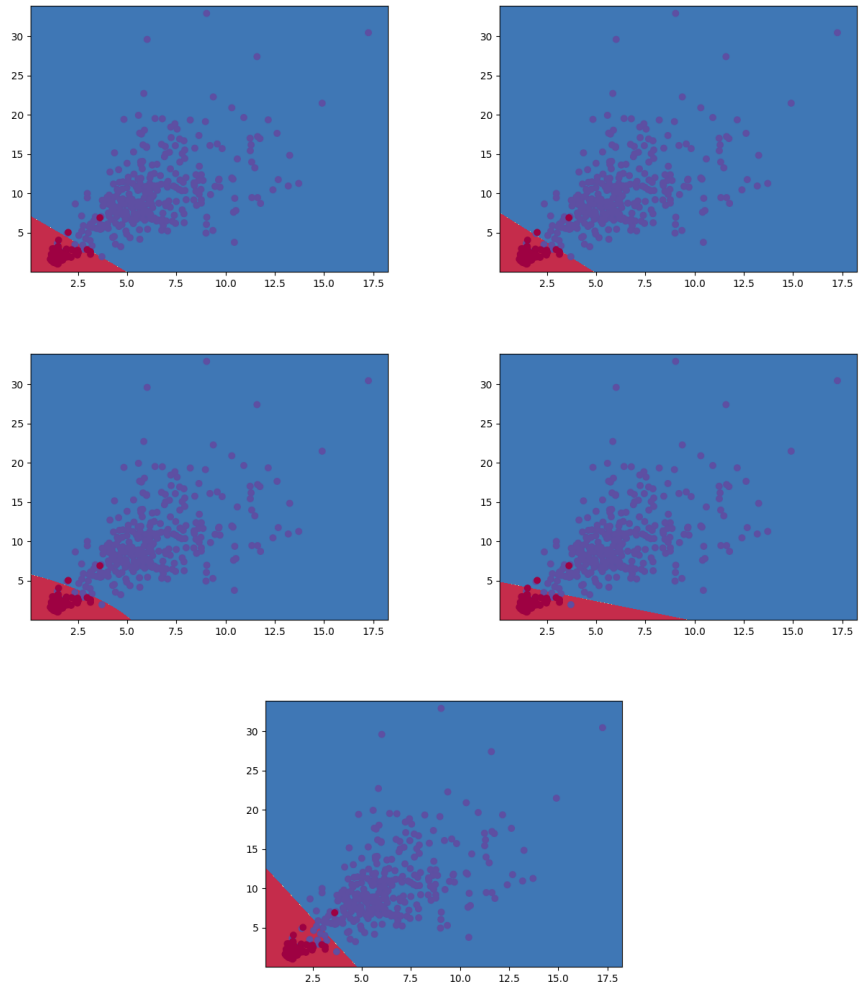
Na natrénovanie klasifikačného modelu som použil knižnicu `scikit`[7] pre jazyk Python, ktorá obsahuje množstvo rôznych klasifikačných modelov. Rozhodol som sa využiť klasifikačný model logistickej regresie, ktorého úspešnosť vyšla najlepšie spomedzi piatich vyskúšaných klasifikačných modelov. Prehľad testovaných klasifikačných modelov spolu s ich úspešnosťou nad vyššie popísanou dátovou sadou môžeme vidieť v tabuľke 7.1. Rovnako je možné vidieť vizualizáciu týchto klasifikačných modelov na obrázku 7.1. Pre natrénovanie bol vytvorený skript `train.py` v jazyku Python, pomocou ktorého bol natrénovaný model logistickej regresie nad upravenou dátovou sadou. Z tohto modelu boli získané váhy a takzvaný bias, z ktorých bola zostavená konečná rozhodovacia funkcia. Táto funkcia má tvar $y = -1.45x + 7.35$.

Model	**Úspešnosť**
Logistic regression	98.15%
Stochastic gradient descent	97.92%
Support vector machine	97.69%
Perceptron	97.45%
Ridge classifier	96.53%

Tabuľka 7.1: Zoznam klasifikačných modelov spolu s ich úspešnosťou.

Po zistení možného pádu je používateľovi zobrazené okno s informáciou o detekovaní pádu. Na tejto obrazovke má používateľ možnosť zrušiť notifikáciu priateľov o páde alebo potvrdiť, že sa naozaj jednalo o pád. Po potvrdení, že sa jednalo o pád alebo po uplynutí stanovenej doby 30 sekúnd je odoslaná požiadavka na náš backend, z ktorého sú následne rozposlané notifikácie na všetky zariadenia priateľov používateľa, ktorého pád bol detekovaný. Tieto notifikácie sú riešené pomocou nástroja *Firebase*, konkrétne `FirebaseMessaging` v mnou implementovanom backende. Táto knižnica odošle notifikácie na zariadenia priateľov na základe ich FCM tokenov. Okno s informáciou o detekovanom páde je zobrazené na obrázku A.10.

[7] https://scikit-learn.org/stable/



Obr. 7.1: Vizualizácia klasifikačných modelov. Na prvom obrázku vľavo hore môžeme vidieť *Logistic Regression*, vedľa *Stochastic gradient descent*, v druhom riadku zľava *Support vector machine* a *Perceptron* a nakoniec *Ridge classifier*. Na ose x môžeme vidieť hodnoty z akcelerometra, na ose y hodnoty z gyroskopu. Červenou farbou sú vyznačené body, ktoré reprezentujú bežné každodenné aktivity. Naopak fialovou farbou zas body reprezentujúce pády. Nakoniec červenou farbou je vyfarbená plocha, v ktorej predpokladáme, že sa nejedná o pád, kým modrou farbou je vyznačená plocha, v ktorej predpokladáme, že sa jednalo o pád.

7.3 Aplikácia pre Wear OS

Poslednou implementovanou časťou bola aplikácia pre inteligentné hodinky s operačným systémom Wear OS. Táto aplikácia zatiaľ poskytuje iba jednu základnú funkciu, a to zdieľanie aktuálnej polohy so svojimi priateľmi v rámci systému. Pred samotným zdieľaním je ale potrebné sa v hodinkách prihlásiť.

7.3.1 Prihlásenie

Keďže inteligentné hodinky majú zväčša výrazne menší displej ako mobilné telefóny, je nutné, aby pri prihlásení nebolo potrebné zadávať príliš veľa alebo príliš dlhé informácie. Z tohto dôvodu nie je možné využiť prihlásenie pomocou emailu a hesla rovnako ako v mobilnej aplikácií. Pre prihlásenie na hodinky som preto vytvoril samostatné prihlasovacie kódy, ktoré slúžia iba pre prihlásenie na hodinkách. Každý takýto kód je unikátny a má platnosť 3 minúty.

Na prihlasovacej obrazovke sa teda nachádza iba jeden vstup na číselné reťazce. Tento vstup je realizovaný pomocou `numericPassword`, ktorý zabezpečí, že je možné zadávať iba čísla a zadaný kód neostane viditeľný na obrazovke. Po kliknutí na tlačítko prihlásiť sa kód odošle na náš backend, kde je kód overený. Ak je kód správny a platný, bude používateľovi vrátený nový prístupový token, ktorý si v aplikácii uložíme pomocou `SharedPreferences` a používateľ bude presmerovaný na hlavnú obrazovku aplikácie. Následne pri opakovanom spustení aplikácie sa na začiatku vždy skontroluje, či v `SharedPreferences` existuje uložený token. Ak sa tam token nachádza, používateľ je presmerovaný priamo na hlavnú obrazovku aplikácie, v opačnom prípade sa zobrazí prihlásenie.

Zatiaľ neimplementovanou, ale v budúcnosti potrebnou funkciou, je zamedzenie náhodného skúšania prihlasovacích kódov. Bude potrebné zaistiť, aby používateľ nemohol skúšať neobmedzený počet prihlasovacích kódov, napríklad tak, že po troch neúspešných pokusoch bude musieť počkať určitú dobu, napríklad 5 minút.

7.3.2 Zdieľanie aktuálnej polohy

Zdieľanie aktuálnej polohy je hlavnou a zatiaľ jedinou funkciou aplikácie pre inteligentné hodinky. Ešte pred zdieľaním samotnej polohy je potrebné zaistiť, že má aplikácia potrebné povolenia na prístup k polohe hodiniek. Ak má aplikácia potrebné povolenia, je možné pristúpiť k zdieľaniu polohy.

Po kliknutí na tlačidlo „Začať jazdu“ je najskôr nutné odoslať požiadavku na náš backend, ktorá informuje o začiatku zdieľania polohy. Všetky požiadavky z aplikácie pre inteligentné hodinky sú odosielané pomocou `StringRequest` z knižnice `com.android.volley`. Po úspešnom odoslaní požiadavky na začiatok jazdy je možné pristúpiť k periodickému získavaniu a zdieľaniu aktuálnej polohy.

Pre periodické získavanie polohy som využil `FusedLocationProviderClient`. Pomocou tohto klienta je možné zaregistrovať si požiadavky na aktualizáciu lokácie v požadovanom intervale. Pre zaregistrovanie tejto požiadavky som vytvoril objekt `LocationRequest`, v ktorom je možné nastaviť všetky parametre požiadaviek na získavanie polohy, ako napríklad prioritu získavania polohy alebo interval získavania tejto polohy. Následne je nutné vytvoriť objekt typu `LocationCallback`, ktorý bude obsluhovať udalosť aktualizácie polohy. V tomto callbacku sa skontroluje platnosť získanej polohy a odošle sa pomocou `StringRequest` na backend. Po vytvorení oboch týchto objektov typu `LocationRequest` a `LocationCallback` je možné zaregistrovať sa k periodickému získavaniu polohy pomocou metódy `requestLocationUpdates` z objektu `FusedLocationProviderClient`.

Ukončenie odosielania polohy je už potom veľmi jednoduché. Stačí pomocou metódy `removeLocationUpdates` opäť z objektu `FusedLocationProviderClient` zrušiť získavanie polohy. Po ukončení získavania polohy je ešte nutné odoslať požiadavku na náš backend na ukončenie jazdy.

Kapitola 8

Testovanie

Testovanie je neoddeliteľnou súčasťou vývoja nielen mobilnej aplikácie. Cieľom testovania je včasné odhalenie chýb, ktoré vyplývajú buď z návrhu používateľského rozhrania alebo chýb, ktoré vznikli počas implementácie. Na to, aby mohlo prebehnúť testovanie mobilnej aplikácie, je potrebné vytvoriť inštalovateľný súbor, ktorý je následne možné distribuovať používateľom v rámci testovania. V nasledujúcich podkapitolách si popíšeme jednotlivé fázy súvisiace s testovaním.

8.1 Testovanie serverovej časti aplikácie

Prvá fáza testovania bola spojená s vývojom serverovej časti aplikácie, keďže implementácia backendu prebehla ako prvá. Toto testovanie prebiehalo pomocou sady jednoduchých HTTP dotazov, ktoré boli navrhnuté tak, aby simulovali reálne používanie aplikácie. Tieto dotazy boli odosielané na náš backend pomocou aplikácie Postman[1], ktorá slúži na vytváranie a testovanie aplikačných rozhraní. V rámci tohto testovania boli overené všetky funkčnosti serverovej aplikácie, ako prihlásenie, začatie a ukončenie jazdy, odosielanie aktuálnej polohy, ale aj správa priateľov, získavanie informácií o jazdách priateľov používateľa či odoslanie oznámenia o detekovanom páde.

Súčasťou backendu nie sú žiadne automatizované testy, keďže sa jedná iba o jednoduchú serverovú aplikáciu, ktorá nevykonáva žiadnu zložitú manipuláciu s dátami a celá požadovaná funkčnosť dokázala byť overená vyššie spomenutou sadou HTTP dotazov.

8.2 Testovanie mobilnej aplikácie

Prvotné testovanie mobilnej aplikácie prebiehalo v simulovanom prostredí pomocou emulátorov v počítači. Konkrétne boli využité ako iOS emulátor, tak aj emulátor systému Android. Oba tieto emulátory ponúkajú možnosť simulovať polohu zariadenia, čo bolo využité pri testovaní zdieľania alebo sledovania polohy používateľov. Výhodou pri využívaní emulátorov je, že nie je nutné vlastniť zariadenia s rôznymi operačnými systémami alebo rôznymi verziami jedného operačného systému. Po otestovaní aplikácie v emulátoroch oboch cieľových operačných systémov som pristúpil k reálnemu testovaniu aplikácie na reálnych zariadeniach so systémom Android.

[1] https://www.postman.com/

8.2.1 Zostavenie a distribúcia mobilnej aplikácie

Pred akýmkoľvek testovaním a distribúciou mobilnej aplikácie na reálnych zariadeniach je potrebné najskôr vytvoriť inštalovateľný súbor, ktorý si používatelia môžu nainštalovať a používať aplikáciu bez toho, aby musela byť aplikácia umiestnená v obchode. Takýto súbor má príponu *APK* (skratka pre *Andriod Package*), a sú v ňom zabalené všetky zdrojové súbory a iné zdroje potrebné pre beh aplikácie. Ešte pred vytvorením tohto súboru bolo potrebné vytvoriť si kľúč, pomocou ktorého bude aplikácie podpísaná a zabezpečená. Takýto kľúč som vytvoril pomocou Java nástroja `keytool`[2]. Počas generovania tohto kľúča je potrebné vyplniť všetky nasledujúce údaje:

- heslo pre prístup ku *keystore* súboru s kľúčom,
- meno a priezvisko,
- názov organizačnej jednotky,
- názov organizácie,
- mesto alebo lokalita,
- štát alebo provincia,
- dvoj písmenový kód krajiny,
- heslo k samotnému kľúču.

Po vyplnení všetkých týchto údajov je úspešne vygenerovaných súbor s príponou *keystore* a tento súbor musí byť nastavený ako podpisový kľúč v konfiguračnom súbore pre zostavenie Android aplikácie. Po vytvorení podpisového kľúča a pridaní tohto kľúča do konfiguračného súboru bolo možné pristúpiť k vygenerovaniu inštalačného *APK* súboru. Ten bol vytvorený pomocou nástroja `Gradle`[3], konkrétne pomocou prepínača `assembleRelease`, ktorý zostaví výsledný *APK* súbor, ktorý môže byť ďalej distribuovaný pre používateľské testovanie.

Vytvorený inštalačný *APK* súbor bol prvotne šírený pomocou odkazu na tento súbor, ktorý sa nachádzal na Google disku. Toto riešenie však neposkytovalo žiadne informácie o testovaní a taktiež neponúka vhodnú správu vydávaných verzií, ktorých je počas vývoja veľké množstvo. Preto som sa následne rozhodol využiť službu *App Distribution* vo webovom nástroji *Firebase*. Pre využitie takejto distribúcie je nutné mať založený projekt v nástroji Firebase, ktorý som už ale mal založený kvôli využitiu služby `Firebase Messaging`. Následne je nutné do služby `App Distribution` nahrať vygenerovaný podpísaný *APK* súbor. Potom zadáme zoznam emailov používateľov, ktorí budú testovať mobilnú aplikáciu. Po tomto bude používateľom zo zoznamu odoslaný email s odkazom na stiahnutie mobilnej aplikácie.

8.2.2 Používateľské testovanie

Hlavnou časťou testovania mobilnej aplikácie je práve používateľské testovanie. Takéto testovanie je postavené na princípe vytvorenia sady úloh, ktoré musia používatelia splniť. Tieto úlohy musia byť definované tak, aby simulovali základné používanie aplikácie a dokázali tak overiť funkčnosť a intuitívnosť výsledného riešenia. Tohto testovania sa zúčastnila podobná

[2] https://docs.oracle.com/javase/8/docs/technotes/tools/windows/keytool.html
[3] https://gradle.org/

skupina používateľov, aká bola opísaná v kapitole 5.1. Jedná sa o používateľov, ktorí by mohli v skutočnosti výslednú aplikáciu používať. Testovanie opäť prebiehalo osobne, kedy testovaný používateľ vykonával definované úlohy a ja som ho pri tom sledoval, poprípade som používateľovi upresnil niektoré informácie, aby bolo možné pokračovať k ďalším úlohám. Celkovo bolo vytvorených 7 testovacích úloh:

1. Otvorte aplikáciu a prihláste sa, prípadne si vytvorte účet a následne sa prihláste.
2. Upravte si meno a priezvisko v profile.
3. Pridajte si medzi priateľov používateľa „Richard Schléger.“
4. Prejdite na mapu a zobrazte detail aktívneho používateľa.
5. Spustite novú jazdu, pár sekúnd nechajte zapnuté zdieľanie polohy a ukončite jazdu.
6. Nahrajte do systému nový GPX súbor.
7. Spustite novú jazdu z naplánovanej trasy, opäť nechajte pár sekúnd spustené zdieľanie a ukončite jazdu.

Poslednou časťou testovania bolo beta testovanie v reálnom prostredí. To prebiehalo približne dva týždne a zúčastnila sa ho rovnaká skupina používateľov ako prvej časti používateľského testovania. Po tejto dobe som opäť uskutočnil osobné rozhovory s testovacími používateľmi. Toto testovanie neodhalilo žiadny veľký problém alebo chybu aplikácie, skôr sa jednalo o drobné úpravy, ktoré sú popísané vo výsledkoch testovania.

8.3 Výsledky testovania

Testovanie odhalilo niekoľko drobných nejasností, ktoré by bolo vhodné upraviť pred produkčným nasadením aplikácie. Hneď prvou nejasnosťou pozorovanou pri používateľskom testovaní bolo, že niektorý používatelia spočiatku nevedeli, ako zatvoriť vysúvacie menu. To je aktuálne implementované formou natívneho tlačidla zariadenia na krok späť z dôvodu, aby nebolo nutné zaberať plochu obrazovky ďalším tlačidlom. Táto nejasnosť by sa dala vyriešiť pridaním úvodného prevedenia používaním aplikácie pri prvom spustení aplikácie.

Druhou pripomienkou bolo pridanie statusu používateľa v zozname priateľov. Aktuálne je v tomto zozname možné vidieť značku, ak je používateľ aktívny. Bolo by dobré rozšíriť túto značku o viac rôznych ikon pre stavy používateľov.

Ďalšou dôležitou pripomienkou bolo neočakávané ukončenie aplikácie počas zdieľania polohy optimalizátorom batérie. Keďže aplikácia pomerne často pristupuje k polohe zariadenia, v niektorých telefónoch je potrebné v nastaveniach povoliť beh aplikácie na pozadí. Niektorí testovací používatelia sa vyjadrili, že na túto skutočnosť by taktiež bolo vhodné upozorniť používateľa pri prvom spustení aplikácie.

Ďalšou pripomienkou bola požiadavka na automatické centrovanie polohy používateľa počas aktívnej jazdy. V súčasnosti síce aplikácia obsahuje tlačidlo na vycentrovanie trasy na obrazovke, niektorí používatelia však poznamenali, že aplikáciu by chceli používať zapnutú v držiaku na telefón priamo na bicykli a teda by bolo dobré, aby sa aktuálna poloha počas jazdy automaticky centrovala.

Celkovo používateľské testovanie dopadlo pozitívne, keďže neboli odhalené žiadne vážne chyby alebo nežiadúce správanie aplikácie. Väčšina používateľov nemala problém splniť zadefinované úlohy testovania, čo nasvedčuje aj o dobrej intuitívnosti aplikácie a jednoduchosti jej používania.

Kapitola 9

Záver

Cieľom tejto diplomovej práce bola analýza, návrh a následná implementácia mobilnej aplikácie slúžiacej pre sledovanie a zdieľanie aktuálnej polohy cyklistov na mape. Výsledná aplikácia bola vytvorená multiplatformovo pre systémy iOS aj Android pomocou frameworku React Native.

Na začiatku bola popísaná samotná oblasť cyklistiky. Pozreli sme sa na rôzne druhy cyklistiky, špecifiká pre zdieľanie aktuálnej polohy, využívané zariadenia. Takisto sme si popísali niekoľko najpoužívanejších aplikácií v oblasti cyklistiky. Z analýzy aktuálnych aplikácií vyplynuli isté nedostatky, ktoré sme sa pokúsili vyriešiť našou výslednou aplikáciou.

V ďalšej časti boli predstavené tri rôzne prístupy pre tvorbu mobilných aplikácií. Každý prístup bol popísaný podrobnejšie, spolu so svojimi výhodami ale aj nevýhodami. Okrem tvorby mobilnej aplikácie bola predstavená aj tvorba aplikácií pre inteligentné hodinky. Tu boli prezentované najpoužívanejšie operačné systémy a následne aj tvorba aplikácií pre tieto systémy. Taktiež sme načrtli možnosti komunikácie takýchto aplikácií so serverom.

Následne bola prezentovaná vykonaná analýza, ktorá bola zameraná na cieľovú skupinu používateľov, ich potreby a súčasný stav v oblasti cyklistiky. Z potrieb používateľov vyplynulo, že aktuálne aplikácie majú isté nedostatky, ktorých vyriešenie by cyklisti uvítali. Z výsledkov analýzy následne vyplynuli požiadavky na našu aplikáciu, a to hlavne podpora detekcie pádu alebo podpora komunikácie s inými aplikáciami. Na základe týchto požiadaviek bol vytvorený návrh aplikácie, ktorý sa skladá z architektúry, návrhu databázy, aplikačného rozhrania a nakoniec grafického návrhu výslednej mobilnej aplikácie.

Výsledná mobilná aplikácie ponúka možnosti zdieľania aktuálnej polohy s neobmedzeným množstvom používateľov a takisto obsahuje detekciu pádu založenú na dátach z akcelerometra a gyroskopu kombinovaných s aktuálnou polohou. Práve tieto dve veci sa ukázali ako najväčšie nedostatky aktuálne používaných aplikácií. Okrem toho aplikácia ponúka možnosť pripojenia k službe Strava, prihlásenie pomocou účtu Google alebo nahrávanie GPX súborov do aplikácie. Popri mobilnej aplikácii bola implementovaná aj aplikácia pre inteligentné hodinky, ktorá slúži na zdieľanie aktuálnej polohy. Vykonané používateľské testovanie neodhalilo žiadne zásadné chyby vo funkčnosti aplikácie, odhalilo však niektoré možné rozšírenia aplikácie do budúcnosti.

Pred nasadením vytvorenej aplikácie bude potrebné vyriešiť niekoľko problémov týkajúcich sa hlavne zabezpečenia aplikácie. Prvou vecou je nutnosť zabezpečiť overovanie emailových adries používateľov pri registrácií. Druhým dôležitým vylepšením zabezpečenia aplikácie je zabezpečenie komunikácie medzi aplikáciami a serverom pomocou protokolu HTTPS.

Do budúcnosti je naplánovaných niekoľko rozšírení aplikácie. Prvým je možnosť o páde notifikovať nielen priateľov používateľa, ale aj iných používateľov, ktorí sa budú aktuálne nachádzať v blízkosti detekovaného pádu. Používateľ by si vedel zvoliť, či bude chcieť mať túto možnosť zapnutú alebo nie. Ďalej by bolo vhodné rozšíriť podporu komunikácie s ostatnými používanými aplikáciami v oblasti cyklistiky. Posledným plánovaným rozšírením je možnosť začať jazdu aj bez pripojenia na internet. Táto jazda by bola vytváraná lokálne a pri pripojení k internetu by bola synchronizovaná so serverom.

Literatúra

[1] Apple Inc.. *About Swift* [online]. [cit. 2021-11-25]. Dostupné z: https://www.swift.org/about/.

[2] Apple Inc.. *Find friends and share your location with Find My* [online]. Január 2021 [cit. 2021-11-12]. Dostupné z: https://support.apple.com/en-us/HT210514.

[3] Apple Inc.. *WatchOS 8* [online]. December 2021 [cit. 2022-01-25]. Dostupné z: https://www.apple.com/watchos/watchos-8/.

[4] Avola, G. a Raasch, J. *Smashing mobile Web development*. John Wiley & Sons, 2012. ISBN 978-1-118-34812-3.

[5] Banks, A. a Porcello, E. *Learning React: functional web development with React and Redux*. Ö'Reilly Media, Inc.", 2017. ISBN 978-1-491-95462-1.

[6] Barnes, R. *Apple iOS Architecture* [online]. September 2018 [cit. 2021-11-25]. Dostupné z: https://www.tutorialspoint.com/apple-ios-architecture.

[7] Bosnic, S., Papp, I. a Novak, S. The development of hybrid mobile applications with Apache Cordova. In: IEEE. *2016 24th Telecommunications Forum (TELFOR)*. 2016, s. 1–4.

[8] Byous, J. Java technology: an early history. *URL: http://java. sun. com/features/1998/05/birthday. html, Artigo pesquisado em 07 de Junho de 2002*. 1998.

[9] Charland, A. a Leroux, B. Mobile application development: web vs. native. *Communications of the ACM*. ACM New York, NY, USA. 2011, zv. 54, č. 5, s. 49–53.

[10] CodingMedic. *Introducing JSX* [online]. November 2020 [cit. 2021-12-01]. Dostupné z: https://codingmedic.wordpress.com/2020/11/10/the-virtual-dom/.

[11] Cotechini, V., Belli, A., Palma, L., Morettini, M., Burattini, L. et al. A dataset for the development and optimization of fall detection algorithms based on wearable sensors. *Data in brief*. Elsevier. 2019, zv. 23.

[12] CyclingTips. *Strava - From the beginning* [online]. Február 2012 [cit. 2021-11-2]. Dostupné z: https://cyclingtips.com/2012/02/strava-from-the-beginning.

[13] Daniel, S. F. *Apple Watch App Development*. Packt Publishing Ltd, 2016. ISBN 978-1-78588-636-2.

[14] Danielsson, W. React Native application development. *Linköpings universitet, Swedia*. 2016, zv. 10, č. 4.

[15] D'Orazio, D. *Google reveals Android Wear, an operating system for smartwatches* [online]. Marec 2014 [cit. 2022-01-27]. Dostupné z: https://www.theverge.com/2014/3/18/5522226/google-reveals-android-wear-an-operating-system-designed-for.

[16] Eisenman, B. *Learning react native: Building native mobile apps with JavaScript*. Ö'Reilly Media, Inc.", 2015. ISBN 978-1-491-92900-1.

[17] Enihe, R. a Joshua, J. *HYBRID MOBILE APPLICATION DEVELOPMENT: A BETTER ALTERNATIVE TO NATIVE*. Máj 2020. DOI: 10.11216/gsj.2020.05.39825.

[18] Facebook Inc.. *Components and Props* [online]. [cit. 2021-12-10]. Dostupné z: https://reactjs.org/docs/faq-state.html.

[19] Facebook Inc.. *Components and Props* [online]. [cit. 2021-12-09]. Dostupné z: https://reactjs.org/docs/components-and-props.html.

[20] Facebook Inc.. *Introducing JSX* [online]. [cit. 2021-11-30]. Dostupné z: https://reactjs.org/docs/introducing-jsx.html.

[21] Flauzino, M., Veríssimo, J., Terra, R., Cirilo, E., Durelli, V. H. et al. Are you still smelling it? A comparative study between Java and Kotlin language. In: *Proceedings of the VII Brazilian symposium on software components, architectures, and reuse*. 2018, s. 23–32.

[22] Fortune Business Insights. *Bicycle Market Size, Share & COVID-19 Impact Analysis, By Type, Technology, End-User and Regional Forecast 2020-2027*. Market Analysis. December 2020. Dostupné z: https://www.fortunebusinessinsights.com/bicycle-market-104524.

[23] Gackenheimer, C. a Paul, A. *Introduction to React*. Springer, 2015. ISBN 978-1-4842-1245-5.

[24] GeeksforGeeks. *Spring Boot – Architecture* [online]. [cit. 2022-05-02]. Dostupné z: https://www.geeksforgeeks.org/spring-boot-architecture/.

[25] Google. *Android Studio* [online]. [cit. 2021-11-24]. Dostupné z: https://developer.android.com/studio.

[26] Google. *Send and sync data on Wear OS* [online]. [cit. 2022-01-27]. Dostupné z: https://developer.android.com/training/wearables/data/data-layer.

[27] Google. *Share your real-time location with others* [online]. [cit. 2021-11-11]. Dostupné z: https://support.google.com/maps/answer/7326816?hl=en&co=GENIE.Platform%3DAndroid.

[28] Google. *Platform Architecture* [online]. August 2021 [cit. 2021-11-24]. Dostupné z: https://developer.android.com/guide/platform.

[29] Hameed, S. a Chida, J. *Mastering Android Wear Application Development*. Packt Publishing Ltd, 2016. ISBN 978-1-78588-172-5.

[30] Komoot. *About Komoot* [online]. [cit. 2021-11-3]. Dostupné z: https://newsroom.komoot.com/about/.

[31] Komoot. *Share your real-time location with family and friends* [online]. [cit. 2021-11-3]. Dostupné z: https://www.komoot.com/premium/live.

[32] Komoot. *The Tech That Makes The Wild Things Happen* [online]. [cit. 2021-11-3]. Dostupné z: https://www.komoot.com/features.

[33] Komoot. *Sport types on komoot* [online]. Jún 2020 [cit. 2021-11-3]. Dostupné z: https://support.komoot.com/hc/en-us/articles/360024733031-Sport-types-on-komoot.

[34] Kruger, L. *Pros & Cons Of Using Hybrid Mobile App Development In Your Business* [online]. [cit. 2021-12-06]. Dostupné z: https://modernweb.com/pros-cons-using-hybrid-mobile-app-development-business/.

[35] Lee, W.-M. *Beginning android 4 application Development*. John Wiley & Sons, 2012. ISBN 978-1-118-19954-1.

[36] Lim, S. *Wear OS Share Surges on Samsung's Highest Quarterly Smartwatch Shipments in Q3 2021* [online]. November 2021 [cit. 2022-01-25]. Dostupné z: https://www.counterpointresearch.com/wear-os-share-surges-samsungs-highest-quarterly-smartwatch-shipments-q3-2021/.

[37] Map My Ride. *App* [online]. [cit. 2021-11-3]. Dostupné z: https://www.mapmyride.com/app.

[38] Meg - Strava support. *Supported Activity Types on Strava* [online]. Apríl 2020 [cit. 2021-11-2]. Dostupné z: https://support.strava.com/hc/en-us/articles/216919407-Supported-Activity-Types-on-Strava.

[39] Meg - Strava support. *Strava Beacon* [online]. August 2021 [cit. 2021-11-3]. Dostupné z: https://support.strava.com/hc/en-us/articles/224357527-Strava-Beacon.

[40] Mehta, N. *Mobile Web Development*. Packt publishing, 2008. ISBN 978-1-847193-43-8.

[41] MuleSoft. *What is an API? (Application Programming Interface)* [online]. [cit. 2022-01-26]. Dostupné z: https://www.mulesoft.com/resources/api/what-is-an-api.

[42] Napoli, M. L. *Beginning Flutter: A Hands On Guide To App Development*. John Wiley & Sons, 2019. ISBN 978-1-119-55087-7.

[43] Oke, O., Bhalla, K., Love, D. C. a Siddiqui, S. Tracking global bicycle ownership patterns. *Journal of Transport & Health*. 2015, zv. 2, č. 4, s. 490–501. DOI: https://doi.org/10.1016/j.jth.2015.08.006. ISSN 2214-1405. Dostupné z: https://www.sciencedirect.com/science/article/pii/S2214140515006787.

[44] Putranto, B. P. D., Saptoto, R., Jakaria, O. C. a Andriyani, W. A Comparative Study of Java and Kotlin for Android Mobile Application Development. In: IEEE. *2020 3rd International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*. 2020, s. 383–388.

[45] Radke, O. *Live Tracking* [online]. Máj 2021 [cit. 2021-11-3]. Dostupné z: https://support.mapmyfitness.com/hc/en-us/articles/1500009133601-Live-Tracking.

[46] Ride With GPS. *Features* [online]. [cit. 2021-11-10]. Dostupné z: https://ridewithgps.com/features.

[47] Ride With GPS. *Live Logging* [online]. [cit. 2021-11-10]. Dostupné z: https://ridewithgps.com/help/live-logging.

[48] Rupesh. *IOS Layered Architecture* [online]. Sep 2017 [cit. 2021-11-25]. Dostupné z: https://codeingwithios.blogspot.com/2017/09/ios-layered-architecture.html.

[49] Setiabudi, D. H., Tjahyana, L. J. et al. Mobile learning application based on hybrid mobile application technology running on Android smartphone and blackberry. In: IEEE. *International Conference on ICT for Smart Society*. 2013, s. 1–5.

[50] StatCounter. *Mobile operating systems' market share worldwide from January 2012 to June 2021*. Jún 2021 [cit. 2021-11-22]. Dostupné z: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/.

[51] Strava, Inc.. *Year in sport 2020*. Data report. December 2020. Dostupné z: https://1n4rcn88bk4ziht713dla5ub-wpengine.netdna-ssl.com/wp-content/uploads/2020/12/USA_YIS_2020.pdf.

[52] Thakur, P. Evaluation and Implementation of Progressive Web Application. Metropolia Ammattikorkeakoulu. 2018.

[53] Tilkov, S. a Vinoski, S. Node. js: Using JavaScript to build high-performance network programs. *IEEE Internet Computing*. IEEE. 2010, zv. 14, č. 6, s. 80–83.

[54] TIOBE Software BV. *TIOBE Index for April 2022* [online]. [cit. 2022-04-25]. Dostupné z: https://www.tiobe.com/tiobe-index/.

[55] Tracy, K. W. Mobile application development experiences on Apple's iOS and Android OS. *Ieee Potentials*. IEEE. 2012, zv. 31, č. 4, s. 30–34.

[56] Vilček, T. a Jakopec, T. Comparative analysis of tools for development of native and hybrid mobile applications. In: IEEE. *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. 2017, s. 1516–1521. DOI: 10.23919/MIPRO.2017.7973662.

[57] WebScrapingAPI. *The 5 Most Popular API Styles and What Sets them Apart* [online]. August 2021 [cit. 2022-01-25]. Dostupné z: https://www.webscrapingapi.com/the-5-most-popular-api-styles-and-what-sets-them-apart/.

[58] Welch, C. *Google just changed the name of Android Wear to Wear OS* [online]. Marec 2018 [cit. 2022-01-27]. Dostupné z: https://www.theverge.com/2018/3/15/17124448/google-wear-os-announced-android-wear-rebranding-smartwatch.

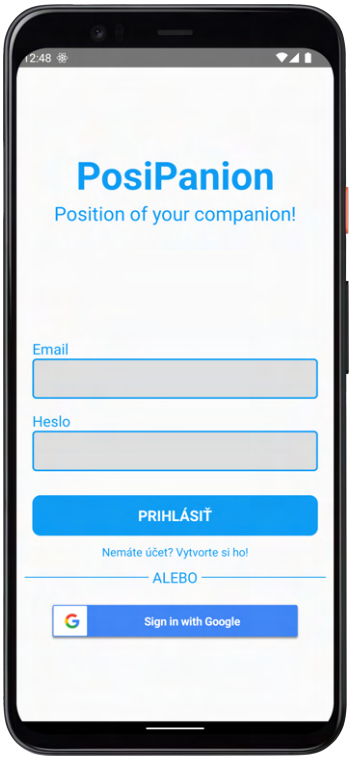
[59] Yusuf, S. *Ionic Framework By Example*. Packt Publishing Ltd, 2016. ISBN 978-1-78528-272-0.

[60] Zielonka, K. *What is Backend in Mobile App Development and How to Choose the Best One for your Project?* [online]. August 2020 [cit. 2021-12-10]. Dostupné z: https://www.thedroidsonroids.com/blog/mobile-app-backend-development-guide-for-app-owners.

[61] Zielonka, K. *What is Rest* [online]. Október 2021 [cit. 2021-12-10]. Dostupné z: https://restfulapi.net/.

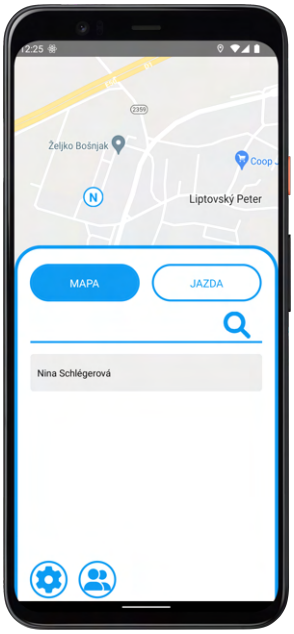
Príloha A

Snímky obrazovky vytvorenej aplikácie

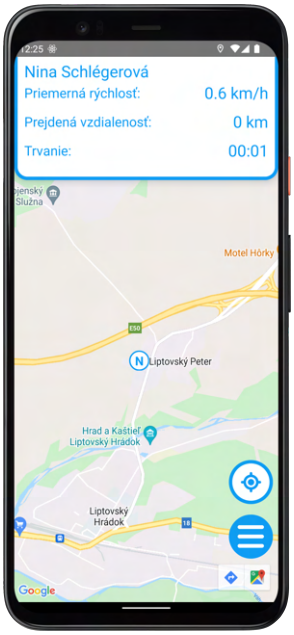


Obr. A.1: Úvodná prihlasovacia obrazovka do aplikácie, ktorá ponúka možnosť prihlásenia pomocou emailu a hesla alebo pomocou účtu Google.

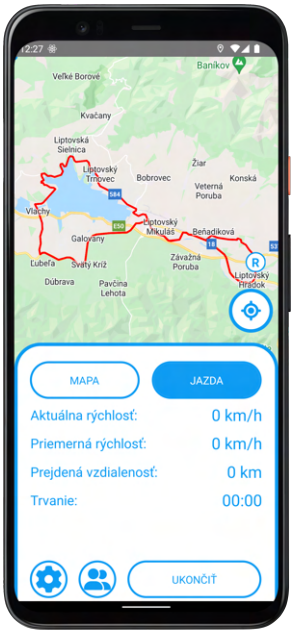
Obr. A.2: Registračná obrazovka, ktorú môže používateľ využiť pre vytvorenie účtu ak nechce využiť prihlásenie pomocou účtu Google.



Obr. A.3: Základná obrazovka aplikácie spolu s vysúvacím menu, na ktorej používateľ môže sledovať polohu svojich priateľov.

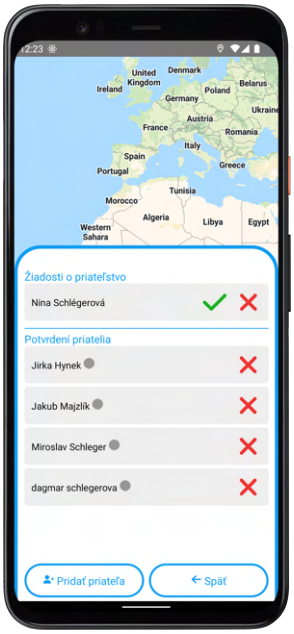


Obr. A.4: Zobrazenie detailu niektorého z aktívnych priateľov spolu s informáciami o jeho aktuálnej jazde.



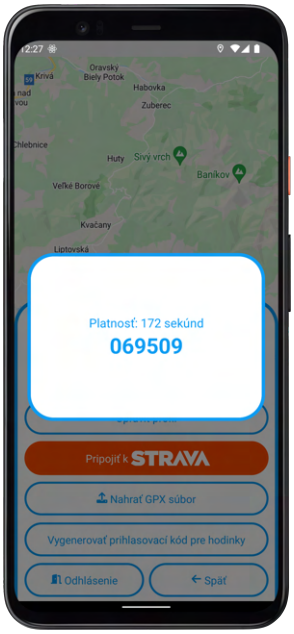
Obr. A.5: Obrazovka zobrazujúca detaily o aktuálne spustenej jazde.

Obr. A.6: Zoznam nahratých trás, ktoré je možné vybrať pred spustením jazdy.

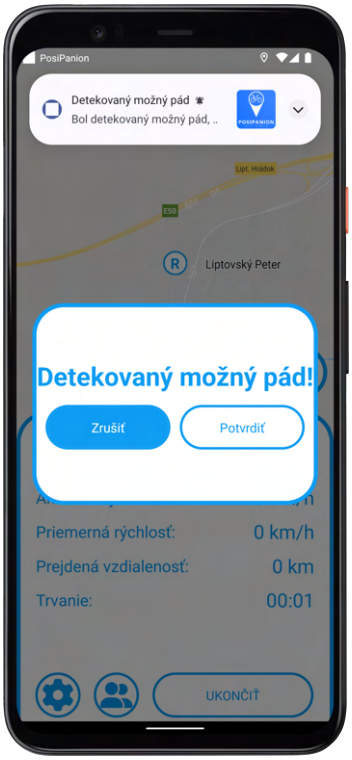


Obr. A.7: Obrazovka slúžiaca na správu priateľov. Zobrazuje aktuálne žiadosti o priateľstvo a tiež už uzatvorené priateľstvá.

Obr. A.8: Nastavenia aplikácie, v ktorých si vie používateľ upraviť profil, pripojiť sa k aplikácií Strava, nahrať GPX súbor alebo vygenerovať kód pre prihlásenie na hodinkách.



Obr. A.9: Zobrazenie vygenerovaného prihlasovacieho kódu pre inteligentné hodinky spolu s jeho platnosťou.



Obr. A.10: Obrazovka, na ktorej vie používateľ zrušiť notifikáciu po tom, ako bol detekovaný možný pád.

Príloha B

Obsah priloženého pamäťového média

```
/posipanion
|- README.MD - obsah média a návod na preklad
|- /sources
   |- /backend
      |- /src
         |- /main
            |- /java - zdrojové súbory
            |- /resources
               |- application.properties - hlavný súbor s nastaveniami
               |- influx2.properties - nastavenia pripojenia k databáze InfluxDB
            |- /sql
               |- init.sql - inicializačný skript pre MySQL databázu
      |- pom.xml - konfiguračný súbor pre nástroj Maven
   |- /frontend
       |- /PosiPanion
         |- /android - špecifické súbory pre operačný systém Android
         |- /components - zdrojové súbory pre framework React Native
         |- /images - obrázky potrebné pre preklad aplikácie
         |- /ios - špecifické súbory pre operačný systém iOS
         |- App.js - hlavný JS súbor pre mobilnú aplikáciu
         |- index.js - JS súbor slúžiaci ako kontajner pre aplikáciu
         |- package.json - konfiguračný súbor s informáciami o potrebných balíčkoch
   |- /wearos - zdrojové súbory pre aplikáciu pre inteligentné hodinky
|- /report - zdrojové súbory textu diplomovej práce
|- /dataset
   |- /Data - súbory dátovej sady
   |- parse_dataset.py - skript pre úpravu tvaru dátovej sady
   |- train.py - skript pre trénovanie modelu detekcie pádu
|- posipanion.apk - inštalovateľný súbor mobilnej aplikácie pre Android
```