VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

APLIKACE CHYTRÝCH HODINEK PRO PODPORU SPORTOVNÍHO TRÉNINKU A ZÁVODŮ

SMARTWATCH APP FOR SPORTS TRAINING AND COMPETITIONS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. PAVEL DOHNALÍK**
AUTHOR

VEDOUCÍ PRÁCE **Prof. Ing. ADAM HEROUT, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2020/2021

Zadání diplomové práce

Student: **Dohnalík Pavel, Bc.**
Program: Informační technologie
Obor: Bezpečnost informačních technologií
Název: **Aplikace chytrých hodinek pro podporu sportovního tréninku a závodů**
Smartwatch App for Sports Training and Competitions
Kategorie: Uživatelská rozhraní
Zadání:

1. Prostudujte možnosti současných chytrých hodinek a problematiku vývoje aplikací pro ně.
2. Vyhledejte a analyzujte existující nástroje pro podporu sportu v chytrých hodinkách.
3. Navrhněte inovativní aplikaci do chytrých hodinek pro podporu sportovního tréninku.
4. Prototypujte a na uživatelích testujte dílčí prvky řešené aplikace - jak technologické možnosti (lokalizace pomocí GPS, použití map, atp.), tak především prvky uživatelského rozhraní a použitelnosti obecně.
5. Implementujte řešenou aplikaci.
6. Testujte řešenou aplikaci v reálném provozu.
7. Iterativně vylepšujte návrh a realizaci řešené aplikace.
8. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- Steve Krug: Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability, ISBN-13: 978-0321965516
- Android Developers: https://developer.android.com/index.html
- Susan M. Weinschenk: 100 věcí, které by měl každý designér vědět o lidech, Computer Press, Brno 2012
- HoJun Jaygarl et al.: Professional Tizen Application Development, Wrox, 2014
- Daniel Knott: Hands-On Mobile App Testing: A Guide for Mobile Testers and Anyone Involved in the Mobile App Business, Addison-Wesley Professional, 2015

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

Vedoucí práce: **Herout Adam, prof. Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 19. května 2021
Datum schválení: 30. října 2020

Abstrakt

Cílem práce je vytvořit aplikaci na chytré hodinky, která umožní měřit závody a tréninky, případně vytvářet lokalizační podklady pro tuto činnost. Aplikace je implementovaná pro mobilní zařízení s operačním systémem Android a iOS. Pro chytré hodinky je podporován operační systém Wear OS. V práci je popsána teorie programování pro mobilní operační systémy a programování na operační systém Wear OS. V praktické části je popsaný návrh, implementace a testování. Pro implementaci mobilní aplikace jsem zvolil framework Flutter spolu s jazykem Dart. Pro aplikaci na chytré hodinky jsem zvolil framework Flutter spolu s jazykem Dart. Výsledná aplikace umožňuje uživatelům měřit závody a tréninky.

Abstract

The aim of the work is to create an application for a smart watch, which will allow you to measure races and trainings, or create localization data for this activity. The application is implemented for mobile devices with the Android and iOS operating systems. The Wear OS operating system is supported for smart watches. The thesis describes the theory of programming for mobile operating systems and programming for the operating system Wear OS. The practical part describes the design, implementation and testing. For the implementation of the mobile application as well as for the smart watch application I decided to choose Flutter framework and programming language Dart. The resulting application allow users to measure races and workouts.

Klíčová slova

Chytré hodinky, Wear OS, Tizen, Android, Flutter

Keywords

Smart watch, Wear OS, Tizen, Android, Flutter

Citace

DOHNALÍK, Pavel. *Aplikace chytrých hodinek pro podporu sportovního tréninku a závodů.* Brno, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Prof. Ing. Adam Herout, Ph.D.

Aplikace chytrých hodinek pro podporu sportovního tréninku a závodů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Prof. Ing. Adama Herouta, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Pavel Dohnalík
18. května 2021

Poděkování

Chtěl bych poděkoval panu prof. Ing. Adamu Heroutovi, Ph.D. za odborné vedení a cenné rady během tvorby této práce. Chtěl bych poděkovat také rodině za velkou podporu během mého celého studia.

Obsah

Kapitola 1

Úvod

Na světě je mnoho lidí, co provozují sport, ať už individuálně, nebo v nějakém sportovním klubu. Při trénincích v klubech je často potřeba provést kontrolní závody, kdy se měří výkonnost sportovců. Při této příležitosti se buď využívají klasické stopky, případně se využívá klasická čipová časomíra.

Ve sportovních klubech jsou alespoň jedenkrát za měsíc pořádány výkonnostní testy. Tyto testy mají poskytnout trenérovi a sportovcům obraz o jejich aktuální formě. Kromě pravidelných výkonnostních testů se před vstupem do závodní sezony pořádají kontrolní závody. Ty mají za účel sportovce dostatečně motivovat ještě k lepším výsledkům než pravidelné výkonnostní testy. Kontrolní závody jsou často pořádané interně ve sportovním klubu, případně si je pořádá několik klubů spolu dohromady.

Organizace kontrolních závodů je organizačně a časově náročný úkol. Zahrnuje výběr vhodné trasy, případně žádost o omezení provozu na trase, a tím zabezpečit trasu pro závodníky. Pokud jsou závody pořádané několika kluby dohromady, je běžné zajistit sponzory a odměny pro nejlepší sportovce. Aby bylo měření závodu objektivní a přesné, je nutné postavit a zprovoznit časomíru. Zprovoznění časomíry zahrnuje přidělení čipů pro závodníky, postavení průjezdných bran a nastavení softwaru pro měření. Následně je potřeba výsledky rozeslat mezi trenéry klubů a závodníky. Dále je potřeba vybrat čipy od závodníků zpátky, aby šly použít znova.

V dnešní době mnoho profesionálních i amatérských sportovců využívá chytré doplňky pro měření jejich výkonnosti a to jak při tréninku, tak při závodu. Sportovci mají na výběr mezi chytrými telefony s velkým množstvím aplikací a chytrými hodinkami, které umožňují navíc měřit i tepovou frekvenci. Problém chytrých hodinek spočívá v jejich neschopnosti komunikovat s mobilní sítí bez přítomnosti připojeného chytrého telefonu. V době psaní této diplomové práce bylo na trhu pouze pár hodinek s technologii LTE a e-sim, které by tohle připojení umožňovalo. Současně nová generace mobilních sítí 5G, které by připojení mnoha chytrých zařízení do sítě podporovala, je v odborné veřejnosti teprve diskutována a jejich vývoj je teprve na počátku.

Cílem mé diplomové práce je navrhnout a implementovat mobilní aplikaci a aplikaci pro chytré hodinky, která umožní organizovat kontrolní závody bez nutnosti použít čipovou časomíru, která je finančně náročná na pořízení. Protože se ve sportovním prostředí setkávám s různými typy uživatelů, kteří mají různá chytrá zařízení s různými operačními systémy, rozhodl jsem se aplikaci implementovat pomocí takové technologie, která by pokryla co možná nejrozmanitější pole operačních systémů. Větší důraz bude kladen na telefony s operačním systémem Android a na chytré hodinky, které využívají operační systém Wear OS.

Čtenář je postupně seznámen s operačním systémem Android a to včetně jeho architektury. Tyhle informace jsou obsaženy v kapitole 2. V další kapitole 3 se dočte o operačním systémech, které jsou využívány v chytrých hodinkách. V kapitole 4 je rozebráno stávající řešení časomíry na závodech a možnosti nahrazení čipové časomíry existujícími aplikacemi pro sportovní aktivity, v krátkosti popsaná zvolená technologie a problém s testovatelností nositelné elektroniky. Současně jsou zde rozebrané požadavky na aplikaci. V kapitole 5 je popsán návrh aplikace na hodinky, telefon, web a server včetně návrhu databáze. V kapitole 6 je popsána má práce na implementaci aplikace pomocí frameworku Flutter s programovacím jazykem Dart a programovacím jazykem Java. V poslední kapitole 7 je zhodnocení mé práce.

Kapitola 2

Operační systém Android

Operační systém Android je v současné době jeden z nejpoužívanějších a nejrozšířenějších operačních systému na poli chytrých telefonů a tabletů. Systém Android našel uplatnění i na poli chytrých hodinek, kde je znám jako Wear OS. Uplatnění operačního systému Android je i v oblasti chytrých televizorů nebo automobilizmu, kde je využíván jako platforma pro infotainment nebo autorádia.

Android je operační systém, který je vyvíjen jako open-source a je založen na bázi Linuxu. O vývoj a údržbu Androidu se stará organizace Open Handset Alliance, která je složena z desítky firem. Firmy v organizaci Open Handset Alliance působí v oblasti mobilních technologii. Jedná se například o společnosti jako je Google, HTC, Intel, NVIDIA, Qualcomm a další. Android se řadí mezi operační systémy, které podporují více platforem procesoru a jiného hardwaru. Informace z této kapitoly jsem převážně čerpal z knihy [12].

2.1 Verze operačního systému Android

Operační systém Android byl původně navržen a vyvíjen Americkou firmou Android Inc., a to až do roku 2003. Tohoto roku byl Android prodám a koupila jej společnost Google, která jej udržuje a vyvíjí až do dnů psaní této diplomové práce.

Android v první své verzi (Android 1.0), byl nasazen v chytrém telefonu HTC Dream (G1). Tento telefon s Androidem se mezi uživatele dostal v září 2008. V této době měl Android na trhu s operačními systémy podíl kolem 0.5%. Operační systém Android, který měl potenciál zaútočit na trh mobilních operačních systémů, byl až Android ve verzi 1.5. Tento operační systém byl kódově označován jako Android Cupcake.

Operační systém Android 1.5. Cupcake podporoval v době, kdy byl na trhu, mnoho funkcí, co je už dnes běžné ve všech chytrých telefonech. Jednalo se například o virtuální klávesnici třetích stran s podporou vlastních slovníků, nahrávání a přehrávání videa ve formátu 3GP a MPEG-4, kopírování a vkládání obsahu pomocí schránky. Dále podporoval přehrávání videa na YouTube a obrázků v Google Picasa. Dále podporoval práci s domovskou obrazovkou, kde umožňoval vkládat widgety a animovat přechody mezi obrazovkami. S Androidem ve verzi 1.6. Donut, který byl vydán v září 2009, přišla podpora rozlišení displeje ve formátu WVGA. Další verze operačního systému, která napomohla jeho rozšíření, byl Android 2.2. Froyo. Tato verze byla vydána v květnu 2010. V této verzi přibyla možnost využít telefon jako přenosný Wi-Fi hotspot.

Dalším zásadním milníkem v historii operačního systému Android je jeho verze 4.0. IceCream Sandwich. Tahle verze byla vydána v říjnu 2011. Zásadní změnou ve vývoji ope-

račního systému Android bylo sjednocení jeho verzí pro telefony a tablety. V operačním systému přibyla implementace rozpoznávání tváře. Byla přidána podpora přehrávání a zaznamenávání videa ve Full HD. Současně proběhla implementace integrující sociální sítě do kontaktů.

Aktuální verze Androidu je verze Android 10, dříve označován jako Android Q. Android 10 byl vydán v září 2019. Od původního systému byla do Androidu přidána možnost uživatelských účtů (Android 4.2), byla zlepšena podpora víceprocesorových zařízení (Android 4.4.). Od Androidu 5.0. byla možnost kompilace aplikací, nebo jejich částí, už při instalaci. Android 7.0. přinesl podporu práce s více okny. Podpora obrazu v obraze přišla s Androidem 8.0. společně s frameworkem, který umožňoval automatické vyplňování platebních údajů, úpravu notifikací a použití vlastních fontů. Samotný Android 10 přináší uživatelům možnost přepnout do tmavého režimu, novinkou je podpora ovládání pomocí gest nebo rozšíření zabezpečení připojování Wi-Fi sítí.

2.1.1 Podporované verze operačního systému Android

Číslo, které přesně určuje verzi operačního systému Android, které je pro běh aplikace minimálně potřeba, se nazývá API verze/level. Jak je vidět v tabulce 2.1, tak číslo, které označuje operační systém Android se neshoduje s číslem API levelu. Před začátkem vývoje aplikace pro Android je potřeba určit, jaké minimální číslo API levelu bude aplikace podporovat. Tím definujeme minimální systémové požadavky pro spuštění a plnou funkčnost aplikace.

Kódové jméno	Verze OS Android	API Level
Android 10	10	API level 29
Pie	9	API level 28
Oreo	8.1.0	API level 27
Oreo	8.0.0	API level 26
Nougat	7.1	API level 25
Nougat	7.0	API level 24
Marshmallow	6.0	API level 23
Lollipop	5.1	API level 22
Lollipop	5.0	API level 21
Honeycomb	3.0	API level 11
Cupcake	1.5	API level 3
-	1.0	API level 1

Tabulka 2.1: Vybraný přehled verzí operačního systému Android a k nim příslušné verze API levelu [12], zpracování vlastní.

Při volbě podporovaného API platí zpětná kompatibilita. To znamená, že veškerá funkčnost zaručovaná právě zvoleným API bude podporována i v novějších verzích operačního systému Android. Při volbě vhodného API pro vývoj aplikace platí úměra, že čím nižší verzi API zvolíme, tím víc zařízení s Androidem budeme podporovat. Pokud ale zvolená verze API bude příliš nízká, nebude možné při vývoji využít nové a pokročilé funkce, kterými systém disponuje. Z tohoto důvodu je nutné volit vhodný kompromis mezi podporovaným zařízením a novou funkcionalitou systému [13].

Při volbě vhodného API je užitečné navštívit stránky udržující statistiku rozšířenosti verzí operačního systému Android, jako vhodné jsou stránky udržované přímo vývojáři [7]. Zde je z tabulky a grafu vidět, že v době psaní této práce by se jako vhodné API hodila buď verze 21 nebo 22 a tím by se pokrylo 89,3% respektive 86,3% používaných Android zařízení. Pokud ale vezmeme v úvahu, že průměrná životnost mobilního telefonu nebo tabletu se pohybuje okolo 3 let [12], mohli bychom z podporovaných zařízení vypustit i ty s verzí operačního systému 5.0 a 5.1, který je starý víc jak 4 roky. Poté bychom stále pokryli více jak 75% používaných zařízení.

2.2 Architektura systému Android

Architektura operačního systému Android je složena z několika vrstev, které jsou vidět na obrázku 2.1. Aby mohla vzniknout kvalitní aplikace pro Android, je potřeba se s touto architekturou seznámit a při programování na ni brát ohledy.

2.2.1 Linux Kernel

Systém Android nevyžívá vlastní jádro systému, ale spoléhá se na jádro systému Linux. Linux Kernel tvoří základ operačního systému Android. Je zde řešena komunikace mezi hardwarem zařízení a jeho software. Za tímto účelem se zde nachází implementace jednotlivých ovladačů - jedná se například o ovladač operující s kamerou, Bluetooth a další. Dále je zde implementována správa procesů, správa paměti, správa napájeni a další.

První Android využíval linuxové jádro ve verzi 2.6.x. Toto jádro mu jen s drobnými změnami vydrželo až do verze Androidu 4.0, kdy bylo vyměněno za Linux Kernel ve verzi 3.0.x. Nejnovější Android 10 využívá linuxové jádro verze 4.4.x, 4.9.x a 4.14.x [20].

2.2.2 Abstraktní Hardwarová vrstva

Abstraktní hardwarová vrstva tvoří standardizované rozhraní pro přístup k jednotlivých hardwarovým částem. Toto rozhraní je vystavené pro nadřazené Java API. Tato vrstva operačního systému Android je složena z knihovních modulů, které jsou v případě dotazování se na příslušný hardware volány a natahovány do systému.

2.2.3 Nativní C/C++ knihovny

Část operačního systému Android, která se nazývá Nativní C/C++ knihovna, je součástí systému z důvodu, že mnoho komponent a služeb využívá ve své implementaci původní knihovny napsané právě ve zmíněných dvou programovacích jazycích. Aplikace mohou díky této vrstvě přistupovat k těmto knihovním funkcím. Například se jedná o přístup k OpenGL ES[1], která je dostupná skrz Android framework Java OpenGL API[2] a tím mohou získat podporu pro manipulaci s 2D a 3D grafikou.

2.2.4 Android runtime

Android runtime je komponenta operačního systému Android, která obsahuje sadu základních knihoven. Od verze Androidu 5.0., která odpovídá API operačního systému na levelu 21, má každá aplikace spuštěná v systému vlastní instanci Android Runtime (ART).

[1] https://developer.android.com/guide/topics/graphics/opengl
[2] https://developer.android.com/reference/android/opengl/package-summary.html

Android Runtime je mnohonásobný virtuální stroj, podobný Java virtual Machine, který spouští soubory s příponou DEX. Navíc je zde optimalizace pro nízkopaměťová zařízení. Soubory s příponou DEX vznikají kompilací klasických souboru JAR a CLASS využívaných Javou.

Ve starších verzích operačního systému Android, to je ve verzích nižších než je Android 5.0., neexistoval Android runtime v podobě, jako je v době psaní této práce. Byl zde použit virtuální stroj Dalvik [12]. Ten na rozdíl od Android Runtime neumožňoval kompilovat část kódu aplikaci už při její instalaci. Android Runtime má na rozdíl od svého předchůdce zdokonalený garbage collector, má lepší podporu pro debugování aplikace a podporuje diagnostické výjimky a hlášení.

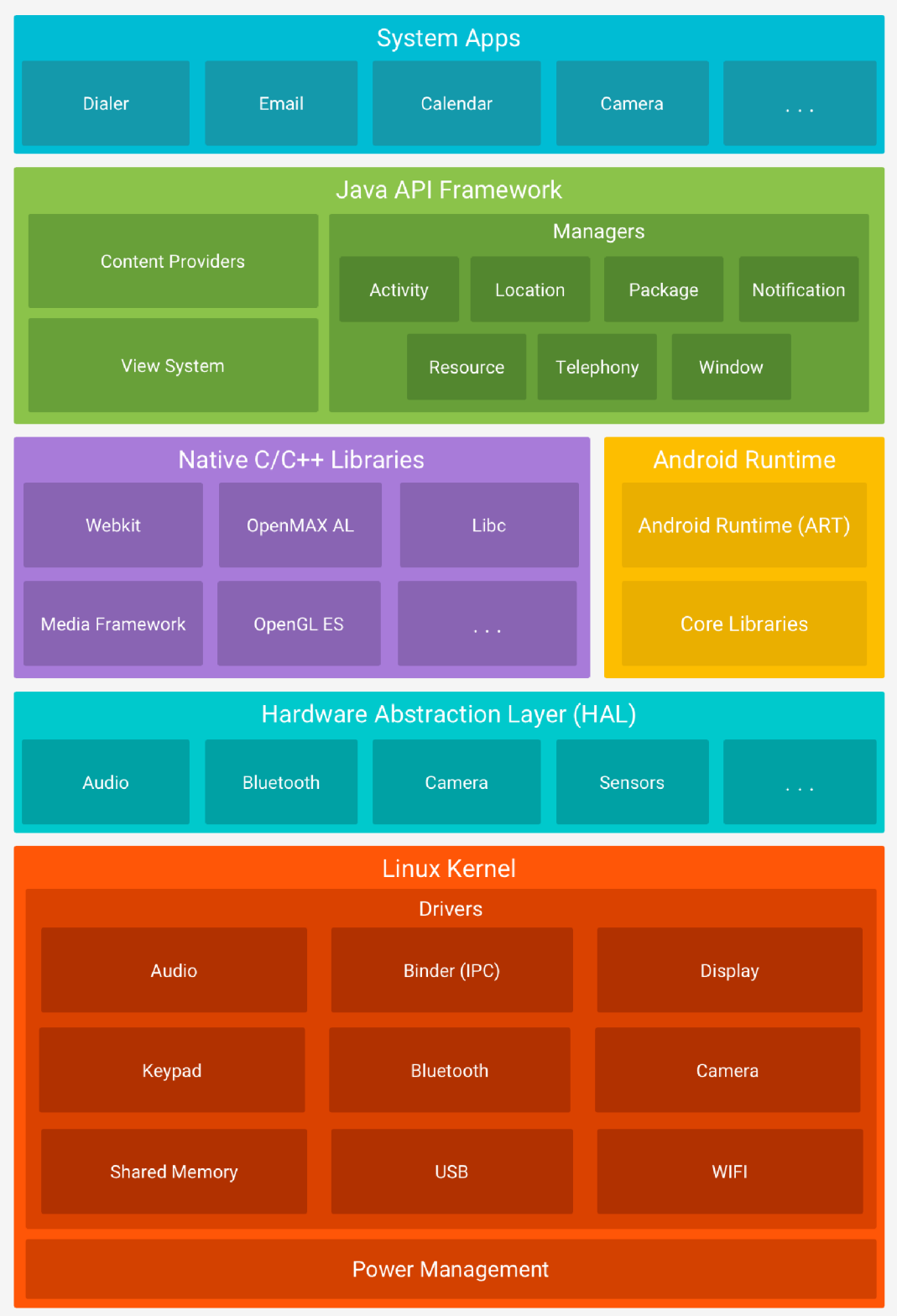
2.2.5 Java API Framework

Z pohledu vývojáře aplikací pro operační systém Android se jedná o nejdůležitější část celého systému. Jedná se o API, které představuje přístupový bod k často používaným částem operačního systému. Jedná se například o:

- *View system*, který se využívá k UI grafickým prvkům aplikace. Obsahuje list, grid manager, text box, tlačítka a další.
- *Resources Manager*, který umožňuje přístup k souborům, které nejsou vyhodnoceny jako zdrojový kód. Jedná se například o textové a grafické soubory.
- *Manager Notifikací* pro umožnění vlastních upozornění z aplikace.
- *Manager aktivit* pro sledování životního cyklu aplikace, umožňuje běh aplikace v době, kdy není spuštěna na hlavní obrazovce.

2.2.6 Systémové aplikace

Operační systém Android disponuje už v čisté verzi několika základními aplikacemi. Jde o aplikace na posílání a přijímání SMS, e-mailu, kalendář, kontakty a další. Systémové aplikace lze při vývoji vlastní aplikace využívat jako služby. Například, pokud budu chtít, aby moje aplikace uměla posílat SMS, nemusím psát znovu funkcionalitu na poslání zprávy, bude stačit využít aplikaci.



Obrázek 2.1: Architektura operačního systému Android [12].

Kapitola 3

Operační systémy chytrých hodinek

Na trhu je nepřeberné množství chytrých hodinek s různými hardwarovými i softwarovými řešeními. Dlouhodobě k nejprodávanějším patří Apple Watch [15]. Další ze stabilních dodavatelů chytrých hodinek je společnost Samsung, Garmin a pak jsou tu hodinky s operačním systémem od Google a to Wear OS.

3.1 Apple Watch

Chytré hodinky společnosti Apple dlouhodobě vládnou prodejním statistikám a Apple je na trhu chytrých hodinek zavedenou společností [18].

3.1.1 Technické specifikace vývoje

Aplikace na chytré hodinky od společnosti Apple se na displeji může objevovat ve třech rozdílných podobách. Hlavní je samotná aplikace. Ta obsahuje strukturu obrazovek s grafickým uživatelským rozhraním a umožňuje interakce s uživatelem. Další je rozhraní glance, kde aplikace ukazuje základní informace. Tato obrazovka se nedá posouvat a je pouze pro čtení. Poslední podobou aplikace jsou notifikace [16].

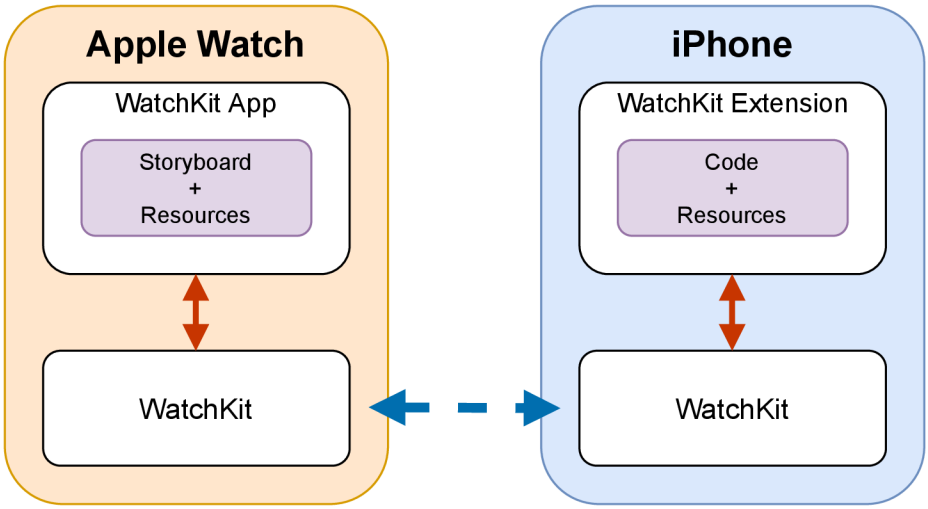
3.1.2 Architektura aplikace pro Apple Watch

Ve společnosti Apple jsou aplikace na chytré hodinky vnímány pouze jako doplněk k aplikaci v mobilním telefonu. To má za následek to, že aplikace třetích stran nemůže fungovat samostatně bez mateřské aplikace v telefonu se kterým jsou hodinky spárované přes bluetoooth [19].

Aplikace na chytré hodinky společnosti Apple je složená ze tří částí a to z WatchKit App, WatchKit Extension a z aplikace v telefonu iPhone. WatchKit App je uložena přímo v hodinkách a obsahuje definici uživatelského rozhraní ve formě storyboardů. Logika aplikace je uložena ve spárovaném telefonu iPhone. WatchKit Extension je uložen v telefonu iPhone a má na starost přesun dat z aplikace v telefonu iPhone do chytrých hodinek [19].

Na obrázku 3.1 je vidět blokové schéma popisující aplikaci pro Apple Watch. Při interakci uživatele s aplikací v hodinkách se pouze vybere příslušná obrazovka reprezentovaná storyboardem. Přes WatchKit se spustí v telefonu WatchKit Extension, který připraví data

pro vykreslení a pošle je do hodinek. Až poté je aplikace zobrazena na displeji. Programátor je od procesu výměny informací mezi hodinkami a telefonem zcela odstíněný.



Obrázek 3.1: Blokové schéma aplikace pro chytré hodinky Apple Watch a logika aplikace ve spárovaném telefonu.

3.2 Wear OS

Operační systém Wear OS je systém společnosti Google. Wear OS podporuje komunikaci přes Bluetooth, Wifi, mobilní sítě a GPS. Stejně jako telefony s operačním systémem Android mají i hodinky s Wear OS přístup do Google play, kde je možné stáhnout aplikace rozšiřující funkčnost operačního systému Wear OS.

3.2.1 Technické specifikace vývoje

Na rozdíl od aplikací pro hodinky od společnosti Apple má vývojář možnost si vybrat jestli jeho aplikace bude plně závislá na aplikaci v telefonu, částečně závislá, nebo zcela nezávislá [6]. Společnost Google ve své dokumentaci doporučuje, aby aplikace vznikající pro hodinky s Wear OS, byly plně nezávisle na telefonu [8]. Plnou nezávislost je potřeba nastavit v konfiguračních souborech aplikace. Pokud programátor plnou nezávislost nenastaví, nebude aplikace dostupná v obchodě Google play. Architektura operačního systému Wear OS vychází z architektury operačního systému Android, která už je popsána v kapitole 2.

3.3 Operační systém Tizen

Operační systém Tizen je založen na linuxovém jádru a je vyvíjen jako open-source. Operační systém Tizen je zaměřený na mnoho různých zařízení jako jsou například chytré telefony, smart televize, zařízení do auta a chytré hodinky. Operační systém Tizen je provozován Linux Foundation a spravován Technical steering Group v níž jsou zapojení zástupci

Samsungu, Intelu a dalších [9]. V době psaní této práce byl nejnovější dostupný operační systém Tizen ve verzi 4.0.0.7 a byl vydaný v září 2019 [9].

3.3.1 Architektura operačního systému Tizen

Operační systém Tizen disponuje webovými aplikacemi a aplikacemi nativními, napsaných v C/C++. Operační systém lze rozdělit do tří vrstev a to na Aplikace, Core a Kernel, vrstvy operačního systému jsou vidět na obrázku 3.2.

Kernel

Tahle vrstva operačního systému Tizen obsahuje Linuxové jádro a ovladače k hardwaru zařízení. Verze linuxového jádra, který nejnovější Tizen využívá, je verze 3.14 [21]. Verze tohoto jádra je využívaná v Tizen 4.0.0 [9].

Core

Vrstva reprezentující jádro operačního systému Tizen je složeno ze dvou částí a to z API a z komponent systému. Komponenty systému představují služby systému, které jsou vystavené do API nad ním.

Webové API

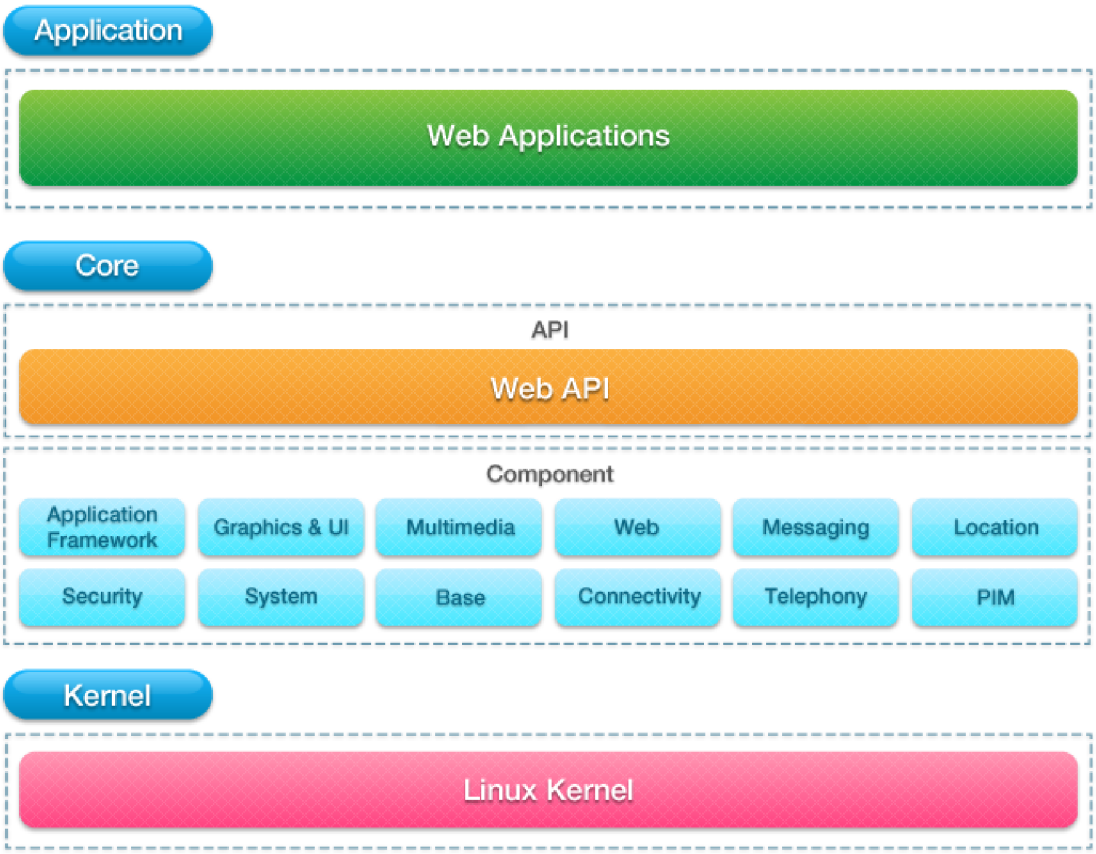
Webové API je určeno pro developery, kteří chtějí využít možnost vyvíjet pro Tizen pomocí webových nástrojů. Tizen web API obsahuje kolekci W3C (například HTML5), Khronos WebGL a další nově definované webové API.

Systémové komponenty

Systémové komponenty jsou převážně služby, které poskytuje operační systém Tizen pro své aplikace. Za zmínku stojí služby pro určování polohy, které zajišťují informace z GPS, WPS (Wi-Fi Positioning System) a další. Z dalších služeb systému je tu podpora pro SMS, MMS a emaily. Z grafických komponent systém disponuje managerem spuštěných oken a OpenGL ES. Z konektivity je zde podpora připojení k Wi-Fi sítím, Bluetooth a mobilním sítím 3G a 4G. Novější chytré hodinky zde mají podporu e-sim [22].

Aplikace

Operační systém Tizen podporuje webové aplikace. Webové aplikace mohou využívat plný výkon a možnosti této platformy, stejně tak jako nativní aplikace napsané v C/C++.



Obrázek 3.2: Architektura operačního systému Tizen [22].

Kapitola 4

Analýza existujících řešení a specifikace požadavků

V této kapitole si rozebereme potřeby, které vznikají při organizaci závodů s čipovou časomírou. Na základě těchto potřeb si zadefinujeme požadavky na aplikaci. Následně se zaměříme na sportovní aplikace, které mají kromě mobilní verze i verzi pro chytré hodinky. U těchto aplikací provedeme analýzu, zda nějaká vyhovuje požadavkům, které jsme si definovali.

4.1 Analýza existujících řešení

Pro zařízení s operačním systémem Android existuje celá řada sportovních aplikací, které ve své verzi zdarma poskytují dostatek údajů pro sportovce o jeho výkonností. Velké množství aplikací má kromě mobilní verze i verzi pro chytré hodinky. Často však chytré hodinky slouží pouze k zobrazení informace a bez mobilního telefonu, který s nimi komunikuje po síti, je celá aplikace nepoužitelná. Většina sportovních aplikací v režimu zdarma poskytuje následující údaje pro sportovce:

- měření aktuální rychlosti pomocí GPS,
- měření průměrné rychlosti,
- měření aktuální ujeté vzdálenosti,
- počítání spálených kalorii,
- snímání trasy.

I přes to, že se jedná o užitečné informace pro sportovce, tak pro závod několika lidí mezi sebou jsou menší prioritou. Aplikace nezvládnou namodelovat hromadný start na konkrétní trase pro konkrétní závodníky a nezvládnou například automaticky odpočítat ani start formou stíhacího závodu. Ve verzi zdarma nezvládnou ani zaznamenat průjezd konkrétním bodem a následně informovat sportovce (závodníka) jak si během závodu vede.

4.1.1 Měření pomocí čipové časomíry

Mezi velmi rozšířené, cenově dostupné a odolné proti poškození se řadí pasivní časomíra. Pasivní časomíra disponuje RFID čipem, který má každý závodník u sebe (obrázek 4.1). Čip je umístěný na kotníku závodníka pomocí pásky. Přesnost měření se zde pohybuje

okolo 0.01 sekundy [1]. Čipy časomíry jsou opakovatelně použitelné pro další závody. Pokud organizátor závodu nechce riskovat nevrácení čipu po závodě, nabízí se i možnost použít čipy jednorázové (obrázek 4.2). Pasivní čipy mají tu výhodou, že nemají v sobě akumulátor a tudíž se nemůžou vybít, to by mělo za následek nezměřený závod pro závodníka.

Obrázek 4.1: Opakovatelně použitelný RFID pasivní čip [17].

Obrázek 4.2: Jednorázový RFID pasivní čip [1].

Při měření závodů pomocí RFID čipů je potřeba na trasu umístit průjezdné body, které jsou tvořeny bránami – anténami (obrázek 4.3) – pro snímáni čipů. Zde se liší podle použitého softwaru pro měření kolik takových průjezdných bodů musí minimálně být. Pokud měřící software umožňuje odstartovat závod ručně a není požadavek měřit reálný čas závodu pro konkrétního závodníka, potom stačí jeden průjezdný bod (anténa), která představuje cíl. Pokud je potřeba měřit konkrétní čas závodníka, například u stíhacího závodu, kdy se čas závodníků na startu liší, je potřeba umístit jednu anténu na start a druhou do cíle. Antény je také možné umístit na průjezdné body.

Toto řešení je ideální pro závody na bruslích, v běhu, na kole ale i klasické lyžování nebo triatlon. Nevýhody jsou vyšší vstupní investice – je potřeba nakoupit čipy, antény, kabeláž a často i licenci na software, kde se dají nastavit parametry závodu. Stejně tak pro kontrolní závody je časová příprava náročná. Je nutné přiřadit čipy k závodníkům v systému, rozmístit antény po trase a zajistit napojení na systém. Možnosti a typy závodů jsou závislé na používaném softwaru v počítači časoměřiče. Pokud se použije jednorázový RFID čip (obrázek 4.2), bývá měření zpravidla méně přesné a vznikají problémy se snímáním průjezdu.

4.1.2 Endomondo Běh Cyklistika Chůze

Kromě čipové časomíry je možné do značné míry realizovat kontrolní závody v klubu pomocí aktuálně dostupných aplikací pro chytré telefony a hodiny. Mezi taková řešení patří

Obrázek 4.3: Anténa pro snímaní čipů během závodu [17].

i aplikace Endomondo Běh Cyklistika Chůze[1]. Aplikace ve verzi zdarma ale nezvládne měřit závod v jeho průběhu a veškeré vyhodnocování probíhá až z naměřených dat po závodě. Z naměřených dat se jen těžko zjišťuje čas závodníka na jednotlivá kola, pokud kolo nemělo délku v celých kilometrech. K porovnání mezi závodníky je k dispozici jenom výsledný čas, vzdálenost, průměrná rychlost a tempo (obrázek 4.4).

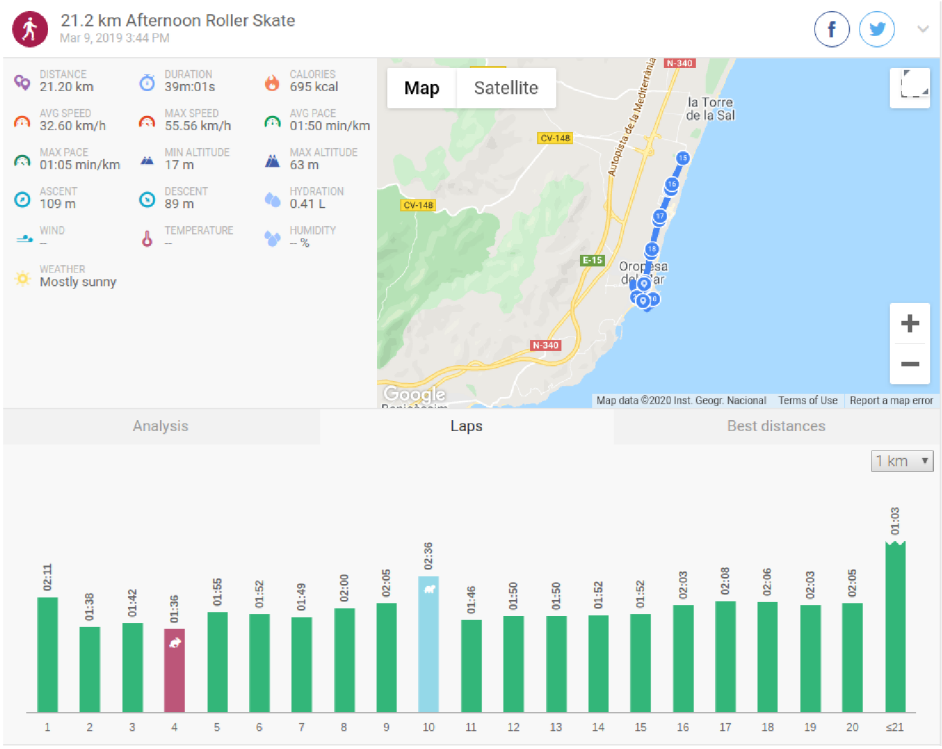
Pokud se chce závodník porovnat v kontrolním měření prováděným v aplikaci Endomondo, je potřeba vytvářet takzvané výzvy. Do nich přizvat závodníky a nastavit cíl - například vzdálenost. Aplikace ale už nezvládne pohlídat, zda všichni zúčastnění projeli stejnou trasu. Dál je zde problém s nastavením soukromí na profilu, kdy závodník může zveřejnit pouze dosažený čas na danou vzdálenost a tím skrýt třeba výškový profil trasy. Informační výstupy, které jsou v aplikaci Endomondo dostupné, jsou vidět na obrázku 4.5, který zobrazuje výsledky pro nejrychlejších 10km na in-line bruslích za rok 2018.

Vývojáři aplikace Endomondo udělali kromě verze na chytrý telefon i aplikaci na chytré hodinky. Aplikace umožňuje z hodinek spustit, pozastavit a ukončit sportovní aktivitu, která je nastavená přes mobilní telefon. Aplikace během sportovního výkonu ukazuje na hodinkách základní údaje o rychlosti a ujeté vzdálenosti. Nedokonalost aplikace na chytré hodinky spočívá v tom, že nezvládne fungovat bez připojeného mobilního telefonu. Aplikace pouze zobrazuje data, která jsou naměřená senzory z telefonu a to i přes to, že chytré hodinky jsou vybavené GPS přijímačem. Hodinky jsou schopné vibrací upozornit na každý ujetý kilometr a následně na displeji přibližně na 5 vteřin ukáží údaje týkající se ujetého kilometru. Na chytrých hodinkách s operačním systémem Wear OS, který vychází z Androidu, se mi nepodařilo zprovoznit snímání srdečního tepu v této aplikaci.

4.1.3 Strava

Další ze zástupců aplikací s podobnou funkcionalitou, jako je aplikace, která vzniká v rámci mé diplomové práce, je aplikace Strava[2]. Aplikace nabízí porovnání mezi sportovci v rámci tzv. segmentů. Aplikace ale bohužel segmenty podporuje pouze pro běh a cyklistiku. Stejně tak nepodporuje statistiky pro sportovce, kteří se věnují jinému sportu než je běh a cyklistika. To znamená, že sportovec věnující se například in-line rychlobruslení, nemá k dispozici

[1] https://play.google.com/store/apps/details?id=com.endomondo.android
[2] https://www.strava.com/



Obrázek 4.4: Ukázka výstupu z aktivity na Endomondo Běh Cyklistika Chůze.

graf výkonu v průběhu sezóny, stejně tak z aplikace nezjistí, kolik kilometrů za časový úsek najel.

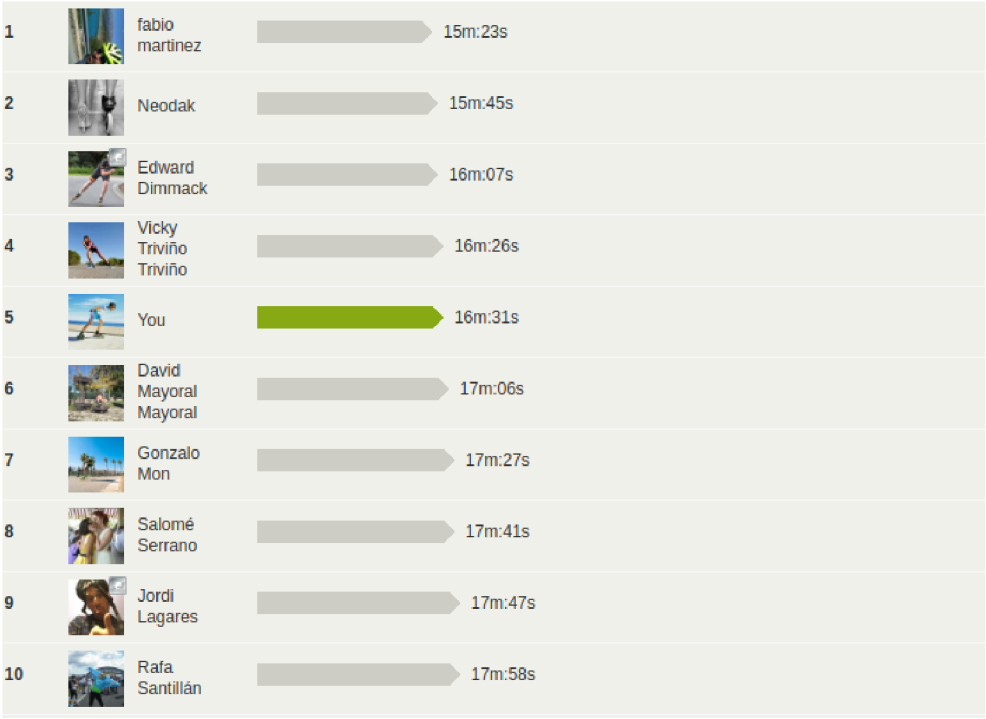
Jediné statistiky, které má závodník k dispozici, je celková ujetá vzdálenost, celkový čas sportovní aktivity, průměrná a maximální rychlost a mapa, kudy závodník jel. Dále je na obrázku 4.6 vidět graf, který zobrazuje výškový profil terénu kudy závodník jel.

Spouštění a zaznamenávání údajů o jízdě na segmentů probíhá automaticky během měření sportovní aktivity a jejich podoba je zobrazena na obrázku 4.7. Je vidět mapa, kde se segment nachází, včetně určení místa startu a cíle. Stejně tak je ve výškovém profilu terénu segment vyznačen tmavou barvou. Dále se sportovec dozví svůj čas, průměrnou a maximální rychlost, nebo nastoupané výškové metry. Stejně tak je pro sportovce k dispozici jeho výsledek a seznam Top 10 sportovců na daném úseku. Sportovci se počítá vždy jeho nejlepší jízda. Největší nevýhoda segmentů je, že nejsou dostupné pro libovolný sport a že jejich vytváření není možné v tarifu, který Strava poskytuje zdarma.

Stejně jako aplikace Endomondo v předchozí kapitole poskytuje i aplikace Strava verzi na chytré hodinky. Ta stejně jako aplikace Endomondo neumožňuje využít GPS senzor přímo v hodinkách. využívá informace o poloze telefonu, kterou přebírá jako polohu, kde je sportovec s hodinkami.

4.2 Specifikace požadavků

Jak je vidět v kapitole 4.1, kde se věnuji popisu existujících řešení pro měření tréninků a závodů, je to pro specifické sporty nevyhovující. Dostatečné přesnosti lze dosáhnout pomocí popsané čipové časomíry, ale její pořízení je finančně nákladné a příprava na měření časově



Obrázek 4.5: Ukázka výstupu z výzev na Endomondo Běh Cyklistika Chůze.

náročná. Aplikace Endomondo poskytuje kvalitní sběr dat a jejich analýzu, ale nedostatečnou podporu pro trénink ve sportovním klubu nebo pro pořádání malých (kontrolních) klubových závodů. Aplikace Strava neumožňuje sběr a analýzu méně populárních sportů – v porovnání s během a cyklistikou. Obě tyto aplikace jsou distribuovány i pro chytré hodinky, ale ani jedna nevyužívá jejich GPS čidlo.

Aplikace Start se snaží poskytnout dostatečnou přesnost pro měření tréninků a malých klubových nebo kontrolních závodů a to za pomocí mobilních telefonů nebo chytrých hodinek. Chytré hodinky s aplikací Start umožní využívat primárně čidla a senzory chytrých hodinek na místo senzorů a čidel telefonu.

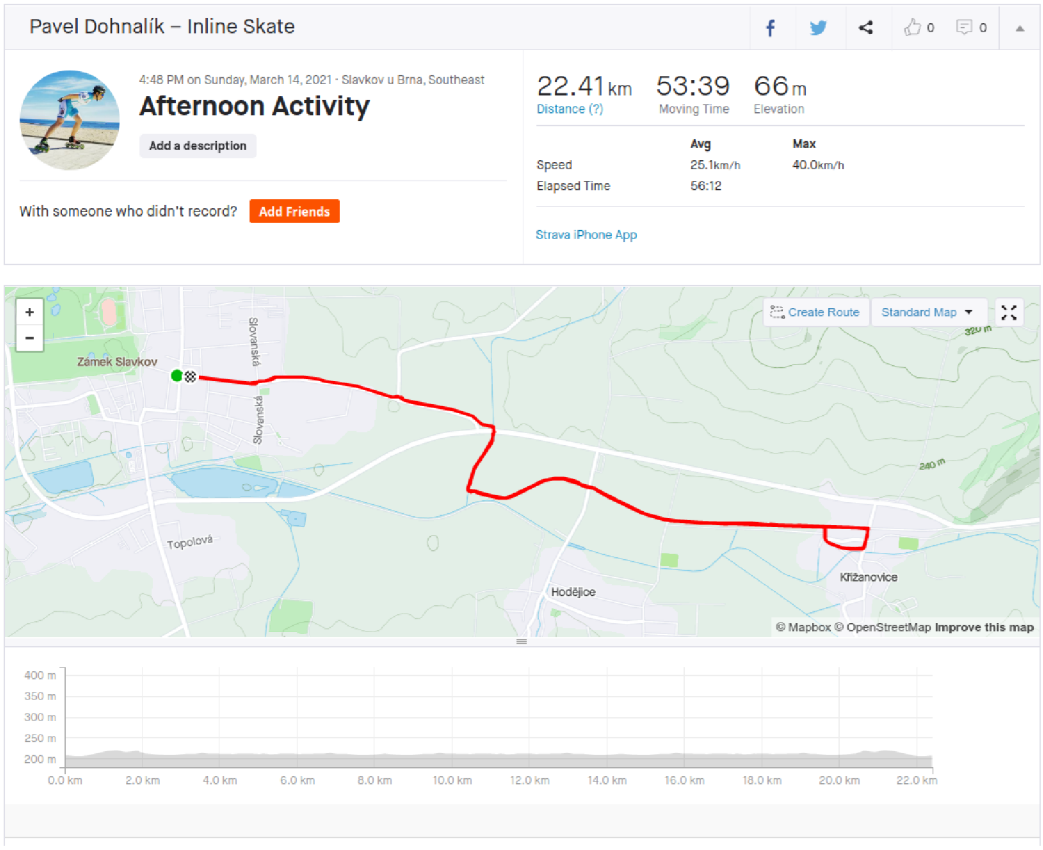
4.3 Použité technologie

V této kapitole jsou popsany základní vlastnosti a přednosti jazyku Dart s frameworkem Flutter, který je použit pro vývoj aplikace na chytré hodinky s operačním systémem Wear OS, pro multiplatformní mobilní aplikaci a pro webovou stránku s možností vytvořit závody.

4.3.1 Framework Flutter

Flutter je framework vyvíjený firmou Google sloužící pro tvorbu multiplatformních mobilních aplikací, webových a desktopových aplikací [24]. Všechny tyto aplikace mohou vznikat z jednoho a toho samého zdrojového kódu.

Při vývoji lze pro snadné ladění grafických prvků využít funkci *hot-reload*, kdy se programátorovi automaticky vykreslí provedené změny bez nutnosti opakované kompilace projektu. Ve frameworku Flutter je možné volat nativní kód dané platformy. Díky tomu si může



Obrázek 4.6: Ukázka výstupu statistik z aplikace Strava.

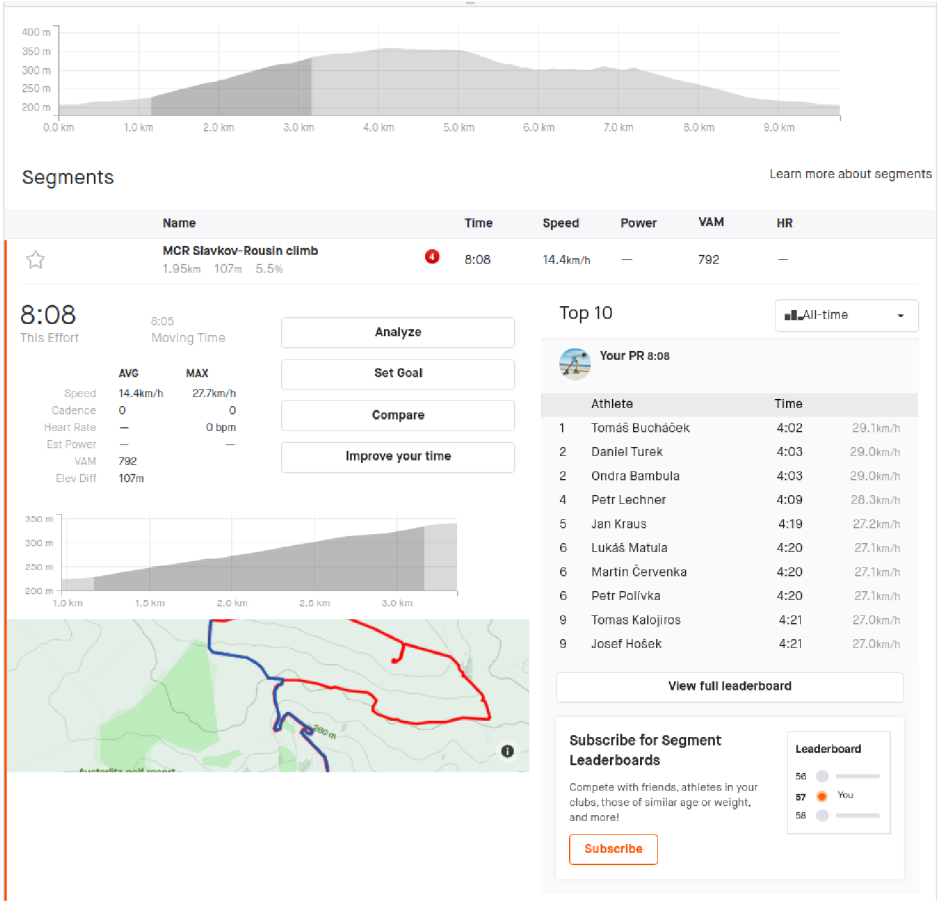
programátor snadno dopsat funkčnost, kterou aktuálně framework na dané platformě nepodporuje. Další výhodou frameworku Flutter je, že nevyužívá nativní prvky platformy kde právě běží, ale jednotlivé grafické elementy vykresluje po pixelech a tedy aplikace vypadá na všech zařízeních stejně [4].

4.4 Testování aplikací na chytrých hodinkách

Testování aplikací na chytré hodinky se dá rozdělit do dvou částí [10]. První z nich je zaměřená na vzhled a uživatele. V ní se věnujeme otázkám, zda se aplikace příjemně ovládá, zda jsou hodinky s aplikací příjemné na nošení a jestli aplikace spuštěná na hodinkách neomezuje běžné činnosti uživatele.

Druhá část testování je zaměřená na komunikaci mezi hodinkami a okolním světem. Zde je zapotřebí brát v úvahu platformu, pro kterou je aplikace na chytré hodinky určená a její vlastností popsané dříve v kapitole 3. Největší výzvou a problémem v oblasti testování nositelné elektroniky (kam patří i chytré hodinky) je jejich častá závislost na dalším softwaru nebo hardwaru ve spárovaném telefonu [10]. Díky tomu je mnohdy technicky náročné nasadit automatické testování pro aplikace běžící na nositelné elektronice, protože je nutné napsat kolem testů prostředí simulující spárovaný telefon.

Framework Flutter umožňuje pro testování využít tři úrovně testů a to unit testy, widget testy a integrační testy. Unit testy pokrývají základní funkčnost metod a tříd. Widget



Obrázek 4.7: Ukázka výstupu z aplikace Strava – segmenty.

testy pokrývají funkčnost widgetu složený z tříd. Integrační testy kontrolují komunikaci a interakce mezi widgety. Pro simulaci okolního prostředí je možné využít plugin `mockito`[3], který částečně zvládne vyřešit závislost aplikace, nebo její části, na dalším subjektu.

[3] https://pub.dev/packages/mockito

Kapitola 5

Návrh aplikace

Samotná aplikace pro chytré hodinky bude pro svou plně dostatečnou funkcionalitu potřebovat aplikační a databázový server pro ukládání a synchronizaci dat. Dále mobilní aplikaci pro prohlížení výsledků ihned po dokončení sportovní aktivity a webovou aplikaci pro plánování závodních okruhů pomocí mapy.

5.1 Návrh aplikace pro chytré hodinky

Aplikace pro chytré hodinky poskytuje sportovci základní informace o sportovní aktivitě, která právě probíhá. Hodinky neustále komunikují se serverem na který odesílají data o času průjezdu branou. Jako odpověď se vrací pozice a ztráta na první místo.

5.1.1 Identifikace uživatele

Běžným uživatelem aplikace je sportovec (rychlobruslař na in-line bruslích). Tento uživatel využívá sportovní hodinky k měření své sportovní výkonosti a nechce během tréninku, nebo závodu u sebe vozit ještě telefon. Hodinky plní funkci měřícího zařízení a s ohledem na toto je grafický vzhled co nejvíce minimalistický a omezený pouze na to nejnutnější.

Aplikace na chytré hodinky musí umožnit rychlý start tréninku. To znamená spustit aplikaci a ihned mít k dispozici tlačítko „start“ pro spuštění měření. Všechna konfigurace a přehled statistik je možný na telefonu.

5.1.2 Grafický návrh aplikace pro chytré hodinky

Aplikace obsahuje 6 obrazovek, pomocí kterých může sportovec dostávat informace o právě prováděné sportovní aktivitě. Jedná se o obrazovku přihlášení, respektive pro párování hodinek s účtem vytvořeném na telefonu, potvrzovací obrazovku, obrazovku s časem od startu, obrazovku se vzdáleností od startu, obrazovku s pozicí v závodě a odhlašovací obrazovku.

Protože displeje hodinek jsou dost malé - rozměrově mají průměr 38mm, 40mm, nebo 44mm, rozhodl jsem se ovládat přesun mezi displeji pomocí gest. Pomocí táhnutí prstem nahoru a dolů se realizuje přepínání mezi informačními displeji o sportovní aktivitě. Toto přepínání tvoří nekonečnou smyčku. Táhnutí prstem zprava doleva se přepne na obrazovku s možnostmi odhlásit a ukončit aplikaci.

Párování aplikace na chytrých hodinkách k uživatelskému účtu na telefonu je realizováno pomocí zadaní generovaného kódu aplikace v telefonu. Grafické uživatelské rozhraní je navrženo tak, aby bylo snadno čitelné a ovladatelné jak na kulatém displeji (obrázek 5.1) tak

Obrázek 5.1: Přihlašování WearOS.

Obrázek 5.2: Přihlašování Apple Wear.

na hranatém displeji (obrázek 5.2). Po zadání unikátního kódu z telefonu proběhne dotaz na server, kde se podle něj vyhledá uživatel. Hodinky si stáhnou údaje nutné pro automatické přihlašování. Při dalším spuštění aplikace už bude uživatel přihlášen automaticky.

Obrazovky hodinek, zobrazující informace o právě probíhané sportovní aktivitě, mají jako hlavní prvek ovládací tlačítko. Toto tlačítko umožní odstartování sportovní aktivity, její pozastavení nebo ukončení a je přístupné ze všech obrazovek (obrázky 5.3, 5.5, 5.4). Kromě ovládacího tlačítka je dalším dominantním prvkem informace o aktivitě. Na obrazovce, která je na obrázku 5.3 to je čas od startu aktivity. Na další obrazovce (obrázek 5.4 je to vzdálenost, co už sportovec během aktivity absolvoval a na poslední obrazovce (obrázek 5.5) je to pozice v závodě se ztrátou na první místo.

Obrázek 5.3: Situace před startem.

Obrázek 5.4: Informace o absolvované vzdálenosti.

Obrazovka předávající informaci o aktuální pozici v závodě a ztrátě na první místo (obrázek 5.5) dostává data od serveru. Tato data jsou doručena v odpovědi na dotaz z hodinek. Dotaz obsahuje identifikaci závodníka, identifikaci detekované brány, kterou závodník právě projel a jeho čas od počátku aktivity.

Detekce průjezdu branou je spuštěna na pozadí všech obrazovek. To znamená, že informace o pozici v závodě se aktualizují i když závodník zrovna není přepnutý na obrazovku s nimi. Hodinky si po celou dobu běhu ukládají všechna data o poloze. Po ukončení sportovní aktivity se odešlou tato data na server. Hodinky tedy neprovádějí finální dopočet, který pomůže zpřesnit časový údaj o průjezdu bránou.

Obrázek 5.5: Informace o pozici.

5.2 Návrh aplikace pro telefon

Aplikace na telefon umožní sportovci sledovat stejné informace jako aplikace na chytré hodinky. Navíc ukáže závodníkovi informace o jeho aktivitě po jejím dokončení. Jedná se například o průměrnou rychlost, maximální rychlost a průměrný čas na kilometr. Navíc bude možné v mobilní aplikaci nahlédnout do mapy aby si sportovec mohl prohlédnout polohu bran, kterými musí projet.

5.2.1 Identifikace uživatele

Běžný uživatel aplikace na telefon je sportovec (rychlobruslař na in-line bruslích), který z nějakého důvodu nemůže nebo nechce pro měření využít aplikaci na chytrých hodinkách. Jako důvod může být nekompatibilita mezi aplikací a operačním systémem hodinek nebo nebo nedostatečná hardwarová vybavenost hodinek.

Hlavním cílem aplikace na chytrém telefonu je umožnit měření sportovní aktivity a to stejně rychle jako na hodinkách. To znamená, že uživatel spustí aplikaci a hned po jejím startu může zahájit měření sportovní aktivity. Sportovec na úvodní obrazovce (obrázek 5.6) hned vidí mapu, na kterou si přes filtr může zobrazit nejbližší závody. Má zvolený svůj hlavní sport a má k dispozici tlačítko „start“. Po stisknutí start tlačítka je zahájeno měření a to buď ve formě volného tréninku nebo vybraného závodu.

5.2.2 Obrazovka sportovní aktivity

První, co závodník uvidí po přihlášení do aplikace, respektive po jejím opětovném spuštění pokud se už jednou přihlásil, je obrazovka, která ihned umožní odstartovat sportovní aktivitu. Jedná se o sportovní aktivitu volný trénink, která umožní sportovci změřit vzdálenost, kterou během aktivity urazil, a čas, jak dlouho mu to trvalo. Pohyb se vykresluje do mapy, která zabírá větší část displeje. Závodník si ještě před odstartováním volného tréninku může vybrat jaký sport bude provozovat. Toto může udělat pomocí ikony, která znázorňuje jím vybraný sport.

Pokud se chce závodník účastnit měřené sportovní aktivity, nebo závodu je nutné aby si nejprve vybral sport, kterému se chce věnovat. Následně si pomocí filtru vpravo dole vybere závod, který se mu vykreslí do mapy pomocí bran. Takto vykreslený závod je vidět na obrazovce, která je na obrázku 5.6. Pokud má závodník spárované hodinky s apli-

kací v telefonu, tak se mu vlevo dole zobrazí ikona hodinek. Pomocí ní může závod odeslat do hodinek a měřit jej pomocí aplikace v chytrých hodinkách.

Obrázek 5.6: Hlavní obrazovka aplikace, kde je vybraný závod a možnost odeslat aktivitu na hodinky.

Měření sportovní aktivity na telefonu probíhá na pozadí a závodník může tedy přepínat mezi obrazovkami. Je nutné aby během aktivity typu závod byl telefon neustále připojený k internetu. Po detekci průjezdu bránou telefon odešle údaje identifikující závodníka, právě projetou bránu a čas průjezdu na server. Server v odpovědi vrátí pozici v závodu a ztrátu na první místo. Tato informace je následně předána závodníkovi.

Telefon během sportovní aktivity ukládá všechny body průjezdu do paměti. Po dokončení sportovní aktivity provede finální zpřesňující dopočet času průjezdů branami. Následně takto zpracovaná data odešle na server, kde dojde k porovnání s ostatními závodníky. Výsledné zpřesnění je tedy provedeno přímo na zařízení u závodníka a ne až na serveru jako v případě komunikace s chytrými hodinkami.

Menu ve spodní části obrazovky umožňuje rychlé přepínání mezi sportovní aktivitou (RUN), přehledem sportovních výkonů (feed) a nastavení (settings). Aktuálně navštěvovaná obrazovka je v menu vždy zvýrazněná.

5.2.3 Obrazovka s přehledem sportovních aktivit

Závodník se může v mobilní aplikaci vracet ke svým sportovním aktivitám, které si může prohlížet. Stejně tak vidí aktivity ostatních sportovců které sleduje. Závodník se může přidat do týmu a tím si přednastavit filtr konkrétních závodníků které chce vidět. Přepínání mezi těmito pohledy probíhá v aplikaci buď pomocí gest - pohyb po prstem po displeji zprava doleva a zpět, nebo pomocí klikání na tlačítka v horní části displeje (obrázek 5.7).

V položce zobrazené v přehledu sportovních aktivit je vidět vždy profilová fotka sportovce, který aktivitu provedl, název sportovní aktivity a typ sportu. Na mapce ukazující sportovní aktivitu jsou vidět brány pouze tehdy, když se jedná o sportovní aktivitu typu závod. U volného tréninku je zobrazená pouze trajektorie jízdy.

Pod mapou jsou hned k dipozici základní údaje o trase. V případě závodu je to výsledná pozice v cíli, ztráta na první místo a celkový čas trvání sportovní aktivity. Pokud je sportovní aktivita typu trénink, zobrazí se zde celková absolvované vzdálenost, celkový čas aktivity a průměrná rychlost. Jak u sportovní aktivity typu závod, tak u sportovní aktivity typu trénink jsou další detaily dostupné v detailu aktivity.

Obrázek 5.7: Návrh obrazovky zobrazující přehled všech sportovních aktivit.

Obrazovka, zobrazující detail sportovní aktivity (obrázek 5.8), nabízí další informace. Závodník si může zobrazit celkovou ujetou vzdálenost v závodě, průměrnou a maximální rychlost a zdolané výškové metry. U sportovní aktivity typu závod navíc dostane přehled průběžných výsledků z bran, které projel. Pomocí těchto informací zjistí, jak si v porovnání

se startovním polem vedl v průběhu závodu. U sportovní aktivity typu trénink zde dostane seznam závodů, které při tréninku absolvoval a kam se může přihlásit i po dokončení sportovní aktivity. Obrazovka zobrazuje mapu, kde je vykreslená trasa provedené sportovní aktivity. Pokud je aktivita typu závod, tak jsou zde navíc vykreslené také brány.

Obrázek 5.8: Návrh obrazovky zobrazující detail sportovní aktivity.

5.2.4 Obrazovka pro párování s hodinkami

Aby uživatel nemusel vypisovat často dlouhý e-mail a heslo na malém displeji hodinek, rozhodl jsem se využít k párování šesticiferný kód. Šest čísel je na malém displeji na numerické klávesnici pohodlnější na psaní.

Generování kódu probíhá na telefonu, kde se ověřuje, zda splňuje dostatečnou délku. Pokud ano, proběhne ověření se serverem, že je kód skutečně unikátní na dobu, po kterou je platný. Pokud server vrátí potvrzující odpověď, zobrazí se kód uživateli tak, jak je vidět na obrázku 5.9. Závodník jej následně zadá do hodinek a proběhne párování.

Tlačítko „Nový kód“ umožní závodníkovi vygenerovat nový kód. Využití to má v případě nákupu nových hodinek, nebo pokud původnímu kódu vypršela platnost a párování s ním už není možné. Aplikace umožní závodníkovi mít spárované vždy jen jedny hodinky.

Obrázek 5.9: Návrh obrazovky pro párování telefonu s hodinkami.

5.2.5 Přihlášení a registrace do aplikace

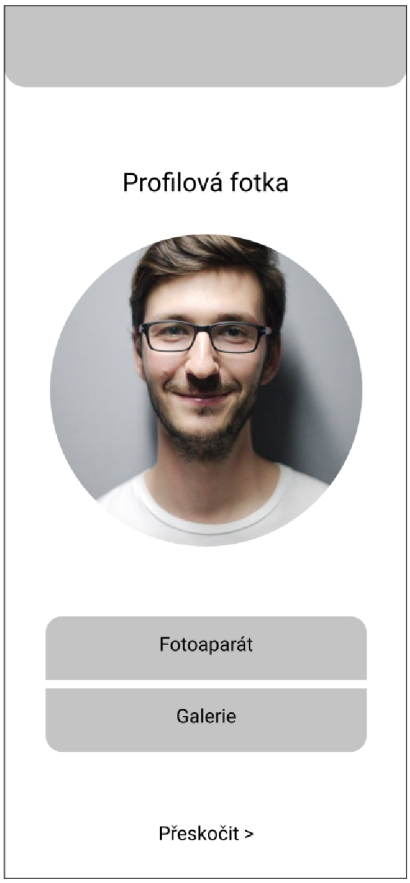
Mobilní aplikace vyžaduje vytvoření uživatelského účtu a následné přihlášení pomocí přihlašovacích údajů. Po prvním přihlášení si aplikace údaje uloží a při dalším spuštěním provede přihlášení už automaticky.

Přihlášení do aplikace je možné buď pomocí e-mailu a hesla nebo pomocí účtu na sociální sítí Facebook. Heslo bude z telefonu na server odesíláno v zahashovaném formátu. Delším stisknutím textového pole, které je určené pro heslo, si uživatel zobrazí heslo v čitelné podobě. Pokud uživatel ještě nemá účet, může se přesunout na obrazovku, kde provede registraci. K registraci vyplní údaje jako jsou přezdívka, e-mail a heslo.

Následně mu je umožněno vybrat si profilovou fotku a to buď z galerie telefonu nebo ji hned pořídit prostřednictvím fotoaparátu v telefonu (obrázek 5.10. Profilová fotka se mu hned ukáže na příslušném místě jak je vidět na obrazovce. Pokud je fotografie příliš velká, dojde ještě před jejím odesláním na server k její komprimaci.

5.3 Návrh webové aplikace

Webová aplikace umožní sportovci prohlížet stejné informace jako v mobilní verzi. Navíc umožňuje vytvářet závody a do mapy zakreslovat průjezdové brány pomocí klikání myší. Dále si zde závodník může prohlížet závody co v minulosti vytvořil. Závod, který ještě nebyl



Obrázek 5.10: Obrazovka pro výběr profilové fotografie.

odstartován, lze editovat a mazat. Závod, co už je odstartován ale ještě není dokončen, je uzamknutý pro veškerou editaci. Závod, co je již dokončen, lze kopírovat a kopii editovat v plném rozsahu pro jakékoliv budoucí datum.

Přihlášení do webové verze aplikace probíhá stejně jako do mobilní a to pomocí e-mailu a hesla nebo s využitím profilu na sociální síti Facebook. Pokud závodník nemá účet, může si ho zde vytvořit. K zapamatování přihlašovacích údajů je nutné využít vlastnosti webového prohlížeče.

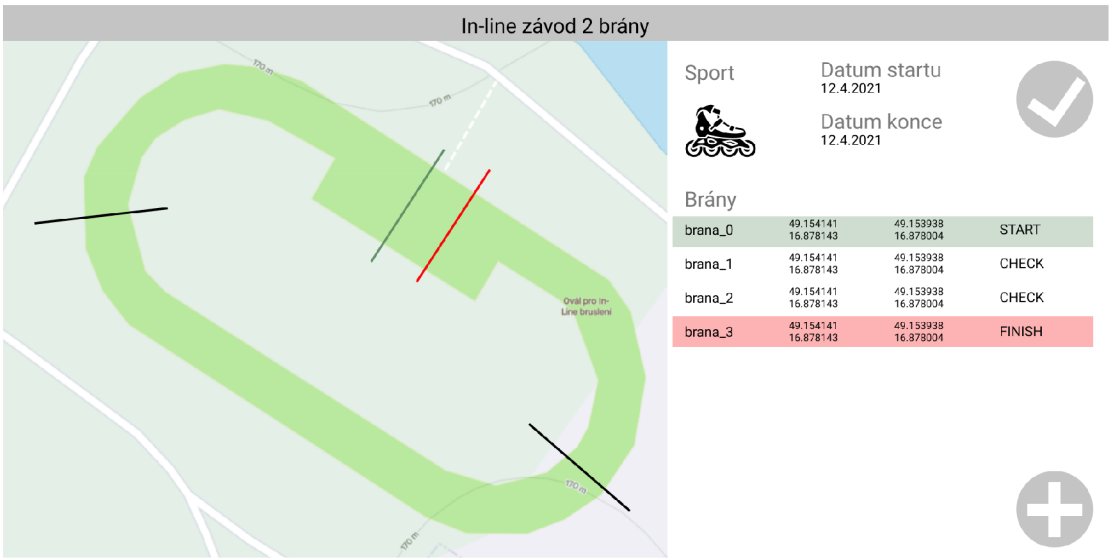
Jako hlavní obrazovka na webu je přehled aktivit přihlášeného závodníka. Přepínání probíhá pomocí tlačítek v horní části obrazovky – podobně jako na telefonu. Detail sportovní aktivity zobrazuje stejné informace jako detail v mobilní aplikaci.

5.3.1 Obrazovka pro vytvoření závodu

Největší část obrazovky pro tvorbu závodů ve webové aplikaci zabírá mapa (obrázek 5.11). Mapa umožňuje závodníkovi snadné plánování bran, kterými je nutné projet při závodu. Mapa umožňuje provádět základní mapové operace jako přiblížení/oddálení nebo posouvání v ní.

V pravé části obrazovky je možné si vybrat sport, pro který je závod tvořen a datum jeho zahájí a ukončení. Následně se pomocí velkého tlačítka plus, které je vpravo dole přidávají průjezdové brány. V mapě se pomocí myši vždy vybere jeden a pak druhý hraniční bod

brány. Následně se provede dokreslení nejkratší rovné cesty mezi těmito body. Brány jsou barevně rozlišeny. Zelená barva je pro startovací bránu, červená barva pro cílovou bránu a černá barva pro průjezdové brány. Barvy mají za úkol ulehčit orientaci v typu průjezdové brány [11].



Obrázek 5.11: Návrh obrazovky pro tvorbu závodu.

Po kliknutí na potvrzovací tlačítko se provede odeslání veškerých zadaných informací na server a závod se zobrazí v přehledu vytvořených závodů. V mobilní aplikaci se zobrazí až v době, kdy je aktuální datum v intervalu mezi zadaným datem startu a datem konce.

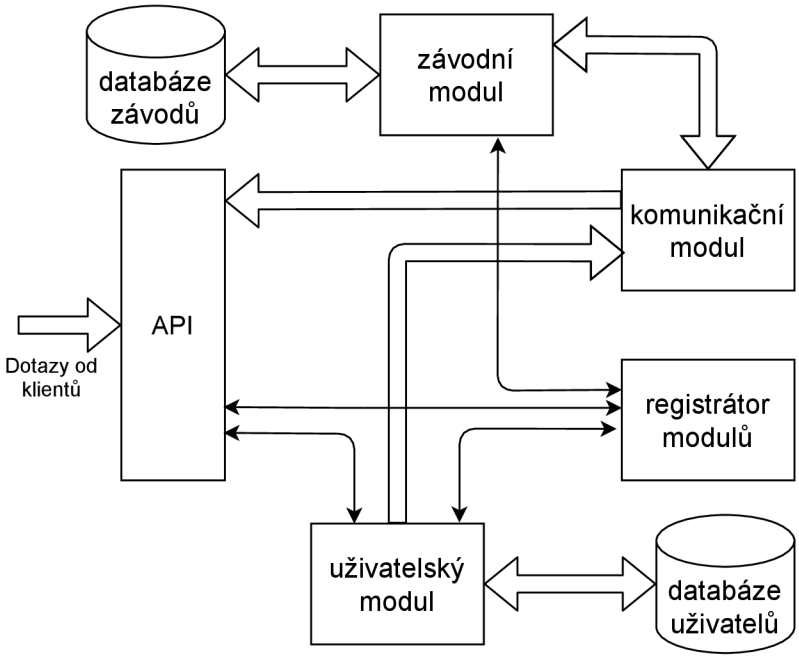
Přehled závodů je tvořen rozevíracím seznamem jmen závodů. Na každém řádku je zobrazen druh sportu o který jde, jméno závodu, datum začátku a konce závodu.

5.4 Návrh serverové aplikace

Serverová aplikace umožňuje ukládání uživatelských dat, jejich synchronizaci a sdílení s ostatními uživateli aplikace. Aplikační serverová část poskytuje komunikační rozhraní pro klienty, komunikační rozhraní pro další služby běžící na serveru a funkcionalitu pro ukládání a čtení dat z databáze.

Server je složen z několika modulů, jak je vidět na schématu na obrázku 5.12. Komunikační modul API přebírá dotazy od klientů (chytré hodinky, mobilní a webová aplikace) a přeposílá je buď do uživatelského modulu nebo jejich požadavek zkusí vyřešit, pokud má uživatel dostatečná oprávnění.

Pomocí uživatelského modulu probíhá také ověření, zda má uživatel přístup k požadované operaci a datům na serveru. Uživatelský modul disponuje rozhraním pro komunikaci s API. Má vlastní databázi uživatelů, kterou si udržuje a poskytuje data z ní dalším službám a to buď pomocí komunikace typu dotaz-odpověď nebo publikováním do streamu zpráv. Uživatelský modul se musí zvládnout registrovat do registračního modulu a to při každém startu aplikace.



Obrázek 5.12: Návrh modulárního serveru pro aplikaci. Registrační modul umožní distribuci jednotlivých modulů na samostatná fyzická zařízení. Komunikační modul zajišťuje doručování zpráv uvnitř systému pomocí mechanizmů producent-konzument.

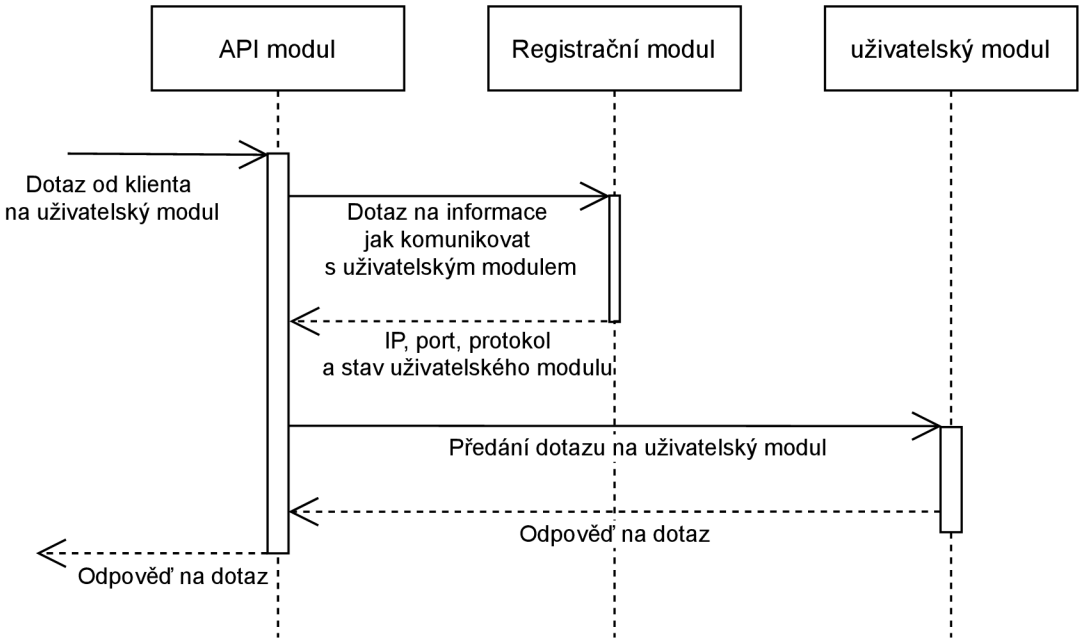
Závodní modul má vlastní databázi, kde si udržuje záznamy o závodech a výsledcích. V závodním modulu je též implementován dopočet přesnosti pro hodinky, které tuto funkcionalitu nemají. Závodní modul publikuje data z databáze pomocí streamu zpráv. Závodní modul se musí zvládnout registrovat do registračního modulu a to při každém startu aplikace

Protože je kompletní server modulární, lze jej distribuovat po více fyzických zařízeních. Proto je nutné mít registrační modul, který zaznamenává adresy služeb, co se k němu přihlašují. Pokud API modul potřebuje komunikovat s uživatelským modulem, je nutné nejprve zjistit, kde uživatelský modul je, pak až je zahájená komunikace s tímto modulem. Proces komunikace je nakreslen na diagramu na obrázku 5.13.

5.4.1 Návrh databáze

V první verzi návrhu databáze vznikl celý návrh tak, jako by byl pro monolitický systém. Tento první návrh je vidět na obrázku 5.14. V relaci user je vidět ukládání aktivačního kódu pro hodinky včetně času aktivace a času expirace vygenerovaného kódu. Pomocí relací team a team_mate je modelováno členství jednoho závodníka ve více týmech. Relace follow modeluje vztah mezi uživateli, tzv sledování.

Relace race v sobě uchovává informace o závodu, jako je jméno nebo sport. Race je ve vztahu 1:n k relaci race_point. Tady je důležité omezení, že každý závod má alespoň dva racePointy a to start a cíl. V relaci run jsou ukládána data o průjezdu uživatele run_pointem, kde relace run_point a race_point mají společného rodiče point. Run_race tvoří vazbu mezi záznamem o absolvování závodu a závodem samotným.

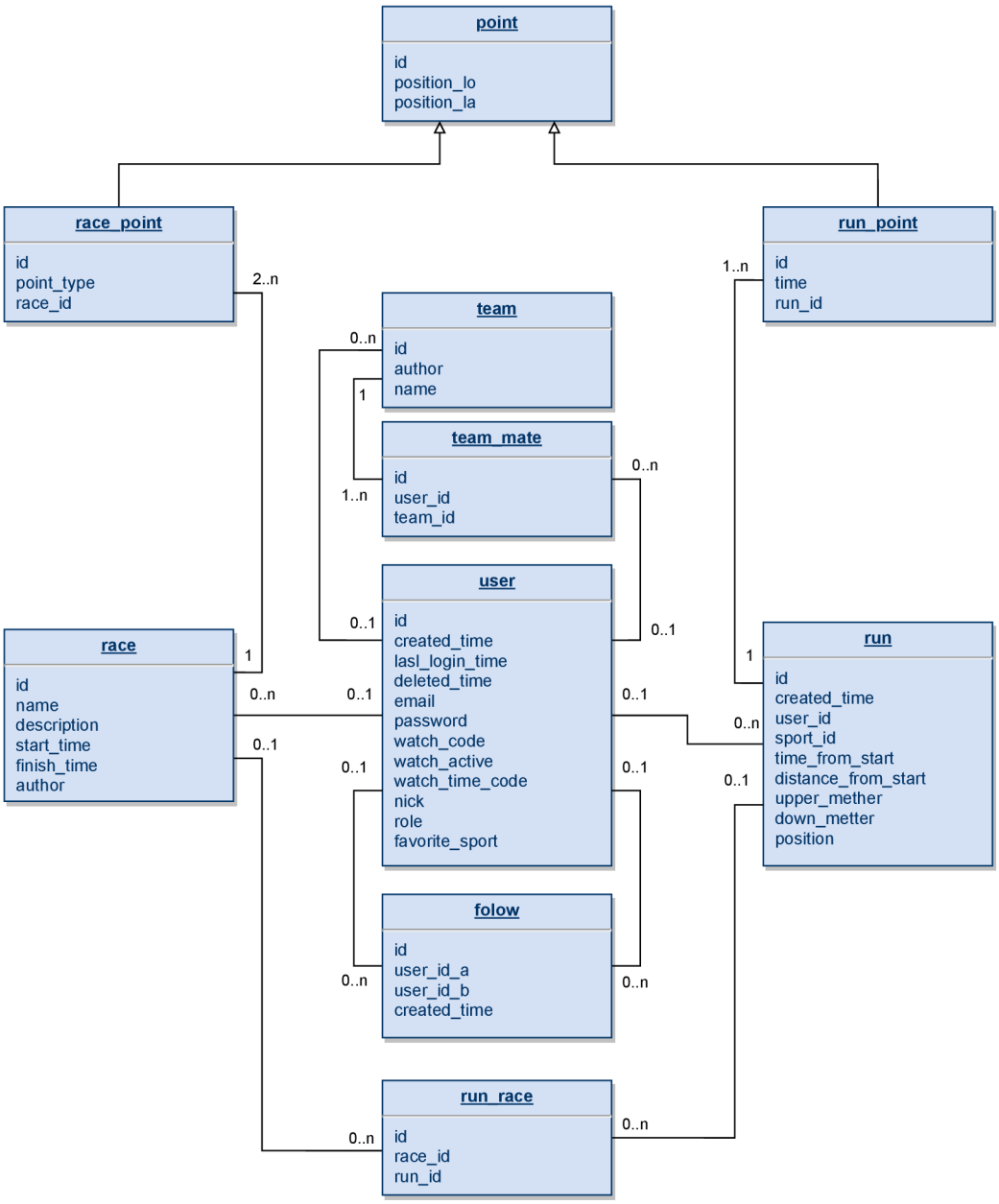


Obrázek 5.13: Diagram komunikace při zaslání dotazu na API. Dotaz je cílen na uživatelský modul. API musí nejprve zjistit, kde uživatelský modul je.

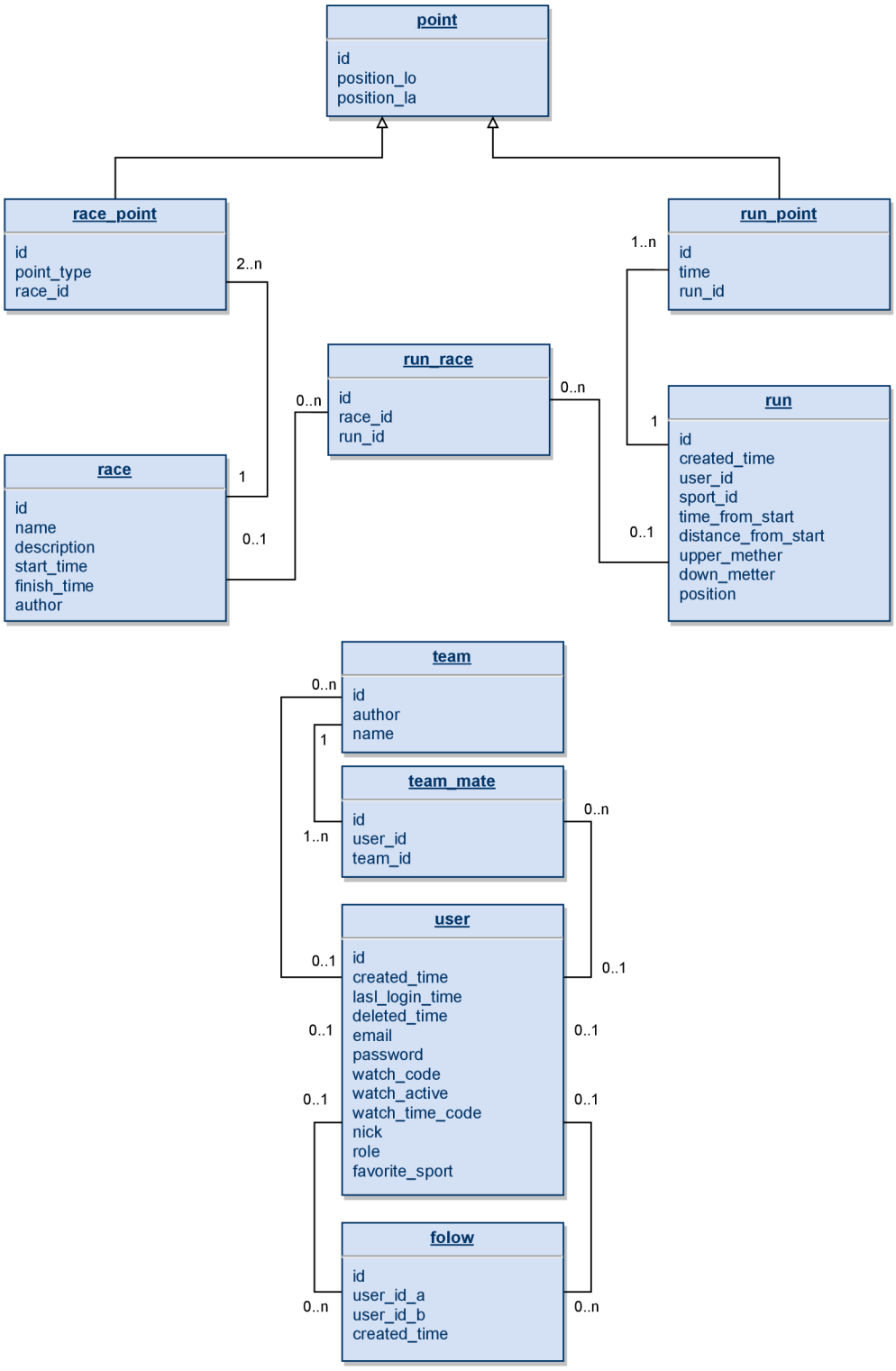
Protože je patrné, že silně je mezi sebou provázaný pouze uživatel s teamy a sledováním a druhý silně provázaný celek je ukládání závodů, rozhodl jsem se databázi rozdělit na tyto dva celky (obrázek 5.15). Komunikace mezi nimi je zařízena na aplikační úrovni.

5.4.2 Návrh komunikace mezi serverem a klienty

Serverový modul API poskytuje komunikační rozhraní pro klientská zařízení. Komunikace mezi serverem a klienty probíhá buď pomocí REST API, nebo pomocí websocketů. Záleží na jejich povaze, kdy například přihlášení je lepší realizovat pomocí dotazu na REST API, ale seřazení závodů od nejnovějších a vytvoření „nekonečného“ seznamu je lepší pomocí socketů.



Obrázek 5.14: ER diagram databáze navržený pro monolitický aplikační server.



Obrázek 5.15: ER diagram databáze navržený pro moduly aplikačního serveru. Na horním obrázku databáze pro ukládání dat o závodu. V dolní části obrázku databáze pro ukládání uživatele.

Kapitola 6

Implementace a vyhodnocení

Tato kapitola se věnuje implementaci jednotlivých částí aplikace na chytré hodinky, telefon, web a server. Tyto části systému byly rozděleny do více služeb, tříť aby se daly znovu použít nebo rozšiřovat. V této kapitole se taky budu věnovat tvorbě uživatelského rozhraní.

6.1 Implementace aplikace pro chytré hodinky

Aplikace pro chytré hodinky obsahuje nejzákladnější funkcionalitu, která je implementována. Tato funkcionalita je následně rozšiřována v aplikaci pro telefony s Androidem a iOS. Další rozšíření je ve webové verzi. Celá aplikace na chytré hodinky je implementována v jazyce Dart pomocí frameworku Flutter.

Aplikace na chytrých hodinkách obsahuje celkem 6 obrazovek. Každá obrazovka je poděděná od třídy `StatefulWidget`. Každá obrazovka má svůj objekt třídy `Model`, který v sobě uchovává data. V aplikaci je implementováno 6 tříd, které operuji nad daty a plní funkci služeb. V těchto třídách je implementována komunikace se serverem, služby pro zjišťování a zpřesňování polohy a služba pro měření času sportovní aktivity.

Všechny třídy v aplikaci, plnící roli služeb, jsou implementovány podle návrhového vzoru singleton. Díky tomu můžu všechny tyto třídy využít ke sdílení dat napříč všemi obrazovkami aplikace. Pro implementaci třídy jako singleton jsem využil flutterový plugin `get_it`[1]. V programu existuje jeden globální objekt třídy `GetIt`. Při startu se do něj registrují všechny třídy, které mají plnit roli singletonu pro aplikaci, registrace je vidět na výpisu 6.2. Pokud widget potřebuje služby, které jsou poskytovány konkrétním singletonem, tak se v metodě `initState()` přihlásí k odběru konkrétního třídy. Při opouštění obrazovky se widget odhlásí od odběru v metodě `dispose()`.

Pro snadný přechod z testovacího prostředí do produkčního jsem vytvořil soubor `app_settings.dart`. V tomto souboru je objekt třídy `Map`, který v sobě má zadefinováno základní nastavení aplikace ve formě klíč-hodnota. Tento soubor obsahuje nastavení URL pro komunikaci se serverem, zda se má použít pro komunikaci protokol `http` nebo `https` a na jaký port na serveru se má klient připojit. Soubor `app_settings.dart` je načten při startu aplikace přes instanci třídy `GlobalConfiguration`. Následně je konfigurace načtená v této instanci dostupná v celé aplikaci. Zdrojový kód pro globální konfiguraci je vidět na výpisu 6.1.

[1] https://pub.dev/packages/get_it

```
final Map<String, dynamic> appSettings = {
  "use_localhost": true,
  "use_https": false,
  "https": "https://",
  "http": "http://",
  "url": "racemaster.coreapp.cz",
  "localhost": "localhost",
  "port": 8005
};
```

Výpis 6.1: Ukázka globální konfigurace, kdy je jako server použit localhost a připojování na server je realizováno přes protokol http a port 8005.

```
GetIt getIt = GetIt.instance;

void main() {
  GlobalConfiguration().loadFromMap(appSettings);

  getIt.registerSingleton<UserService>(UserService(), signalsReady:true);
  getIt.registerSingleton<GeoService>(GeoService(), signalsReady:true);
  getIt.registerSingleton<RandomNumService>(RandomNumService(),
      signalsReady:true);
  getIt.registerSingleton<TimerService>(TimerService(), signalsReady:true);
}
```

Výpis 6.2: Ukázka zaregistrování tříd plnících roli služeb jako singletonu do instance třídy `GetIt` a načtení globální konfigurace.

6.1.1 Implementace měření sportovní aktivity

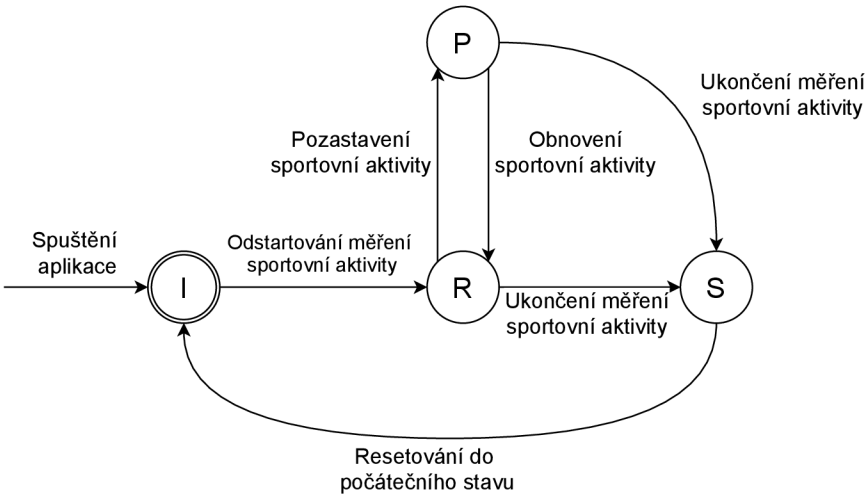
Aplikace na chytré hodinky si ve vztahu k měření sportovní aktivity projde několika stavy, které je vidět na obrázku 6.1. Po spuštění a přihlášení uživatele, které je realizováno ve třídě `UserService` a popsáno v kapitole 6.1.3 se aplikace dostane do stavu *I*. Všechny hodnoty ve třídách `GeoService` a `TimerService` jsou vynulovány.

Po interakci s uživatelem a spuštění měření sportovní aktivity se zavolá metoda `_startButtonAction()`. V těle této metody se provede ověření, zda jsou povolené a zapnuté geolokační služby a to pomocí třídy `GeoService`. Spustí se měření času, které je implementováno ve třídě `TimerService` a následně aplikace začne poslouchat data z metody `positionStream`, která je ve třídě `GeoService`. Pokud všechny tyto metody nevyvolají výjimku, tak jako poslední se zavolá metoda z `ControlButtonService`, která zajistí překreslení grafiky ze stavu *I* do stavu *R*.

Ve stavu *R* probíhá měření sportovní aktivity. Třída `GeoService` publikuje do aplikace data o aktuální poloze a následně detekuje a zpřesňuje čas průjezdu branou tak, jak je popsáno v kapitole 6.1.4. Uživatel na displej dostává informace o průběhu aktivity. Jedná se o celkový čas, který je dostupný v třídě `TimeService`, celkovou ujetou vzdálenost a pozici v závodě ze třídy `GeoService`. Po interakci s uživatelem buď dojde k ukončení měření (přesun do stavu *S*) nebo k jeho pozastavení (přesun do stavu *P*).

Ve stavu *P* je pozastaveno měření a to jak času v `TimerService` tak se neukládají hodnoty do pole s objekty třídy `Position`. I přes to, že neprobíhá sběr dat o poloze, si třída `GeoService` udržuje spojení se satelity a publikuje aktuální polohu. Je zde předpoklad, že si uživatel dává pouze krátkou přestávku například v intervalovém tréninku. Přesun ze stavu *P* je po interakci s uživatelem možný buď do stavu *R*, kde dojde k obnovení měření nebo do stavu *S*, kde se měření ukončí.

Přesun do stavu *S* znamená ukončení měření. Aplikace ponechá na displeji údaje o aktivitě a vytvoří objekt třídy `ResultData`. Ten převede na JSON pomocí třídy `HttpService` a metody `postRequest(String url, ResultData data)` jej odešle na server k synchronizaci s telefonem a upřesnění času průjezdů branami. Interakce s uživatelem vynuluje všechny naměřené hodnoty v aplikaci a dostane aplikaci zpátky do stavu *I*. Zde proces měření končí.



Obrázek 6.1: Diagram stavů, ve kterých se může aplikace nacházet. Přechod mezi stavy je realizovaný interakcí s uživatelem.

6.1.2 Komunikace se serverem

Pro komunikaci se serverem je implementována třída `HttpService`. Třída v sobě má implementovánu metodu `String createUrl(List<String> params)`, která vezme list parametrů a načtené údaje z globální konfigurace a vytvoří odpovídající URL adresu serveru.

Pro komunikaci se serverem jsou zde implementovány metody pro HTTP dotazy typu GET, POST, PUT a DELETE, Kde každá metoda je pro každý typ dotazu implementována dvakrát (výpis 6.3). První z nich vykoná dotaz na server bez položky `authorization` v hlavičce HTTP požadavku a odešle dotaz. Druhá z metod přidá do hlavičky HTTP požadavku autorizační token, pokud jej má z předchozí komunikace k dispozici. Všechny metody, zajišťující HTTP komunikaci z aplikace na server, jsou asynchronní a vyhodnocení se provádí v bloku `then`.

Tělo HTTP dotazu je tvořeno řetězcem v JSON formátu. Aby práce s objekty, které slouží jako model pro ukládání dat, byla co nejjednodušší, je v aplikaci využitý plugin `json_serialiyable`[2]. Všechny třídy v modelu aplikace mají anotaci `@JsonSerializable()`. Každá třída modelu je implementována s konstruktorem, který na vstupu má všechny třídní proměnné, do každé třídy je dopsaná metoda `fromJson(Map<String, dynamic> json)` a metoda `toJson()`. Obě tyto metody pracují se zdrojovým kódem generovaným při kompilaci a realizují převod z objektu programátorem vytvořené třídy na objekt třídy `Map`, který je serializovatelný do řetězce v JSON formátu. Druhá z uvedených metod realizuje zpětný převod z objektu třídy `Map` do objektu třídy vytvořené programátorem.

V těle kódu v bloku `then` je asynchronně počkáno na odpověď ze serveru. Ta až dorazí, tak je podle návratového kódu HTTP protokolu vykonána další akce s doručenými daty.

```
Future<Response> postRequestWithoutToken(String url, Object data) {
  return post(
    url,
    headers: <String, String>{
       'Content-Type': 'application/json; charset=UTF-8',
    },
    body: jsonEncode(data)
    );
}

Future<Response> postRequest(String url, Object data) {
  return post(
    url,
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
      'authorization': 'Bearer ' + getIt()<UserService>.loginUser.token
    },
    body: jsonEncode(data)
    );
}
```

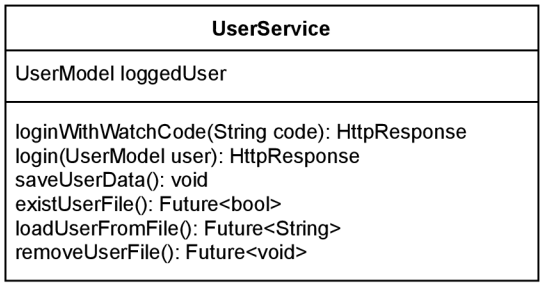
Výpis 6.3: Ukázka implementace dvou metod pro dotaz POST, kde jedna využívá data dostupná ze singletonu `UserService` získaného pomocí instance třídy `GetIt`.

[2] https://pub.dev/packages/json_serializable

6.1.3 Párování hodinek na uživatele

Proces párování hodinek k uživatelskému účtu je implementován v metodě `loginWithWachCode(String code)` v třídě `UserService` (na obrázku 6.2). Aplikace při prvním spuštění pomocí `RandomNumService` vygeneruje náhodné šesticiferné číslo. Unikátnost tohoto čísla je ověřena přes HTTP dotaz na server prostřednictvím metody `getRequestWithoutToken(String url)`. Pokud dojde ze serveru potvrzení, že jde o unikátní číslo, je uživatel vyzván, aby kód zadal do aplikace telefonu. Následně potvrdí na hodinkách, že kód zadal. Pomocí metody `loginWithWachCode(String code)` je zaslán dotaz na server. V odpovědi se vrací uživatelská data, co se uloží do proměnné `loggedUser` třídy `UserModel` nebo chybový kód.

Metoda `saveUserData()` je volána bezprostředně po úspěšném přihlášení a uložení dat do třídní proměnné `loggedUser`. Metoda využije metodu `getApplicationDocumentsDirectory()` z pluginu `path_provider`[3], kde její návratová hodnota je cesta k adresáři s daty aplikace. Zde metoda `saveUserData()` vytvoří soubor s uživatelskými daty. Při startu aplikace ověřuje metoda `existUserFile()` jeho existenci a ve spolupráci s metodou `loadUserFromFile()` a `login(User user)` je implementováno automatické přihlášení a aktualizace uživatelských dat v souboru. Metoda `removeUserFile()` je volaná při akci odhlásit a zajišťuje vymazání souboru s daty uživatele. Tím znemožňuje automatické přihlášení při dalším spuštění aplikace.



Obrázek 6.2: Struktura třídy `UserService`, která poskytuje metody pro přihlášení uživatele a párování hodinek s uživatelem.

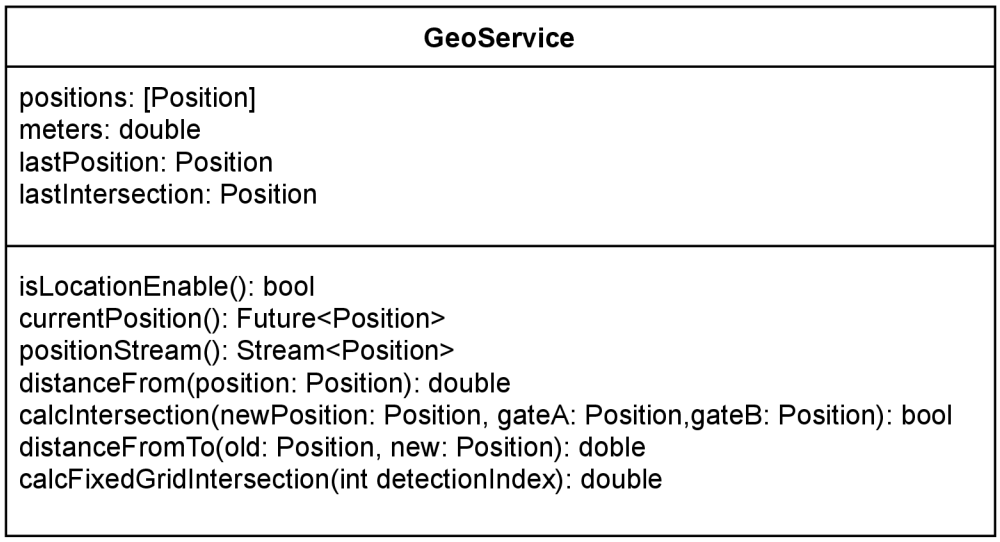
6.1.4 Zjišťování polohy a její zpřesnění

Další částí aplikace, co bylo potřeba implementovat byla práce s aktuální polohou uživatele. Jak pro systém Android tak pro operační systém Wear OS je nutné, aby uživatel povolil oprávnění přistupovat k geolokačním službám. K zobrazení povolení je nutné upravit soubor `AndroidManifest.xml`, který obsahuje XML strukturu nesoucí v sobě informace o aplikaci včetně oprávnění, kterými aplikace disponuje. Do kořenového uzlu této struktury jsem přidal následující oprávnění:

- `android.permission.ACCESS_FINE_LOCATION`,
- `android.permission.ACCESS_BACKGROUND_LOCATION`,
- `android.permission.ACCESS_COARSE_LOCATION`.

[3] https://pub.dev/packages/path_provider

Celá logika pro práci s polohou uživatele je implementována ve třídě s názvem `GeoService`, kde její struktura je vidět na obrázku 6.3. Tato třída poskytuje přístup k metodám pro zjišťování aktuální polohy, výpočtu vzdálenosti mezi body, detekci průjezdu branou a první zpřesnění času průjezdu branou.



Obrázek 6.3: Struktura třídy `GeoService`, která poskytuje metody pro práci s polohou, detekci průjezdu branou a zpřesnění času průjezdu branou.

Pro získání aktuální polohy jsem využil plugin Geolocator[4]. Moje třída `GeoService` využívá jeho instanci pro čtení bodů souřadnicového systému. Data, která z Geolocatoru získám ukládám do objektu třídy `Position` (struktura je na výpisu 6.4). Když je aplikace ve stavu R, který je na diagramu na obrázku 6.1, jsou data o poloze poskytována pomocí metody `positionStream()`. Tato metoda využívá metodu `getPositionStream()` z `Geolocator` pluginu. Jako parametr nastavuji `desiredAccuracy` na hodnotu nejlepší. Toto nastavení má za následek, že zařízení se pokusí určit aktuální polohu co možná nejpřesněji. Jako druhý parametr nastavuji `timeLimit` na hodnotu 5 vteřin. Je to čas, po kterém dojde k `TimeoutException` z důvodu nenavázaní GPS spojení a nezjištění aktuální polohy.

```
class PositionStruct {
  double latitude;
  double longitude;
  DateTime timestamp;
  double altitude;
  double accuracy;
  double heading;
  int floor;
  double speed;
  double speedAccuracy;
}
```

Výpis 6.4: Struktura třídy `Position`, která se využívá jako model pro ukládání dat o pozici.

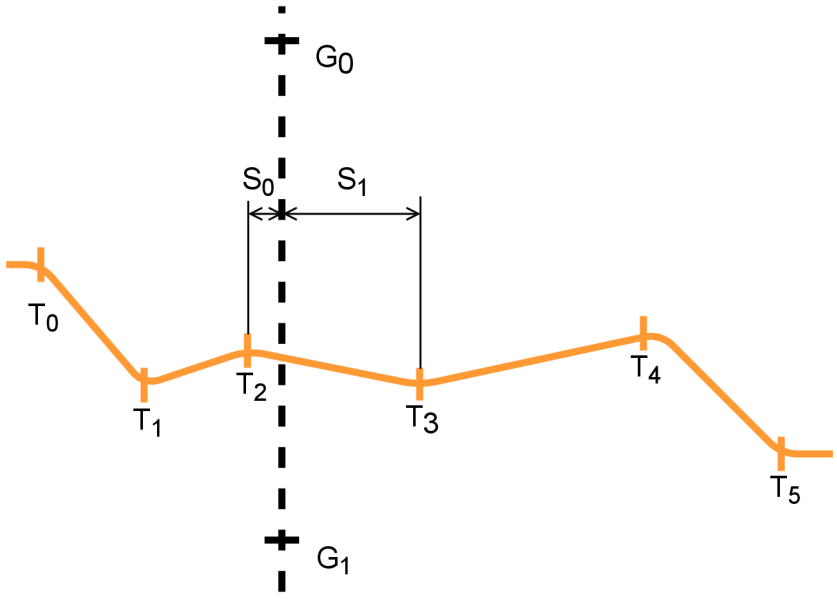
[4] https://pub.dev/packages/geolocator

Při obdržení nových souřadnic s aktuální polohou, jsou ty původní uloženy do proměnné `lastPosition`. Po jejich uložení se spustí metoda `calcIntersection()`, která má na vstupu aktuálně novou polohu a krajní souřadnice brány. Z proměnné `lastPosition` si načte předchozí hodnotu polohy uživatele. Následně je implementovaný výpočet směrového vektoru pro dvě přímky a pomocí determinantu pro dvě rovnice o dvou neznámých je stanoveno, zda existuje průsečík. Pokud je determinant různý od nuly, je ověřeno, že se nejedná o dvě rovnoběžné úsečky. Další výpočet je ověření, zda bod, co vyšel jako průsečík, leží v obdélníku, který je definován body polohy a body pozice brány. Pokud vyjde, že průsečík leží v této oblasti, pak aplikace vyhodnotí, že došlo k průjezdu branou a spustí metodu *calcFixedGridIntersection()* pro zpřesnění času průjezdu. Jinak nedělá nic. Tento způsob dopočtu průjezdu branou je základní a je využíván i v mobilní aplikaci.

Zpřesnění času průjezdu branou pomocí metody `calcFixedGridIntersection()` je založeno na předpokladu, že se uživatel na krátké vzdálenosti své trajektorie pohybuje konstantní rychlostí. Ve třídě `GeoService` jsou ukládány všechny hodnoty o poloze včetně času. Následně lze tedy modelovat situaci jako je na obrázku 6.13. Oranžová čára je trajektorie pohybu, kde v bodech T_x znám souřadnice a čas, kdy v nich byl uživatel. Body G_0 a G_1 jsou krajní body průjezdové brány. Hodnoty v S_0 a S_1 jsou vzdálenosti mezi průsečíkem brány s trajektorií pohybu a prvním bodem před branou a za ní. Naměřené údaje mi umožňují využít následující vzorec:

$$t_I = (T_3 - T_2)\frac{S_0}{S_0 + S_1} + T_2 \tag{6.1}$$

kde výsledné t_I je zpřesněný čas průjezdu na základě prvního bodu před bránou a prvního bodu za bránou.



Obrázek 6.4: Schéma dopočtu času průjezdu branou na základě dvou bodů a poměru vzdáleností mezi body a průsečíkem s branou. Oranžová čára je trajektorie pohybu, kde v bodech T_x znám souřadnice a čas, kdy v nich byl uživatel. Body G_0 a G_1 jsou krajní body průjezdové brány. Hodnoty v S_0 a S_1 jsou vzdálenosti mezi průsečíkem brány s trajektorií pohybu a prvním bodem před branou a za ní.

Tabulka 6.1: Barva pozadí v závislosti na pozici v závodě.

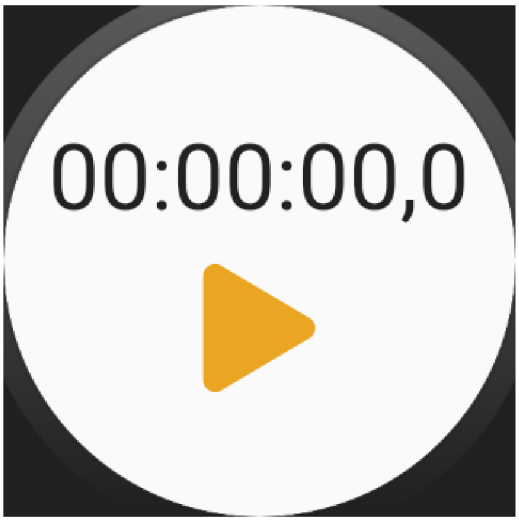
umístění v závodě	**barva pozadí aplikace**
1. místo	zlatá (#FFD700)
2. místo	stříbrná (#C0C0C0)
3. misto	bronzová (#cd7f32)
4. - 10. misto	zelená (#32CD32)
první polovina	modrá (#4169E1)
druhá polovina	tmavě rudá (#500000)

6.1.5 Tvorba uživatelského rozhraní aplikace na chytré hodinky

Doposud představené principy fungování vybraných služeb v aplikaci pro měření sportovních aktivit jako jsou tréninky a klubové závody. Výsledky z těchto služeb jsou následně prezentovány na několika obrazovkách v aplikaci. Každá obrazovka je reprezentována svou třídou, která je poděděna od třídy `StatefulWidget`. Jednotlivé komponenty obrazovky jsou rozděleny do tříd (Widgetů), které jsou znovupoužitelné na jakékoliv další obrazovce v aplikaci.

Obrazovka Home

Tato obrazovka ve výchozím stavu (stav I z obrázku 6.1) obsahuje pouze jedno tlačítko a vynulovanou časomíru. Na tlačítko je navázaná akce, která když žádná z jejich operací nevyvolá výjimku, tak provede přechod ze stavu I do stavu R znázorněných na obrázku 6.1.



Obrázek 6.5: Domácí obrazovka ve stavu po přihlášení do aplikace před spuštěním měření.

Abych na malém displeji hodinek předal maximální množství informací současně, implementoval jsem do obrazovky Home widget třídy `Stack`. Tím jsem umožnil, že můžu využít barvu pozadí k předávání informace o aktuální pozici v závodě. Když je aplikace ve stavu R a dojde k detekci průjezdu branou proběhne zaslání informací o průjezdu na server. V odpovědi ze serveru dorazí ztráta na první místo a pozice. Ve třídě `PositionService` dojde k vyhodnocení odpovědi a překreslení pozadí na základě umístění v závodě na barvu podle

tabulky 6.1. Barvy na prvních třech místech respektují zažité zvyky ze závodního prostředí, kdy reflektují barvy cenných kovů, ze kterých se vyrábějí medaile. To dovoluje závodníkům rychle číst informaci z displeje [23].

```
class SwipeGestureDetector extends StatefulWidget {

  Function up;
  Function down;

  SwipeGestureDetector({this.up, this.down});

  @override
  _SwipeGestureDetectorState createState() => _SwipeGestureDetectorState();
}

class _SwipeGestureDetectorState extends State<SwipeGestureDetector> {
  @override
  Widget build(BuildContext context) {
    return Stack(
      children: [
        _down(),
        _up(),
        _right()
      ],
    );
  }
}
```

Výpis 6.5: Vlastní třída `SwipeGestureDetector` poděděná od třídy `StatefulWidget`. V metodě `Widget build(BuildContext context)` se vytváří instance třídy `Stack` s metodami, které v sobě zapouzdřují implementaci `GestureDetector` z výpisu 6.6.

Jako problematické se zde ukázalo původní přepínání na další z Home obrazovky na pak a zpět. Toto bylo původně realizované dotykem na horní a dolní část displeje. Uživatelé (sportovci – in-line rychlobruslaři) využívají k ochraně při pádu rukavice. Rukavice dělají dotyk nepřesný a při testování to často vedlo k nechtěnému ukončení měření. Jako řešení tohoto problému s uživatelským rozhraním se nabízelo místo komplikovaného přesného dotyku na displej použít gesto. Pro toto řešení jsem vytvořil nový widget `SwipeGestureDetector`, který v sobě zapouzdří widget `Stack`. Ve widgetu `Stack` jsou na sobě umístěny 3 instance třídy `GestureDetectoru`, kde v callback metodě `onPanUpdate` je detekováno jakým směrem swipe gesto šlo.

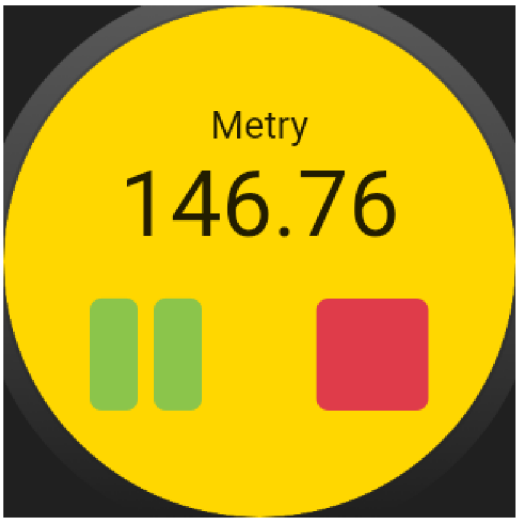
```
GestureDetector(
  onPanUpdate: (details){
    if (details.delta.dy > 0) {
      widget.down();
    }
  },
```

```
  child: Container()
)
```

Výpis 6.6: Detekce v jedné z instancí třídy `GestureDetector` ve widgetu `SwipeGestureDetector`. Gesto tažení směrem dolů a následné provolání metody `widget.down`, která byla předána jako vstupní parametr do třídy `SwipeGestureDetector`.

Obrazovky ujeté vzdálenost a pozice v závodě

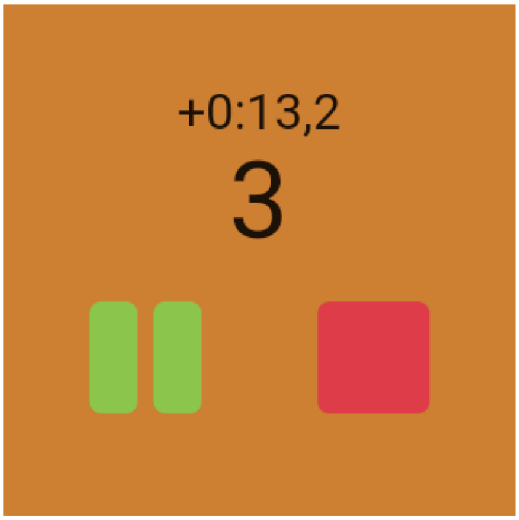
Jedná se o další dvě obrazovky, které předávají informaci uživateli (závodníkovi) během výkonu sportovní aktivity. Přepnutí na ně je realizované pomocí třídy `SwipeGestureDetector`. Aplikace na hodinkách ve stavu `R` znázorněném na diagramu, který je na obrázku 6.1, dává uživateli k dipozici dvě tlačítka (obrázek 6.6). Tlačítka jsou umístěna co nejvíc od sebe aby nedocházelo ke špatné detekci stisknutí ve sportovních rukavicích.



Obrázek 6.6: Hodinky ve stavu *R* znázorněném na diagramu (obrázek 6.1). Pozadí informuje závodníka o tom, že na posledním průjezdu branou je na prvním místě. Text na hodinkách předává informaci o vzdálenosti od startu do současnosti.

Poslední obrazovka, prezentující data o aktuálně probíhané sportovní aktivitě, je ta, která nese informace o pozici v závodě a ztrátě na první místo (obrázek 6.7). Aby byl text dobře čitelný ne jen na kulatém displeji, ale také na hranatém, je celá kontextová část zapouzdřená ve třídě `Center`.

Všechny obrazovky prezentující informace o aktuálně probíhané sportovní aktivitě spolu sdílí unifikovaný vzhled. To umožňuje znovupoužití tříd widgetů poděděných od třídy `StatefulWidget` nebo `StatelesWidget`. Takto použitá je v aplikaci třída `ControlButton`. Ta má na starosti zobrazení tlačítek, které vyvolávají přechod mezi jednotlivými stavy aplikace. Třída `ControlButton` volá v sobě metody ze třídy `ControlButtonService`, pomocí kterých dochází k notifikaci dalších služeb aplikace.



Obrázek 6.7: Hodinky ve stavu R znázorněném na diagramu (obrázek 6.1). Pozadí informuje závodníka o tom, že na posledním průjezdu branou je na třetím místě. Text na hodinkách předává informaci o pozici a ztrátě na první místo.

Obrazovka pro párování s telefonem

Obrazovka pro párování s mobilním zařízením se zobrazí pouze tehdy, když se jedná o první spuštění aplikace, nebo se uživatel z aplikace odhlásil. Při vykreslování obrazovky, která je na obrázku 6.8, je v metodě `initState()` ve třídě `OpenScreen` vygenerován pomocí třídy `RandomNumService` a ověřen dotazem na server unikátní náhodný šesticiferný kód. Po zadání do aplikace v telefonu a potvrzení úkonu na hodinkách dojde ke spárování hodinek s uživatelem.

6.2 Implementace mobilní aplikace

Aplikace pro chytrý telefon rozšiřuje funkcionalitu z hodinek. Oproti aplikaci na chytré hodinky je zde navíc implementována podpora pro operační systém iOS, mapové služby využívající knihovnu pro MapBox[5] mapy, obrazovka zobrazující detail sportovní aktivity a „nekonečný“ list přehledu sportovních aktivit.

Pro přepínání mezi obrazovkami je v samostatném souboru `rounting` implementována metoda spolu s konstantami, nutnými pro přepínání obrazovek s podporou animace. Pro vybranou komunikaci se serverovou aplikací je implementován websocket. Mobilní aplikace navíc oproti hodinkám disponuje zpřesňováním času průjezdu branou pomocí lineární regrese.

6.2.1 Zjišťování polohy

Pro zjišťování polohy na mobilních telefonech s operačním systémem Android jsem nastavil oprávnění v souboru `AndroidManifest.xml` úplně stejně, jako jsem popsal v kapitole 6.1.4 pro aplikaci na hodinky.

[5] https://www.mapbox.com/

Obrázek 6.8: Párování hodinek s uživatelem v aplikaci na chytrých hodinkách. Aplikace vygeneruje kód, co je zadaný v aplikaci na telefonu. Po potvrzení dojde ke stažení uživatelských dat potřebných k přihlášení do hodinek.

Pro telefony s operačním systémem iOS jsem musel nastavit v souboru s XML strukturou `Info.plist`, který slouží k nastavování uživatelských oprávnění v operačním systému iOS [14] [5]. Jako nový uzel jsem přidal klíč `NSLocationAlwaysAndWhenInUseUsageDescription`, kde hodnota klíče byl důvod, proč vyžaduji oprávněni k přístupu ke službám o poloze zařízení. Tento důvod se zobrazí uživateli, který jej musí pro správnou funkčnost aplikace schválit. Následná práce s polohou zařízení je už pro oba mobilní operační systémy shodná.

6.2.2 Lineární regrese

Mobilní aplikace po dokončení sportovní aktivity provádí dopočet pomocí matematické lineární regrese a to přímo v telefonu uživatele. Pro tyto účely je využit plugin `ml_algo`[6]. Pro výpočet dvojce čísel, představující parametry přímky, si vytvořím objekt ze třídy `DataFrame`. V konstruktoru této třídy jsou vstupními parametry dvě pole, kde první z nich představuje hodnoty pro osu X (zeměpisná šířka, respektive zeměpisná délka) a druhé pro osu Y (čas naměřený v bodu). Takto vytvořený objekt se použije jako parametr pro konstruktor třídy `LenearRegressor`. Z takto vytvořeného objektu čtu hodnotu proměnné `coefficients`, kde jsou uložena čísla pro parametrický popis přímky.

Pro zpřesnění se provede výpočet lineární regrese dvakrát a to v prvním případě na závislost mezi zeměpisnou šířkou a časem v něm naměřených a podruhé na závislost mezi zeměpisnou délkou a časem v něm naměřených. Pro výpočet se použije 6 naměřených bodů před detekcí průjezdu branou a 6 naměřených bodů po detekci průjezdu bránou. Výsledkem jsou dvě dvojice čísel, které popisují parametricky přímku funkce času v závislosti k zeměpisné šířce, resp. zeměpisné délce.

Do parametrických rovnic přímky, vytvořených pomocí parametrů získaných z lineární regrese, dosadím zeměpisnou délku, respektive zeměpisnou šířku, a vypočítám dva časy

[6] https://pub.dev/packages/ml_algo

Tabulka 6.2: Tabulka se zaznamenanými body průjezdu z telefonu zobrazující vstupní data pro aplikování lineární regrese. Modrý řádek značí bod, kdy byl detekován průjezd. Dopočet se provádí z 6 bodů před detekci a šesti bodů za detekcí průjezdu.

index do pole hodnot	zeměpisná délka	zeměpisná šířka	čas průjezdu ve vteřinách
-6	49.1539405	16.8781919	8.555
-5	49.153934	16.8781512	9.289
-4	49.1539342	16.8781461	9.376
-3	49.1539351	16.878126	9.751
-2	49.153938	16.8781148	9.924
-1	49.1539414	16.8781037	10.139
bod detekce průjezdu	49.1539475	16.8780946	10.336
1	49.1539519	16.8780844	10.566
2	49.1539602	16.8780765	10.796
3	49.1539666	16.8780681	10.983
4	49.1539808	16.8780509	11.392
5	49.1539871	16.8780397	11.619
6	49.1539905	16.8780276	11.834

průjezdů, které následně zprůměruji. Tato funkcionalita je implementována ve třídě `LinearRegresionService`.

Popis zpracování dat lineární regresí v telefonu

Po dojetí závodu má aplikace k dispozici pole naměřených hodnot ze kterého zvládne určit bod detekce průjezdu bránou. Tato detekce průjezdu probíhá v telefonu úplně stejně jako na hodinkách a je popsána v kapitole 6.1.4. Bod detekce průjezdu je v tabulce 6.2 označen modrým řádkem.

Lineární regresí získám dvojici koeficientů A a B, které popisují parametrické vyjádření přímky. Ty dosadím do rovnice pro lineární funkci:

$$y = A + Bx \tag{6.2}$$

kde x je následně buď zeměpisná šířka nebo délka. Výsledné y je čas, kdy bylo zeměpisné šířky (respektive délky) dosaženo.

Pokud bych body z tabulky 6.2 spolu s přímkou, kterou získám aplikováním lineární regrese, zanesl do grafu, vypadalo by to jako na obrázku 6.9. Výsledný vzorec pro přímku popisující vztah zeměpisné délky k času vypadá pro hodnoty z tabulky 6.2 následovně:

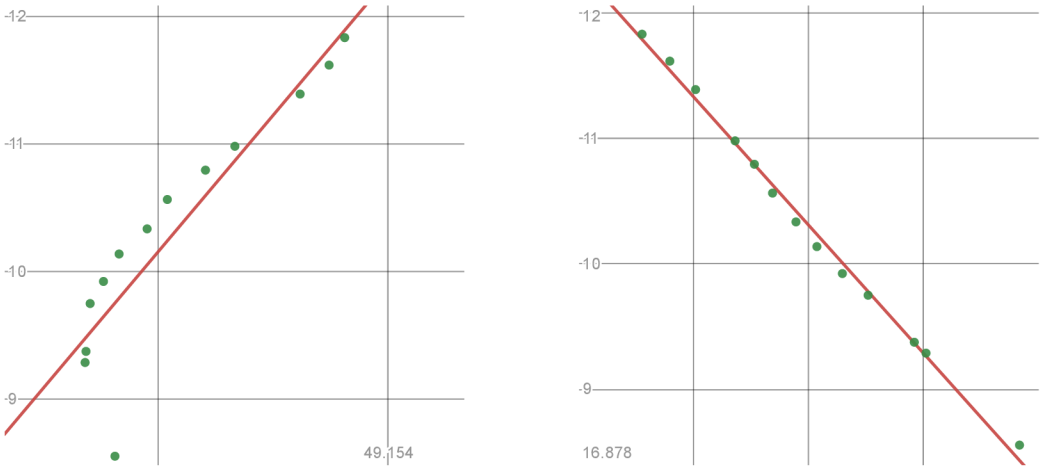
$$y = 42737.752473498x - 2100719.1874441 \tag{6.3}$$

kde konstanty 42737.752473498 a -2100719.1874441 jsou získány jako výsledek lineární regrese. Vzorec pro přímku popisující vztah mezi časem a zeměpisnou šířkou pro data uvedená v tabulce 6.2 má následující podobu:

$$y = 345131.67656424 - 20447.880053525x \tag{6.4}$$

kde konstanty 345131.67656424 a -20447.880053525 jsou získány jako výsledky lineární regrese.

Protože znám souřadnice brány a první bod před branou a první za branou, můžu dopočítat souřadnice pravděpodobného bodu, kde závodník projel branou. Dopočet je realizován stejně, jako v aplikaci na hodinkách, a je popsán v kapitole 6.1.4. Souřadnici průsečíku popisující zeměpisnou délku, respektive šířku, dosadím do vzorce pro přímku času a zeměpisné délky, respektive šířky. Výsledkem jsou dvě hodnoty času, kdy došlo k průjezdu branou. Tyto hodnoty zprůměruji a získám tak výsledný čas průjezdu.



Obrázek 6.9: Vlevo vynesený graf závislosti času na zeměpisné délce spolu s přímkou, která vznikne aplikováním lineární regrese. Vpravo vynesený graf závislosti času na zeměpisné šířce spolu s přímkou, která vznikne aplikováním lineární regrese.

6.2.3 Komunikace se serverem po webSocketu

Pokud klient potřebuje získat data ze serveru je k tomu určená třída `HttpService`. Nevýhodou tohoto řešení je, že komunikaci musí vždy inicializovat klient. To bylo potřeba vyřešit na obrazovce zabývající se zobrazováním informací o závodníkem absolvovaných závodech a o závodech jeho přátel, které sleduje. Tyto informace jsou totiž uloženy na serveru a periodické dotazování se z klienta na server, zda náhodou není k dispozici aktualizace dat, je zatěžující jak pro server, tak pro klienta.

Z tohoto důvodu jsem naprogramoval třídu `SocketService`. Tato třída služeb je jako ostatní třídy registrována pomocí pluginu `get_it` jako singleton a využívá plugin `websocket`[7] pro navázání komunikace po socketu. Navázání komunikace probíhá ihned po startu aplikace a následná aktualizace dat, která přichází ze serveru, se děje na pozadí pomocí asynchronní metody `listen`.

6.2.4 Router pro přepínání obrazovek

Z důvodu vyššího počtu obrazovek, jsem implementoval metodu `rounting(setting)`, která je vidět na výpisu 6.7. V těle metody je implementován (switch), který na základě parametru `name` z objektu `setting` rozhoduje na kterou stránku bude přepnuto. Objekt `setting` v sobě uchovává argumenty, které mohou být využity jako vstup v konstruktoru obrazovky kam přepínám. V implementaci metody `routing` transformuji položku `setting.arguments` na objekt třídy `Map`. Následně mohou k parametrům přistupovat přes jejich klíč.

[7] https://pub.dev/packages/websocket

```
routing(setting) {
  switch (setting.name) {
    case INIT:
      return switchPage(HomeScreen(), setting);
      break;
    case HOME_SCREEN:
      return switchPage(HomeScreen(), setting);
      break;
    case LOGIN_SCREEN:
      return switchPage(LoginScreen(), setting);
      break;
    case ITEM_SCREEN:
      Map map = Map<String, dynamic>.from(setting.arguments);
      return switchPage(LoginScreen(id: map['id'])), setting);
      break;
  }
}

PageTransition switchPage(pageToPeon, RouteSettings setting) {
  return PageTransition(
      settings: setting,
      child: pageToPeon,
      type: PageTransitionType.fade,
      duration: Duration(milliseconds: 300));
}
```

Výpis 6.7: Ukázka medoty `rounting` pro přepínání mezi obrazovkami a metody `switchPage`, přidávající animaci přechodu mezi obrazovkami. Ukázka práce s argumenty v proměnné `setting`.

Implementace metody `rounting` v souboru mimo hlavní metodu `main()` umožňuje její lepší testování pomocí widget testů. Provolání metody `routing` je v parametru `OnGenerate-Route` při volání konstruktoru třídy `MaterialApp`. Stejným způsobem je metoda `rounging` připojena do widget testů, kde takto testuji přepínání mezi obrazovkami. Využití konstant pro přepínání mezi obrazovkami zase vede k menší chybovosti při psaní routy.

Implementování metody `switchPage` mi umožnilo spravovat animace pro celou aplikaci z jednoho místa. Pro animaci jsem využil plugin page_transition[8]. Metoda `swichPage` má na vstupu instanci obrazovky, která je cílová a kam má přepnout. Dále proměnou typu `RouteSettings`, která předává informace odkud a kam je přepínáno a v případě portování na web umožňuje mít funkční tlačítko zpět v prohlížeči. Další parametry, co ve funkci nastavuji je typ animace (`PageTransitionType.fade`) a její trvání v milisekundách.

6.2.5 Jazykové mutace

Mobilní aplikace obsahuje podstatně větší množství textu než hodinková. Proto jsem se rozhodl implementovat třídu `AppLocalizations`. Tato třída rozpozná systémově nastavený

[8] https://pub.dev/packages/page_transition

jazyk v telefonu a vybere vhodný překladový soubor, který je uložen v `assets/json/`. Pokud nenajde překladový soubor pro jazyk, který je použit v systému, tak je nastaven jako výchozí jazyk aplikace angličtina. Překlad probíhá přes statickou metodu `translate()`, která na vstupu očekává klič ukazující do překladových JSON souborů.

6.2.6 Uživatelské rozhraní

Mobilní aplikace obsahuje několik obrazovek. Jedná se například o obrazovku pro registraci uživatele a jeho přihlášení, obrazovku pro párování s hodinkami, nebo obrazovku nastavení. V této kapitole si rozebereme tři nejpodstatnější obrazovky a to hlavní obrazovku (tzv. Home), obrazovku s výsledky závodu a obrazovku předávající informace o výkonech ostatních uživatelů (tzv. feed).

Home

Tato obrazovka se ukáže ihned po přihlášení do aplikace, případně, pokud se jedná o opakované spuštění a přihlašovací údaje jsou uložené v telefonu, tak se zobrazí ihned po startu aplikace. Obrazovka využívá služeb třídy `MapService`, která ovládá funkce poskytované pluginem `mapbox_gl`[9].

V konstruktoru třídy `MapService` aplikuji token získaný při registraci na stránkách poskytovatele map. Následně se nastaví kamera pomocí třídy `CameraPosition`, které předám získané souřadnice o aktuální poloze a nastavím počáteční hodnotu pro zoom. Pokud závodník vybere ze seznamu závod, tak pomocí třídy `Polyline` vykreslím do mapy čáry, představující kontrolní brány v závodu. Při pohybu během závodu je pomocí třídy `Polyline` vykreslována také trajektorie pohybu závodníka.

Obrazovka s výsledky závodu

Aplikace pro mobilní telefon umožňuje navíc oproti hodinkám zobrazit informace o absolvovaném závodu. Obrazovka (obrázek 6.10) je vytvořena z několika tříd, které dědí od `StatefulWidget` třídy. Pro získání dat o pozici a času na průjezdech jednotlivými branami se využije třída `HttpService`, která pomocí metody `getRequest()` získá ze serveru řetězec v JSON formátu. Ten je následně pomocí funkce `fromJson()` převeden na objekt třídy `RaceResult`. Objekt je předán příslušným třídám zajišťujícím zobrazení dat, které po jeho obdržení provolají metodu `setState()`. To má za následek zobrazení obdržených dat na displeji zařízení.

Feed

Obrazovka Feed je naprogramovaná ve třídě `FeedWidget`, která dědí vlastnosti od třídy `StatefulWidget`. Jejím ústředním prvkem je instance třídy `InfinityScrollList`. Jedná se o vlastní implementaci nekonečného seznamu položek v rolovacím seznamu. Třída `InfinityScrollList` využívá služeb třídy `SocketService` pro získávání dat. Pro detekci zahájení stahování nových dat ze serveru je ve třídě `InfinityScrollList` implementována metoda `_listenToScrollChange`. Ukázka kódu zajišťujícího načítání dat pomocí třídy `SocketService` je vidět na výpisu 6.8.

[9] https://pub.dev/packages/mapbox_gl

Obrázek 6.10: Detail dokončeného závodu. Uživatel obdrží informace o svém umístění v cíli, ztrátě na první místo, celkovém času a absolvované vzdálenosti. Dále dostane informace o časech na jednotlivých kontrolních bodech, tzv branách.

Ve třídě, zajišťující zobrazení feedu dat, je v metodě `initState()` volána metoda `loadData()`. Tato metoda poslouchá na socketu a třídí data, která server posílá přímo pro ni. Jedná se o aktualizace dat, kde komunikaci zahajuje server a klient je pouze v roli posluchače.

6.3 Implementace webové aplikace

Webová aplikace navazuje implementačně na mobilní aplikaci, ze které přebírá služby a objekty pro zobrazení dat o závodu. Z implementace vypouští GPS služby, protože není předpokládáno jejich využití přes webový prohlížeč. Cílem webové aplikace je poskytnout pohodlné klikací prostředí pro plánování závodů na mapě.

6.3.1 Mapové služby

Pro vykreslení map a jejich využití jsem vytvořil třídu `MapService`. Tato třída využívá plugin `mapbox_gl`[10]. Pro zobrazení map je potřeba registrace na stránkách[11], kde je získán přístupový token.

[10] https://pub.dev/packages/mapbox_gl
[11] https://www.mapbox.com/

```
void _requestData(RequestData requestData) {
  _currentCountNum = 0;
  _page = _page + 1;
  String jsonString = jsonEncode(requestData.toJson());
  getIt<SocketService>().requestData(jsonString);
}

void _listenToScrollChange() {

  if ((_scrollController.position.pixels > _scrollController.position.
      maxScrollExtent * 0.9 && !_loading) && !_all) {
    requestData = RequestData(request: "data", filter: widget.category.
        backEndTranslate, ofset: _page, page: _pageSize);
    _requestData(requestData);
    setState(() {
      _loading = true;
    });
  }
}
```

Výpis 6.8: Ukázka metody `_listenToScrollChange`, která vytvoří dotaz a zahájí načítání ve chvíli, kdy pomocí instance třídy `ScrollController` detekuje, že se uživatel dostal do spodních 10% obrazovky.

V konstruktoru třídy `MapService` aplikuji získaný token. Následně nastavím kameru pomocí třídy `CameraPosition`, které předám získané souřadnice o aktuální poloze a nastavím počáteční hodnotu pro zoom. Při kliknutí do mapy jsou z mapy získány souřadnice. Jakmile existuje dvojce souřadnic, je tato dvojce použita jako jeden z argumentů pro kontruktor třídy `Polyline`. Ta provede zakreslení čáry představující bránu do mapy.

6.4 Implementace serverové aplikace

Serverová aplikace je implementována v programovacím jazyce Java pomocí frameworku Spring Boot. Aplikace je rozdělena do balíků `model`, `controller`, `service` a `repository`.

Třídy v balíku `model`, jsou využívány jak pro zachycení databáze, tak pro tvorbu objektů, které jsou následně odesílány nebo přijímány přes interní a externí komunikační rozhraní. Třídy v balíku `controller` tvoří komunikační body pro ostatní moduly a aplikace. Tyto třídy v sobě volají metody tříd z balíku `service`, kde je implementována práce s daty.

6.4.1 Komunikační a registrační modul

Pro interní komunikaci mezi moduly serverové aplikace se využívá REST API a program Apache Kafka spolu s programem Apache ZooKeeper[12].

Aby každý modul mohl běžet samostatně na fyzickém zařízení, je potřeba ho nejprve registrovat do Apache Zookeeper. Ten si vytvoří záznam obsahující IP adresu, port a název

[12] https://zookeeper.apache.org/

modulu. Následně je schopný o něm informovat ostatní moduly. Připojení jak uživatelského modulu, tak závodního modulu do programu Apache Zookeeper je definován v souboru `application.properties` pomocí následující proměnných:

- `spring.application.name`,
- `spring.cloud.zookeeper.connect-string`,
- `spring.cloud.zookeeper.discovery.enabled`.

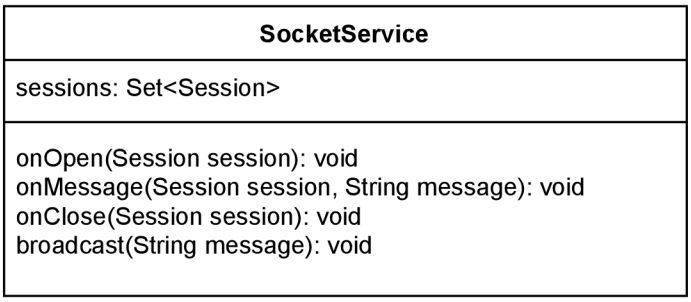
První proměnná definuje jméno, pod kterým bude modul do registrátoru Apache ZooKeeper přihlášen. Druhá oznamuje modulu na jaké URL adrese a portu má program Apache ZooKeeper hledat. Poslední povoluje zaslání informací na ostatní moduly registrované v programu Apache Zookeeper.

Pokud chce modul komunikovat s jiným po REST API, je potřeba se nejprve zeptat programu Apache ZooKeeper, kde modul je. To se provede pomocí mnou naprogramované metody `getModuleInfo()` ze třídy `DiscoverService`. Tato metoda vrací objekt obsahující IP adresu a port. To je následně spolu s daty předáno do třídy `HttpService`, kde je dokončeno vytváření cílové URL adresy následné odeslání dat.

Pro interní komunikaci mezi moduly je využit také program Apache Kafka. Přes tento modul je v režimu producent-konzument zasíláno větší množství dat. Uživatelský modul má třídu `KafkaService`, která obsahuje metodu `userPublisher()`. Tato metoda zastává roli producenta a dává k dispozici všechny uživatele, co jsou aktuálně přihlášení do aplikace. Tyto uživatele čte závodní modul pomocí metody `userConsument()` ve třídě `KafkaService`. Závodní modul je producentem zpráv obsahující výsledky závodů. K jejich zasílání do programu Apache Kafka využívá metodu `resultPublisher()` ve třídě `KafkaService`. Konzumentem těchto zpráv je API modul, který je pomocí třídy `WebSocketService` převádí z programu Apache Kafka do websocketu, přes který je odesílá do právě připojených zařízení, které jsou identifikované jako mobilní telefony.

6.4.2 WebSocket pro komunikaci s klienty

Protože aplikace napsané ve programovacím jazyku Dart pomocí frameworku Flutter neumí komunikovat s programem Apache Kafka, musel jsem implementovat komunikační most, který by data obdržená z programu Apache Kafka posílal dál. Jako tento komunikační most slouží třída `SocketService`, která je vidět na obrázku 6.11.



Obrázek 6.11: Schéma třídy `SocketService` s metodami pro zahájení komunikace, komunikaci, ukončení komunikace a pro hromadné odesílaní zpráv.

Při připojení klienta na komunikační socket se zavolá metoda `onOpen`. V jejím těle se provede přidání session do třídní proměnné `sessions`, která je typu `Set<Session>`.

Pokud je přijata zpráva pomocí metody `OnMessage` tak dojde k dekódování požadavku a následně k předání konkrétní třídě, která má na starost jeho vyřízení. Metoda `onClose` reaguje na odpojení klienta a odebírá jeho session z proměnné `sessions`. Metoda `broadcast` provede rozeslání zprávy, kterou má na vstupu, na všechny položky, které má aktuálně uložené v množině `sessions`.

6.4.3 Filtrace dotazů na endpointy

Filtraci dotazů na externí koncové komunikační body serveru provádí třída `WebSecurityConfig`, která dědí od třídy `WebSecurityConfigurerAdapter`. Ve třídě je implementována metoda `configure`, kde na vstupu je proměnná typu `HttpSecurity`. Filtrace je realizována pomocí nadefinovaní white listu koncových komunikačních bodů, na které lze přistoupit bez vlastnictví autorizačního tokenu. Všechny ostatní koncové komunikační body, které nejsou přidány do proměnné typu `HttpSecurity`, vyžadují mít v hlavičce požadavku zaslaného od klienta autorizační token.

6.4.4 Restové Cotrollery

Třídy plnící roli kontrolérů jsou v balíku `controler`. Pokud třída plní roli kontroleru pro komunikaci po REST API, má tato třída anotaci `@RestController` a `@RequestMapping("/api")`, kde řetězec `"/api"` definuje část URL, pod kterým je třída dostupná. Takto označená třída musí obsahovat alespoň jednu metodu, implementující metodu GET, POST, PUT nebo DELETE. Implementaci metody typu GET provádím pomocí anotace `@GetMapping("/url")` a to přímo nad příslušnou metodou. Obdobně implementuji metody pro POST, PUT a DELETE.

6.5 Vyhodnocení metod pro zpřesňování času průjezdu branou

V aplikaci jsou celkem tři metody, které slouží ke zjištění co nejpřesnějšího času průjezdu závodníka kontrolní bránou. První metoda se soustředí na první bod po průjezdu a v průběhu vývoje se stala detekční metodou průjezdu. Druhá metoda pracuje s poměrem mezi vzdáleností od průsečíku trajektorie pohybu závodníka s kontrolní bránou. Třetí metoda implementuje lineární regresi.

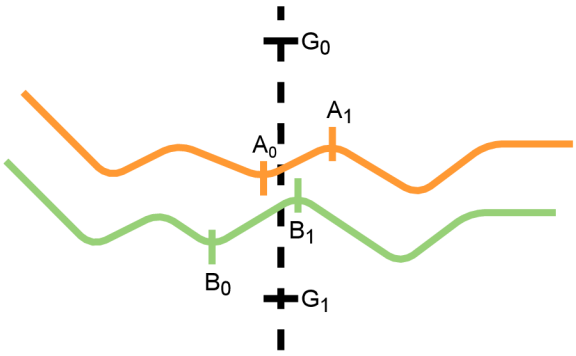
Testování přesnosti bylo realizováno pomocí dvou stejných telefonů s operačním systémem android. Na obou telefonech byla spuštěna mobilní verze aplikace. Tester měl za úkol opakovaně s oběma telefony současně projít předem stanovenou trasu s testovacími branami. Telefony ukládaly naměřená data na server do databáze a data byla následně vyhodnocena a to postupně všemi třemi metodami.

6.5.1 Metoda 0 – detekce průjezdu

Na obrázku 6.12 je znázorněna situace, kdy telefon A a telefon B překonaly bránu ve stejný čas, ale telefon B získal souřadnici B_1 blíž k bráně v lepším čase než telefon A souřadnici A_1. To uměle zvýhodní telefon B.

Chybovost v měření času průjezdu bránou touto metodou se v extrémně špatných případech pohybovala na hranici těsně pod 3 vteřinami. V extrémně dobrých případech těsně

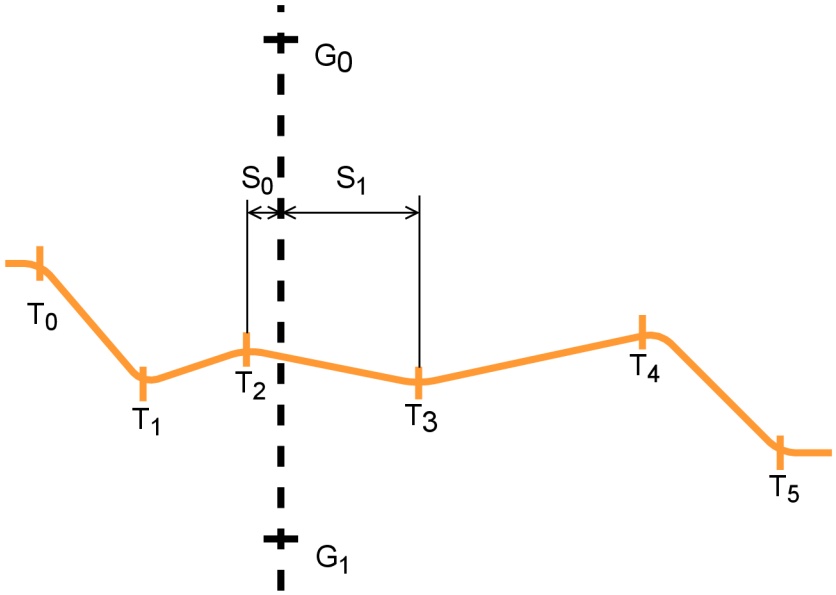
nad hodnotami 0.1 vteřiny. Střední hodnota chyby se po několika měřeních zastavila na hodnotě mezi 0.15 a 0.55 vteřiny, jak je vidět na obrázku 6.14 pro hodnotu *Metoda 0*.



Obrázek 6.12: Schéma situace, kdy telefony A a B překonají bránu ve stejný čas, ale kvůli získání času a polohy v různých místech mají rozdílný čas průjezdu.

6.5.2 Metoda 1 – dopočítání z poměrů

Tato metoda se snaží vyřešit problém vzdálenosti času a bodu od průsečíku s branou, který byl uvedený v předchozí kapitole 6.5.1. Toto se snažím vyřešit tím, že do výpočtu zpřesnění polohy zahrnuji více bodů. Pokus zahrnoval postupné spojení bodů a výpočty pro data z dvojce T_2 a T_3, T_1 a T_4, T_0 a T_5 až do vzdálenosti 6 bodů před bránu a 6 bodů za bránu.



Obrázek 6.13: Spojování dvojce bodů pro výpočet poměru mezi vzdáleností a časem mezi body před branou a za branou a průsečíkem průjezdu.

Při tomto pokusu se v extrémně špatném případě měření pohybovala maximální chyba pod hodnotu jedné vteřiny. Při uplatnění algoritmu, popsaného v kapitole 6.1.4 je střední

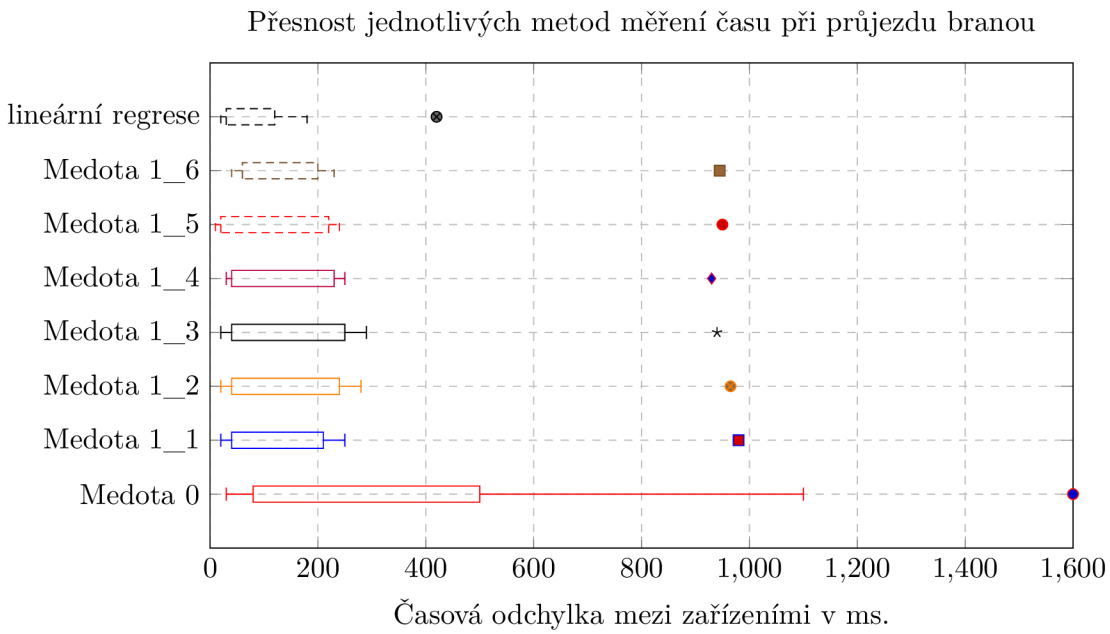
naměřená chyba těsně nad 0.2 vteřinami. Tato hodnota je pro orientační měření během závodu dostatečná. Pokud by se počítal čas průjezdu z 6. bodu před branou a 6. bodu za branou, docházelo by k přesnosti na rovném úseku pod 0.2 vteřiny. Graf pro všech šest variant je vidět na obrázku 6.14, položka *Metoda 1_1* značí první bod před a první bod za branou, *Metoda 1_2* druhý bod před a druhý bod za branou. Analogicky to platí až po položku *Metoda 1_6* v grafu.

6.5.3 Metoda 2 – aplikování lineární regrese

Metoda zpřesňování pomocí lineární regrese, která je popsaná v kapitole 6.2.2, dosahuje nejpřesnějších výsledků v porovnání s ostatními metodami. Nejvyšší chyba, která byla při experimentech naměřena, dosahovala přibližně 0.42 vteřiny. Jinak byla nejčastěji měřena chyba v hodnotách od 0.03 do 0.12 vteřiny. Na obrázku 6.14 je chyba znázorněna položkou *lineární regrese*.

6.5.4 Zhodnocení metod

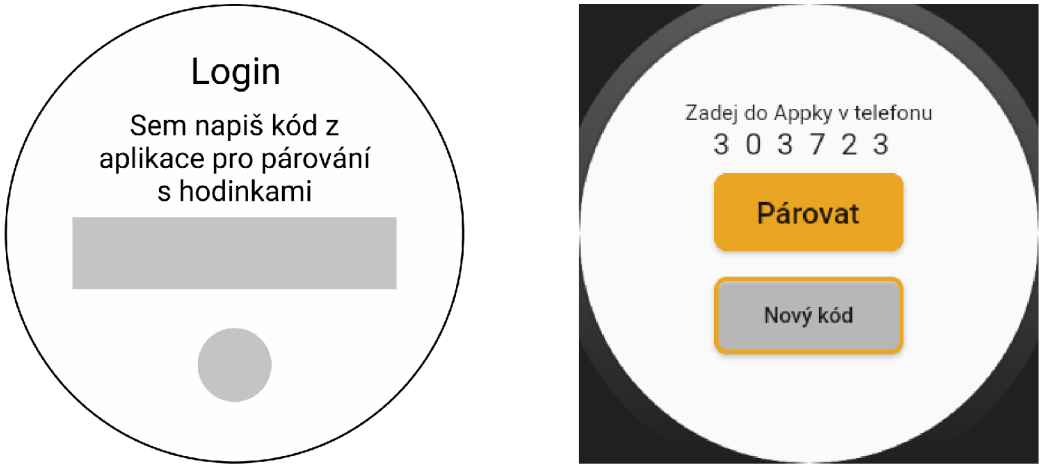
Na základě provedených experimentů a měření jsem dosáhl nejvyšší přesnosti při dopočítávání času pomocí lineární regrese. Metoda 0 uvedená v kapitole 6.5.1 dosahuje velkých nepřesností a je na měření nevhodná. Na detekci průjezdu je dostatečná. Metoda 1 popsaná v kapitole 6.5.2 dosahuje téměř o polovinu lepších výsledku než metoda 0. Proto jsem jí použil na dopočet času průjezdu branou během závodu. Lineární regrese je jako nejpřesnější metoda určena pro finální korekturu.



Obrázek 6.14: Přesnost jednotlivých metod měření času při průjezdu branou v milisekundách. Předpoklad byl, že odchylka naměřená na dvou zařízeních, které spolu projedou kontrolní branou, se bude blížit nule. Nejpřesnější se po provedených experimentech projevil dopočet přesnosti pomocí lineární regrese.

6.6 Testování na uživatelích

Aplikaci jsem v průběhu vývoje testoval ve spolupráci s členy rychlobruslařského oddílu in-line Třinec. Testování probíhalo přímo během tréninků, kde uživatelům byly zapůjčeny hodinky s nainstalovaným prototypem aplikace a telefon s mobilní verzí aplikace pro možnost otestovat párování hodinek a uživatele. Při testování párování hodinek na telefon byl nejčastější problém s ovládáním malého displeje hodinek ve sportovních rukavicích. Na základě zpětné vazby byl z hodinkové aplikace odstraněn vstup pro zadání kódu pro párování. Kód pro párování je nově zadáván pomocí telefonu a na hodinkách probíhá pouze jeho generování. Na obrázku 6.15 je vidět původní záměr obrazovky určené pro párování s telefonem a výsledek po sesbírání zpětné vazby od uživatelů.



Obrázek 6.15: Vlevo je původní původní návrh přihlašovací obrazovky, kde uživatel má pomocí vstupu zadat z klávesnice hodinek párovací kód. Na pravé straně je výsledná obrazovka po zapracování připomínek, že malý displej hodinek je ve sportovních rukavicích neovladatelný.

Další změnou na základě uživatelského testování prošlo samotné přepínání mezi obrazovkami na hodinkách. Z původního záměru kliknutí na horní nebo dolní okraj obrazovky se stalo gesto. Uživatelé si nejčastěji stěžovali, že při přepínání během sportovního výkonu je téměř nemožné trefit správný bod na displeji, aby došlo k přepnutí na další obrazovku.

6.7 Rozšíření a pokračování projektu

Jako rozšíření aplikace pro chytré hodinky vznikla aplikace pro telefony s operačním systémem Android a iOS. V telefonu může uživatel pohodlně sledovat svoje výsledky a to ihned po dojezdu. Dále vznikla webová aplikace, kde uživatel může vytvářet závody a to pomocí klikání bran do mapy. Další z funkcí co bude přidána v další verzi aplikace na hodinky je monitoring tepové frekvence během sportovní aktivity. Do telefonu budou přidány notifikace a možnost vyjádření podpory ostatním uživatelům na základě jejich zaznamenaných sportovních aktivit (obdoba funkce „to se mi líbí“).

Z výhledem do budoucnosti a díky tomu, že už teď je aplikace v hodinkách zcela nezávislá na čidlech, senzorech a komunikačních modulech telefonu, můžou hodinky, resp. aplikace využívat komunikaci na 5G sítích a to téměř okamžitě, jak budou tyto sítě k dispozici. Jedna z nejzajímavějších funkcionalit, co by měla v 5G síti být k dispozici, je samotná

architektura zaměřená zařízení (Device-Centric Architectures). Chytré hodinky s aplikací by pak měly snadnější komunikaci mezi účastníky v závodu, protože komunikace by probíhala přímo mezi nimi [3]. Dále se bude dát využít vyšší přenosová rychlost a odolnost proti výpadkům například pro SOS volání sportovce, kde oznámí svoji polohu a dostupné informace o zdravotním stavu přímo zdravotnímu dispečinku. 5G modul slibuje lepší správu napájení a tím prodlouženou výdrž baterie [2].

Kapitola 7

Závěr

Cílem diplomové práce bylo navrhnout a vytvořit aplikaci na chytré hodinky, která pomůže v oblasti sportovního tréninku a závodů. Výsledkem je aplikace Sport spustitelná na chytrých hodinkách s operačním systémem Wear OS. Aplikace zvládne sledoval polohu závodníka, informovat ho o celkovém čase od startu sportovní aktivity a celkové vzdálenosti, kterou od startu urazil. Současně zvládne aplikace v reálném čase během závodu informovat závodníka jeho pozici a ztrátě na první místo.

K aplikaci na chytré hodinky vznikla také mobilní verze, která je spustitelná na mobilních telefonech s operačním systémem Android a iOS. V mobilní verzi jsou vidět výsledky závodu a tréninku. Současně je možné provést měření sportovní aktivity pomocí mobilní aplikace. Dále vznikla webová aplikace, kde je možné plánovat a ukládat závody pomocí mapových služeb.

Aplikace na chytré hodinky, telefon a web využívají služby mnou vytvořeného modulárního aplikačního serveru, který pro ukládání dat o uživateli a sportovních aktivitách využívá databázi.

Aplikace byla iterativně vyvíjena a testována na uživatelích. Z testování na uživatelích vzniklo několik požadavků na úpravu ovládání aplikace v hodinkách (popsáno v kapitole 6.6). Uživateli byla kladně hodnocena funkce barvy pozadí, která reprezentuje umístění v závodě. Kromě uživatelského testování probíhalo také testování snímání lokalizačních dat v závislosti na čase průjezdu kontrolními body. Závěrem testování bylo ustanovení metody detekce průjezdu a zpřesňování času průjezdu pomocí dvou bodů a lineární regrese.

Výsledky diplomové práce byly prezentované na studenské konferenci `Excel@FIT`. Ve sportovním klubu byla aplikace kladně přijata a vznikly zde požadavky na rozšíření o možnost snímání tepové frekvence pomocí hodinek.

Všechny požadavky, uvedené v zadání, byly splněné a práci považuji za přínosnou v oblasti amatérského a poloprofesionálního sportu, kde umožní s minimální vstupní investicí a technickou znalostí provést kvalitní měření výsledků sportovní aktivity.

Literatura

[1] AMCHRONOS. *Chronos* [online]. [cit. 2020-01-11]. Dostupné z: http://www.amchrono.cz/cz/s3679/c2573-Technologie-a-SW.

[2] APILO, O., UITTO, M. a MÄKELÄ, J. 5G test Network: testing the Mobile Communications for Sports Wearables and Media broadcasting. *ERCIM News*. European Research Consortium for Informatics and Mathematics (ERCIM). [online]. 2019, roč. 4, č. 117. https://ercim-news.ercim.eu/en117/special/5g-test-network-testing-the-mobile-communications-for-sports-wearables-and-media-broadcasting. ISSN 0926-4981.

[3] AUN, N. F. M., SOH, P. J., AL HADI, A. A., JAMLOS, M. F., VANDENBOSCH, G. A. et al. Revolutionizing Wearables for 5G: 5G Technologies: Recent Developments and Future Perspectives for Wearable Devices and Antennas. *IEEE Microwave Magazine*. IEEE. 2017, roč. 18, č. 5, s. 108 – 124. https://ieeexplore.ieee.org/abstract/document/7893084. ISSN 1527-3342.

[4] CHENG, F. *Flutter Recipes: Mobile Development Solutions for iOS and Android*. Apress, 2019. ISBN 978-1484249819.

[5] CLAYTON, C. *IOS 12 Programming for Beginners: An introductory guide to iOS app development with Swift 4.2 and Xcode 10*. 3. edice. Packt Publishing, 2018. ISBN 978-1789348668.

[6] DANIEL, S. F. *Android Wearable Programming*. Packt Publishing, 2015. ISBN 978-1785280153.

[7] DEVELOPERS, A. *Distribution dashboard* [online]. [cit. 2020-01-11]. Dostupné z: https://developer.android.com/index.html. Path: https://developer.android.com/about/dashboards.

[8] DEVELOPERS, A. *Independent versus dependent Wear OS apps* [online]. [cit. 2021-05-09]. Dostupné z: https://developer.android.com/training/wearables/apps/independent-vs-dependent.

[9] JAYGARL, H., LUO, C., KIM, Y., CHOI, E., BRADWICK, K. et al. *Professional Tizen application development*. Wrox, 2014. ISBN 978-1118809266.

[10] KNOTT, D. *Hands-On Mobile App Testing: A Guide for Mobile Testers and Anyone Involved in the Mobile App Business*. Addison-Wesley Professional, 2015. ISBN 978-0134191713.

[11] Krug, S. *Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability.* 3rd. New Riders, 2013. ISBN 978-0321965516.

[12] Lacko Luboslav. *Vývoj aplikací pro Android.* Computer Press, 2015. ISBN 978-80-251-4347-6.

[13] Lacko Luboslav. *Mistrovství - Android: kompletní průvodce vývojáře.* Computer Press, 2017. ISBN 978-80-251-4875-4.

[14] Lacko Luboslav. *Vývoj aplikací pro iOS.* Computer Press, 2018. ISBN 978-80-251-4942-3.

[15] LIM, S. *Global Smartwatch Shipments Rise 1.5% in 2020; Price Trends Going Premium* [online]. [cit. 2021-05-09]. Dostupné z: https://www.counterpointresearch.com/global-smartwatch-shipments-rise-1-5-2020-price-trends-going-premium/.

[16] Mísař, J. *Vyvíjíme pro Apple Watch: Architektura* [online]. [cit. 2021-05-09]. Dostupné z: https://zdrojak.cz/clanky/vyvijime-pro-apple-watch-architektura/.

[17] raceresult. *Pasivní časomíra* [online]. [cit. 2020-01-11]. Dostupné z: https://www.raceresult.com/en/systems/passive.php.

[18] Shirer, M. *Shipments of Wearable Devices Reach 118.9 Million Units in the Fourth Quarter and 336.5 Million for 2019, According to IDC* [online]. [cit. 2021-05-09]. Dostupné z: https://www.idc.com/getdoc.jsp?containerId=prUS46122120.

[19] Smyth, N. *WatchOS 2 App Development Essentials: Developing WatchKit Apps for the Apple Watch.* CreateSpace Independent Publishing Platform, 2015. ISBN 978-1517365059.

[20] source.android.com. *Linux Kernel* [online]. [cit. 2020-01-11]. Dostupné z: https://source.android.com/devices/architecture/kernel/releases.

[21] Tizen. *Tizen - linux* [online]. [cit. 2020-01-11]. Dostupné z: https://wiki.tizen.org/Linux.

[22] Tizen. *Tizen Architecture* [online]. [cit. 2020-01-11]. Dostupné z: https://developer.tizen.org/tizen-architecture.

[23] Weinschenk, S. M. *100 věcí, které by měl každý designér vědět o lidech.* 1. vyd. Computer Press, 2012. ISBN 978-80-251-3649-2.

[24] Windmill, E. *Flutter in Action.* Manning Publications, 2019. ISBN 978-1617296147.

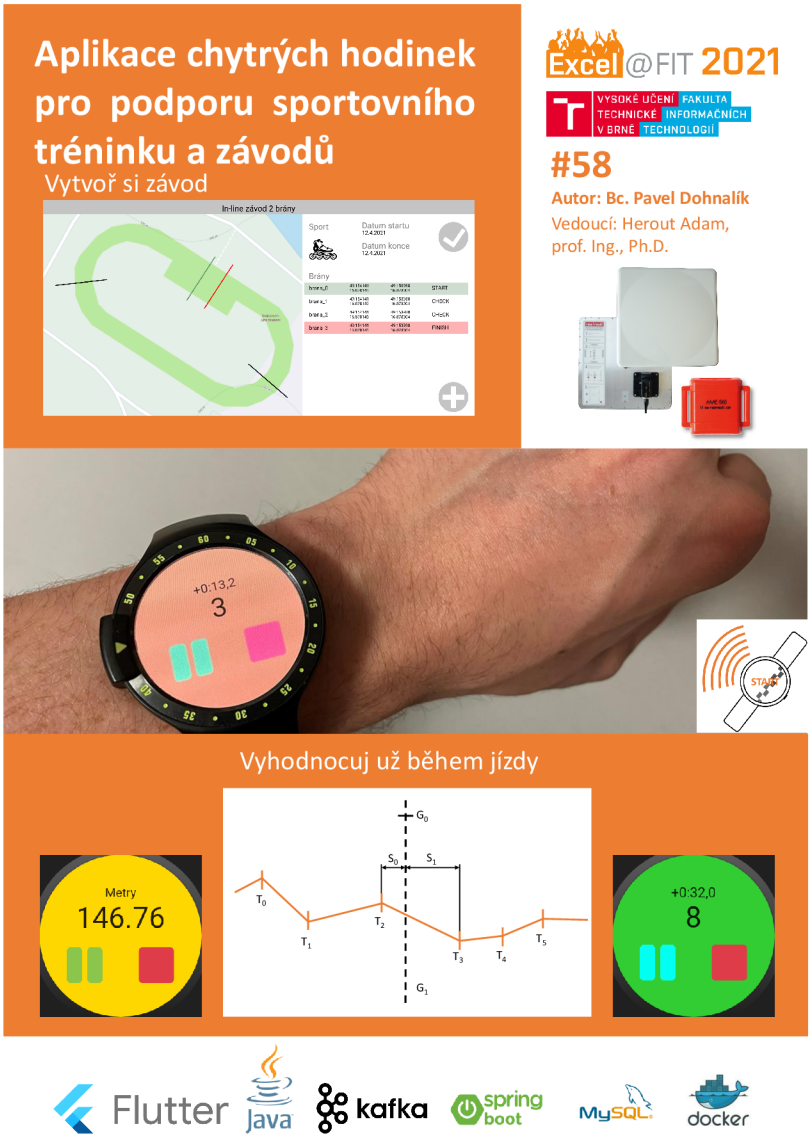
Příloha A

Obsah CD

/
- `Latex/` ..... Složka se zdrojovými LaTeXsoubory teto práce.
- `Thesis/` ..... Složka obsahující tuto práci ve formátu pdf.
- `Poster/` ..... Složka obsahující plakát.
- `Video/` ..... Složka obsahující prezentační video.
- `SourceCode/` ..... Složka obsahující zdrojový kód aplikací.
  - `SmartWatchApp/` ..... Složka obsahující zdrojový kód pro aplikaci na Wear OS.
  - `MobileApp/` ..... Složka obsahující zdrojový kód pro aplikaci na Android a iOS.
  - `WebApp/` ..... Složka obsahující zdrojový kód pro webovou aplikaci.
  - `ServerApp/` ..... Složka obsahující zdrojový kód pro serverovou aplikaci.
- `README` ..... Informační soubor.

Příloha B

Plakát



Obrázek B.1: Plakát ze soutěže `Excel@FIT`.