UNIVERZITA HRADEC KRÁLOVÉ

FAKULTA INFORMATIKY A MANAGEMENTU
KATEDRA INFORMATIKY A KVANTITATIVNÍCH METOD

SMART APLIKACE PRO SPORTOVNÍ AKTIVITY
SMART APPLICATION FOR SPORT ACTIVITIES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. LUKÁŠ KOPECKÝ

VEDOUCÍ PRÁCE doc. Mgr. TOMÁŠ KOZEL, Ph.D.

Hradec Králové 2022

ANOTACE

Cílem práce je vytvořit sportovní trackovací aplikaci s pomocí moderních multiplatformních frameworků, a to konkrétně frameworku Flutter. Uživateli je umožněno využívat aplikaci při různých typech sportovních aktivit, a sledovat tak průběžné informace o sportovní aktivitě přímo v aplikaci v reálném čase. Aplikace poskytuje kompletní datovou analýzu nad daty, které uživatel nasbíral v průběhu aktivity. Data jsou zpracována a vykreslena do přehledných grafů v aplikaci tak, aby měl uživatel co nejlepší zpětnou vazbu o provedené aktivitě, a mohl tak porovnat aktivitu s aktivitami předešlými.

ANNOTATION

The goal of this thesis is to create a sports tracking application using a modern multi-platform framework, specifically the Flutter framework. The user is allowed to use the application during different types of sports activities, to track continuous information about the sports activity directly in the application in real time. The application provides complete data analysis over the data collected by the user during the activity. The data is processed and plotted into clear graphs in the app, so that the user has the best possible feedback on the performed activity and can compare the activity with previous activities.

PROHLÁŠENÍ

Prohlašuji, že jsem bakalářskou/diplomovou práci zpracoval/zpracovala samostatně a s použitím uvedené literatury.

Hradec Králové ....

podpis autora(-ky)

PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce doc. Mgr. Tomáši Kozlovi, Ph.D za metodické vedení práce, vedení projektu a poznámky k implementaci aplikace. Poděkovaní patří také Bc. Štěpánu Zálišovi za poznatky k implementaci aplikace. Dále děkuji panu Mgr. Janu Draesslerovi, Ph.D. za poznatky a připomínky ke statistickým metodám použitých v diplomové práci.

Obsah

Seznam obrázků

SEZNAM ZDROJOVÝCH KÓDŮ

Seznam tabulek

Seznam vzorců

ÚVOD

Rychlý vývoj mobilních zařízení a pokrok v oblasti bezdrátových sítí, vedl k rozsáhlému rozvoji chytrých technologií. Všudypřítomná zařízení umožňují uživatelům komunikovat s jinými zařízeními prostřednictvím internetu. Díky svým schopnostem, které uživatelům umožňují vyvíjet software, který potřebují, vznikla celá řada aplikací pro mobilní zařízení. V této diplomové práci je představeno získávání dat při sportovních aktivitách pomocí mobilních sportovních aplikací. V práci je dále vysvětleno dolování a analýza dat, což umožňuje sportovcům analyzovat tréninky a předpovídat další tréninkové aktivity. Sportovní trackovací aplikace sledují a monitorují sportovní aktivity v reálném čase. Když se tedy sportovec během své sportovní aktivity pohybuje, zaznamenávají se informace týkající se doby trvání, délky a nadmořské výšky trasy. K tomu je zapotřebí pouze mobilní zařízení, včetně zařízení GPS. Pro sledování údajů o tréninkových aktivitách trenérem, musí být k dispozici bezdrátové připojení přes internet.

Cílem práce je vytvořit demonstrativní sportovní aplikaci pro sledování a zaznamenávání sportovních aktivit, pomocí multiplatformního frameworku Flutter. Práce vysvětluje principy multiplatformního vývoje, vývoje mobilních aplikací a představuje sportovní aplikace na trhu. Výsledkem práce bude nová sportovní aplikace, včetně uvedených a vysvětlených výpočtů, které používají komerční aplikace.

Teoretická část práce popisuje funkčnost a metody získávání dat z mobilních trackovacích aplikací. Představuje nejznámější aplikace na trhu a nabízí základní popis těchto aplikací. Důležitým prvkem je ukládání a přenosnost dat. Tato problematika je zde vysvětlena včetně příkladů dvou hlavních formátů KLM/KMZ a GPX. V těchto aplikacích se více využívá formát GPX, a proto je zde představena struktura formátu, včetně vysvětlení jednotlivých elementů a logiky ukládání dat. Poté, co je čtenář uveden do problematiky sportovních trackovacích aplikací, jsou představeny mobilní platformy a vývojové technologie pro vývoj těchto aplikací. Speciálně je pak představen multiplatformní vývoj a hlavní technologie multiplatformního vývoje, kterými jsou: React Native, Xamarin, Ionic, Flutter. Detailněji je rozvedena problematika a technika vývoje pomocí frameworku Flutter, který byl vybrán jako hlavní implementační framework cílové aplikace. Pro získávání dat z aplikace jsou představeny senzory mobilních zařízení, a pak speciálně technologie GPS. Vysvětleny jsou metody lokalizačních služeb, včetně výpočtu vzdálenosti mezi geografickými body pomocí metody Haversina a Vincenta. Popsány jsou i sekundární metody lokalizace pomocí WiFi a mobilních sítí. Poté, co je čtenář zasvěcen

do vývojových technologií mobilních aplikací, jsou představeny technologie serverové pro ukládání velkých dat do NoSQL databáze pomocí webového frameworku Java Spring Boot. Dále jsou v práci popsány knihovny a technologie pro analýzu dat pomocí programovacího jazyka Python.

Praktická část se zabývá analýzou a návrhem sportovní trackovací aplikace. Uvedeny jsou funkční a nefunkční požadavky, včetně drátěných modelů, podle kterých bude aplikace implementována. Aplikace bude podporována platformou Android a iOS. Následně je v této části práce popsána implementace mobilní aplikace, včetně výpočetních modulů a ukládaní dat na server. Dále jsou zde vysvětleny analýzy a statistiky, které lze získat z dat z mobilní aplikace.

1 SPORTOVNÍ TRACKOVACÍ APLIKACE

V dnešní době díky rozšířeným mobilním zařízením lze snadno sledovat, shromažďovat a sdílet informace o zdravotním stavu jedince na sociálních sítích. [57] V současnosti je již nepředstavitelné zaznamenávat sportovní aktivity bez chytrého mobilního telefonu. Tyto aplikace mohou pomoci sportovcům zůstat zdravý a vyvodit závěry z předchozích tréninků. Prvním zařízením pro zlepšování lidského výkonu byl monitor srdečního rytmu. Jednalo se o hrudní pásy se senzory umístěnými kolem hrudníku, které sledovaly srdeční rytmus. Přelomovým krokem v oblasti sportovních aplikací bylo použití prvních mobilních sledovacích systémů s použitím GPS v roce 2005. Tato metoda založená na GPS se používá dodnes. Například pro cyklisty to znamená, že není třeba nadále kupovat různé senzory na kolo, ale lze použít pouze mobilní sportovní aplikaci. S postupem času se mobilní aplikace začaly objevovat také na chytrých hodinkách, kde výhodou byla vysoká přenosnost, malé rozměry a estetický vzhled. Většina aplikací na mobilních zařízeních si tyto data o aktivitě uchovává ve své lokální paměti a poté data přepošle na server, který následně data analyzuje a vyhodnotí sportovní aktivitu. Hodnocením může být vzdálenost, grafy s rychlostí v průběhu tréninku, nadmořská výška, srdeční frekvence a další. Sportovní aplikace mapují a monitorují sportovní aktivity v reálném čase. Tedy pokud se sportovec jakkoliv pohne z místa, tak informace o pozici, trvání, délce a nadmořské výšce jsou ihned zaznamenány. Pro tyto věci je nutné, aby zařízení disponovala GPS modulem. Tréninková data se ukládají do souborů, například ve formátu GPX. Formát GPX je určen pro popis trasových bodů, tras a cest, což je užitečné pro různé softwarové aplikace. Více informací o formátu GPX v kapitole 1.2.2. [29] GPS data pro sledování jízdních kol mají často potenciál poskytovat detailnější informace o cestování na kole. Dokáží odhalit, které trasy jsou oblíbené, a kterým se lidé vyhýbají, jaké jsou čekací doby na křižovatkách a jízdní špičky. Může také poskytovat některé charakteristiky cyklistické komunity v oblastech, jako je rozložení podle věku, nebo pohlaví. Následně mohou být tato data použita ke zlepšení cyklistické infrastruktury. [62] Existuje několik specializovaných softwarů, které umožňují komplexní analýzu dat ze senzorů sportovce s možností vizualizace v tabulkách, popř. 2D, či 3D grafů. [55] Problematice sportovních aplikací a příkladů komerčních aplikací je věnována následující kapitola 1.1.

1.1 Mobilní aplikace pro Android a iOS

Sportovní aplikace pomáhají uživateli dostávat zpětnou vazbu na jejich sportovní aktivitu. Tyto získané informace mohou uživatele motivovat k překonání vlastních limitů při další aktivitě. Další možností motivace uživatele k další aktivitě je získávání odznaků a odměn v rámci sportovní aplikace. S tím souvisí přidání herních prvků do aplikace, která mohou motivovat uživatele a zároveň lépe představit funkce aplikace uživateli. [5] Další strategií, jak zlepšit výsledky aktivity, je využít virtuálního trenéra v aplikaci. Tuto funkci nabízí kolem 50 % dostupných aplikací na trhu. Trenér může komunikovat pomocí SMS zpráv v průběhu dne, či průběhu aktivity, což vede ke zvýšení výsledků aktivity. [68] Většina sportovních aplikací má implementované prvky sociálních sítí. Je tedy možné sdílet aktivity s přáteli, nebo přesdílet aktivitu na sociální sítě, které mohou být propojeny s účtem uživatele. [5] Strava a Endomondo jsou nejznámějšími aplikacemi na trhu pro oblast zaznamenávání sportovních aktivit. Endomondo již bohužel zaniklo a bylo nahrazeno aplikací MapMyRun. Mimo tyto aplikace existují například aplikace: Runtastic, Runkeeper, BikeMap, MapMyFitness a mnoho dalších.

1.1.1 Strava

Aplikace byla vytvořena pro potřeby zaznamenávání sportovních aktivit: cyklistiky a běhu. V roce 2017 již Strava podporovala více něž 31 druhů sportovních aktivit. Jedná se o nejpopulárnější sportovní aplikaci na trhu. Pokud je aktivita dokončena, data jsou odeslána a následně analyzována. Vyhodnocena je průměrná rychlost, tempo, srdeční frekvence (pokud je dostupný patřičný senzor) a další. Důvodem, proč se Strava stala tak populární, je soutěživost mezi uživateli, která je podpořena funkcí překonávání vlastních rekordů, žebříčku nejlepších sportovců v regionech nebo v klubech a sbíráním bodů za ukončené aktivity. V prémiové verzi je možnost vygenerovat si běžeckou mapu, která znázorňuje nejpopulárnější cesty uživatelů, nebo osobních cest. Strava také nabízí možnost exportovat data o aktivitě do formátu GPX k další práci s daty. Mobilní aplikaci Strava lze použít k podpoře sportovců tak, aby dosáhli lepších výsledků, ale není příliš vhodná pro začátečníky, kteří se sportovními aplikacemi začínají, jelikož je aplikace složitější a obsahuje velké množství funkcí, které začátečník nepotřebuje. Hodí se tedy spíše pro pokročilejší sportovce, kteří chtějí výsledky porovnávat s komunitou, či přáteli. Aplikace je velmi přesná na znamenávání vzdálenosti sportovních aktivit, analýzu průběhu aktivit, či průjezdy

trasovými segmenty. Aplikace nabízí jak volně přístupnou část, tak placenou, která dovoluje zobrazit více analýz, nebo například funkce plánování trasy. [74]

1.1.2 Endomondo

Endomondo bylo založeno v Dánsku a v roce 2015 získala aplikaci společnost Under Armour. Aplikace byla dostupná na Android, iOS a Windows Phone. Endomondo umožňovalo sledování 70 kategorií aktivit včetně čtyř typů halových cyklistických aktivit. [24] Z výzkumu [19] v roce 2014 bylo zjištěno, že třemi kontinenty s nejvíce aktivními uživateli byly Evropa, která představovala 60,84 % všech uživatelů, následovala Amerika (24,28 %) a Asie (11,53 %). Země s nejvyšším počtem registrovaných uživatelů bylo USA, kde tvořili 18,5 % všech uživatelů, následovalo Španělsko (13,8 %) a Spojené království (8,42 %). Nejoblíbenější sportovní aktivitou byl běh, který představoval 42,6 % všech aktivit, následovala cyklistika (22,87 %), chůze (19,95 %) a další (14,53 %) [19]. Bohužel aplikace byla ukončena k 31. prosinci 2020, kdy vznikla nová aplikace MapMyRun od společnosti Under Armour, aby tuto aplikaci nahradila. Výhodou je, že lze data z aplikace Endomondo převést na platformy MapMyRun, a tak uživatel neztratí žádné své údaje. [24]

1.1.3 Runkeeper

Runkeeper je aplikace navržena pro trackování sportovních aktivit. Aplikace využívá služeb Google Maps. Nabízí základní analýzy sportovních aktivit. I v této aplikaci je dostupná placená verze, která nabízí rozšířené statistiky. Placená verze je však dražší a nenabízí takové funkce, jako například placená verze aplikace Strava. Zároveň Runkeeper neobsahuje audio trenéra, který dává informaci o aktuálním dosaženém kilometru, či míli. Aplikaci je možné napojit na účet přes sociální sítě a sdílet aktivity na sociálních sítích. Runkeeper nabízí, až 13 jazyků a obsahuje herní prvky jako výzvy a cíle. [43]

1.2 Formáty dat sportovních aplikací

Pro přenášení dat ze sportovních aplikací je nutné využívat ucelený formát dat. Pro sdílení a přenos dat z aplikací jako například Google Earth, slouží formáty KLM/KMZ. Z mobilních sportovních aplikací je to většinou formát GPX. Tyto formáty jsou dále popsány v následujících kapitolách.

1.2.1 Keyhole Markup Language

Keyhole Markup Language (KLM) je notací Extensible Markup Language (XML) pro popis informací v geografickém kontextu. KML verze 2.2 se stala mezinárodním standardem, protože byla přijata jako implementační standard OGC (Open Geospatial Consortium) pro vizuální reprezentaci geografických informací. KML i KMZ jsou formáty souborů používané v aplikacích Google, konkrétně v aplikacích Google Earth a Google Maps. Lze jej vytvářet a upravovat pomocí základního textového editoru, i když nejčastějším způsobem tvorby je export dat do KML, nebo uložení prvků pomocí grafického uživatelského rozhraní pomocí různých nástrojů. Uživatelské rozhraní desktopového Google Earth má řadu nástrojů, které umožňují definovat prvky KML, a umožňují tak ukládat data v tomto formátu. Bohužel existují některé prvky KML, které nelze vytvořit prostřednictvím uživatelského rozhraní. [80] Soubor KML specifikuje sadu prvků (místa, obrázky, polygony, modely, textové popisy atd.), které lze zobrazit na mapách softwaru, který podporuje formát KML. Každé místo má vždy zeměpisnou délku a šířku. Struktura souboru KML pro čáry, body a obrázky je obdobná a obsahuje záhlaví a deklaraci jmenného prostoru. [50] Ukázka syntaxe KML pro vytvoření bodu je vypsána na zdrojovém kódu č. 1 [50]

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Placemark>
    <name>Simple placemark</name>
    <description>Attached to the ground. Intelligently places itself
       at the height of the underlying terrain.</description>
    <Point>
      <coordinates>-122.0822035425683,37.42228990140251,0</coordinates>
    </Point>
  </Placemark>
</kml>
```

Zdrojový kód 1: Ukázka zápisu dat pomocí KML. Zdroj: Vlastní

1.2.2 GPS eXchange Format

GPX (GPS eXchange Format) je schéma XML navržené jako formát dat GPS pro softwarové aplikace. Lze jej použít k popisu trasových bodů, cest a tras. Jedná se o formát podporovaný největšími poskytovateli map a navigačních softwarů, jako jsou Google Earth, MapSource a další. [21] Formát umožňuje ukládání polohy, nadmořské výšky a času. Dále formát obsahuje i hlavičkové značky, které obsahují metadata. Zeměpisná šířka a délka jsou vyjádřeny v desetinných stupních pomocí souřadnicového

systému WGS84. [55] Data a časy nejsou uvedeny v místním čase, ale ve světovém koordinovaném čase (UTC) ve formátu ISO 8601. Ve starších verzích hlavní část souboru GPX obsahovala dělení pouze na elementy *<trk>* a *<trkseg>*. V nejnovější verzi formátu jsou hlavními značkami *<wpt>* pro "waypoint"a *<rte>* pro "route". Tyto značky dále obsahují jednotlivé body (trackpoints), které jsou uloženy pod značkou *<trkpt>*. Každý z bodů obsahuje parametry `lat` (zeměpisná šířka) a `lon` (zeměpisná výška), které udávají přesnou pozici bodu. Informace o nadmořské výšce je uložena pod značkou *<ele>* a pro časový otisk, je obsažena v hodnotě značky *<time>*. [66]. Ve zdrojovém kódu č. 2 je vypsán příklad zápisu jedné z aktivit ve formátu GPX, který byl exportován z aplikace Strava.

```
<gpx creator="StravaGPX Android">
 <metadata>
  <time>2022-02-04T15:47:59Z</time>
 </metadata>
 <trk>
  <name>POHA 10</name>
  <type>9</type>
  <trkseg>
   <trkpt lat="50.1281280" lon="16.1423290">
    <ele>260.1</ele>
    <time>2022-02-04T15:47:59Z</time>
    <extensions>
     <gpxtpx:TrackPointExtension>
      <gpxtpx:cad>0</gpxtpx:cad>
     </gpxtpx:TrackPointExtension>
    </extensions>
   </trkpt>
  </trkseg>
 </trk>
</gpx>
```

Zdrojový kód 2: Ukázka zápisu dat pomocí GPX. Zdroj: Vlastní

1.2.2.1 Waypoint

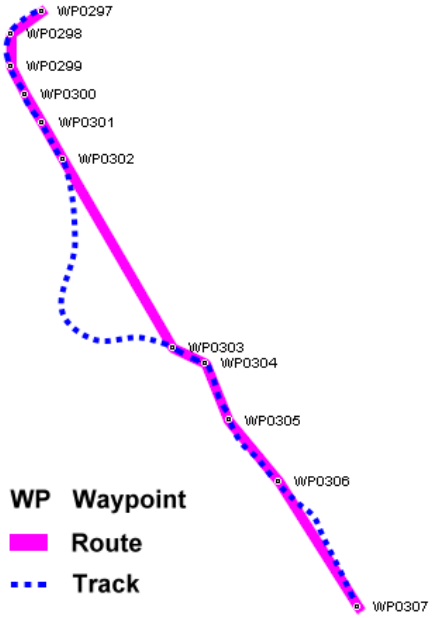
Waypoint je samostatný bod v listě bodů, který nemá žádný vztah s ostatními body. [52] Kromě souřadnic mají waypointy název a obvykle i symbol (letiště, náměstí, bod, stadion, most, vlajka, budova, autobusová zastávka atd.). Waypointy jsou nezávislé na ostatních bodech. Uživatel obvykle vytváří waypointy buď označením místa pomocí přijímače GPS, nebo pomocí mapy (nejlépe pomocí mapovacího softwaru v počítači). [8]

1.2.2.2 Route

Cesta je soubor bodů, které tvoří cestu z jednoho bodu do druhého. Cesta se skládá pouze ze stovek, či jen několika traťových bodů (v závislosti na přijímači), zatímco trasa může mít tisíce traťových bodů. Přijímač GPS má specifické funkce pro navigaci po trase. Vzhledem k tomu, že trasa a cesta jsou si podobné, novější přijímače dostávají funkce pro navigaci po trase. Trasa není určena ke kreslení liniových prvků na obrazovce mapy. [8]

1.2.2.3 Trackpoint

Přijímač obvykle zaznamenává body trasy během jízdy. Trackpointy definují stopu vytvořenou spojením bodů čarami. Tato stopa představuje cestu, stezku, pěšinu atd., po které se uživatel se zařízením vydal. [52] Křivky jsou tvořeny krátkými úsečkami. Obecně platí, že traťové body nemají názvy ani symboly. Mohou mít datum, nebo časový otisk, který umožňuje výpočet rychlosti pro traťový úsek. Traťovým úsekem se nazývá linie spojující dva trackpointy. Vzdálenost se vypočítá ze souřadnic trackpointů. Často se zaznamenává i převýšení, a je tedy možné získat výškový profil trati, nebo její trojrozměrné zobrazení. Rozdíly mezi waypoint, route a trackpoint jsou znázorněny na obrázku č. 1. [8]



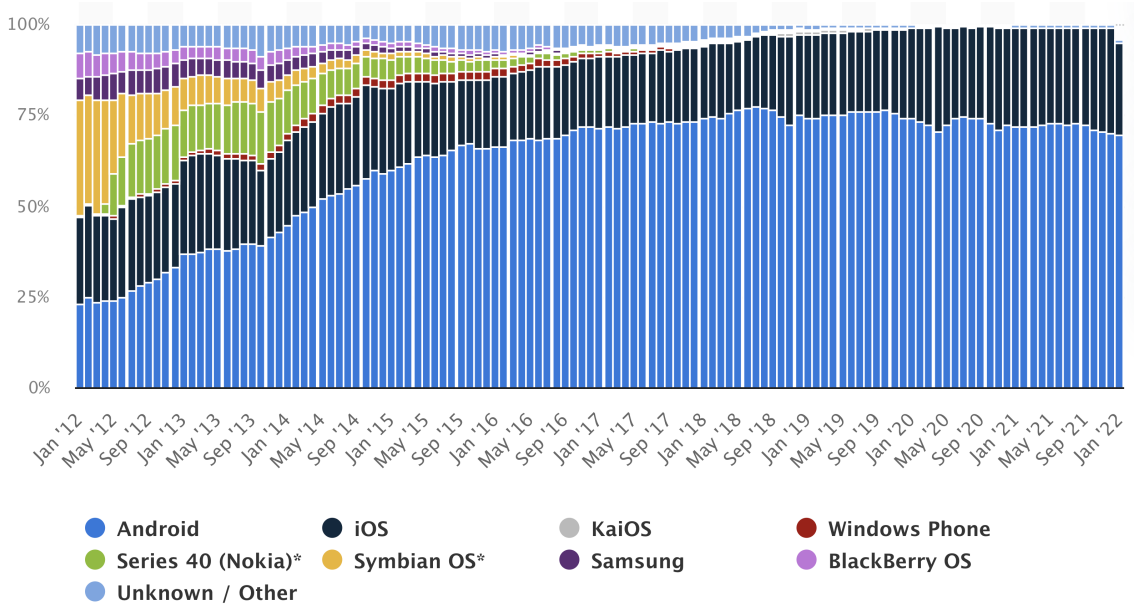
Obrázek 1: Znázornění rozdílu mezi: Waypoint, Route a Track. Zdroj: [40]

2 TECHNOLOGIE PRO VÝVOJ

V následující kapitole jsou popsány technologie, které umožňují vytvářet sportovní trackovací aplikace. Popsány jsou technologie, jak pro vývoj mobilních aplikací, tak technologie pro ukládání dat s následnou analýzou geolokačních dat.

2.1 Mobilní platformy

Na trhu s mobilními zařízeními existuje nespočet odlišných mobilních platforem. Nejvýraznějšími platformami dnes jsou Android a iOS. [69] Soutěživost a konkurence mezi různými operačními systémy je hlavním faktorem, který nutí vývojáře přidávat a vymýšlet nové funkce. [34]. Na obrázku č. 2 je znázorněn graf využití mobilních operačních systémů v rozmezí roků 2012 až 2022. V roce 2012 byl dle [3] nejpoužívanějším operačním systémem Symbian OS, který představoval 31.89 % podílu na trhu. Dále s 24.04 % se nacházel iOS, 23.21 % Android, 6.94 % BlackBerry OS, 5.84 % Samsung OS, 0.36 % Windows Phone a zbytek tvořily ostatní operační systémy. V roce 2022 se situace na trhu s mobilními operačními systémy radikálně změnila. Hlavní podíl na trhu v roce 2022 měl s 69.74 % operační systém Android. Druhou příčku drží iOS s 25.49 % a zbytek ostatní operační systémy. [3] Nejpoužívanější mobilní platforma Android je dále popsána v kapitole č. 2.1.1 a platforma iOS je detailněji popsána v kapitole č. 2.1.2.



Obrázek 2: Vývoj trhu s mobilními operačními systémy mezi roky 2012 a 2022. Zdroj: [3]

2.1.1 Android

Android je operační systém pro mobilní zařízení, který byl vyvinut společností Google. Android se skládá z UNIX-ového operačního systému založeném na jádře Linuxu verzi 2.6. Android je primárně vyvinut pro mobilní zařízení, která podporují dotykové obrazovky. [28] Mimo mobilní zařízení se Android využívá u chytrých hodinek (Android Wear), tabletů a v systémech pro automobily (Android Auto). Android byl vydán pod bezplatnou licencí s otevřeným zdrojovým kódem. Bezplatná licence umožňovala vývojářům vyvíjet a distribuovat vlastní modifikovanou verzi operačního systému prostřednictvím Android Open Source Project (ASOP). [27] Mezi všemi mobilními operačními systémy je Android nejpopulárnější operační systém, který konkuruje systémům iOS a dříve i Windows Phone. Android aplikace jsou psány v programovacím jazyce Java a jsou spouštěny v Dalvik Virtual Machine (VM) a Android Runtime (ART) v novější verzi. Každá Android aplikace běží ve svém vlastním procesu s vlastní instancí Dalvik VM. Dalvik VM spouští soubory ve formátu Dalvik Executable (*.dex*), který je optimalizovaný pro minimální zatížení paměti. S Androidem 5.0 (Lollipop) Google nahradil Dalvik VM za ART, pro zvýšení výkonu. [17]

2.1.1.1 Architektura

Ve spodní vrstvě architektury spoléhá Android na Linuxové jádro, které zajišťuje zabezpečení, správu paměti, správu procesů, síťové prvky a ovladače. Android obsahuje sadu knihoven, které nabízí většinu funkcionalit dostupných v knihovnách Javy. [27] Další vrstva obsahuje sadu C/C++ knihoven používaných různými částmi systému Android. Vzhledem k tomu, že byl Android vyvinut, aby běžel na nízko-výkonových Contorll Process Unit (CPU) a Graphics Process Unit (GPU) zařízeních s omezenou pamětí. Stejně tak knihovny jako *libc*, nebo *libm*, byly vyvinuty tak, aby zajišťovaly nízkou paměťovou náročnost. Vrstva obsahuje knihovny jako Surface Manager, která je odpovědná za přístup k obrazovce. Dále knihovnu Media Framework optimalizovanou pro přehrávání audio a video formátů, Structured Query Language (SQL) databázi a nativní engine webového prohlížeče (WebKit). Tyto prostředky jsou nabízeny vývojářům přes aplikační framework. Nejdůležitější částí vrstvy aplikačního frameworku je Activity Manager, odpovědný za ovládání životního cyklu aplikace. Android Software Development Kit (SDK) nabízí kompletní nástroje potřebné k vývoji aplikace. Android je vybaven sadou aplikací včetně emailového klienta, SMS programu, prohlížeče, kalendáře a dalších. [28]:

- Android Asset Packaging Tool (AAPT) - Nástroj dovoluje vývojáři vytvářet, prohlížet a aktualizovat kompatibilní archivy, jako jsou: *.zip*, *.jar*, *.apk*. Také dokáže kompilovat zdroje do binární podoby. Tyto soubory mohou být instalovány na jakémkoliv Android zařízení nebo emulátoru.
- Android Debug Bridge (ADB) - Nastavuje připojení s fyzickým Android zařízením, nebo emulátorem za účelem přenosu a instalace souborů *.apk*. Zároveň nabízí vývojářům funkci spouštění vzdálených shell dotazů.
- Android Interface Definition Language (AIDL) - Dovoluje definovat programovací rozhraní, na kterém se klient a server dohodnou, aby spolu mohli komunikovat podle InterProcess Communication (IPC).
- Dalvik Debug Monitor Service (DDMS) - Poskytuje služby směrování portů, záznamu obrazovky, informace o zařízení, logcat, informace o procesech, příchozí hovory, Short Message Service (SMS) a lokalizační údaje.
- Dalvik cross-assembler (DX) - Konvertuje bytecode souborů tříd do binární souboru .dex, které spouští Dalvik VM.

2.1.2 iOS

iOS (původně iPhone OS) je mobilní operační systém vytvořený vývojáři ze společnosti Appple Inc. a distribuovaný pouze pro zařízení Apple. Jedná se o operační systém, který pohání mobilní zařízení jako: iPhone, iPad a iPod touch. [33] iOS je derivátem z operačního systému MacOS X. Vývoj iOS vyžaduje počítače Macintosh, který alespoň využívají operační systém MacOS X 10.6 (Snow Leopard). [30] Operační systém iOS nemůže běžet na jiných stanicích, než od firmy Apple, protože zde neexistují licence k instalování iOS. V dnešní době však existují simulátory pro běh iOS na jiných typech operačního systému. [54] V roce 2015 podle [33] Apple oznámil, že se podařilo prodat jednu miliardu iOS zařízení od doby, kdy byl představen iPhone v roce 2007. [33] Apple nabízí aktualizace iOS systému, přes Over-the-Air (OTA) rozhraní. [54] iOS aplikace jsou psány pomocí XCode moderního Integrated Development Environment (IDE). Aplikace jsou většinou psány v programovacím jazyce Objective-C, nebo Swift. [30]

2.1.2.1 Architektura

iOS architektura obsahuje čtyři vrstvy. Spodní Core OS vrstva obsahuje jádro sytému, ovladače zařízení, správu paměti, zabezpečení a další. Core OS vrstva je nejblíže multi-vláknovému UNIX jádru. Zde se nachází Application Programming Interface (API) napsáné v jazyce C. Jádro systému iOS se nazývá XNU a je jádrem operačního systému Darwin. [54] Vrstva Core Service nabízí základní služby jako je práce s administrací, interakce se sítí, správa kontaktů a voleb. Tyto služby dovolují využití hardwarových vlastností zařízení (GPS, kompas, akcelerometr, nebo gyroskop). [54] [37] Vrstva Media umožňuje práci s grafickými a audio a video technologiemi. Grafické technologie obsahují Core Graphics, Core Animation a OpenGL systémy, které zajišťují práci s 2D vektory, animaci 2D a 3D objektů včetně animace pohledů. Audio technologie podporují formáty jako Advanced Audio Codec (AAC), Apple Lossless Audio Codec (ALAC), A-law a Linear PCM. Video technologie podporují přehrávání souborů, jako: .mov, .mp4, .m4v a .3gp [37] Cocoa Touch vsrtva obsahuje klíčový framework k vytváření iOS aplikací. Jedná se o nejvyšší vrstvu v architektuře. [37] Jelikož byl iOS odvozen z MacOS X, i zde se vrstva Cocoa Touch nachází. Knihovny na této vrstvě nabízí základní strukturu pro vývojáře. Apple nahlíží ke knihovnám jako k frameworku. [33] Vrstva definuje základní struktury a podporu multitaskingu, dotykových polí, notifikací a tak dále. Hlavním programovacím jazykem je zde Objective-C, nebo Swift. Swift je kompilovaný programovací

jazyk, který byl vytvořen a představen firmou Apple pro iOS v roce 2014. Swift je navržen tak, aby interagoval s Cocoa a Cocoa Touch frameworky s existujícími Apple produkty napsanými v jazyce Objective-C. [54] Zejména elementy z knihovny UIKit jsou používány k vytváření uživatelského rozhraní. Nejvíce používanými elementy jsou UIViewController, UIView, UIButton a další prvky. [33]

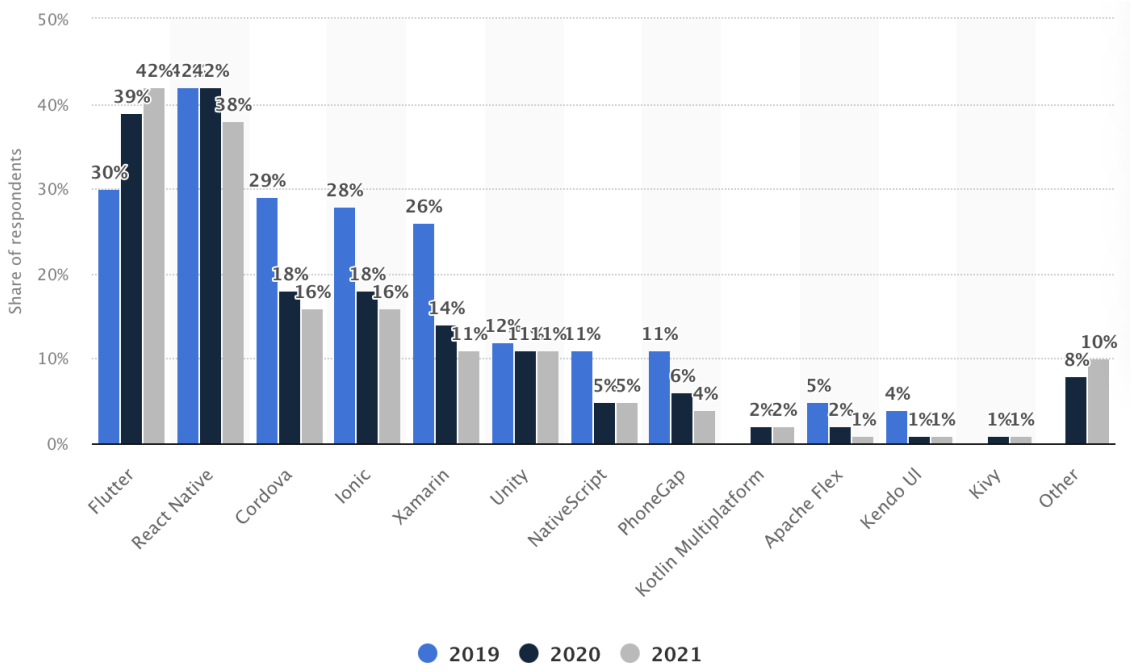
2.2 Způsoby vývoje mobilních aplikací

Tradičním způsobem při vyvíjení mobilních aplikací, bylo použití nástrojů navržených pro specifickou vývojovou platformu. Tento typ vývoje je označován jako nativní vývoj. V tomto přístupu jsou programovací jazyky, vývojová prostředí a SDK nativní pro cílovou vývojovou platformu. Hlavními vývojovými platformami jsou nyní Android od společnosti Google a iOS od společnosti Apple. Nativní aplikace musí být vyvíjeny odděleně pro každou z platforem. To znamená mnoho vývojových platformních nástrojů a programovacích jazyků. Pro iOS je to vývojové prostředí XCode a programovací jazyk Swift/Objective-C. Pro Android se jedná o vývojové prostředí Android Studio a programovací jazyk Java, dnes již spíše Kotlin. [16] Alternativní cestou vývoje aplikací je použití multiplatformního vývoje, který dovoluje použití jednoho vývojového kódu na více platformách. Vývoj nativních aplikací je finančně a časově velmi náročný. [44] Překonání těchto nevýhod nativního vývoje, je hlavním cílem multiplatformního vývoje. Ačkoli multiplatformní vývoj zahrnuje spoustu výhod, často zmiňované jsou i nevýhody. V roce 2012 [16] společnost Facebook zjistila, že jejich multiplatformní aplikace nesplnila úroveň požadavků, kvůli složitosti aplikace. To společnost donutilo, aby opustila multiplatformní vývoj a vrátila se zpět k vývoji nativních aplikací. Jejich předchozí multiplatformní aplikace byla vyvinuta pomocí hybridního přístupu. Více o hybridním přístupu v kapitole 2.2.1. [16]

2.2.1 Hybridní vývoj

Hybridní přístup kombinuje nativní vývoj s mobilními webovými aplikacemi (Web-Based). Využívá kódovou základnu nativních aplikací. Aplikace mohou přistupovat k hardwarovým vlastnostem zařízení a taktéž mohou být distribuovány pomocí platformních mobilních obchodů. Hybridní aplikace jsou vytvářeny pomocí Cross Platform Tools (CPT). CPT nabízí dva přístupy k tvorbě aplikací. První přístup dovoluje

použití webových technologií jako jsou HyperText Markup Language (HTML), Cascading Style Sheets (CSS), JavaScript (JS), které vytvoří základní kód. Tento kód poté bude spuštěn v interním prohlížeči (WebView), který je obsažen v nativní aplikaci. [6] Jak vývoj hybridních aplikací pokračoval, organizace a vývojářská komunita musela vážně přemýšlet o klíčových změnách a posunech v oblasti zabezpečení aplikací. Pravděpodobnost útoku na hybridní aplikace je velmi vysoká, protože aplikace mají možnost přistupovat k nativním funkcím zařízení přes middleware hybridní platformy. [7] Jedním z takových middlewarů je Apache Cordova, která je popsána v kapitole 2.2.1.1. Příkladem používající tento přístup jsou technologie PhoneGap, Ionice, Trigger.io, Appery.io a další. Jiný přistup použití CPT dovoluje psát kód v programovacích jazycích jako je C#, nebo JavaScript, který poté bude překompilován do nativního kódu pro každou z platforem. Příkladem jsou technologie Xamarin, Appcelator Titanium, nebo Adobe Air, které využívají tento přístup vývoje. [6]



Obrázek 3: Porovnaní využití multi-platformních frameworků v letech 2019, 2020, 2021. Zdroj: [2]

2.2.1.1 Apache Cordova

Apache Cordova je knihovna, která dovoluje vytváření mobilních aplikací za pomocí HTML, CSS a Javascriptu. Logika aplikace je psána pomocí programovacího jazyka Javascript. Apache Cordova cílí na více platforem s jedním zdrojovým kódem.

Podporuje 8 platforem včetně platforem Android, iOS a Blackberry. Platformy využívající tyto knihovny jsou například PhoneGap, či Ionic. Cordova aplikace jsou implementovány jako WebView založené na prohlížeči v rámci nativní mobilní platformy. Platforma Cordova umožňuje přidávat závislosti a rozšíření (pluginy). Tím je možno přistupovat a pracovat s fotoaparátem, kontakty či geolokací. Největší nevýhodou Apache Cordova je horší výkon. [7]

2.2.1.2 Ionic

Framework Ionic je vyvinut pro vývoj interaktivních hybridních aplikací nad stávajícími komponentami nativních aplikací. Framework je postaven na Apache Cordova. To znamená, že obsahuje strukturu souborů Apache Cordova pro překlad HTML, CSS a JS. Ionic také nabízí rozšíření, které umožní používat nativní vlastnosti, ke kterým má poté prohlížeč přístup. Ionic lze psát pomocí JavaScript frameworků AngularJS, ReactJS a VueJS. Aplikace je spouštěna jako WebView, ale díky Cordova, či knihovně Capacitator, lze využívat nativních prvků zařízení. [26]

2.2.2 Xamarin

Xamarin je framework pro tvorbu multiplatformních uživatelských rozhraní. Byl vytvořen společností Microsoft. Xamarin podporuje schémata Model-View-Controller (MVC) a Model-View-View-Model (MVVM). [72] Byl integrován do IDE Visual Studio pod licencí open-source. Obsahuje vlastní abstrakci pro uživatelské rozhraní, které bude s použitím nativních nástrojů vy-renderována do platforem iOS, Android, nebo Universal Windows Platform (UWP). [53] Xamarin nabízí přístup k platformním SDK přes programovací jazyk C#, nebo .NET. [72] Xamarin obsahuje dvě hlavní SDK: Xamarin.Android a Xamarin.iOS. V případě iOS je zdrojový kód kompilován přímo do nativního Advanced RISC Machines (ARM) strojového kódu, zatímco při kompilace Xamarin Android aplikace je nejdříve aplikace kompilovaná do Intermediate Language (IL), a poté do nativního strojového kódu. Xamarin.Forms je ideální pro multiplatformní aplikace, kde není potřeba mnoho nativních funkcí platformy. S použitím Xamarin.Forms lze vytvořit nativní uživatelské rozhraní, které může být sdíleno mezi platformami Android, iOS a Windows Phone. [53] Nejvíce používaný přístup je vytvoření přenosné knihovny (Portable Library), nebo sdíleného projektu pro uložení kódu, a poté vytvořit aplikace, které z knihoven, nebo projektů budou čerpat. [72] Existují dva způsoby jak definovat uživatelské rozhraní v

Xamarin.Forms. První variantou je vytvořit uživatelské rozhraní přímo ve zdrojovém kódu pomocí jazyka C#. Druhou variantou je použití Extensible Application Markup Language (XAML) a deklarativně popsat uživatelské rozhraní. Takto vytvořené uživatelské rozhraní bude při kompilaci pře-renderováno do nativních User Inteface (UI) elementů na každou z platforem. [53]

2.2.3 React Native

React Native byl představen v roce 2015 společností Facebook a je stále vyvíjen. Tato platforma, založená na programovacím jazyce Javascript, byla vytvořena na základě webové knihovny React. React je velmi populární webovou knihovnou, proto se vývojáři z Facebooku rozhodli, že vytvoří verzi zaměřenou na mobilní aplikace, která by umožňovala vytvářet vysoce výkonné mobilní aplikace. [31] Je tedy aplikačním vývojovým frameworkem, který používá standardní webové technologie. Vývojář může použít jakékoliv IDE, které podporuje Javascript. [78]

JSX Jak React, tak React Native používají speciální syntaxi JavaScript XML (JSX). JSX je kombinace tří jazyků: HTML, CSS a JS. JSX kód je transformován do čistého JS, který se spustí za běhu. JSX pro vývoj není nutný, ale velice usnadní a zrychlí vývoj. [78]

Virtual DOM Stejné jako je tomu u webové knihovny React, i React Native využívá Virtual Document Object Model (DOM). DOM je stromová struktura, která reprezentuje elementy na stránce. Každý element odpovídá jednomu uzlu ve stromu DOM. Pokud uživatel provedl změnu, DOM se aktualizuje a prohlížeč použije DOM k vykreslení změn. S použitím DOM je prohlížeč nucen překreslit celou stránku, což může být velmi pomalé. Virtual DOM využívá jiného algoritmu k provedení nezbytných změn. React naslouchá změn ve stavech. Jakmile se vnitřní stav změní, React aktualizuje Virtual DOM. React ví, které elementy ve Virtual DOM byly změněny, proto aktualizuje pouze tyto elementy. Následně je porovnán předchozí Virtual DOM s aktualizovaným. Tím je o mnoho zvýšen výkon, než pracovat s celou strukturou stromu DOM. Pokud nastala změna stavu v React Native a je čas na vykreslení DOM na obrazovku, místo aktualizování DOM v prohlížeči a vykreslení stránky v prohlížeči, React Native vytvoří nativní komponenty a vykreslí je na obrazovku s použitím nativních metod. Ve webové verzi Reactu je element vykreslen jako element v DOM prohlížeče. Zatímco v React Native aplikacích

je <View> komponenta překreslena jako UIView komponenta na platformě iOS a View komponenta na platformě Android. Tím se z komponenty <View> stává nativní komponenta platformy. [78]

Nativní most React Native nabízí Javascriptové rozhraní k platformním API. To znamená, že je možno psát kód v Javascriptu a React Native bridge se postará o vše ostatní. Nativní most komunikuje s platformními nativními API, které používá k vytvoření nativních komponent a vykreslení komponent na obrazovku. [78] Most se nachází mezi dvěma hlavními komponentami: Nativním modulem (napsán v jazyce Java pro Android a Objective-C, nebo Swift pro iOS) a Javascript VM.[20] Při startu aplikace je spuštěno jedinečné hlavní vlákno, které je automaticky přiřazeno operačnímu systému zařízení. Ačkoliv je většina UI komponent napsána v Javascriptu, nakonec budou všechny vykresleny jako nativní.[78] React Native musí vytvořit nativní pohledy a namapovat je na JS komponenty.
O tento krok se postará UIManagerModul. Dále musí React Native skladovat a zobrazovat nativní pohledy. Tento úkol zajišťuje kontejner RootView, kde jsou nativní pohledy organizovány do velkého stromu. [20]

2.2.4 Flutter

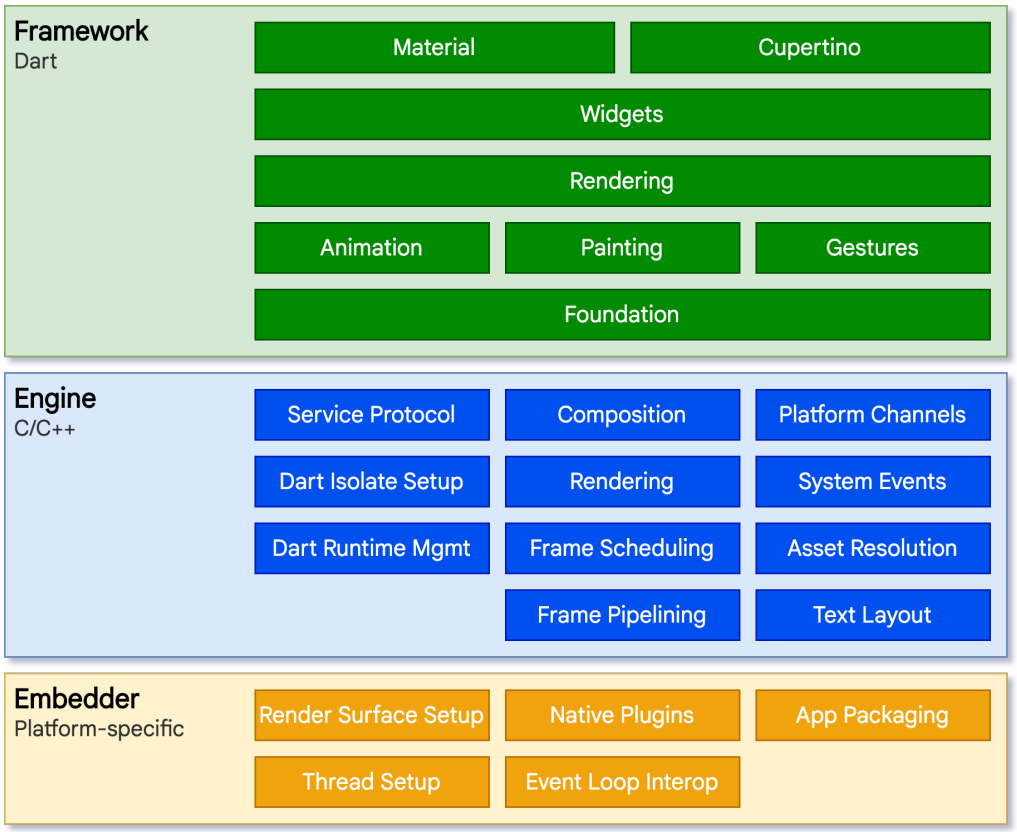
Flutter je framework pro vytváření nativně kompilovaných aplikací pro mobilní zařízení, počítač a web stejným kódem. [49] Programovací jazyk pro UI nástroj Flutter je Dart. Flutter byl představen v roce 2016 společností Google. Flutter je vybrán společností Google jako aplikační framework pro jejich další operační software. [70] Flutter využívá vlastní výkonný vykreslovací engine. Princip vykreslování widgetů je popsán v kapitole 2.2.4. [49] Z hlediska architektury C/C++ kód enginu zahrnuje kompilaci s Android Native Development Kit (NDK) a Low Level Virtual Machine (LLVM) iOS. Během kompilace je kód jazyka Dart zkompilován do nativního kódu. Funkce Hot Reload se ve Flutteru využívá při vývoji aplikací a jedná se o hlavní zrychlující faktor vývoje. Hot Reload je implementován tak, že posílá aktualizovaný kód do Dart VM, aniž by se změnila vnitřní struktura aplikace, a tedy přechody a akce aplikace budou zachovány. [70]

Flutter engine Flutter využívá vlastní výkonný vykreslovací engine, nazývaný Skia, k vykreslování UI komponent, takže není potřeba Original Equipment Manufacturer (OEM) widgetů. Engine je zodpovědný za nízko-úrovňovou implementaci

základního Flutter API, včetně podpory dostupnosti běhového prostředí Dart, grafické a textové rozvržení a architekturu pluginů. Platforma, kterou může být Android nebo iOS, nabízí sadu OEM widgetů, které tvoří UI platformy. Tyto widgety jsou zásadní, protože dávají možnost vykreslovat UI, používat canvas a reagovat na akce vyvolané například klepnutím prstu. [35] Skutečná síla Flutteru se tedy opírá o fakt, že aplikace jsou tvořeny vlastními vykreslovanými prvky a není třeba vykreslovat rozhraní s použitím OEM widgetů, které jsou závislé na platformě, a kde každá platforma nabízí odlišné OEM widgety a odlišné API. [4]

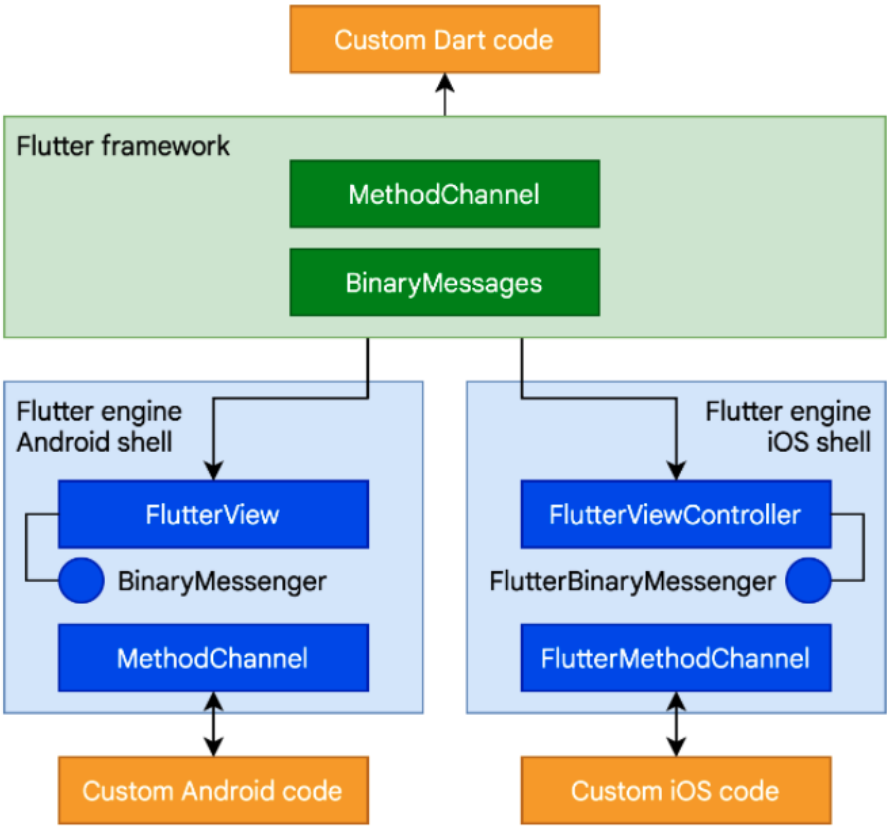
Architektura Flutter je vícevrstvý, rozšiřitelný systém. Existuje jako řada nezávislých knihoven, kde každá knihovna závisí na spodní vrstvě. Architektura frameworku Flutter je znázorněna na obrázku č. 4. Žádná vrstva nemá privilegovaný přístup do nižší vrstvy. Flutter aplikace jsou zabaleny stejným způsobem jako nativní aplikace. Embedder vrstva, závislá na konkrétní platformě, zajišťuje vstupní bod. Komunikuje s platformním operačním systémem. Zajišťuje přístup ke službám, jako jsou vykreslování povrchu, přístupnost a zajišťuje správu nad zprávami. [49] Embedder vrstva je napsaná v programovacím jazyce, který přísluší dané platformě. Java a C++ pro Android, Objective-C/C++ pro iOS a MacOS, C++ pro Windows. [35] Použitím vrstvy Embedder, může být Flutter kód integrován do existujících aplikací jako modul. Engine vrstva je nejdůležitější a zde se nachází samotný Flutter engine. Princip Flutter enginu je popsán v kapitole 2.2.4. Vrstva frameworku je dělena do několika vrstev, kde zespoda jsou [49]:

- Základní stavební bloky a služby jako animace, kreslení, gesta a další.
- Vykreslovací vrstva poskytuje abstrakci pro práci s rozložením. S touto vrstvou lze vytvořit strom vykreslovaných objektů, kde poté lze dynamicky s objekty manipulovat, přičemž strom se aktualizuje podle vykonaných změn.
- Vrstva widgetů je kompoziční abstrakcí. Každý vykreslovací objekt obsažený ve vykreslovací vrstvě má korespondující třídu v této vrstvě. Také nabízí využívat kombinace tříd, které lze znovu využít. V této vrstvě je představen reaktivní model programování.
- Knihovny Material a Cupertino. Tyto knihovny nabízejí komplexní sadu ovládacích prvků, které jsou implementovány v designu iOS, nebo Material designu pro Android.



Obrázek 4: Diagram architektury Flutter. Zdroj: [39]

Platformní kanály Flutter nabízí sadu mechanismů, kde lze přistupovat ke kódu nebo k API napsaném v jazyce Kotlin, či Swift. Jedná se o jednoduchý mechanismus pro komunikaci mezi nativním kódem na platformě a aplikací psané v jazyce Dart. Vytvořením nativního společného kanálu lze zasílat a přijímat zprávy mezi aplikací v jazyce Dart a platformními komponentami napsaných v jazyce Java, Kotlin, nebo Swift. Data jsou serializovaná z Dart standardizovaného typu Map, a poté de-serializovaná do ekvivalentní reprezentace v jazyce Kotlin (HasMap), nebo Swift (Dictionary). Na straně Dart aplikace se kanál vytvoří pomocí rozhraní MethodChannel. Na straně nativní platformy Android je to rozhraní MethodChannel a FlutterMethodChannel na platformě iOS. [35]



Obrázek 5: Komunikace přes platformní kanály. Zdroj: [39]

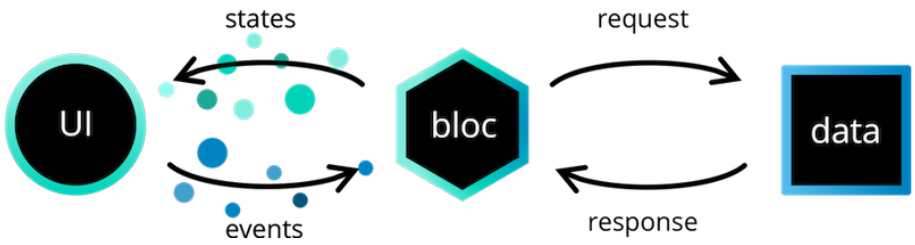
Widgety Flutter využívá widgety jako základní stavební jednotku aplikace. Widgety jsou stavební bloky UI rozhraní Flutter aplikace, kde každý widget je immutable deklarací každé části UI. Widgety tvoří hierarchii založenou na kompozici. Každý widget se vnoří do rodičovského widgetu a je schopen přistupovat ke kontextu právě od rodiče. Tato struktura se tvoří až do vrchního hlavního widgetu, kterým bývá kontejner aplikace jako MaterialApp, nebo CuppertinoApp. Aplikace aktualizuje UI v reakci na událost tím, že sdělí frameworku, aby nahradil widget v hierarchii jiným widgetem. Framework poté porovná nové a staré widgety a efektivně aktualizuje UI. [15]

Stateless Widget Tento typ widgetu se využívá, když je potřeba vytvořit uživatelské rozhraní, které se nebude v průběhu času měnit. Jedná se o samostatný blok, který není závislý na externích událostech nebo zdrojích, a spoléhá se pouze na interní data nebo data přes vlastní konstruktor. [39]

Statefull Widget Tento typ widgetu se využívá, pokud je potřeba vytvořit uživatelské rozhraní, které se bude v průběhu času měnit. V tomto případě se uživatelské rozhraní bude dynamicky měnit v důsledku vnějších událostí, jako je přijatá odpověď na požadavek, nebo zpětné volání vyvolané klepnutím na tlačítko. Hlavním rozdílem mezi Stateless a Statefull widgetem je, že Statefull widget může ukládat a pracovat s vnitřním stavem widgetu. [39]

Kompozice Widgety jsou obvykle složeny z kombinace malých a jednoúčelových widgetů. Flutter využívá stejného principu práce s widgety k reprezentaci kreslení na obrazovku, tvoření rozložení, pozicování, držení stavu widgetu, animacím a mnoha dalším reprezentacím. [15] Hiearchie tříd je záměrně abstraktní,
aby se docílilo co největší kombinace malých a skladatelných widgetů. Například nejpoužívanější widget Container je tvořen několika widgety zodpovědnými za tvoření rozložení, kreslení, pozicování a změnu velikosti. Je tedy složen z widgetů: LimitexBox, ConstrainedBox, Align, Padding, Decorator Box. [39]

Flutter Bloc Balíček Flutter Bloc vytvořil Felix Angelov. Je k dispozici v oficiálním úložišti balíčků (pub.dev). Jedná se o implementaci vzoru pro správu stavu BLoC, který byl představen na konferenci Google I/O v roce 2018. BLoC je zkratka pro Business Logic Components. Podstatou BLoC je, že vše v aplikaci by mělo být reprezentováno jako asynchronní proud událostí: widgety odesílají události, ostatní widgety na ně reagují. BLoC stojí uprostřed a řídí tuto konverzaci. [49] Tato architektura se příliš neliší od klasické MVC. Widget může komunikovat pouze s vrstvou BLoC. Vrstva BLoC posílá události datové vrstvě a vrstvě uživatelského rozhraní
a zpracovává servisní logiku. Tato struktura se dá s růstem aplikace dobře škálovat. Vzor BLoC je ve skutečnosti jen rozhraním kolem Dart Streams. [38] Celý Bloc využívá Event objektů a State objektů. Podstatou bloku je převod události na stav. Widget odešle událost do bloku, což je třída implementující určitou logiku. Blok je informován o tom, že do Streamu přišla nová událost. Zpracovává požadavky a poté vytvoří nový stav Widgetu. Naslouchající Widget obdrží nový stav předaný blokem a znovu se překreslí. [49] Flow diagram architektury BLoC je znázorněn na obrázku č. 6.



Obrázek 6: BLoC architektura. Zdroj: [59]

Flutter IOC *„V softwarovém inženýrství je injekce závislostí technika, při které objekt přijímá další objekty, na kterých závisí. Tyto další objekty se nazývají závislosti.“* [56] Zde přichází na řadu služba Get_it. Pomocí knihovny Get_it lze jednoduše zaregistrovat třídy/objekty Dart a třídy, na kterých jsou závislé, k nimž lze pak snadno přistupovat odkudkoli. [56] Pomocí knihovny Get_it lze třídy registrovat dvěma způsoby :

- Factory: Pokud je vyžádána instanci třídy, aplikace získá pokaždé novou instanci. Hodí se pro registraci ViewModel, které potřebují při spuštění spustit stejnou logiku, nebo které musí být nové při otevření View. [71]
- Singleton: Singletony lze registrovat dvěma způsoby. Poskytnout instanci třídy při registraci (registerSingleton), nebo poskytnout anonymní funkci, která bude vyvolána při první potřebě třídy (registerLazySingleton). Service locator uchovává jednu instanci registrovaného typu třídy po zbytek životnosti aplikace a na vyžádání tuto instanci vrátí.[71]

2.2.5 Dart

Dart je optimalizovaný, objektově orientovaný programovací jazyk pro tvorbu výkonných aplikací spustitelných na jakékoliv platformě. Dart se používá především ve Flutter frameworku. Dart je široce používaný společností Google a ověřený na tvorbu velkých aplikací, jako je například AdWords. Původně byl Dart navržen tak, aby nahradil jazyk Javascript. Z jazyka Javascript si však Dart převzal několik charakteristik, jako například klíčová slova `async` a `await` [70]. Pro vývojáře, kteří nejsou spokojeni s JS, je výhodou velice podobná syntaxe s programovacím jazykem Java. [49]

Podporované platformy Dart je velmi flexibilní jazyk díky prostředí, ve kterém může běžet. Kód, který byl napsán a testován, může být nasazen pomocí více cest:

Samostatně (Stand-alone) Stejně jako programovací jazyk Java nemůže běžet bez Java Virtual Machine (JVM), tak Dart nemůže být spuštěn bez Dart Virtual Machine (DVM). Je potřeba stáhnout a nainstalovat DVM, který spustí Dart v prostředí příkazové řádky. SDK kromě kompilátoru a knihoven obsahuje spoustu nástrojů jako [49]:

- pub - Balíčkovací systém.
- dart2js - Nástroj, který dokáže zkompilovat kód Dartu do nasaditelného Javascriptu.
- dartdoc - Generátor dokumentace k jazyku Dart.
- dartfmt - Nástroj, který pomáhá formátovat a udržovat čistotu kódu podle platných a oficiálních stylů a standardů.

Jinak řečeno tato varianta kompilování a běhu vyžaduje přítomnost nainstalovaného DVM.

AOT kompilování Ahead-Of-Time (AOT) je proces překládání vyššího jazyku, jako je Dart, do nativního strojového kódu. Ze zdrojového Dart kódu lze dostat jeden binární soubor, který může být spuštěn nativně v dané platformě. AOT je důvod, proč je Flutter rychlý a přenosný. S pomocí AOT, již není potřeba nainstalovaný DVM, protože na konci kompilace je binární spustitelný soubor. Pro Android je to typ souborů *.apk*, nebo *.aab*, pro iOS *.ipa*, a pro Windows *.exe*.
Od verze Flutter 1.21 je Dart SDK již obsazeno v SDK Flutteru, takže není potřeba jednotlivá SDK instalovat odděleně. Obě jsou zahrnuta v jednom balíku. Od verze 2.6 příkaz **dart2native** (podporovaný na Windows, iOS a Android) dokáže AOT kompilovat Dart do x64 nativního strojového kódu. Výstupem mohou být samostatné spustitelné soubory jako: *.apk*, *.aab*, *.api* a další. [49]

Web Pomocí nastroje dart2js může být Dart projekt přeložen do rychlého a kompaktního Javascript kódu. Dart pohání například webový framework AngularDart, který byl použit na tvorbu aplikací jako je AdSense a AdWords. AngularDart je taktéž produktem společnosti Google. [49]

Proč Flutter využívá Dart Je hodně důvodů, proč si Google vybral programovací jazyk Dart pro Flutter. Výčet několika důvodů [49]:

- Objected Oriented Programming (OOP) - Naprostá většina vývojářů má znalosti objektově orientovaného programování, a proto je Dart lehký na naučení

a adaptaci nejvíce využívaných OOP vzorů. Vývojáři se tedy nemusí vyrovnávat s novým typem programování, ale mohou využít dosavadní znalosti a začlenit je do jazyka Dart.

- Výkon - Aby byl zaručen vysoký výkon a předcházelo se padání frekvence obnovování snímků při běhu aplikace, je potřeba stabilní a předvídatelný jazyk. Dart může garantovat efektivitu a poskytuje výkonný alokátor paměti, který zpracovává malé a krátkodobé paměťové alokace.
- Produktivita - Flutter umožňuje vývojářům psát Android, iOS, webové a počítačové aplikace jedním kódem, který zachovává stejnou výkonnost na každé platformě. Vysoce produktivní jazyk jako Dart urychluje proces vývoje aplikací a vytváří framework více atraktivnější pro vývoj.

2.3 Využití mobilních senzorů

Senzory zaznamenávají a překládají fyzické atributy okolního světa do pozorovatelných elektrických impulzů. Mezi tyto atributy patří teplota, hmotnost, rychlost, tlak a další. Mikroprocesor zpracovává elektrické impulsy a poskytuje výstupy, které odpovídají hodnotám měřené veličiny. Systém postupně odesílá výstup do určených zařízení. Technologie použití senzorů se využívá v mnoha směrech každodenního života, zejména v chytré domácnosti jako prevence požáru a přehřívání nebo také správy osvětlení. Senzory se využívají celosvětově k zlepšení dopravy, léčebných procesů, nano-technologií, či umělé inteligenci. Staly se tak součástí každodenního života. Příkladem může být automatický mycí robot, který využívá vodní senzory, které chrání robota před poškozením. Samořídící vozidla Tesla využívají ultrazvukových senzorů k určení vzdálenosti mezi automobilem a objektem. [36]. V neposlední řadě i chytré mobilní telefony disponují celou řadou senzorů, které napomáhají analyzovat pohyby a práci s mobilním zařízením. [41]. Poskytují tedy neomezené možnosti tvorby aplikací, které pomáhají a mění životy lidí. [77] Toho se využívá například pro rozpoznání aktivity, kterou člověk právě provádí. Právě pomocí rozšiřujícího se množství senzorů v mobilních telefonech je oblast rozpoznání aktivity uživatele rychle rostoucí oblastí. Výhodou použití mobilních telefonů pro rozpoznání aktivity je přenosnost zařízení a mobilita zařízení v prostředí. Přístupy použití mobilních senzorů pro rozpoznaní aktivity byly popsány v článku [67] v roce 2015. Autoři kategorizovali typy existujících senzorů mobilních zařízení (akcelerometr, senzor teploty, gyroskop, světelný senzor, lineární akcelerátor, magnetometr, barometr), tak i aktivity, které lze těmito senzory rozpoznat. Podařilo se rozpoznat jak jednoduché

aktivity jako: chůze, sezení, stání, tak i složité aktivity jako: nakupování nebo řízení motorového vozidla. Oblast která velmi rezonuje s rozpoznáváním aktivit je sport, zejména pak cvičení a běh. Na trhu existuje nespočet aplikací, které využívají senzory pro zaznamenávání dat o cvičení. Vedle mnoha dalších aplikací existují aplikace, které se zaměřují pouze běh a chůzi (Nike+, Endomondo), nebo také cyklistiku a plavaní (Strava, MapMyRun). Ačkoliv jsou aplikace zaměřeny pouze na limitovaný počet aktivit, se kterými pracují, dosahují velmi dobrých výsledků v oblasti rozpoznání aktivit, kde přinášejí rozšíření jako automatické pozastavení aktivity, pokud se uživatel při aktivitě zastaví. Senzory se využívají také v oblasti počítačových her, kde se rozpoznání aktivity stalo aktivní součástí s technologiemi jako Kinect, PlayStation Move a Nintendo. [73]

2.3.1 Akcelerometr

Akcelerometr je typ převodníku, který převádí mechanický pohyb na elektrické signály. Měří zrychlení, které se projevuje při volném pádu, a které pociťují lidé a předměty. Takové zrychlení se populárně měří jako síla g. Princip akcelerometru využívá tedy setrvačnou sílu. [41] Tyto senzory se nyní nacházejí v mnoha předmětech včetně chytrých telefonů. Snímání vibrací, náklonu a zrychlení jsou jen některé z jejich využití. Skvěle se hodí pro sledování vozů, natáčení obrazovky telefonu nebo pro použití chytrého krokoměru. Akcelerometr dobře měří posunutí objektu, avšak nepřesně měří otáčivý pohyb zařízení. [36]

2.3.2 Gyroskop

Gyroskop je velmi citlivé zařízení, které dokáže dobře detekovat otáčivý pohyb. Nejčastěji se používá pro navigaci a měření úhlové a rotační rychlosti ve třech osách. Stejně jako akcelerometr, gyroskop vrací trojrozměrné hodnoty. Hodnota, kterou gyroskop vrací, je úhlová rychlost, která udává, jak rychle se zařízení otáčí kolem os. [36] Mezi jejich nejtypičtější využití patří navigační systémy do automobilů, herní ovladače, mobilní telefony a fotoaparáty, spotřební elektronika, ovládání robotů a drony.[41]

2.3.3 Magnetometr

Magnetometr je měřicí přístroj používaný k měření intenzity a případně směru magnetického pole. Akcelerometr a gyroskop jsou schopny zjistit směr pohybu, avšak

směr je relativní a řídí se souřadnicovým systémem, který používá mobilní zařízení. Někdy je zapotřebí různé mobilní zařízení synchronizovat, proto je k získání absolutního směru zapotřebí magnetometr, kde se směr řídí souřadnicovým systémem planety Země. [41]

2.3.4 Mikrofon

Mikrofon je velmi rozšířeným senzorem. Obvykle se používá k záznamu zvuku. Problémem je, jak se zaznamenaným zvukem pracovat. Nejběžnějším způsobem je najít ve zvukovém záznamu známou periodu zvuku. [41]

2.3.5 Optický senzor

Optický senzor je zařízení, které detekuje fyzikální množství světelných paprsků a převádí je na elektrický signál, který dokáže přečíst elektronické zařízení. Díky tomu se tyto senzory používají v různých odvětvích včetně zdravotnictví, monitorování životního prostředí, energetiky, letectví, kosmonautiky a mnoha dalších. [36]

2.3.6 Lokalizační technologie

Sportovní trackovací aplikace jsou velmi rozšířené a dostupné na trhu mobilních aplikací. Trackovací aplikace mapují aktivitu uživatele a dodávají zpětnou vazbu uživateli s výsledky jeho sportovní aktivity. Nicméně je nezbytné pozastavit se nad přesností jednotlivých aplikací, kde jsou prokazatelné rozdíly v měření pomocí GPS. Typicky se trasa sportovní aktivity uživatele vykresluje do map a grafů. Základem těchto aplikací jsou lokalizační technologie, které lze považovat za standardní součásti dnešních chytrých telefonů. Určovat polohu lze pomocí identifikátoru buňky (Cell ID), prostřednictvím bezdrátové místní sítě (WLAN), nebo prostřednictvím systému GPS. Avšak trackovací aktivity založené na WLAN nebo Cell ID nejsou pro účely těchto aplikací vhodné, protože výsledky určování polohy nejsou příliš přesné. Díky své přesnosti a celosvětové dostupnosti je GPS považována za nejvhodnější typ měření pro sportovní trackovací aplikace. Mobilní chytré telefony jsou osazeny základními GPS přijímači. V důsledku toho mohou být velké odchylky ve srovnání s vysoce kvalitními GPS přijímači. Protože chytré telefony jsou vybaveny různými čipovými sadami, úrovně přesnosti závisí také na příslušných zařízeních. [13] Ačkoli určovaní polohy pomocí GPS je velmi přesné, nehodí se na vnitřní lokalizaci např. v budovách, a v důsledku toho se využívají doplňkové polohovací systémy. [79]

Například určovaní polohy v metropolitním měřítku pomocí Wifi se stalo v USA realitou.

2.3.6.1 Určovaní polohy pomocí GPS

GPS je satelitní navigační systém vyvinutý Ministerstvem obrany USA pro vojenské účely. Globální polohovací systém (GPS) je nejrozšířenějším Global Navigation Satellite System (GNSS). Po mnoho let byl Global Positioning System (GPS) jediným GNSS dostupným pro civilní obyvatelstvo. Global Navigation Satellite System (GLONASS) začal být podporován mobilními zařízení v roce 2011. V roce 2016 více než 150 modelů chytrých mobilních zařízení podporovalo GPS i GLONASS. GLONASS nenahradil GPS, ale spíše zvýšil jeho pokrytí a přesnost. Historicky byl GPS jediným široce rozšířeným GNSS a stále je přední technologií GNSS. [63] K lokalizaci je zapotřebí nejméně čtyř nezávislých satelitů k určení přesné polohy zařízení. Vojenská verze GPS dosahuje mnohem vyšší přesnosti než veřejná, která je limitována na 5 až 10 metrů. Funkčnost GPS je však omezena, protože je vyžadována přímá viditelnost satelitů. Ačkoli rozšířenost přijímačů GPS dramaticky snížila náklady, velikost a požadavky na napájení, snímání GPS je ve srovnání s jinými technologiemi určování polohy poměrně náročné na napájení. [13] Moderní chytré telefony jsou vybaveny funkcí A-GPS. A-GPS využívá sítě chytrých telefonů v kombinaci s anténou GPS ke zvýšení rychlosti určení nebo fixace polohy. A-GPS vylučuje potřebu zahřívací doby, která je vyžadována u tradičních GPS jednotek. A-GPS nabízí vynikající přesnost, dostupnost a pokrytí za rozumnou cenu. A-GPS je složen z bezdrátového koncového přijímače (mobilní telefon), A-GPS serveru s propojením na GPS přijímač a bezdrátovou mobilní síť s mobilními vysílači. Síť dokáže přesně předpovědět GPS signál, který přijímač přijme a přenést tyto informace do mobilního telefonu, čímž výrazně snižuje velikost vyhledávacího prostoru a zkrácení TTFF z minut na sekundy, nebo méně.[25]

Lokalizační služby v aplikacích Existuje nespočet aplikací, které dokáží pracovat s kolekcí dat z GPS. V závislosti na aplikaci může člověk pracovat s obrázky poskytnutými vývojáři aplikace a exportovat uložené pozice pro použití v jiných aplikacích či softwarech. Chytrý telefon v kombinaci s vhodnou aplikací může poskytovat podobně kvalitní funkce jako základní GPS jednotka pro rekreační účely (např. Garmin). Většina současných výzkumů zahrnujících funkce GPS pro chytré telefony se zaměřuje na dopravu, nebo aplikace na monitorování lidského pohybu a sledování zdraví. [48]

Výpočet vzdálenosti mezi geografickými body Existují dvě široce používané metody pro výpočet vzdálenosti na kouli a eliptických útvarech. Je to Haversinova a Vincentyho metoda. Haversinův vzorec dokonale slouží k výpočtu vzdálenosti na sférickém útvaru, zatímco Vincentův vzorec se používá na eliptické útvarech. Odlišností algoritmů se zabývali Hagar Mahamoud a Nadine Akkari. V článku z roku 2016 [42] byly popsány rychlostní a přesnostní rozdíly v algoritmech. Testováním algoritmů na různém množství bodů (od 10 do 100000) bylo zjištěno, že Vincentyne algoritmus má až 2x delší výpočetní dobu, než algoritmus Haversine. Složitost algoritmu Haversin je lineární $O(n)$, kdežto u Vincentine algoritmu se jedná o složitost $O(\lambda * n)$, kde n je počet bodů a λ je startovací proměnná u algoritmu Vincent. Výpočet mezi dvěma body trvá, dokud je proměnná λ zanedbatelná. Test odhalil, že průměrně je iterováno 8-12x pro každé dva body v listu, což velmi zpomaluje celý výpočet. V druhém testu byla ověřovaná přesnost výpočtů algoritmů. Algoritmy byly využity na spočítání celkového obvodu země, kde výsledky byly porovnány s konstantní dobře známou hodnotu obvodu země (6378.1 km). Výpočtem Haversine algoritmu bylo dosaženo celkové vzdálenosti 6356.8km, kde jednoduchý podělením s konstantou se došlo k přesnosti 0.9966604 %, tedy chyba byla 0.334 %. Vincenty algoritmus předpokládá eliptický tvar Země. Vyžaduje, aby byla splněna podmínka $a = b > c$, kde a objemový poloměr, b je poloměr rovníku a c je poloměr po pólu. Výpočtem bylo docíleno vzdálenosti 6371km, a tedy přesnost byla 0.998886 %, kde chyba byla 0.1113 %. Výsledky bylo zjištěno, že přesnějším algoritmem je Vincenty algoritmus. [42]

Haversinova metoda Obecně se Haversinova metoda používá pro výpočet vzdáleností mezi dvěma souřadnicemi. Přibližný tvar Země je oblý sféroid. Neexistuje žádný geometrický tvar, který by planetu Zemi přesně popisoval. [75]. Haversinův vzorec se dokonale aplikuje na výpočet vzdálenosti na sférických útvarech.[64] Haversinův vzorec vypočítá vzdálenost mezi hlavním polohovým bodem a cílovým bodem na základě délky přímky, přičemž se bere hodnota vstupní zeměpisné délky a šířky. Zeměpisná šířka je reprezentována symbolem fí ϕ a je definována jako úhel mezi rovnoběžkou a rovníkem, přímkou v určitém místě a rovinou rovníku. Zeměpisná délka je reprezentována symbolem lambda λ a je definována jako úhel, který ukazuje na západ, nebo na východ od Greenwichského poledníku. Když je zeměpisná délka a zeměpisné šířky na obou místech, je třeba určit převést desetinné hodnoty na radiány, a poté se tato čísla použijí v algoritmu Haversine [75]. Haversine vzorec pro výpočet vzdálenosti je definován jako [32]:

$$
\begin{aligned}
&\Delta lat = lat2 - lat1 \\
&\Delta long = long2 - long1 \\
&a = sin^2(\frac{\Delta lat}{2}) + cos(lat1) * cos(lat2) * n^2(\frac{\Delta long}{2}) \\
&c = 2 * atan^2(\sqrt{a}, \sqrt{1-a}) \\
&d = R * c
\end{aligned}
\tag{2.1}
$$

kde:

R	: Konstanta obvodu země = 6371 (km)
Δlat	: Rozdíl zeměpisné šířky
$\Delta long$	: Rozdíl zeměpisné délky
d	: Vzdálenost (km)

Vzorec 5.1: Haversine vzorec. Zdroj: [32]

Vincenty metoda Vincentyho vzorec jsou dvě iterační metody, které vyvinul Thaddeus Vincenty. Vincentův vzorec používá přesný elipsoidický model Země. Poskytuje přesné výsledky pro vzdálenosti do 0,5 (mm) na použitém elipsoidu. Řeší dva geodetické problémy: Přímý problém a inverzní problém [42].

- Přímá úloha: Na elipsoidu je dán první bod (určený zeměpisnou šířkou θ a délkou λ), směr nebo azimut α tohoto bodu a vzdálenost od tohoto bodu k druhému bodu, určete zeměpisnou šířku θ a zeměpisnou délku λ tohoto druhého bodu.
- Inverzní úloha: Jsou dány dva body p1 (θ, λ) a p2 (θ, λ) a je nutné určit vzdálenost s mezi nimi.

2.3.6.2 Určování polohy pomocí WiFi

Určování polohy pomocí WiFi využívá WiFi AP k určení pozice. Všechny tyto AP opakovaně vysílají do okolí signál oznamující svou existenci. Tyto signály se obvykle pohybují několik set metrů všemi směry. Hustota AP v městských oblastech je tak vysoká, že se signály často překrývají a vytváří tak přirozený referenční systém pro určování polohy. WiFi polohovací software identifikuje signály WiFi v dosahu mobilního zařízení a vypočítá aktuální polohu zařízení. Pokrytí polohy WiFi je nejlepší v hustě obydlených oblastech. Výsledkem je, že určování polohy WiFi má vynikající pokrytí a výkon ve vnitřních prostorách. Tyto atributy jej odlišují od GPS, která

má potíže s poskytováním informací o poloze ve vnitřním prostředí. [79] Zjišťování polohy WiFi nevyžaduje navázání připojení k síti WiFi. Využívá se intenzity přijímaného signálu RSS. [13] Ve fázi určování polohy jsou signály na neznámém místě porovnány s databází dříve zaznamenaných lokací, aby se určila nejbližší shoda. Na tomto mechanismu polohování je přesnost určena s odchylkou 20 až 40 metrů a funguje uvnitř i venku. [79]

2.3.6.3 Určování polohy pomocí mobilních sítí

Buněčné sítě se rychle vyvinuly v rozsáhlou bezdrátovou komunikační infrastrukturu s téměř celosvětovým pokrytím. Oblasti mobilních služeb jsou rozděleny do buněk a každá z těchto buněk má spojenou základnovou stanici (mobilní věž). Byly vyvinuty techniky pro sledování klienta, když se pohybuje sítí. Tyto techniky správy polohy spoléhají na obousměrnou komunikaci mezi mobilním zařízením a sítí. Když se uživatel připojí k síti, mobilní zařízení je připojeno k vysílači s nejsilnějším signálem. Principem je tedy využít známé lokace vysílače, ke kterému je mobilní telefon připojen. Tato metoda je známa jako buněčná identifikace (Cell ID). Přesnost polohy závisí pouze na velikosti buňky. [79] GSM buňka může mít vzdálenost od 2 do 20 kilometrů průměr. [65] Některé buňky jsou rozděleny do různých sektorů směrovými anténami vysílače, což může podstatně snížit polohovou chybu. Dalšího zlepšení lze dosáhnout použitím síly přijímaného signálu, ačkoli síla signálu se může značně lišit v důsledku slábnutí, topografie, překážek a dalších faktorů. [79] Informace o ID buňky lze kombinovat s hrubým odhadem doby zpáteční cesty mezi zařízením a vysílačem („hodnota předstihu časování“), ze které lze odvodit vzdálenost mezi zařízením a vysílačem. [13] Tato technika je označována jako E-CID. S touto technikou lze dosáhnout lepší přesnosti. Bez ohledu na to, jaký konkrétní algoritmus určování polohy se použije, přesnost určování polohy buňky značně závisí na hustotě základnových stanic. Bylo tedy zjištěno, že horizontální chyba se velmi liší napříč městskými a venkovskými gradienty se střední chybou řádově 50 až několik set metrů v městských oblastech a řádově několik set metrů až kilometrů ve venkovských oblastech. V jedné ze studií [51] s použitím tří různých mobilních operátorů ve Spojeném království byla chyba 246 m v městském prostředí a 626 m ve venkovském prostředí. [79]

2.4 Ukládání a zpracování dat

Ke zpracování dat z klientské aplikace jsou využívány dva servery. Hlavní server je napsán pomocí frameworku Java Spring a ukládá zpracovaná data do NoSQL databáze. Framework Java Spring je více popsán v kapitole 2.4.2 a NoSQL databáze je popsaná v kapitole 2.4.1. Komunikaci mezí hlavním serverem a NoSQL databází zajišťují databázové repositáře, které jsou popsány více v kapitole 2.4.2.6. Druhý server je čistě výpočetní a je napsán v jazyce Python (viz. kapitola 2.4.3), který se více hodí na práci s daty, než uvedený Java Spring, jelikož podporuje nespočet knihoven, které velmi urychlý práci nad velkými formáty dat. Podrobná implementace serverů je popsána v kapitole 4.2.

2.4.1 NoSQL

V době, kdy systémy operují s masivními objemy dat v různých formátech a zdrojích, je zapotřebí flexibilní úložný prostor, který bude s takovými daty umět pracovat. Not Only Structured Query Language (NoSQL) je technologie, která reprezentuje třídu produktů, které nejsou principiálně stejné jako Relational Database Management System (RDBMS). [60] NoSQL databáze podporují velké svazky strukturovaných, nestrukturovaných, nebo polo-strukturovaných dat. [10] Svoji aplikaci našli v práci s velkými objemy dat a v realtime aplikacích. Vlastnosti jako horizontální škálovatelnost, flexibilní schéma, spolehlivost a odolnost proti chybám jsou jedny z výhod databází NoSQL. [60]

2.4.1.1 Dokumentové databáze

Dokumentové databáze jsou nejpopulárnější typ NoSQL databáze. Dokumenty jsou strukturovaná data ve formátech jako JavaScript Object Notation (JSON), XML. [60] Dokumenty jsou ukládány do kolekcí. Výhodou je rychlé vyhledávání, flexibilní struktura, která může být dosažena pomocí vnořených hierarchií. Nevýhodu je velká komplexnost pro implementaci. Operace, které mohou být aplikovány na dokumenty obsahují akce jako vkládání, aktualizování klíčů a dokumentů, nebo navracení, či vymazávání klíčů. [12] Dokumenty jsou dosažitelné pomocí klíčů a hodnoty v dokumentu mohou být indexovány a vyhledávány pomocí query dotazů. Nejznámější dokumentovou databází je MongoDB. Skládá se z dynamických schémat a využívá Binary JavaScript Object Notation (BSON) formát, který reprezentuje binární JSON dokumenty, a který má vliv na rychlost a jednoduchost datové integrity.

MongoDB podporuje dotazy, kde mohou být navráceny celé dokumenty, nebo jenom jmenovitá pole, či lze některé pole vynechat z odpovědi. [11]

2.4.2 Java Spring

Spring je nejpopulárnější vývojový framework pro programovací jazyk Java. Spring framework je open source platforma Javy a byl původně napsán Rodem Johnsonem, a vydán pod licencí Apache 2.0 v červnu 2003. Základní verze Spring frameworku má velikost přibližně 2 MB. Základní funkce Spring frameworku lze použít při vývoji libovolné aplikace v Javě, ale existují i rozšíření pro vytváření webových aplikací nad platformou Java EE. Cílem frameworku Spring je usnadnit vývoj a podpořit programování tím, že umožňuje pracovat s modely založenými na POJO. Spring je modulární a umožňuje tedy vybrat si, které moduly zapojit, a které vynechat. [23]

2.4.2.1 IOC kontejner

Kontejner Spring je jádrem frameworku Spring Framework. Kontejner vytváří objekty, propojuje je, konfiguruje a řídí jejich kompletní životní cyklus od vytvoření až po zničení. Kontejner Spring používá ke správě komponent, které tvoří aplikaci, metodu DI (Dependency injection). Tyto komponenty se nazývají Spring Beans. Kontejner získá instrukce, jaké objekty má instancovat, konfigurovat a sestavit, přečtením poskytnutých konfiguračních metadat. Konfigurační metadata mohou být reprezentována buď XML, anotacemi jazyka Java, nebo kódem jazyka Java. Kontejner Spring IoC využívá třídy Java POJO a konfigurační metadata k vytvoření plně nakonfigurovaného a spustitelného systému nebo aplikace.[23]

2.4.2.2 Spring BeanFactory

Jedná se o nejjednodušší kontejner poskytující základní podporu pro DI. BeanFactory a související rozhraní, jako BeanFactoryAware, InitializingBean, DisposableBean, jsou ve Spring stále přítomny z důvodu zpětné kompatibility s velkým množstvím frameworků třetích stran, které se integrují se Springem.[23]

2.4.2.3 Application Context

Tento kontejner přidává další specifické podnikové funkce jako je možnost překládat textové zprávy ze souboru vlastností a možnost publikovat události aplikace zainteresovaným posluchačům událostí. Tento kontejner je definován rozhraním

org.springframework.context.ApplicationContextinterface.[23]

2.4.2.4 Beans

Objekty, které tvoří základní komponenty aplikace, a které spravuje kontejner Spring IoC, se nazývají Beans. Bean je objekt, který je instancován, sestaven a spravován kontejnerem Spring IoC. Tyto Beany jsou vytvářeny pomocí konfiguračních metadat, které kontejner čtě například ve formě definic XML <bean/> z konfiguračního souboru, nebo anotací @Bean a dalších.[23]

2.4.2.5 Dependency injection ve Springu

Technologie, se kterou je Spring nejvíce ztotožňován, je Dependency Injection (DI). Inversion of Control (IoC) je obecný koncept, který lze vyjádřit mnoha různými způsoby a Dependency Injection je pouze jedním z konkrétních příkladů IoC. Při psaní komplexní aplikace v jazyce Java, by měly být třídy aplikace co nejvíce nezávislé na jiných třídách jazyka Java, aby se zvýšila možnost opakovaného použití těchto tříd a jejich testování nezávisle na jiných třídách při provádění Unit testů. Dependency injection může probíhat formou předávání parametrů do konstruktoru, nebo dodatečně pomocí metod setter. [23]

2.4.2.6 Práce nad MongoDB databází

MongoDB je dokumentová databáze. Dokumenty jsou většinou ukládány jako Map, které jako hodnoty mohou obsahovat textové řetězce, kolekce, nebo vnořené dokumenty. MongoDB ukládá data ve formátu BSON, což je binární derivace od formátu JSON.

MongoTemplate Třída MongoTemplate je hlavní třídou podpory MongoDB ve Springu a poskytuje sadu funkcí pro interakci s databází. Šablona nabízí pohodlné operace pro vytváření, aktualizaci, mazání a dotazování dokumentů a poskytuje mapování mezi doménovými objekty a dokumenty MongoDB. Třída MongoTemplate implementuje rozhraní MongoOperations. Metody rozhraní MongoOperations jsou v co největší míře pojmenovány podle metod dostupných v objektu ovladače MongoDB Collection, aby bylo API známé stávajícím vývojářům MongoDB, kteří jsou zvyklí na API ovladače MongoDB. MongoOperations má také rozhraní API pro operace Query, Criteria a Update.

```
public Activity pushLikeActivity(Activity activity, Account account) {
        UpdateResult updateResult = mongoTemplate.updateMulti(
                new Query(where("id").is(activity.getId())),
                new Update().push("likes", account.getId()),
                Activity.class
        );
        if (!updateResult.wasAcknowledged() || updateResult.getModifiedCount() < 1) {
            throw new UserPropagableException("Cannot do opeartion to like activity");
        }
        return getNotNull(activity.getId());
    }
```

Zdrojový kód 3: Využití MongoTemplate pro dotaz do databáze. Zdroj: Vlastní

MongoRepository Někdy aplikace vyžadují použití více než jednoho modulu Spring Data. V takových případech musí definice úložiště rozlišovat mezi technologiemi persistence. Hlavním rozhraním abstrakce úložiště Spring Data je Repository. Jako typové argumenty přijímá třídu doménového objektu, kterou má spravovat a typ ID třídy. Toto rozhraní slouží především jako značkovací rozhraní, které zachycuje typy, s nimiž se má pracovat. MongoRepository rozšiřuje rozhraní PagingAndSortingRepository a QueryByExampleExecutor, která dále rozšiřují rozhraní CrudRepository. Dotazy do databáze jsou tedy vytvářeny automaticky v závislosti na parametrech a názvu metody.

2.4.3 Python

Python je velmi populární, vysoko úrovňový programovací jazyk, který vznik v roce 1991 a od té doby se stal nejpoužívanějším programovacím jazykem na světě. Jedná se o objektově orientovaný, dynamicky typovaný, mnoho paradigmatický programovací jazyk. Python se stále více používá ve vědeckých aplikacích, v nichž tradičně dominují skriptovací jazyky jako: R, MATLAB, Stata a mnoho dalších. [46]

2.4.3.1 Numpy

NumPy (Numerical Python) je balíček pro vědecké výpočty v Pythonu. Nabízí rychlé a efektivní multi-dimenzionální práce nad poli, či práci s maticovými operacemi. Jedná se o knihovnu, která dala směr a primární účel využití Pythonu na datovou analýzu. [47]

2.4.3.2 Pandas

Knihovna Pandas nabízí bohaté datové struktury a funkce pro práci s rychlými, jednoduchými a stručnými operacemi nad strukturami. Jedná se o kritickou knihovnu pro využití Pythonu na náročné a produktivní datové analýzy. Primární objekt knihovny Pandas se nazývá DataFrame. Jedná se o dvou-dimenzionální, sloupcově orientovanou tabulku s řádkovými a sloupcovými popisky. Pandas kombinují vysoce výkonné funkce NumPy pro výpočet polí s flexibilními možnostmi manipulace s tabulkovými daty a daty relačních databází. Nabízí sofistikované indexovaní pro rychlé operace nad daty jako jsou: přetvoření, půlení, slepování, agregování, vybíraní a další. Jedná se o knihovnu, která bude nejvíce využívána v této práci. [47]

2.4.3.3 Matpotlib

Jedná se o populární knihovnu pro vytváření grafů a 2D datových vizualizací. Knihovna byla vytvořena Johnem D. Hunterem a nyní je vyvíjena velkým týmem vývojářů. Velice se hodí pro vytváření grafů a vizualizací do vědeckých publikací. I v této práci jsou grafy tvořeny knihovnou Matpotlib. Integrace s Pythonem je velmi komfortní a nabízí interaktivní prostředí pro prozkoumání vizualizace nad daty. Grafy jsou interaktivní. Je možné přibližovat nebo oddalovat sekce v grafech. [47]

2.5 Využití chytrých hodinek

Chytré hodinky jsou mini-počítače, které mají nespočet funkcí a jsou jedním z nejnovějších vývojových trendů v oblasti informačních technologií. Odborná literatura neposkytuje přesnou definici technologie chytrých hodinek a postrádá jasné rozlišení od příbuzných technologií jako jsou chytré náramky nebo „fitness trackery“. Chytré náramky nebo „fitness trackery“ jsou zařízení, která sledují fyzické funkce uživatele (např. puls) a poskytují informace zobrazované na vlastních displejích. To znamená, že primárním účelem těchto zařízení je sběr dat, která může uživatel analyzovat na jiném zařízení (např. notebooku nebo chytrém telefonu). Prezentace informací je velmi omezená (např. tep nebo čas) a chytré náramky nenabízejí možnost instalace aplikací. Naproti tomu chytré hodinky jsou větší než chytré náramky a jsou často ještě větší než většina tradičních hodinek. Ciferník chytrých hodinek je obvykle dotykový displej. Operační systém umožňuje uživatelům instalovat různé aplikace. Chytré hodinky by mohly zvýšit pozornost uživatele, protože se stávají

komunikačním centrem, které umožňuje přístup k e-mailům, zprávám a mnoha dalším informacím. Zvýšený zájem o novou technologii se odráží také v obrovském množství aplikací nabízených pro chytré hodinky. Na rozdíl od chytrých náramků přinášejí chytré hodinky největší výhody, když jsou připojeny k internetu (WiFi, mobilní internet nebo Bluetooth). Primárním účelem chytrých náramků je sběr dat. Prezentace relevantních informací (např. oznámení z Facebooku, e-maily) je primární funkcí chytrých hodinek. [18]

Obrázek 7: Sportovní aplikace Strava na Apple Watch. [1]

Řada licencovaných hodinek Google „Android Wear“ a Apple s vlastním názvem „Apple Watch“ přebírají funkce z chytrých mobilních telefonů, a to v novém provedení a v nové podobě s řadou nových designových inovací. Android Wear a Apple Watch dosáhly v posledních letech obrovské popularity. Nejzákladnější funkcí chytrých hodinek je zobrazování času a možnost výběru vzhledu z knihovny ciferníků, které zobrazují čas spolu s dalšími údaji, jako je počasí nebo ukazatel počtu kroků. Hodinky Apple Watch dokáží vypnout displej, aby šetřily energii baterie, pokud uživatel s hodinkami nepracuje a rozsvítí se jen tehdy, když je rozpoznán pohyb ruky uživatele nebo když se uživatel dotkne obrazovky. Každé hodinky jsou propojeny s telefonem uživatele a přenášejí informace z telefonu do hodinek. Oznámení, která přicházejí na telefon (např.: textové zprávy, telefonní hovory nebo aplikace), jsou přeposílány do hodinek, kde vytvářejí zvuk nebo vibrace na zápěstí uživatele. Pokud uživatel zvedne ruku během několika sekund, zobrazí se podrobnosti o notifikaci. Po přečtení se oznámení na hodinkách ukládá pro případné znovu-nahlédnutí, dokud jej uživatel nesmaže. Chytré hodinky Android Wear nebo Apple Watch umožňují posílat zprávy, pouštět aplikace, nebo provádět různé úkoly pomocí hlasových příkazů. Kroky uživatelů se měří pomocí různých senzorů, které monitorují fyzickou aktivitu. [58]

3 NÁVRH APLIKACE

Cílovou aplikací bude sportovní trackovací aplikace pro běh, cyklistiku a in-line brusle. Jedná se o nejčastější sporty, které jsou zaznamenávány do mobilních aplikací. Uživatel aplikace bude mít možnost zaznamenat svoji aktivitu, kde v průběhu aktivity bude schopen nahlížet na aktuální data. Uživatel může v aplikaci prohlížet historii aktivit a jejich detailní analýzy. Aplikace nebude podporovat chytré hodinky z důvodu aktuální neoficiální podpory vývoje pomocí Flutter frameworku. V této kapitole jsou detailně popsány funkční a nefunkční požadavky na aplikaci, včetně návrhu jednotlivých obrazovek aplikace.

3.1 Funkční požadavky

V následující sekci jsou zobrazeny funkční požadavky na aplikaci. Funkční požadavky jsou vyobrazeny na diagramu požadavků na obrázku č. 9. Základní případy užití jsou vyobrazeny na obrázku č. 8.

3.1.1 Požadavky na zaznamenávání polohy

Pro přesné určení polohy uživatele je nutné využít přesné pozice GPS. Je nutné zaznamenávat čas, zeměpisnou šířku, zeměpisnou výšku a nadmořskou výšku každé zaznamenané polohy uživatele.

3.1.2 Požadavky na mapu v aplikaci

Při aktivitě je nutné pohybovat mapou v aplikaci tak, aby kamera mapy byla vždy nad aktuální pozicí uživatele. Uživatel musí mít možnost pohybovat mapou tak, aby měl přehled o své poloze. Zaznamenané body trasy je nutné propojit a zvýraznit jednotnou barvou, která bude udávat trasu aktivity. Mapa musí zobrazovat typ „uliční mapy“, aby se uživatel mohl orientovat podle detailních informací na mapě, které v jiných typech map nejsou detailně zobrazeny.

3.1.3 Požadavky na sekci klubů

Uživatel má možnost se připojit ke klubu, a sdílet tak aktivity v klubu. Každý klub obsahuje statistiky aktivit v klubu. Zároveň jsou zobrazeny všechny aktivity,

které jsou zde sdíleny. Uživatel si může volně prohlížet další uživatele, kteří jsou zde připojeni. Může však také klub jednoduše opustit.

3.1.4 Požadavky na uživatele

Pro přístup do aplikace je nutné, aby se uživatelé přihlásili, a to zahrnuje předchozí registraci. Uživatel může sledovat další uživatele tak, aby měl přehled o aktivitách sledovaných uživatelů. Uživatel vidí v aplikaci všechny aktivity, včetně aktivit sledovaných uživatelů. Na profilu uživatele vidí aktivity za poslední týden, včetně dalších statistik jednotlivého uživatele (viz. kapitola 3.1.5).

3.1.5 Požadavky na statistiky uživatelů

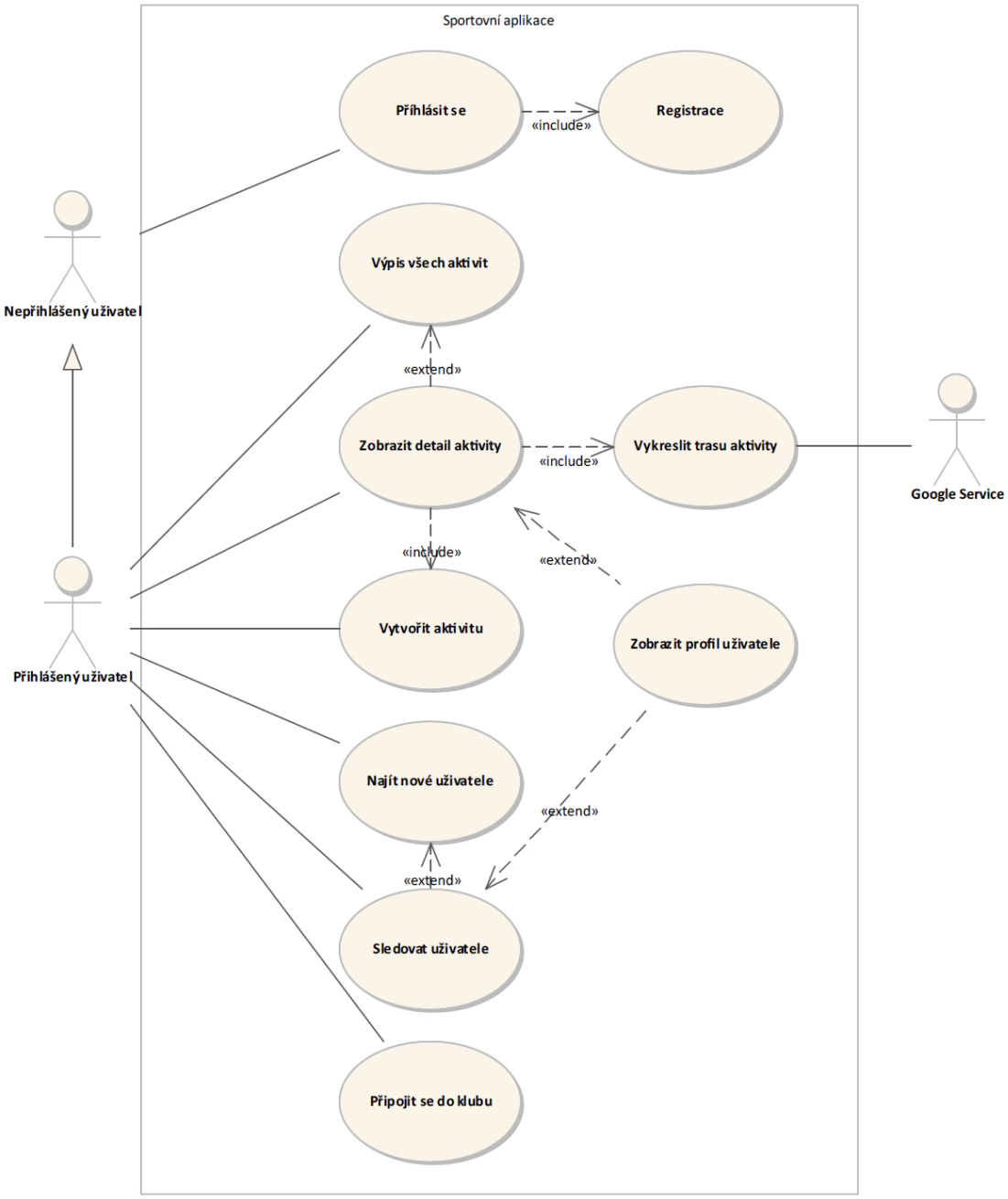
Uživatel má k dispozici statistiky o jeho profilu, které jsou seskupeny podle typu aktivity. Mezi statistiky patří například nejrychlejší časy na dané úseky. Tak uživatel může zjistit jaký je rekordní čas na trase například 10 km nebo celková vzdálenost všech aktivit podle typu, či celkový čas všech aktivit.

3.1.6 Požadavky na detail aktivity

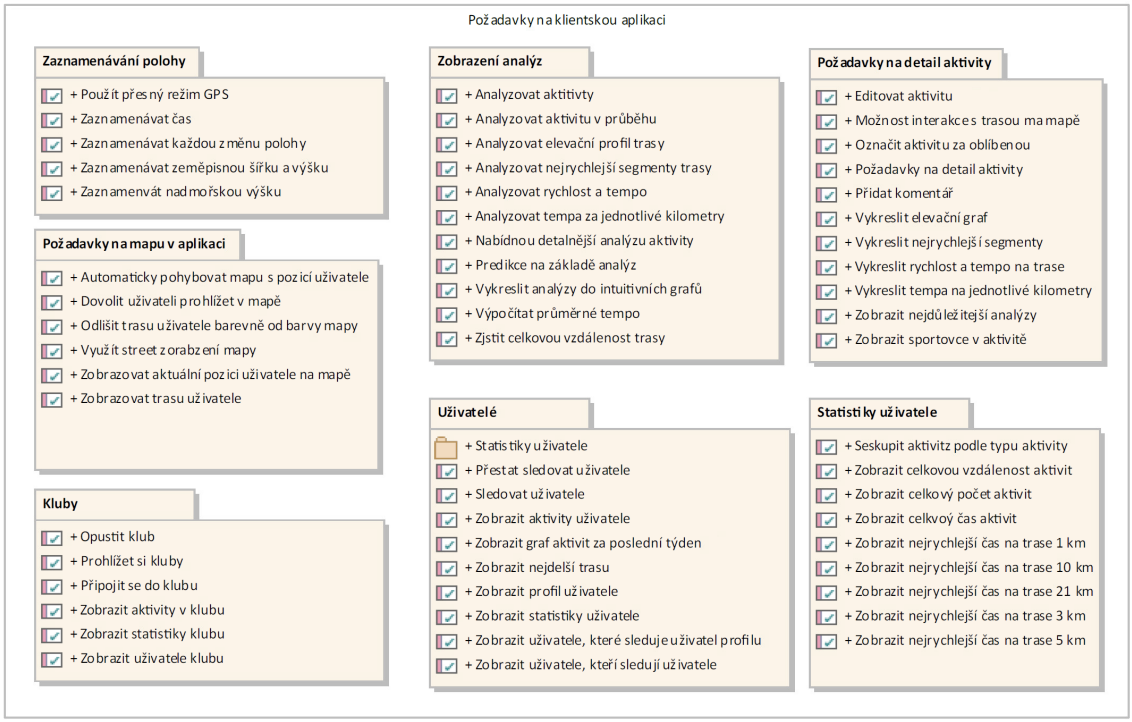
Detail aktivity je možné editovat, pokud je uživatel vlastníkem aktivity. Je možné si detailně a interaktivně prohlížet trasu aktivity na mapě. Lze přidat komentář k aktivitě, nebo aktivitu označit za oblíbenou, jako je tomu u sociálních sítí. Uživatel může lehce analyzovat aktivitu pomocí interaktivních grafů, které se nacházejí u aktivity. Mezi grafy patří například graf převýšení, nejrychlejších segmentů na trase, tempa za jednotlivé kilometry, či graf gradientu. Ostatní statistiky jsou vypsány do tabulky. Uživatel také vidí sportovce, kteří byli přidání k aktivitě.

3.1.7 Požadavky na analýzy aktivit

Je nutné analyzovat každou ukončenou aktivitu. Částečná analýza je nutná už i v klientské aplikaci, kde je nutné zjišťovat průběžné tempo, celkovou vzdálenost a čas aktivity. Podle těchto údajů pak uživatel může při aktivitě nahlížet na aktuální výsledky. Po ukončení aktivity je nutné analyzovat převýšení trasy, rychlosti na trase, tempa na trase, vykreslit údaje do grafů a zjistit základní parametry aktivity.



Obrázek 8: Diagram případů užití. Zdroj: Vlastní



Obrázek 9: Diagram funkčních požadavků. Zdroj: Vlastní

3.2 Nefunkční požadavky

Nefunkční požadavky byly rozděleny na požadavky klientské aplikace a požadavky serverové části aplikace. Všechny nefunkční požadavky na aplikaci jsou zobrazeny na Unified Modeling Language (UML) diagramu nefunkčních požadavků č. 10.

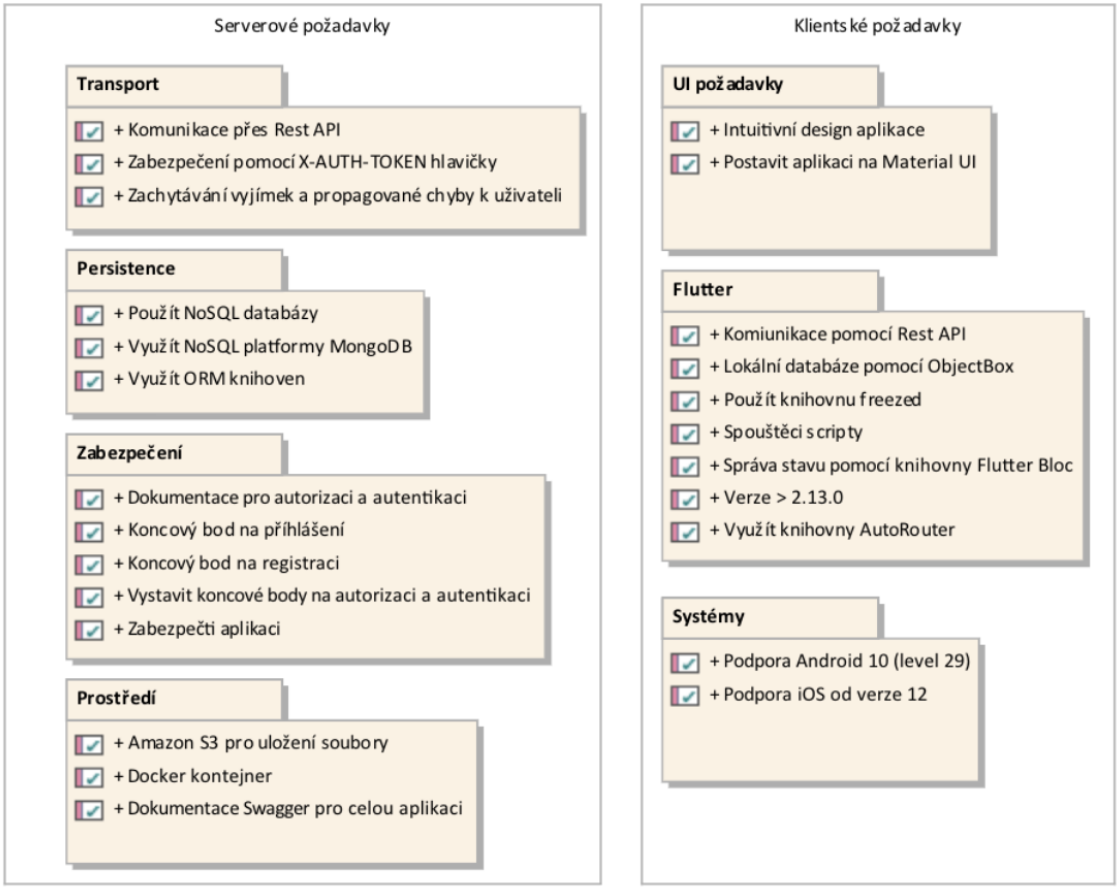
3.2.1 Klientské nefunkční požadavky

Nefunkčními požadavky klientské aplikace je nutné zajistit intuitivní design aplikace, který je postavený na hotovém Material UI designu, který plně Flutter podporuje, a nabízí nespočet hotových komponent, které lze rychle a jednoduše využít. Klientská aplikace bude se serverem komunikovat pomocí Representational state transfer (REST) API, bude podporovat lokální databázi ObjectBox pro lokální ukládání pozic trasy aktivity a používat knihovnu Freezed, která velmi zrychlí psaní tříd pro serializaci/deserializaci JSON do objektů. Aplikace musí být lehce použitelná a připravená i pro vývojáře, proto je nezbytné zajistit a připravit scripty na správu Flutter aplikace. Hlavním prvkem nad stavem celé aplikace bude Flutter Bloc, který

bude obsluhovat komunikaci mezi REST API a UI Widgety v aplikaci. Pro navigaci v aplikaci je nezbytné použít knihovny AutoRouter a podporovat verzi Flutteru 2.13.0 a vyšší.

3.2.2 Serverové nefunkční požadavky

Celá serverová část bude s klientem komunikovat pomocí REST API, proto je nutná kompletní dokumentace všech koncových bodů kontrolérů. Celá dokumentace je dostupná pomocí nástroje Swagger a Open-API. Jelikož klientská aplikace vyžaduje přihlášení uživatele, je nutné aby server poskytoval API pro přihlášení a registraci, dokázal rozpoznat token v hlavičce požadavku a zachytával běhové chyby serveru, které by se neměly dostat do klientské aplikace. Databázová vrstva využívá NoSQL databáze s platformou MongoDB, kde je možné aplikovat Object Relation Mapping (ORM) na databázi pro jednodušší práci s objekty. Pro ukládání obrázků a souborů se využívá služby třetích strany Amazon S3. Pro lepší spouštění a vysazení je připraven Docker kontejner, který spouští databázi, výpočetní Python server, Amazon S3 lokální úložiště a vlastní Java server.

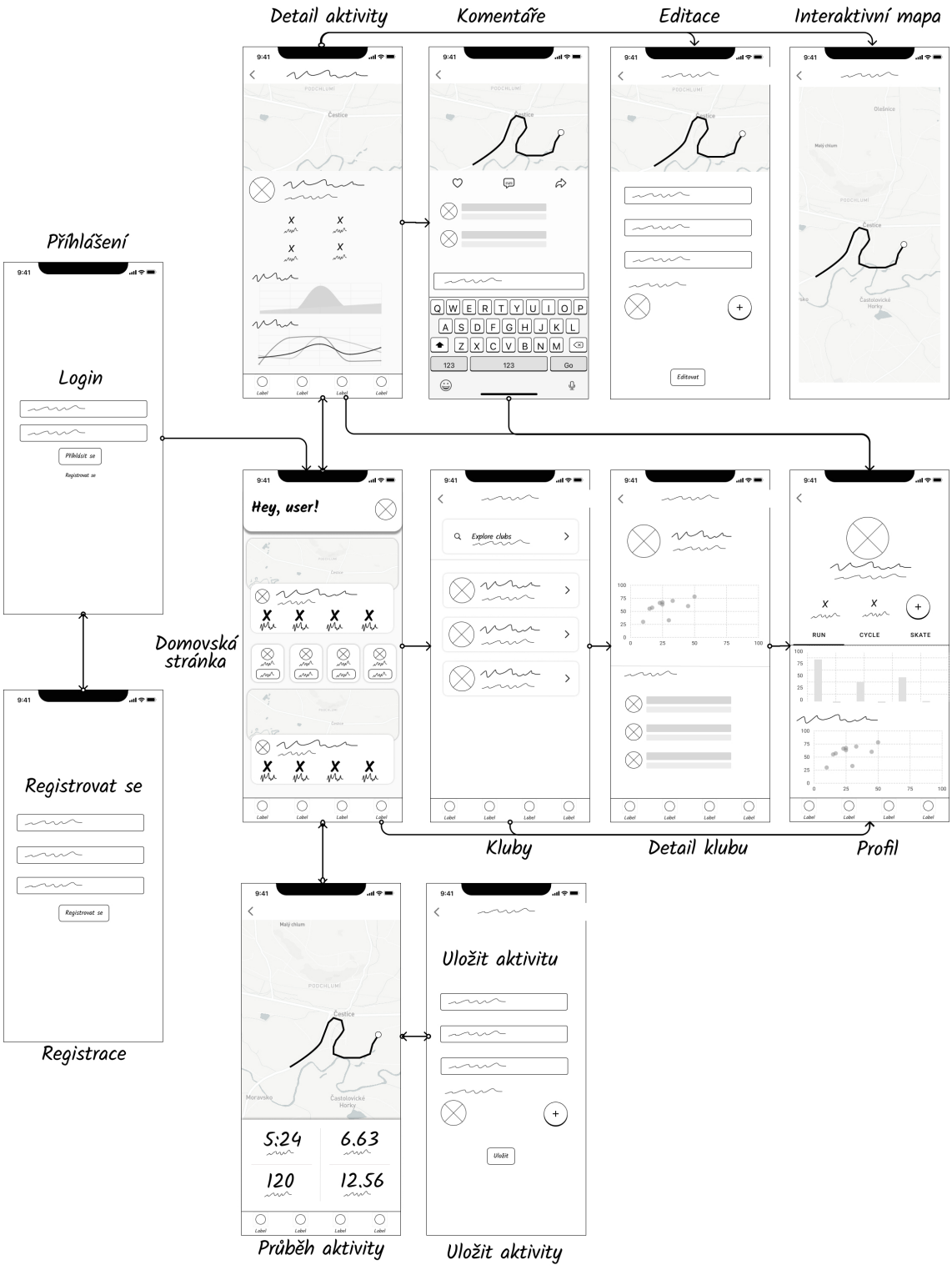


Obrázek 10: Diagram nefunkčních požadavků. Zdroj: Vlastní

3.3 Navigace mezi obrazovkami

Na flow diagramu č. 11 je zobrazen diagram přechodu mezi obrazovkami. Uživatel jako první vidí obrazovku přihlášení. Pokud uživatel není zaregistrován, může se jednoduše dostat na stránku registrace, kde může registraci provést. Následně se přihlásí přes obrazovku přihlášení, kde se po úspěšném přihlášení dostane na hlavní obrazovku. Z hlavní obrazovky se může navigovat pomocí dolní navigace, či přecházením na detail aktivity, či sledování uživatele. Pokud se uživatel dostane na detail aktivity, je zde možnost se navigovat na editaci, interaktivní mapu, sekci komentářů, či profil uživatele. Pokud se uživatel z hlavní obrazovky přesune pomocí dolní navigace na obrazovku klubů, tak zde může objevit nové kluby, připojit se ke klubu, nebo se navigovat na detail klubu, ze kterého se lze dostat na profil každého člena klubu. Pokud uživatel z hlavní stránky vstoupí na obrazovku nové aktivity, může ji zde i začít. Po dokončení aktivity se lze dostat na uložení aktivity, a tak aktivitu

uložit.



Obrázek 11: Diagram přechodů mezi obrazovkami. Zdroj: Vlastní

3.4 Obrazovky

V následující sekci jsou podrobně popsány funkcionality jednotlivých obrazovek, které byly zmíněny v diagramu přechodu v kapitole č. 3.3.

3.4.1 Domovská stránka

Domovská stránka slouží jako rozcestník celé aplikace. Na hlavní stránku se lze dostat pouze jako přihlášený uživatel, tedy pouze přes stránku přihlášení. Hlavní stránka v hlavičce obsahuje Avatar uživatele, kde se lze pomocí dotyku na Avatar přesměrovat na profil uživatele. V posuvném Widgetu, hned pod hlavičkou se bude nacházet výpis všech aktivit, jak přihlášeného uživatele, tak uživatelů, které uživatel sleduje. Tento prvek vychází z definice většiny sociálních sítí, kde se uživatelé mohou navzájem sledovat, tudíž mít přístup a vidět aktivitu sledovaného uživatele. Trasa aktivity bude nastíněna pomocí statické Google mapy (viz. kapitola č. 4.1.4.1), tedy mapa bude zobrazena jako obrázek a nebude nijak interaktivní. Pod mapou se budou nacházet základní informace o aktivitě jako je: vzdálenost trasy, celkový čas, tempo, titulek a popisek od uživatele, ikona uživatele s možností přesměrování na profil uživatele. Takto budou aktivity, které uživatel může vidět poskládány ve sloupci postupně pod sebou. Vloženou sekcí mezi aktivitami bude komponenta, kde bude moci uživatel objevovat nové uživatele. Doporučení uživatelé budou zobrazeni v kartách postupně za sebou v řádku, kde v případě velkého množství uživatelů, lze rolovat do strany. Drátěný diagram této stránky je zobrazen na obrázku č. 12 pod písmenem a).

3.4.2 Detail aktivity

Detail aktivity obsahuje detailní informace a analýzy konkrétní aktivity. Nachází se zde o mnoho více údajů, něž ve výpisu na hlavní stránce. Nejvýše na stránce bude umístěna statická Google mapa s trasou aktivity. Pokud uživatel dvakrát poklepe na mapu bude přesměrován na interaktivní mapu na další stránku. Hned pod statickou mapu se nachází informace o uživateli, včetně připojeného titulku a popisku. Zároveň jde zde označit aktivitu jako oblíbenou, či se přesměrovat na stránku komentáře pomocí příslušných ikon. Hlavní sekcí celé stránky jsou statistiky o aktivitě. Tyto statistiky jsou počítány na serveru, a poté jsou zobrazovaný do pravidelné mřížky. Mezi statistiky bude zobrazeno například: vzdálenost celkové trasy, tempo, průměrná rychlost, převýšení, počet nastoupaných metrů, počet sestoupaných metrů

a průměrné tempo. To jsou základní statistiky, které obsahuje většina sportovních aplikací. Pokud uživatel běžel s nějakými dalšími uživateli a při uložení aktivity uživatele přidal do aktivity, budou zobrazeny v řádcích pod statistikami. Důležitou součástí statistik jsou grafy k dané aktivitě. Zde se budou nacházet grafy jako je průběh převýšení, průběh tempa, tempo na kilometry, či gradient trasy. Potřebné grafy k aktivitě včetně výpočtů jsou uvedeny v kapitole č. 4.2.3. Drátěný diagram této stránky je zobrazen na obrázku č. 12 po písmenem b).

3.4.3 Profil uživatele

Profil uživatele je dostupný téměř z každé stránky aplikace. Lze se sem přesměrovat z každého Avataru uživatele v aplikaci. Na stránce jsou zobrazeny informace a záznamy ke konkrétnímu uživateli. V horní části se nachází sekce představení uživatele, kde je vykreslen Avatar s uživatelským jménem. Poté následuje sekce sociální sítě, kde je možné si zobrazit uživatele, jaké uživatel sleduje, či kým je uživatel sledován, popřípadě pomocí tlačítka s ikonou +, lze uživatele začít sledovat. I na této stránce se nachází kumulované grafy k uživateli. Na prvním grafu jsou zobrazeny dny, týden od aktuálního dne, kde jsou kumulativně se-sčítány celkové uběhnuté kilometry za jednotlivé dny v týdnu. Takto se lze jednoduše orientovat, jak často, a jak náročně uživatel sportuje. Grafy jsou seskupeny podle typu aktivity, kde pro každý typ aktivity jsou kumulovány pouze dané typy aktivity. Další grafy jsou vztažené k typu aktivit a kumulativním vzdálenostem, kde lze predikovat další aktivity uživatele například pomocí lineární regrese. Grafy a výpočty jsou v kapitole č.4.2.3. Drátěný diagram této stránky je zobrazen na obrázku č. 12 pod písmenem c).

3.4.4 Obrazovka průběhu aktivity

Pro sportovní aplikaci se jedná o nejvýznamnější obrazovku. Hlavní komponentou na stránce je interaktivní mapa, která zaznamenává pohyb sportovce a vykresluje trasu kudy se sportovec pohyboval. Zároveň je nutné, aby mapa byla interaktivní. To se může hodit v případě, kdy sportovec chce vědět více informací o poloze, kde se nachází, či při orientaci na mapě nebo případné ztrátě v terénu. Trasa uživatele je obarvena pro lepší orientaci v mapě tak, aby nebyla zaměnitelná s jinými prvky na mapě. Pro sportovce je taktéž důležité mít přehled o sportovních průběžných údajích při aktivitě. To je například celková vzdálenost, dynamické aktuální tempo, čas a další údaje. Tyto údaje mohou pomoci sportovci překonávat vlastní limity. Průběh aktivity jde pozastavit, nebo ukončit v jakýkoliv moment. Údaje o poloze jsou

zaznamenávány postupně při každé změně polohy. Tyto polohy jsou analyzovány a ukládány do lokální databáze tak, aby se zamezilo případné ztrátě dat. Zároveň je každá poloha analyzována tak, aby bylo možné dopočítávat sportovní údaje v jakýkoliv moment průběhu aktivity. Drátěný diagram této stránky je zobrazen na obrázku č. 12 pod písmenem d).

3.4.5 Obrazovka kluby

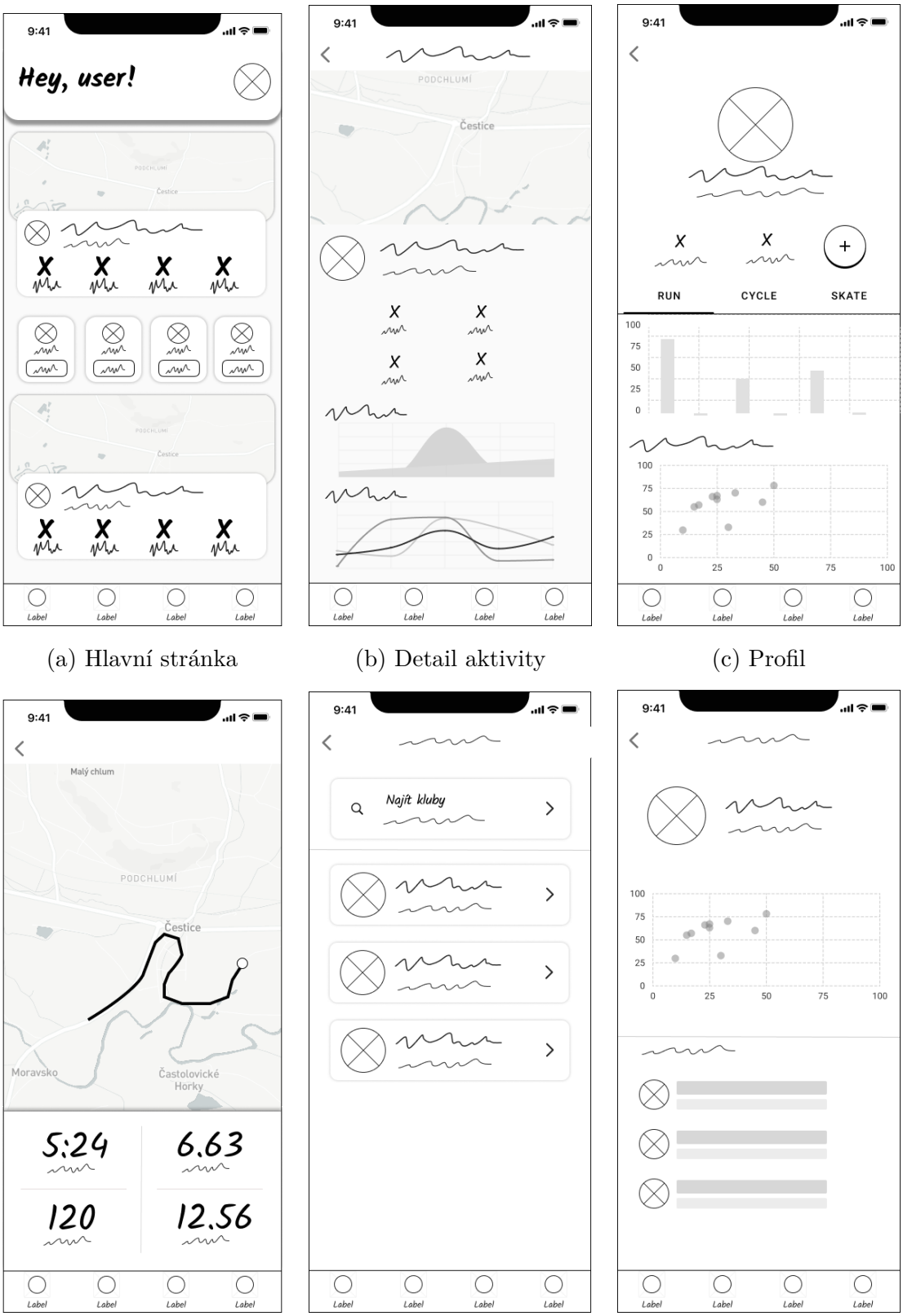
Jako například v aplikaci Strava, i v této aplikaci se lze přidávat ke klubům. Klub znamená nějakou množinu uživatelů, kteří vzájemně sdílí své aktivity. Tato obrazovka zobrazuje pole dostupných klubů, kam se uživatel může připojit. Zároveň jsou zobrazeny kluby, ve kterých se již uživatel nachází. Proklikem na klub se uživatel dostává do detailu klubu. Drátěný diagram této stránky je zobrazen na obrázku č. 12 pod písmenem e).

3.4.6 Detail klubu

V detailu klubu uživatel může najít informace o aktivitách v klubu od jiných uživatelů, kteří jsou zde také přihlášeni. Aktivity jsou řazeny podle data. Lze zde najít i uživatele, kteří se ve skupině nachází, a lehce se tak přesměrovat na jejich profil. Aktivity jsou vykresleny do grafu, který znázorňuje kumulativní vzdálenost aktivit pro každého uživatele za poslední týden. Lze tak velmi jednoduše odvodit, kdo je nejaktivnějším sportovcem v celém klubu.

3.4.7 Další obrazovky

V aplikaci se nachází další obrazovky, které zde však nebudou detailně popsány. Pro přihlášení do aplikace je jako první zobrazována přihlašovací stránka, ze které se lze dostat na stránku registrace. Drátěný diagram přihlašovací stránky je zobrazen na obrázku č. 13 pod písmenem a) a pod písmenem b) je zobrazena registrační stránka. Obrazovky na obrázku č. 13 pod písmeny b) a c), slouží k uložení a editaci aktivity. Lze zde uvést základní informace jako titulek, popisek nebo přidání, či odebrání kooperačního uživatele aktivity. Jelikož aplikace má prvky sociální sítě, ke každé aktivitě se nachází i obrazovka pro vkládání komentářů k aktivitě. Tato stránka je zobrazena na obrázku č. 13 pod písmenem d).

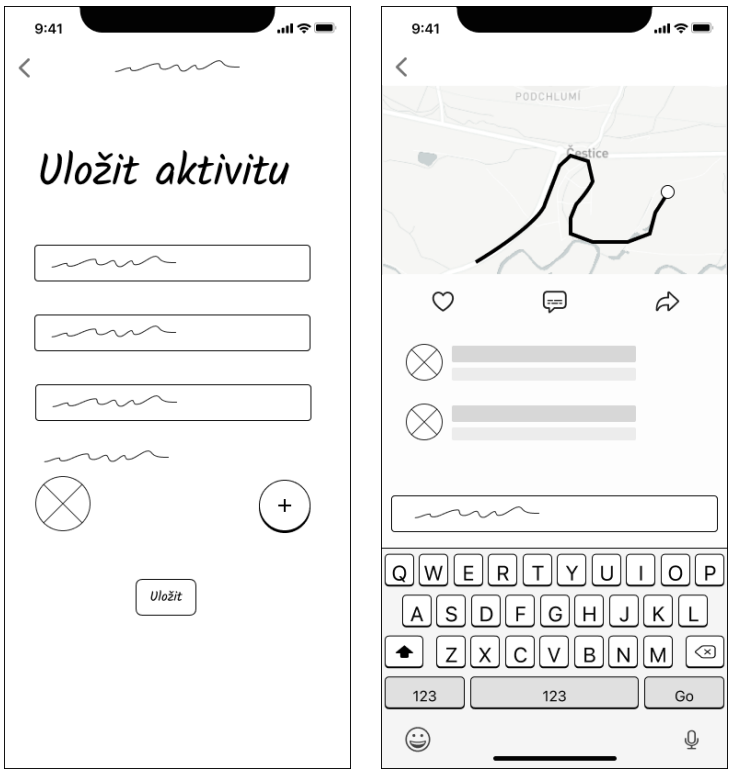


(a) Hlavní stránka (b) Detail aktivity (c) Profil

(d) Průběh aktivity (e) Přehled klubů (f) Detail klubu

Obrázek 12: Drátěné diagramy obrazovky aplikace. Zdroj: Vlastní

(a) Přihlášení uživatele (b) Registrace uživatele (c) Editace aktivity

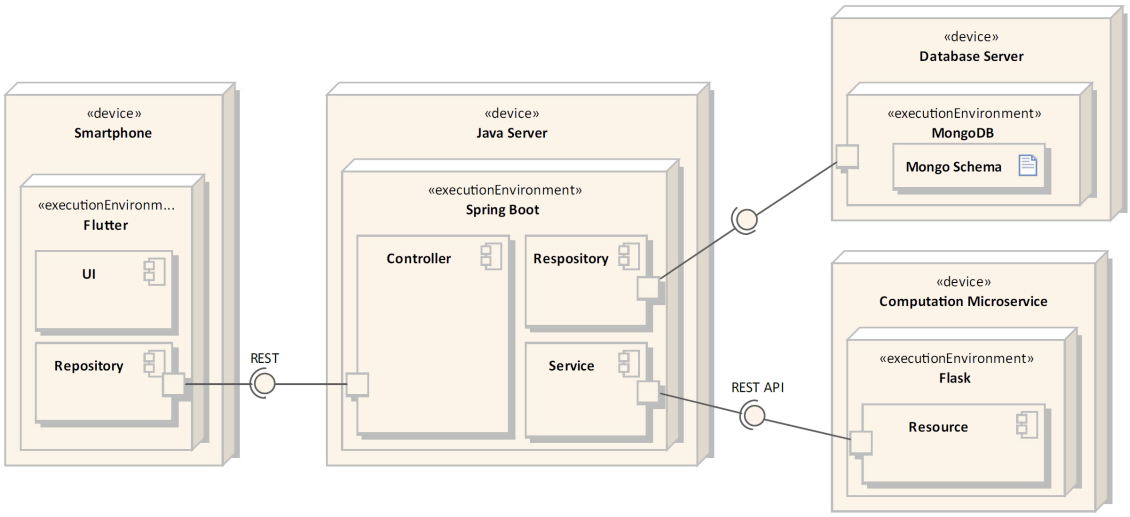


(d) Uložení aktivity (e) Komentáře aktivity

Obrázek 13: Další drátěné diagramy ostatních obrazovek aplikace. Zdroj: Vlastní

4 IMPLEMENTACE APLIKACE

Aplikace je rozdělena na několik zařízení, které spolu vzájemně komunikují. Hlavním přenosovým protokolem je Hypertext Transfer Protocol (HTTP) protokol, který využívá architekturu REST. Klientská aplikace běží na chytrém mobilním zařízení. Jako spouštěcí prostředí využívá framework Flutter. Aplikace obsahuje repositáře, které komunikují s hlavním serverem. Komunikace se serverem z klientské aplikace je detailně popsána v kapitole č. 4.1.7. Komunikace probíhá mezi klientskými repositáři a serverovými kontroléry. Samotný hlavní server využívá jako své běhové prostředí Spring Boot, který spouští vlastní Tomcat server. API serveru je vystaveno přes kontroléry. Ty jsou děleny vždy podle toho, nad jakou entitou kontroléry pracují. Server využívá připojení k MongoDB databázi, a to pomocí doménových repositářů, které využívají dedikované knihovny pro komunikaci s databázovým serverem. Server pracuje s MongoDB databází, která se řadí mezi NoSQL dokumentové databáze. Databázové schéma obsahuje vlastní kolekce databáze, které jsou definovány pomocí migrací na hlavním serveru. Pro vlastní výpočty a analýzy nad daty aktivity se využívá sekundárního serveru. Tento server je postaven na Flask frameworku, který nabízí jednoduché definování REST API pomocí programovacího jazyka Python. Hlavní server s výpočetním serverem komunikuje pomocí servisních tříd, kde výpočetní server má vystavené dostupné koncové body. Jak u hlavního serveru, tak u výpočetního, je vystavené dokumentační API, kde lze testovat jednotlivé volání. Dokumentace je vytvořena pomocí knihovny Swagger. Výpočetní server nevyužívá společnou databázi, pouze vypočte požadované statistiky a dané informace odešle v odpovědi nazpět.



Obrázek 14: Deployment diagram aplikace. Zdroj: Vlastní

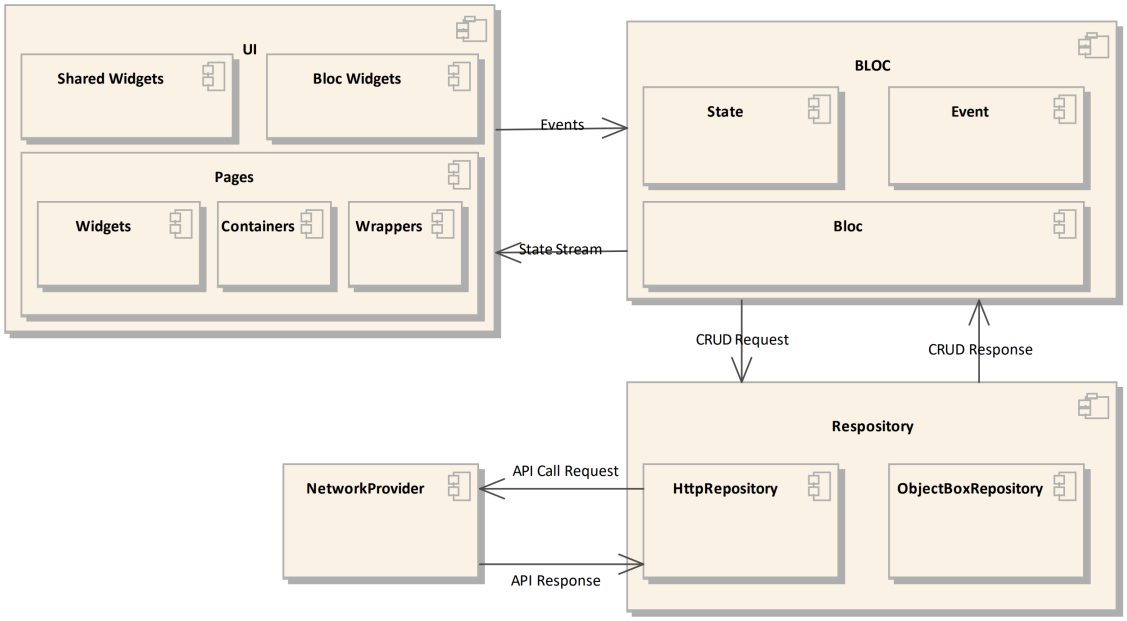
4.1 Klientská aplikace

V následující kapitole je popsána architektura a implementace klientské aplikace. Aplikace je psána ve frameworku Flutter s využitím moderních knihoven pro navigaci, práci s lokální databází, práci s mapou a práci s držením stavu v aplikaci.

4.1.1 Architektura klienta

Architektura klientské aplikace se odvíjí od použitých technologií. Hlavním prvkem a zároveň prostředníkem komunikace je Flutter Bloc. Ten zajišťuje správu nad daty, která jsou zobrazována v UI aplikace. Bloc pracuje se stavy a událostmi. Zpravidla jeden Bloc je vázán k jedné obrazovce a spravuje právě danou obrazovku aplikace. Stránka zobrazovaná v UI komunikuje s Bloc pouze prostřednictvím zasílání událostí a reaguje na změnu stavu v Blocu pomocí naslouchání stavu. Aplikační UI je děleno hiearchisticky na stránky, obsahující kontejnery a dále Widgety. Widget má přístup k Blocu pouze pokud je potomkem Bloc poskytovatele. K naslouchání stavu má Flutter Bloc připravené Widgety, se kterými lze snadno pracovat o ověřovat jaký je aktuální stav Blocu. Pokud aplikace vyžaduje data ze serveru, při inicializaci obrazovky se zavolá událost, která v příslušném rodičovském Blocu vyvolá požadavek na server. Tedy UI nikdy nekomunikuje přímo s API serveru, ale pouze přes Bloc. Bloc zajišťuje tedy kompletní aplikační logiku nad daty. Bloc k volání API serveru využívá repositáře, které jsou zpravidla dělené podle doménového objektu,

kde každý repositář spravuje právě jeden doménový objekt. Repositáře nemusí vždy komunikovat se serverem, ale pod repositáři se rozumí i komunikace například s lokálními zdroji, jako jsou lokální databáze. Serializaci a deserializaci objektu vždy řeší právě jenom repositář, kde návratovou hodnotu metody repositáře jsou již naplněné objekty daty ze severu. Samotný repositář neví, jak komunikovat se serverem, a také záleží, jaká technologie a protokol je pro komunikaci se serverem zvolena. Proto repositáře využívají externí knihovny, které komunikaci zajistí a dokáží pracovat s požadavky, které převede na daný komunikační protokol.



Obrázek 15: Architektura klientské aplikace. Zdroj: Vlastní

4.1.2 Navigace

K navigaci mezi stránkami v aplikaci byla využita knihovna AutoRouter, která velmi zjednodušuje navigování mezi stránkami ve Flutter aplikacích. „*AutoRouter je směrovací balíček Flutteru, který umožňuje předávání silně typovaných argumentů, snadné propojování a používá generovaného kódu pro zjednodušení nastavení navigačních tras, což vyžaduje minimální množství kódu pro použití navigace v aplikaci.*“ [9] Nejprve je třeba vytvořit hlavní třídu směrovače, která bude obsahovat odkaz na všechny stránky, které mohou být cílem směrování. Nastavení navigačních obrazovek a jmenných cest je zobrazeno na obrázku č. 4. Poté je nutné vygenerovat pomocné třídy, které se postarají o implementaci navigačních cest pro danou aplikaci. Pak už

stačí takto vygenerované třídy předat hlavnímu Widgetu Material routeru. Implementace předání je vyobrazena na obrázku č. 5. Pro navigaci ve Widgetu je nutné balíček importovat. Poté lze přes context navigovat na danou stránku pomocí vygenerovaných tříd s koncovkou Route, kde je možné předávat argumenty, které jsou generovány podle nastavení @AutoRouter. Příkladem je navigace na detail aktivity v aplikaci, kde je použita cesta ActivityDetailRoute s parametrem objektu aktivity. Použití je zobrazeno na obrázku č. 5.

```
part 'app_router.gr.dart';

@MaterialAutoRouter(routes: [
  AdaptiveRoute<dynamic>(
      page: AppRouterWrapper,
      path: '',
      initial: true,
      children: [
        AdaptiveRoute<dynamic>(page: HomeScreen, initial: true),
        AdaptiveRoute<dynamic>(page: LoginPage, path: 'login'),
        AdaptiveRoute<dynamic>(page: PrepareRunningNew),
        AdaptiveRoute<dynamic>(page: SplashPage),
        AdaptiveRoute<dynamic>(page: ActivityDetailMap),
        AdaptiveRoute<dynamic>(page: ActivityComments),
        AdaptiveRoute<dynamic>(page: AllClubsScreen),
        AdaptiveRoute<dynamic>(page: ProfileScreen, path: 'profile/:id'),
        AdaptiveRoute<dynamic>(page: ActivityEdit, path: 'activity/:id'),
        AdaptiveRoute<dynamic>(
            page: ActivityDetailScreen, path: 'activity/:id'),
        AdaptiveRoute<dynamic>(page: SavingActivityScreen, path: 'run/saving')
      ])
])
// extend the generated private router
class $AppRouter {}
```

Zdrojový kód 4: Navigace v aplikace pomocí AutoRouter. Zdroj: Vlastní

```
context.router.push(ActivityDetailMapRoute(activity: activity));
```

Zdrojový kód 5: Navigace pomocí AutoRouter na detail aktivity. Zdroj: Vlastní

4.1.3 Implementace IOC na klientu

IOC je v aplikaci implementována pomocí knihovny `injectable`, ta dovoluje registrování tříd pomocí jednoduchých anotací. K implementaci je potřeba vytvořit globální objekt aplikace tzv: injector. Injector se vytvoří pomocí knihovny Get_it, která

byla popsána v kapitole č. 2.2.4. Poté je zapotřebí vytvořit funkci `configureDI`, která je anotována pomocí anotace `@injetableInit`. Tato metoda, pomocí build_runner, vygeneruje novou funkci, která používá injector k registrování tříd. Funkce `configureDI` je nutné volat ještě před vytvořením hlavního Widgetu aplikace. Pomocí `injectable` je pak možné registrovat třídy pomocí anotací `@Singleton` a `@LazySingleton`. Pokud nějaká třída přijímá v konstruktoru nějaký z registrovaných objektů, je nutné třídu anotovat pomocí `@injectable`, která dovolí automatické injektování instancí tříd do objektu. V aplikaci jsou registrovány všechny repositáře a Bloc moduly tak, aby mohly být automaticky injektovány.

4.1.4 Google Maps

Jakožto sportovní aplikace je nutné zobrazovat polohu uživatele, zaznamenávat předešlé polohy uživatele a dát uživatelovi možnost orientovat se v mapě v průběhu aktivity. Na trhu existuje nespočet služeb, které poskytují využití a integraci map v mobilních aplikacích. Nejznámějšími službami jsou Goole Maps, Map Box, nebo OpenStreetMap a další. Sportovní aplikace využívá služeb Google Maps, a to především kvůli jednoduché integraci na platformy Flutter a velké podpoře. Před použitím je nutné získat API klíče, a to ve vývojářské administraci služby Google. Google Maps nabízí přímo dostupný Widget v balíkovacím systému (pub.get). Google Maps Widget nabízí velké množství funkcí pro přepínání typů zobrazení mapy, vykreslení cest, vykreslení geolokačních bodů, nebo podporu vykreslovaní BitMapových deskriptorů. V případe sportovní aplikace se jako výchozí pozice kamery nastavuje počáteční bod aktivity s úrovní přiblížení 16. Důležitou metodou je `onMapCreated`, který se volá hned po vytvoření Widgetu, a jako parametr obsahuje objekt `GoogleMapController`, který umožňuje další práci s Widgetem v průběhu aplikace. `GoogleMapController` je kontrolér, který spravuje funkce kamery (poloha, animace, zoom). Tento vzor je podobný jiným kontrolérům dostupným ve Flutteru, například `TextEditingController`. Objekt `GoogleMapController` se ukládá do lokálního stavu Widgetu ihned po provolání metody `onMapCreated`. Zároveň je nutné nastavit správně kameru, tak aby zachycovala celou aktivitu, a ne pouze počáteční bod, kde by nebyla vidět kompletní trasa aktivity. Proto je nutné ihned po vytvoření Widgetu najít vhodný zoom a výseč pro danou aktivitu. Algoritmus funguje na způsobu hledání maximálních a minimálních hodnot zeměpisné šířky a výšky. Takto nalezené hodnoty se předají do metody `CameraUpdate.newLatLngBounds`, kde je automaticky, v závislosti na hodnotách zeměpisné šířky a výšky, vytvořen ohrani-

čující box, který odpovídá přesnému zobrazení aktivity. Celá metoda je vypsána ve zdrojovém kódu č. 6.

```
void _setMapFitToTour(Set<Polyline> p) {
    double minLat = p.first.points.first.latitude;
    double minLong = p.first.points.first.longitude;
    double maxLat = p.first.points.first.latitude;
    double maxLong = p.first.points.first.longitude;
    p.forEach((poly) {
      poly.points.forEach((point) {
        if (point.latitude < minLat) minLat = point.latitude;
        if (point.latitude > maxLat) maxLat = point.latitude;
        if (point.longitude < minLong) minLong = point.longitude;
        if (point.longitude > maxLong) maxLong = point.longitude;
      });
    });
    mapController.moveCamera(CameraUpdate.newLatLngBounds(
        LatLngBounds(
            southwest: LatLng(minLat, minLong),
            northeast: LatLng(maxLat, maxLong)),
        20));
  }
```

Zdrojový kód 6: Automatické nastavení kamery mapy vzhledem k trase aktivity. Zdroj: Vlastní

4.1.4.1 Vykreslení statické mapy

Rozhraní API Maps Static vrací obrázek (Graphics Interchange Format (GIF), Portable Network Graphics (PNG), nebo Joint Photographic Experts Group (JPEG)) jako odpověď na požadavek HTTP prostřednictvím adresy Uniform Resource Locator (URL). Pro každý požadavek lze určit umístění mapy, velikost obrázku, úroveň přiblížení, typ mapy a umístění volitelných značek na místech mapy. Staticky generované mapy se hodí tam, kde je nutné rychle vykreslit například právě cestu na mapě, bez nutnosti načítat Google Maps Widget. Některé parametry adresy URL pro získání statické mapy jsou povinné a některé nepovinné. Parametr **path** definuje sadu jednoho nebo více míst spojených cestou, která se vykresluje na obrázku mapy. Pro velké a dlouhé cesty, které obsahují tisíce bodů, je nepraktické všechny body vypisovat do parametru postupně. Od toho Google představil algoritmus pro kódování takovýchto velkých dat, kde výstupem je zakódovaná cesta ve formátu Base64. Tímto řetězcem lze pak nahradit hodnotu parametru path. Ukázka generování statické mapy pomocí URL je vypsána ve zdrojovém kódu č. 7 a výsledek

generování je zobrazen na obrázku č. 16. Krom parametru path, lze pak definovat i barvu vykreslované cesty, šířku cesty, nebo také velikost generovaného obrázku.

```
class GoogleMapsUtils {
  /* ... */
  static String buildStaticGoogleMapWithPolylines(List<LatLong> locations) {
      final encodePath = GoogleMapsUtils.getEncodePath(locations);
      final url = 'http://maps.googleapis.com/maps/api/staticmap?size=600x300' +
          '&path=color:0x${FitnessAppTheme.purple.value.toRadixString(16).
              substring(2, 8)}' +
          '%7Cweight:5%7Cenc:${encodePath}&key=${dotenv.env['GOOGLE_MAPS_API']}';
      return url;
  }
}
```

Zdrojový kód 7: Generování statické mapy se zakódovanou cestou. Zdroj: Vlastní

Obrázek 16: Vygenerovaná mapa pomocí statických map Google. Zdroj: Vlastní

4.1.5 Zaznamenávání aktivity

Hlavní funkcí celé aplikace je zaznamenávání sportovní aktivity. V průběhu aktivity jsou zaznamenávány geolokační informace pro další zpracování a analýzu. Celý stav aktivity obsluhuje Flutter Bloc, který při inicializaci vytvoří objekt z balíčku `location`, který dovoluje naslouchat na změnu GPS pozice zařízení. Pokud se GPS pozice změní objekt `location` vyvolá callback s novými informace o pozici zařízení. Po každém vyvolání se předají informace do Flutter události, která je odchycena a dále zpracována. Pokud uživatel spustí aktivitu, je spuštěn časovač, který aktualizuje každou sekundu celkový čas aktivity. Zároveň spuštěním aktivity, jsou příchozí změny GPS pozice zaznamenávaní do stavu Blocu, a dále ukládány do lokální databáze. Pozice jsou do lokální databáze ukládány každých 30 sekund, do té doby jsou

neuložené pozice drženy ve stavu Blocu. Více o ukládání pozic do lokální databáze je popsáno v kapitole 4.1.6. Pro každou změnu pozice, pokud je aktivita spuštěna, je nutné vypočítávat celkovou vzdálenost průběhu aktivity tak, aby uživatel měl aktuální informace o aktivitě. K tomuto výpočtu je využívána knihovna `geolocator`, která nabízí výpočet vzdálenosti pomocí Haversine vzorce (viz. kapitola č. 2.3.6.1). Každou změnou GPS pozice se aktualizuje pozice kamery na mapě, tak se docílí sledování uživatele, který za sebou zanechává body předchozích pozicí, které jsou propojeny do úseček.

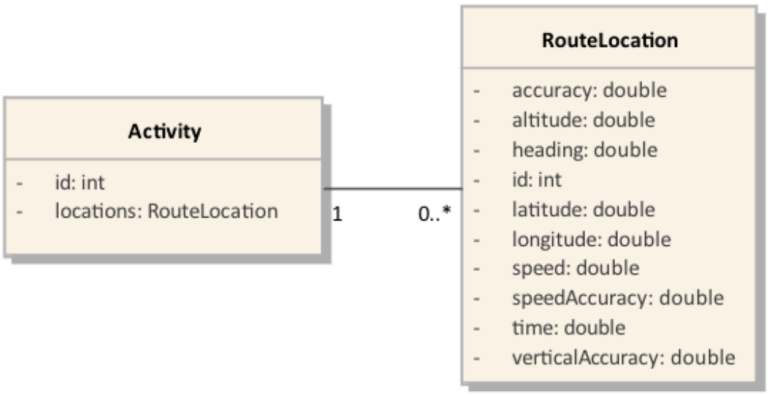
```
FutureOr<void> _onInit(RunBlocInit e, Emitter emit) async {
 emit(MapState.initial());
 location.onLocationChanged.listen((event) {
   add(RunBlocLocationChanged(location: event));
 });
}
```

Zdrojový kód 8: Změna pozice GPS. Zdroj: Vlastní

4.1.6 Lokální databáze

Aplikace obsahuje lokální databázi, a to pro účely ukládání aktivit a pozic, které k aktivitě patří. Tímto se zamezí ztrátě aktivit při náhlém ukončení aplikace. Na implementaci byl využit balíček ObjectBox. Jedná se o velmi rychlou lokální databází, která umožňuje volání Create Read Update Delete (CRUD) operací nad Dart objekty. ObjectBox podporuje Atomicity, Consistency, Isolation, Durability (ACID) vlastnosti databáze. ACID je soubor vlastností databázových transakcí, které mají zaručit platnost dat navzdory chybám, výpadkům napájení a jiným nehodám, kde by mohla být data ztracena. Zároveň ObjectBox podporuje definovat vztahy mezi entitami, vytvářet dotazy nad daty a migrace. Databáze pracuje s dvěma entitami. První entitou je entita Run. Jedná se o entitu běhu, která pouze obsahuje atribut „id“, který je automaticky generovaný, a je referenčním klíčem na objekt. Každá entita, která je ukládána do databáze, musí být anotována anotací `@Entity`. Tím zajistí ObjectBox vygenerování podpůrných tříd na práci s touto entitou. Každá aktivita může obsahovat N lokací. Lokace jsou ukládány postupně a to z Blocu aktivity. Počet lokací záleží na délce aktivity, kde jedna aktivita většinou obsahuje tisíce lokací. Lokace obsahuje informace o dané lokaci v okamžiku změny, proto je možné získat i například rychlost v daný moment. Krom toho lokace obsahuje informace

o nadmořské výšce, zeměpisné šířce a výšce, času a dalších podpůrných informací, které však k další analýze nejsou využity.



Obrázek 17: ObjectBox diagram entit. Zdroj: Vlastní

4.1.7 Komunikace se serverem

Pro komunikaci se serverem se využívá repositářů v aplikaci (viz. kapitola č. 4.1.1). Repositáře jsou jedinými zdroji komunikace se serverem, a to pomocí balíčku `http`. Tento balíček obsahuje sadu vysokoúrovňových funkcí a tříd, které usnadňují používání prostředků HTTP. Je multiplatformní a podporuje mobilní zařízení, počítače i prohlížeče. Pro komunikaci byla vytvořena třída `BaseRepository`, ze které všechny ostatní repositáře dědí a nabízí základní generické funkce pro volání http metod. Třída zajišťuje automatické přidání hlaviček do požadavku, jako jsou `content-type`, či `x-auth-token`, pro autentizaci uživatele pomocí tokenu. Nastavení požadavku probíhá přes jednoduchý objekt, který s sebou nese url požadavky, případně nepovinné parametry a hlavičky, či tělo požadavku. Tímto objektem se automaticky naplní metody knihovny `http` a požadavek je odeslán. Pokud je status odpovědi `Http.OK` je tělo odpovědi převedeno z formátu JSON do HashMap. Ve většině případů je HashMapa předána metodě `fromJson` jednotlivých doménových objektů.

```
Future<Result<T, Failure>> get<T>(Request request, T Function(Map<String, dynamic>
    json) fromJson) async {
  try {
    final response = await http.get(
        Uri.http(baseUrl!, request.url, request.query),
        headers: await getHeaders(request)
    );
    return parseResponse(response, fromJson);
  } catch(e) {
    return const Result.failure(Failure.badRequest());
  }
}
```

Zdrojový kód 9: Generická metoda get v repositáři. Zdroj: Vlastní

```
Result<T, Failure> parseResponse<T>(http.Response response, T Function(Map<String,
    dynamic> json) fromJson) {
  if (response.statusCode == 200) {
    try {
      final object = fromJson(json.decode(response.body) as Map<String, dynamic
          >);
      return Result.success(object);
    } catch(e) {
      return const Result.failure(Failure.entityParseError());
    }
  } else {
    if(response.statusCode == 401 || response.statusCode == 403) {
      return const Result.failure(Failure.unauthorized());
    }
    return const  Result.failure(Failure.badRequest());
  }
}
```

Zdrojový kód 10: Parsování odpovědi a volání zpětného volání fromJson. Zdroj: Vlastní

Příklad využití v repositáři ActivityRepository, která dědí z abstraktního repositáře BaseRepository je vyobrazen na obrázku č. 11. Názorně je ukázáno použití metody `get` z abstraktní třídy BaseRepository, kde je návratovým typem třída `Activity`. Jako první parametr metody je předán objekt Request s koncovou URL, a jako druhý parametr je předána metoda `fromJson` ze třídy Activity, která automaticky naplní objekt Activity.

```
Future<Result<Activity, Failure>> getActivityById(String id) async {
  return get<Activity>(
      Request(url: '/api/public/activity/$id'), Activity.fromJson);
}
```

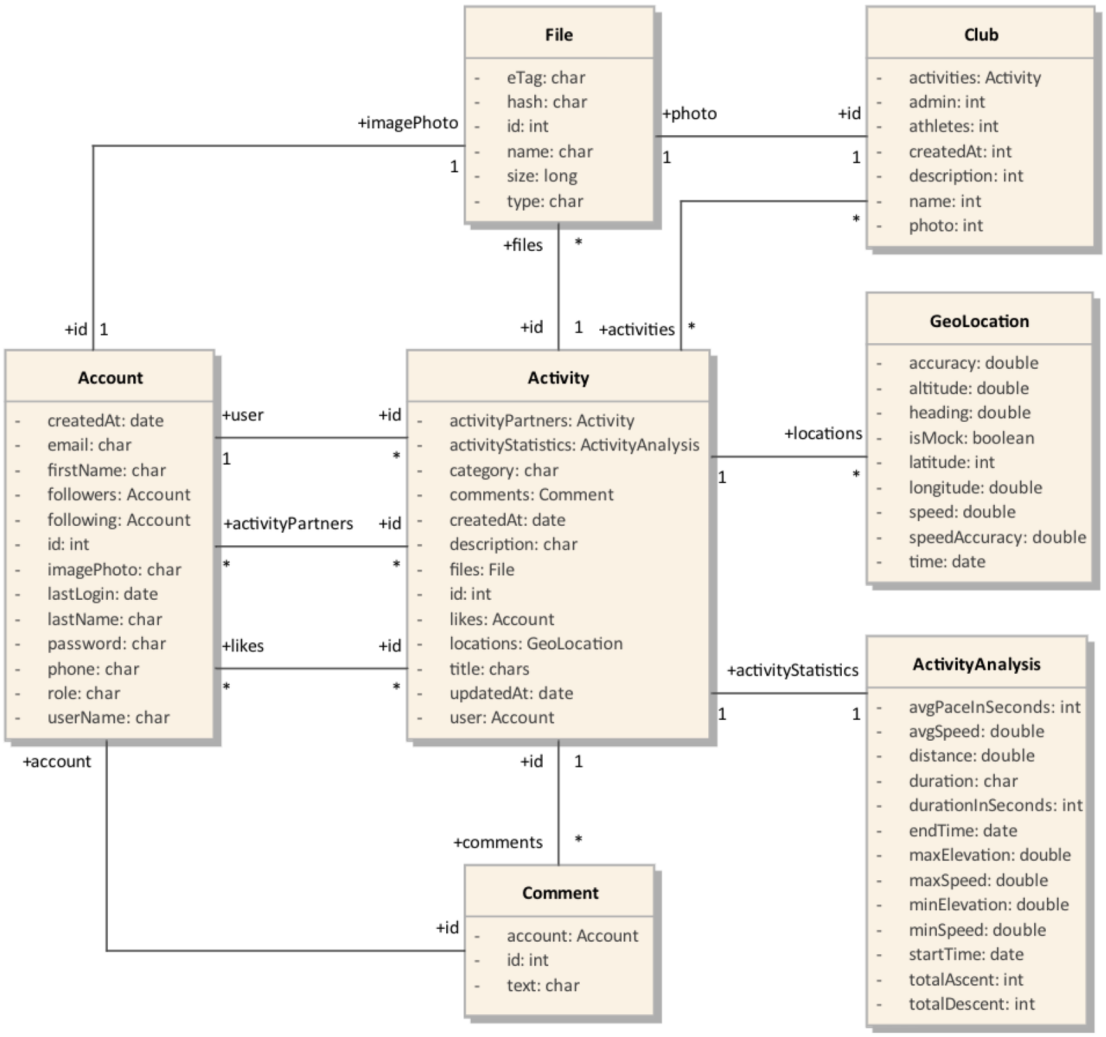
Zdrojový kód 11: Využití abstraktní třídy Base Repository. Zdroj: Vlastní

4.2 Server a výpočetní klient

V následující kapitole je popsána implementace severu, včetně serveru pro výpočet statistik aktivit.

4.2.1 Databáze

Jak již bylo zmíněno, aplikace využívá MongoDB databázi. Entity databáze jsou tedy objektové dokumenty. Na diagramu č. 18 je znázorněn modelový diagram tříd, který kopíruje strukturu databáze. Hlavní entitou databáze je entita Aktivity, která nese informace o aktivitě uživatele. Ta obsahuje vazbu na entitu Account, kde lze pomocí této vazby přiřadit uživatele k aktivitě. Jak již bylo zmíněno, v aktivitě mohou být uživatelé, kteří se na aktivitě taktéž podíleli. Ti jsou uloženi jako `activityPartners`. Uživatelé, co označili aktivitu jako oblíbenou, jsou ukládáni pod atributem `likes`. K aktivitě je možné přidávat komentáře, kterým odpovídá entita `Comment`. Samotný komentář obsahuje text komentáře a vazbu na uživatele, který komentář přidal. Entita `Club` znázorňuje klub, do kterého se uživatelé mohou přihlásit. Samotný klub obsahuje reference na aktivity, které do klubu patří a mají se zobrazit pod klubem. Klub obsahuje logo, proto je nutné mít vazbu na entitu `Files`, která obsahuje informace o uložených obrázcích. Takto je to i u entity `Account`, která využívá vazby na `Files`, kde uživatel má profilový obrázek. Objekty `GeoLocation` a `ActivityAnalysis` jsou pouze vnořené do `Activity`. Samostatně nemohou existovat a vážou se pouze na aktivitu. `GeoLocation` je objekt surových dat, který přišel z klienta při uložení aktivity. Takovýchto objektů může mít aktivita několik. `ActivityAnalysis` je objekt, který je vracen z výpočtového serveru. Kvůli náročnosti výpočtů nad surovými daty, jsou analýzy ukládány přímo do databáze, kde je lze velmi jednoduše a rychle využít, protože není nutné analýzy počítat vícekrát pro ty samé informace o aktivitě.



Obrázek 18: Databázové schéma aplikace. Zdroj: Vlastní

4.2.2 Databázové repositáře

Pro práci s databází na serveru se využívá databázových repositářů. Repositáře jsou členěny vždy podle doménového objektu, se kterým pracují, proto vždy jeden repositář pracuje pouze nad jednou doménovou entitou. Repositáře dědí z třídy `AbstractRepository`, která definuje základní CRUD metody pro práci nad entitou, to zamezí zbytečnému duplicitnímu kódu v každém z repositářů. Třída je generická, proto jí lze použít na jakýkoliv doménový objekt. Implementace samotného repositáře vyžaduje moduly Spring Data, pro vytváření dotazů do databáze. Základní moduly pro práci s databází pomocí Spring Data jsou popsaný v kapitole č. 2.4.2.6.

4.2.3 Implementace výpočtu analýz nad daty

V následující kapitole jsou popsány výpočty, které jsou potřebné pro analýzu sportovních aktivit a zpětnou vazbu uživateli.

4.2.3.1 Vykreslení trasy

Hlavním prvkem analýzy sportovní aktivity je vykreslení průběhu trasy. Vykreslení trasy probíhá pomocí všech zaznamenaných bodů na trase. Tyto body obsahují informace o zeměpisné šířce a výšce, proto je velice jednoduché tyto body postupně vykreslit a propojit. V Pythonu lze využít knihoven jako je `folium`, která dovoluje vykreslit body do mapy. Výsledek vykreslení trasy pomocí `folium` je znázorněn na obrázku č. 19. V klientské aplikací je trasa vykreslována pomocí Google Maps.

Obrázek 19: Kód pro vykreslení cesty do mapy za pomocí knihovny folium.py. Zdroj: Vlastní

4.2.3.2 Výpočet nastoupaných metrů v průběhu aktivity

Výpočet nastoupaných a sestoupaných metrů lze velice jednoduše spočítat. Prochází se všechny body trasy, a pokud je rozdíl nadmořské výšky oproti předchozímu bodu větší než nula, hodnota je přičtena k celkovému výsledku. Sestoupané metry fungují obdobně, ale zde se sčítají body, které mají oproti předchozímu bodu rozdíl výšky menší než nula.

```
def total_uphill(self):
   return self.df[self.df['elevation_diff'] >= 0]['elevation_diff'].sum()

def total_downhill(self):
   return self.df[self.df['elevation_diff'] <= 0]['elevation_diff'].abs().sum()
```

Zdrojový kód 12: Výpočet celkového převýšení do kopce a z kopce. Zdroj: Vlastní

4.2.3.3 Extrémy nadmořské výšky

Výpočtu nejvyšší dosažené nadmořské výšky a nejnižší dosažené nadmořské výšky lze dosáhnout použitím funkcí knihovny `pandas`. Nejdříve je nutné vyfiltrovat pozice, kde by mohla být nadmořská výška nedefinovaná. Body jsou pro-iterovány a jednoduše jsou nalezeny extrémy v datech pro minimální hodnotu a maximální hodnotu.

```
def elevation_extremes(self) -> tuple:
    elevations = [x.elevation for x in self.data if x.elevation is not None]
    if not elevations:
        return 0, 0
    return min(elevations), max(elevations)
```

Zdrojový kód 13: Nalezení extrémů nadmořské výšky na trase. Zdroj: Vlastní

4.2.3.4 Výpočet vzdálenosti mezi dvěma body

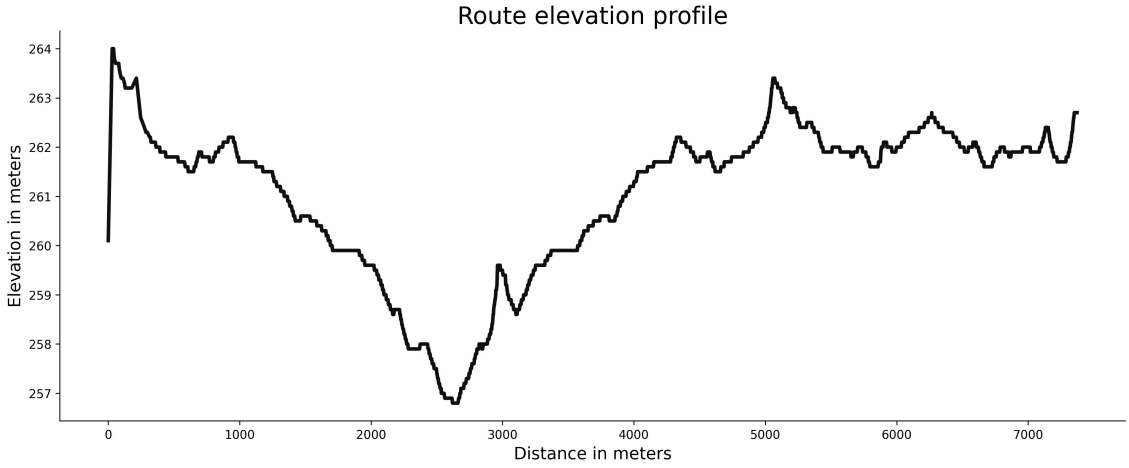
Jak již bylo zmíněno, vzdálenost mezi geografickými body je počítána pomocí Haversine vzorce (viz. kapitola č. 2.3.6.1). Tato metoda však nepočítá s elevačním profilem, tedy při výpočtu nebere v potaz nadmořskou výšku bodů. Vstupem pro výpočet jsou zeměpisná šířka a výška dvou bodů. Výsledkem je vzdálenost uvedena v metrech. Takto se vypočítá vzdálenost mezi každými dvěma body v datech z klientské aplikace.

```
_AVG_EARTH_RADIUS_KM = 6371.0088
EARTH_RADIUS = _AVG_EARTH_RADIUS_KM * 1000

@staticmethod
def haversine_distance(latitude_1: float, longitude_1: float, latitude_2: float, longitude_2: float) ->
    d_lon = radians(longitude_1 - longitude_2)
    lat1 = radians(latitude_1)
    lat2 = radians(latitude_2)
    d_lat = lat1 - lat2

    a = pow(sin(d_lat / 2), 2) + \
        pow(sin(d_lon / 2), 2) * cos(lat1) * cos(lat2)
    c = 2 * asin(sqrt(a))
    d = EARTH_RADIUS * c

    return d
```

Zdrojový kód 14: Výpočet vzdálenosti podle Haversine vzorce. Zdroj: Vlastní

4.2.3.5 Profil nadmořské výšky

Graf, který se objevuje v každé sportovní aplikaci, je graf závislosti vzdálenosti a nadmořské výšky. Ten udává, jak se měnila nadmořská výška v průběhu aktivity. Lze si tak jednoduše vykreslit profil trasy a lehce si poté analyzovat úseky s největším převýšením a náročností.



Obrázek 20: Graf nadmořské výšky v závislosti na kumulované vzdálenosti. Zdroj: Vlastní

4.2.3.6 Celková délka

Celková vzdálenost trasy aktivity je vypočítána jako poslední bod v listu dat, kde celkovou vzdáleností je kumulovaná vzdálenost všech bodů, které byly vypočteny pomocí Haversine vzorce (viz. kapitola 2.3.6.1).

```
def get_total_distance(self):
    return self.df.iloc[-1]['distance']
```

Zdrojový kód 15: Výpočet vzdálenosti. Zdroj: Vlastní

4.2.3.7 Výpočet rychlosti

Výpočet rychlosti je odvozen ze známého vzorce podílu vzdálenosti a času. Samotná klientská aplikace sama zasílá údaje o rychlosti. Tyto údaje jsou však velmi nepřesné, a proto jsou ignorovány. Výpočet rychlosti probíhá tak, že se bere aktuální bod a najde se k němu bod předchozí a následující. Vypočítají se rychlosti vzhledem ke vzdálenosti a času k předchozímu bodu a následujícímu bodu. Tyto vypočítané rychlosti se následně zprůměrují, a tím je docílena konečná vypočítaná rychlost.

```
def get_speed(self, point_no: int) -> Optional[float]:
    point = self.data[point_no]

    previous_point = None
    next_point = None

    if 0 < point_no < len(self.data):
        previous_point = self.data[point_no - 1]
    if 0 <= point_no < len(self.data) - 1:
        next_point = self.data[point_no + 1]

    speed_1 = point.speed_between(previous_point) if previous_point else None
    speed_2 = point.speed_between(next_point) if next_point else None

    if speed_1:
        speed_1 = abs(speed_1)
    if speed_2:
        speed_2 = abs(speed_2)

    if speed_1 and speed_2:
        return (speed_1 + speed_2) / 2.

    if speed_1:
        return speed_1

    return speed_2
```

Zdrojový kód 16: Výpočet rychlosti Zdroj: Vlastní

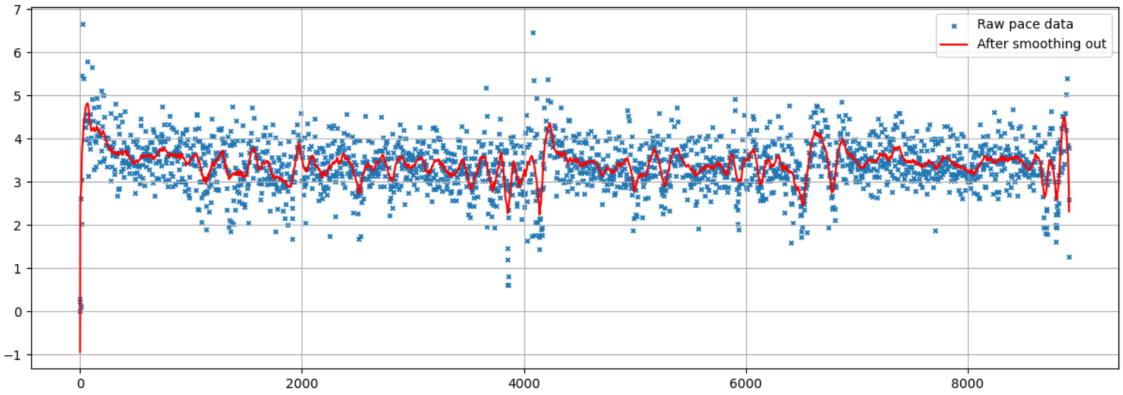
4.2.3.8 Aplikování filtru na data o rychlosti

Geolokační pozice v průběhu aktivity mohou obsahovat chybu, tak že GPS signál označí pozici s nezanedbatelnou chybou. Tyto chyby jsou ve sportovních aplikacích časté. Chyby se projeví na celkové délce trasy, ale hlavně na rychlosti a tempu v daný moment, kde rychlosti mohou nabývat až nereálných hodnot, které člověk nedokáže vyvinout. V grafu rychlosti nebo tempa poté mohou vznikat velké odchylky. K odstranění těchto odchylek se využívá Savitzky-Golay filtr. Ten slouží k aproximaci původní funkce, zachování důležitých rysů a zbaví data výkyvů. Za tímto účelem se na po sobě jdoucí množiny bodů vytvoří polynomiální funkce, která minimalizuje chybu. Na aplikaci Savitzkyho-Golayův filtru lze použít funkci `savgol_filter` z balíčku `scipy.signal`. Tato funkce přijímá jako první vstup pole obsahující signál, velikost okna, které se použije v každé iteraci pro vyhlazení signálu a řád polynomiální funkce použité pro vyhlazení původních dat. Ve zdrojovém kódu č. 17 je vypsáno využití filtru na datech s velikostí okna 33 a řádem polynomiální funkce 3. Výsledky aplikace filtru na datech o rychlosti jsou znázorněny na obrázku č. 21. Na

obrázku je jasně viditelné zanedbání extrémů a odchylek ve výpočtu.

```
self.df['pace_smooth'] = scipy.signal.savgol_filter(self.df['pace'], 33, 3)
```

Zdrojový kód 17: Výpočet rychlosti. Zdroj: Vlastní



Obrázek 21: Vyhlazení a odstranění chyb ve výpočtu rychlosti. Zdroj: Vlastní

4.2.3.9 Výpočet tempa na jednotlivé kilometry na trase

Nejdůležitějším grafem pro sportovce je graf závislosti tempa na jednotlivých kilometrech na trase. Sportovec tak může jednoduše analyzovat tempo běhu a dostat zpětnou vazbu, kdy a jak rychle daný kilometr běžel. Z dat posbíraných na trase lze určit tempa pro celý kilometr, či pro zbytek kilometru, který uživatel nedoběhl. Pokud uživatel skončil například v polovině kilometru, dochází k predikci tempa. K predikci tempa se využívá Riegelův vzorec. Riegelův vzorec predikuje čas potřebný k dosažení dané vzdálenosti. Vzorec je nutný jak k predikci posledního kilometru, který může být nedokončen, tak k dopočítání přesného času celých kilometrů. Problém je, že data nikdy nebudou obsahovat vzdálenost jako celou hodnotu kilometru, vždy bude pár metrů chybět, nebo přebývat. To je dáno chybou měření GPS. Pomocí Riegelova vzorce (viz. vzorec č. 4.1) lze dopočítat přesný čas kdy byl, nebo bude celý kilometr dosažen. Výsledkem může být tabulka, kde je vypsáno jakého tempa uživatel dosáhl a v jakém kilometru. Tabulka je znázorněna na obrázku č. 22.

$$t = t_{in} * (\frac{d}{d_{in}})^{1.06} \tag{4.1}$$

t_{in} : Naměřená hodnota trvání daného kilometru
d_{in} : Naměřená hodnota vzdálenosti daného kilometru
d : Vzdálenost, která má být predikována
t : Výsledný čas predikce

4.1: Vzorec: Riegelův vzorec. Zdroj: [14]

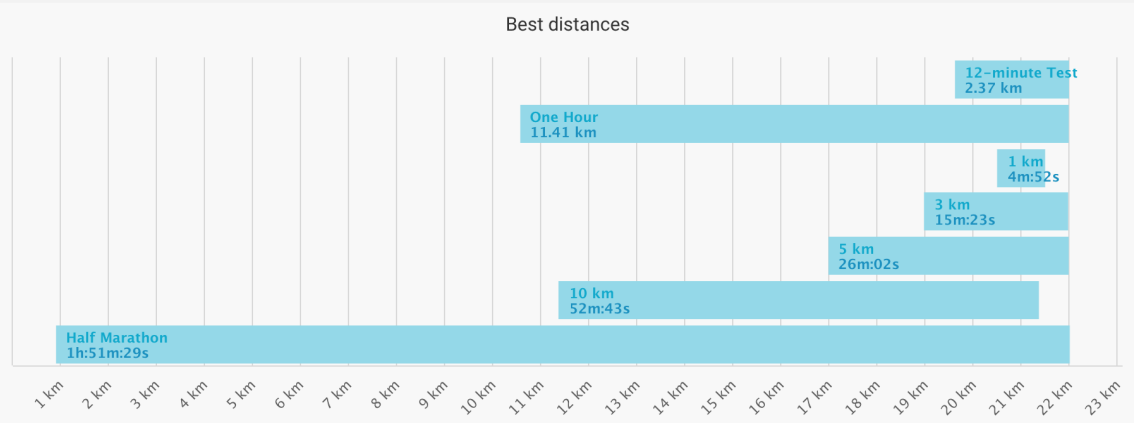
KM	Pace	Elev
1	4:04 /km	2 m
2	5:24 /km	-2 m
3	5:37 /km	0 m
4	5:37 /km	2 m
5	5:32 /km	1 m
6	5:52 /km	0 m
7	5:26 /km	0 m
0.36	5:37 /km	1 m

Obrázek 22: Výsledná tabulka tempa na jednotlivé kilometry. Zdroj: Vlastní

4.2.3.10 Nejrychlejší segmenty

Aplikace jako Endomondo nabízely funkci hledání nejrychlejších segmentů v trase. Znamená to že se hledá nejrychlejší segment kdekoliv v trase. Většinou se hledají segmenty jako jsou jednotky kilometrů, či mil. To do aplikace přidává herní složku, kdy se uživatel pokaždé snaží rekord překonat, a tím zlepšuje výkony. Aplikace jako Strava označuje nové rekordy ikonou poháru a na překonaný rekord upozorní. Za touto metodou se nachází poměrně složitý algoritmus, který takovéto segmenty hledá. Složitost algoritmu je $O(n^3)$, a tedy algoritmus je velmi pomalý. Nejdříve se definují segmenty, které se snaží algoritmus najít v trase. Pokud je celková délka trasy kratší než délka segmentu, segment se přeskočí a nebude počítán. Poté se začnou procházet všechny body trasy. Vezme se vzdálenost aktuálního bodu v iteraci a odečte se od celkové vzdálenosti. Pokud tento rozdíl je menší, nebo roven hledanému

segmentu, jsou data oříznuta na požadovaný interval. Pokud nalezený interval není prázdný, počítá se doba trvání intervalu a to odečtením prvního a posledního času bodů v intervalu. To samé se udělá pro výpočet vzdálenosti intervalu. Z času a vzdálenosti lze jednoduše spočítat tempo. Takto spočítané údaje se uloží do globálního DataFrame pro daný segment. Pokud jsou pro-iterovány všechny body trasy, nalezne se nejnižší tempo pro daný segment. Bod, který byl vybrán jako nejrychlejší interval pro daný segment, obsahuje indexy začínajícího bodu a posledního bodu, proto lze pak jednoduše lokalizovat interval na trase. Takto se postupuje pro všechny dotazované segmenty. Výsledek může být vizualizace grafem, jako je tomu na obrázku č. 23.



Obrázek 23: Vizualizace nejrychlejších segmentů na trase. Zdroj: [76]

4.2.3.11 Sklon terénu na trase

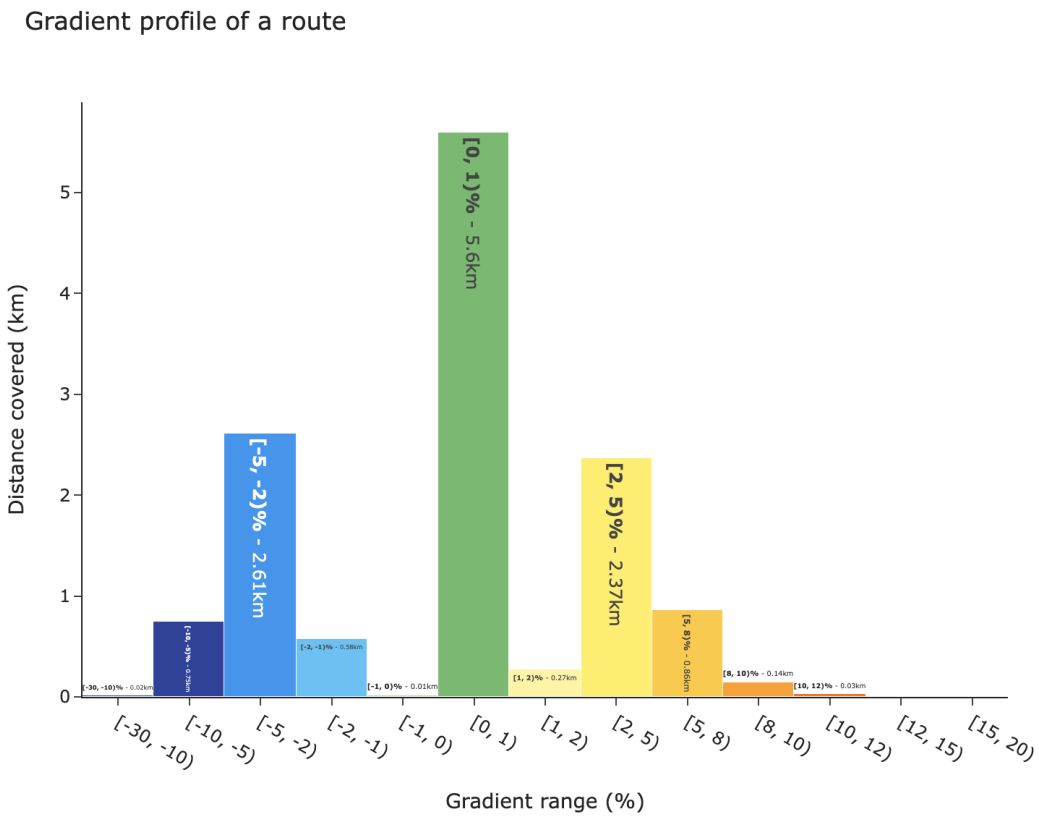
Zajímavou analýzou je sledování změny náklonu povrchu na trase. Náklon povrchu odráží stoupání a klesání na trase. Lze tak docílit statistik, například jak dlouho uživatel pod jakým úhlem stoupal, či klesal. Náklon povrchu se vypočítá jako podíl změny nadmořské mezi dvěma body ku vzdálenosti mezi body. Pokud se výsledek vynásobí 100, výsledkem je náklon povrchu vyjádřený procenty. Vzorec je vypsán ve vzorci číslo 4.2.

$$grade = \frac{elevation_{diff}}{distance} * 100 \tag{4.2}$$

4.2: Vzorec: Výpočet náklonu povrchu.

Z výsledných sklonů lze poté vypočítat intervaly náklonu na trase a k nim dopočítat vzdálenost ke každému intervalu. Tím lze uživateli dát zpětnou vazbu, jak

daleko běžel pod sklonem například +10 % stupňů. Je zapotřebí definovat příslušné intervaly sklonu na trase. Rozmezí se pohybuje od -30 % do +30 %. Nepředpokládá se větší převýšení než v tomto rozmezí, pokud k tomu dojde, nejspíše se bude jednat o chybu měření a hodnoty budou ignorovány. Poté již jednoduchým seskupením podle náklonu lze dopočítat vzdálenosti ke každému intervalu. Výsledkem je graf, který je rozdělen podle intervalů na ose X a na ose Y udává celkovou zaznamenanou vzdálenost pod jednotlivým intervalem. Příklad výsledného grafu je znázorněn na obrázku č. 24.



Obrázek 24: Intervaly náklonu na trase s celkovými vzdálenostmi. Zdroj: [22]

4.2.3.12 Predikce

Následující kapitola se zabývá problematikou, zda je možné predikovat čas aktivity v závislosti na předpokládané celkové vzdálenosti aktivity. Následující výpočty budou prováděny a znázorněny na vlastních datech exportovaných z aplikace Strava. Nutno podotknout, že se jedná o aktivitu typu běh. K predikci lze využít metodu lineární regrese. Jedná se o model jedné závislé a nezávislé proměnné. V případě predikce času na vzdálenosti běhu je nezávisle proměnnou vzdálenost běhu a zá-

visle proměnnou předpokládaný čas. Lineární regrese představuje aproximaci daných hodnot přímkou metodou nejmenších čtverců. Pokud tuto přímku vyjádříme rovnicí $y = \beta_0 + \beta_1 x + \epsilon$, jedná se o nalezení optimálních hodnot koeficientů β_0 β_1. Absolutní člen β_0 vyjadřuje geometrický průsečík přímky se svislou osou. Koeficient β_1 je regresní koeficient a geometricky se jedná o směrnici přímky. Výsledkem je vektor β, obsahující koeficienty β_1, β_0. V datech exportovaných z aplikace Strava se nacházelo 100 různých hodnot aktivit, všechny byly stejného typu aktivity, a to běhu. Data byla použita na lineární regresním model, kde výsledek byl koeficient $\beta_1 = 309.76$ a koeficient $\beta_0 = 98.3$. Regresní přímku znázorňuje červená přímka proložená daty na obrázku č. 25.

$$\beta = (A^T A)^{-1} A^T Y \tag{4.3}$$

kde:

Y : sloupcový vektor n pozorování hodnot závisle proměnné
A : matice n x (k + 1) pozorování hodnot vysvětlujících proměnných
β : sloupcový vektor k + 1 neznámých parametrů

4.3: Vzorec: Výpočet lineární regrese. Zdroj: [45]

Zdrojový kód č. 18 ukazuje výpočet v programovacím jazyce Python. K výpočtu je využit vzorec 4.3. Data jsou naplněna z pandas.DataFrame, kde závisle proměnná je vzdálenost (Distance) a nezávisle proměnná je čas aktivity (ElapsedTime). K maticovým funkcím je využita knihovna `numpy`.

```
# generate x and y
x = numpy.array(df['Distance'].tolist())
y = numpy.array(df['ElapsedTime'].astype(int).tolist())

# assemble matrix A
A = np.vstack([x, np.ones(len(x))]).T

# turn y into a column vector
y = y[:, np.newaxis]

# Direct least square regression
alpha = np.dot((np.dot(np.linalg.inv(np.dot(A.T,A)),A.T)),y)
```

Zdrojový kód 18: Lineární regrese v programovacím jazyce Python. Zdroj: Vlastní

K posouzení míry kvality regresního modelu na datech lze použít index determinace. Ten vysvětluje, jaký podíl variability závislé proměnné model vysvětluje. Index determinace nabývá maximální hodnoty 1 (100%), což značí dokonalou predikci hodnot. Vzorec pro výpočet indexu determinace udává podíl mezi součtem čtvercových odchylek hodnot vysvětlované proměnné od teoretických hodnot Error Sum of Squares (SSe) a sumou čtvercových odchylek závisle proměnné y od průměru Total Sums of Squared (SSt). V případě výpočtu na datech z aplikace Strava, index determinace vychází 0.736. To znamená, že závislost času na vzdálenosti je popsána z 73.6% modelem a 26.4% chybou. Tento lineární model je vhodný pro predikci času v závislosti na vzdálenosti.

$$I_2 = 1 - \frac{SSe}{SSt} = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y}_i)^2} \tag{4.4}$$

kde:

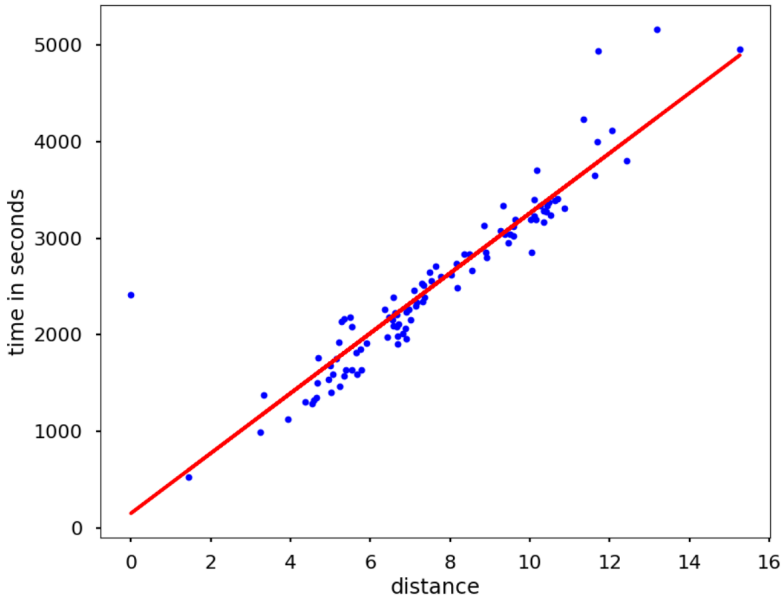
I_2 : výsledný index determinace
y_i : hodnota vysvětlované proměnné
$\bar{y}_i$: aritmetický průmět vysvětlované proměnné
$\hat{y}_i$: teoretické vysvětlované proměnné

Vzorec 10.2: Výpočet indexu determinace. Zdroj: [61]

```
predicted_values = compute_predicated_values(x, alpha[0], alpha[1])
mean = numpy.mean(predicted_values)
sse = numpy.sum((y - predicted_values)**2)
ssr = np.sum((predicted_values - mean) ** 2)
sst = ssr + sse
i2 = 1-(sse/sst)
```

Zdrojový kód 19: Výpočet indexu determinace. Zdroj: Vlastní

S regresní přímkou pro daného uživatele je nyní jednoduché predikovat jakoukoliv vzdálenost aktivity uživatele. Pro ukázku lze vypsat uživateli, jaký bude předpokládaný čas běhu, pokud by další aktivitou uživatele byl běh na 10 km. Jednoduše lze dosadit do rovnice: $309.76 * 10 + 98.3$, kde výsledkem je predikovaný čas 53 minut 15 sekund.



Obrázek 25: Vizualizace nejrychlejších segmentů na trase. Zdroj: Vlastní

5 SHRNUTÍ VÝSLEDKŮ

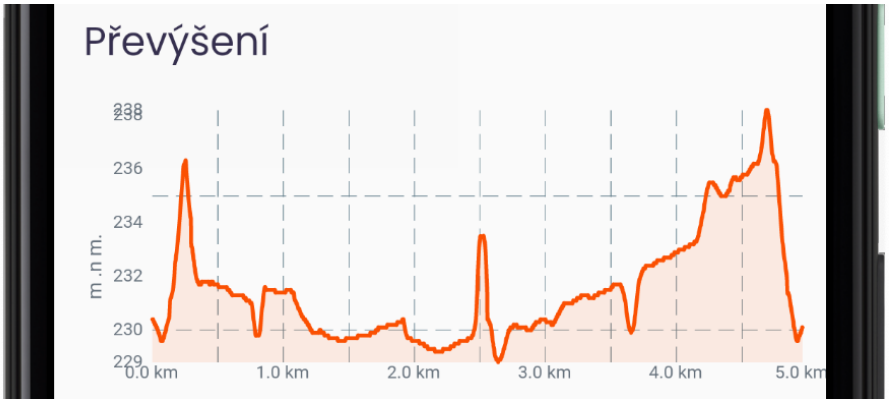
Klientská aplikace byla vyvíjena v IDE Android studio s použitím technologie Flutter. Hlavní server byl vyvíjen pomocí IntelliJ IDEA v programovacím jazyce Java za pomocí webového frameworku Java Spring Boot. Pro ukládaní velko-objemových dat byla využita MongoDB databáze. Výpočetní server byl vyvíjen pomocí PyCharm IDE v programovacím jazyce Python. Programovací jazyk Python pro tuto problematiku byl vybrán, jelikož je na práci s velkými objemy dat připraven a za pomocí externích knihoven, je velmi silným nástrojem v oblasti datové analýzy. Aplikace byla vyvíjena postupně tak, aby byly splněny všechny funkční i nefunkční požadavky na aplikaci. Zároveň byl navýšen počet analýz aktivit oproti konkurenčním aplikacím, které tyto analýzy nabízejí v prémiových sekcích. Byla dodržena navrhovaná flow mezi obrazovkami a obrazovky odpovídají zadání aplikace.

5.1 Analýzy v detailu aktivity

Všechny analýzy vypočítané z dat o aktivitě jsou zobrazeny v detailu aktivity, jako je tomu u jiných sportovních aplikací. V následující kapitole jsou zobrazeny a popsány výsledné grafy, které podrobně popisují aktivitu. Metody výpočtu těchto grafů byly popsány v kapitole 4.2.3.

5.1.1 Profil nadmořské výšky na trase

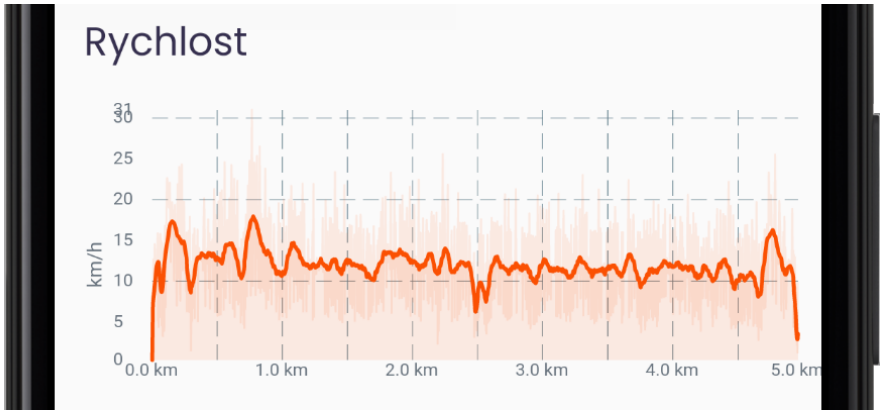
Stejně jako byl tento graf vyobrazen a popsán v kapitole 4.2.3.5, i zde je vykreslen uživateli a to interaktivně, kde je možné si detailně projíždět profil trasy. Výsledný graf v detailu aktivity je zobrazen na obrázku č. 26.



Obrázek 26: Profil nadmořské výšky aktivity v klientské aplikaci. Zdroj: Vlastní

5.1.2 Rychlost

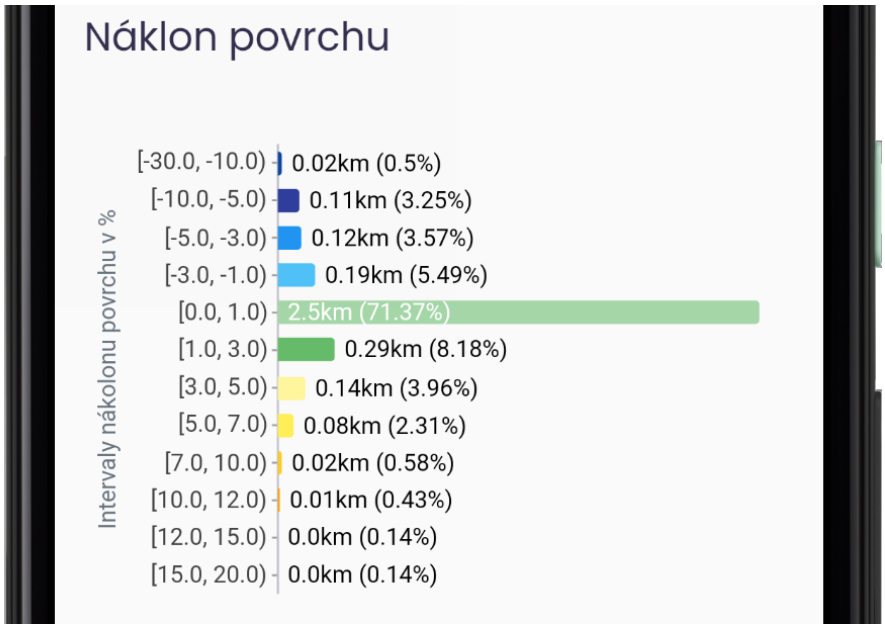
Jak bylo popsáno v kapitole 4.2.3.8, na vypočítaná data o rychlosti se aplikuje filtr, který aproximuje původní funkce. Tato aproximovaná funkce je zobrazena výraznější barvou v popředí a na pozadí jsou vidět původní vypočítané rychlosti. I tento graf je interaktivní a lze zjisti jednotlivé hodnoty v bodech. Výsledný graf v detailu aktivity je zobrazen na obrázku č. 27.



Obrázek 27: Graf rychlosti na trase. Zdroj: Vlastní

5.1.3 Gradient

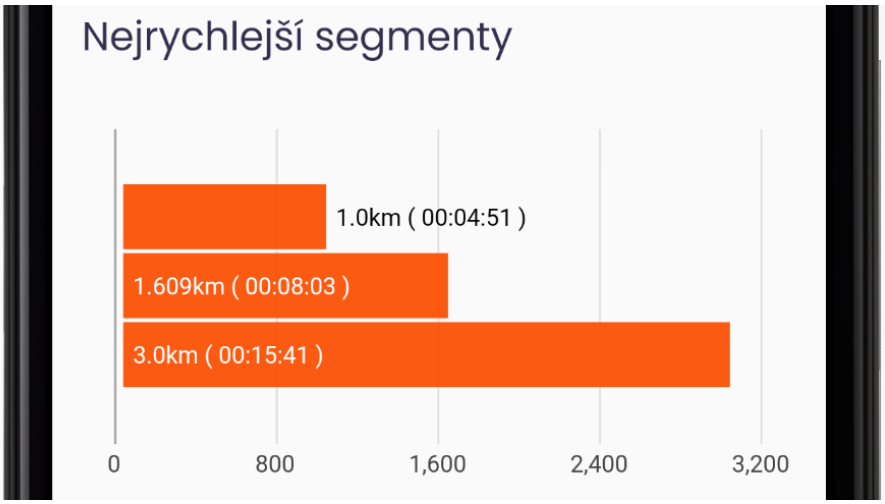
Gradientní profil byl popsán v kapitole 4.2.3.11. Tento graf může dát uživateli ještě jasnější představu o profilu trasy. Podle grafu na obrázku 28 je evidentní, že zaznamenaná aktivita byla z 71.37 % bez významného, či velmi mírného stoupání na trase.



Obrázek 28: Graf stoupání a klesání na trase. Zdroj: Vlastní

5.1.4 Nejrychlejší segmenty

Dalším implementovaným grafem, který je předložen uživateli v detailu aktivity, je graf nejrychlejších segmentů na trase. Ten dává uživateli informaci, v jakém úseku daný segment zaběhl nejrychleji. Jak je vidět z obrázku č. 29, nejrychlejší segment na 6km trase pro úsek 3km je až od 1.5 km. Tento údaj je zajímavým atributem, jelikož uživatel může sledovat vlastní průběžný výkon v rychlosti.



Obrázek 29: Nejrychlejší segmenty na trase. Zdroj: Vlastní

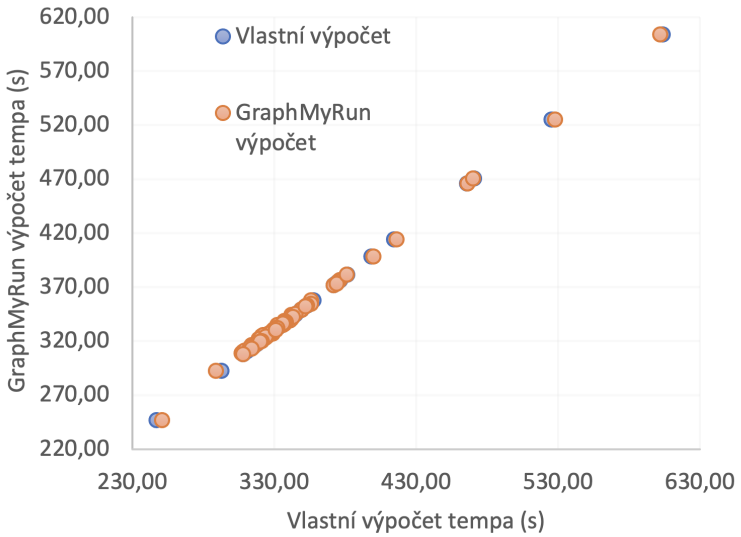
5.2 Testování aplikace

Aplikace byla testována již při vývoji. Postupně byly ověřovány všechny funkcionality potřebné pro běh aplikace. Obzvláště bylo testováno zpracování dat z klientské aplikace a následná analýza. Testování výpočtu analýz je více popsáno v následující kapitole č. 5.2.1.

5.2.1 Testování výpočtu analýz

Je důležité ověřit, zda vypočítávané statistiky z dat jsou validní a dobře počítány. Jednoduše to lze ověřit pomocí softwaru třetích stran, kde lze jednoduše nahrát soubor GPX. Výsledkem budou obdobné statistiky, jako jsou výstupy z této aplikace. Jedním z veřejně dostupných softwarů pro analýzu GPX souboru je webová služba GraphMyRun.com[1]. Ta nabízí možnost importu GPX souboru, který následně analyzuje a vykreslí do obdobných grafů, jako byly ukázány v předešlých kapitolách. Lze tedy velmi jednoduše exportovat všechny aktivity z aplikace a ověřit si výpočty v této službě. Celkem bylo exportováno 10 aktivit, které byly typu běh. Postupně byly nahrávány do služby GraphMyRun a zapisovány do tabulky k porovnání. Hlavní porovnávanou analýzou byl rozdíl v časech na jednotlivé kilometry. Takto získaná data lze zakreslit do grafu, kde budou porovnávány časy jednotlivých kilometrů ze všech deseti běhů. Celkem se jednalo o výpočet 81 jednotlivých kilometrů. To je dostatečný počet vzorků, aby se dalo určit, zda jsou výpočty správné. Vykreslený graf je zobrazen na obrázku č. 30.

[1]Veřejná služba pro výpočet analýz z formátů GPX: http://www.graphmyrun.com/graph.html



Obrázek 30: Graf rozdílu času mezi výpočtem GraphMyRun a vlastním výpočtem. Zdroj: Vlastní

K ověření přesnosti dat je využit Studentův párový T-test. Párový t-test se využívá k porovnání středních hodnot dvou populací. Porovnávají se vzorky z jednoho pozorování, které jsou spárovány se vzorky druhého pozorování. Pod označením x jsou časy z aplikace GraphMyRun a y jsou časy z testované aplikace. K testu byla zvolena 5% hladina významnosti. K ověření přesnosti dat je nutné stanovit statistické hypotézy:

Hypotéza H_0 (Testovaná hypotéza)**:** $\mu_x - \mu_y = 0$

Hypotéza H_1: $\mu_x - \mu_y \neq 0$

Hypotéza H_0 říká, že výpočty jsou stejné a na hladině 5% se neliší. Naopak hypotéza H_1 popírá hypotézu H_0 a tedy říká, že se výsledky aplikací liší. Pro implementaci výpočtu bylo využito statistického softwaru SPSS. Na obrázku č. 31 jsou vyobrazeny základní statistiky obou souborů. Jak je patrné, velikost souboru byla 81 vzorků, to zaručuje normalitu dat pomocí centrální limitní věty (CLV). Vlastní výpočet je vypsán na obrázku č. 32, kde průměr rozdílu párových hodnot vyšel $m_z = 0.2455$, směrodatná odchylka průměrů $SSE(m_z) = 0.1504$ a nakonec testové kritérium $t = 1.632$. P-hodnota pro oboustranný t-test vyšla 0.107, což je vyšší než testovaná hladina významnosti 0.05. Je tedy zřejmé, že se hypotéza H_0 nezamítá, tudíž výpočetní výsledky aplikací se na 5% hladině neliší a lze je považovat za stejné.

Paired Samples Statistics

		Mean	N	Std. Deviation	Std. Error Mean
Pair 1	GraphMyRun	342,825802469135800	81	47,176304731141170	5,241811636793464
	VlastniData	342,58	81	47,279	5,253

Obrázek 31: Základní statistiky populací. Zdroj: Vlastní

Paired Samples Test

		Paired Differences			
					95% Confidence Interval of the ...
		Mean	Std. Deviation	Std. Error Mean	Lower
Pair 1	GraphMyRun - VlastniData	,24555555556	1,3537820818	,15042023131	-,0537902446

Paired Samples Test

		Paired ...			Significance
		95% Confidence Interval of the ...			
		Upper	t	df	One-Sided p
Pair 1	GraphMyRun - VlastniData	,54490135570	1,632	80	,053

Paired Samples Test

		Significance
		Two-Sided p
Pair 1	GraphMyRun - VlastniData	,107

Obrázek 32: Výsledky vypočtu párového t-testu, včetně p-hodnoty. Zdroj: Vlastní

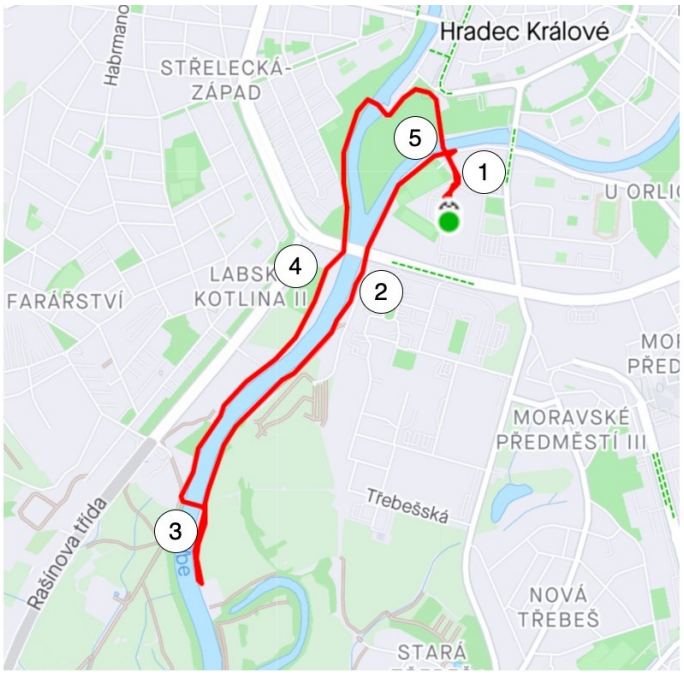
5.2.2 Testování při běhu

Aplikace byla testována na univerzitní akci Akademik run 2022 na zařízení Realme 8 Pro, s verzí Android 11. Jednalo se o běh na 5 km. Aplikace byla testována společně se zapnutou aplikací Strava. Z aplikace Strava byla exportována data a analyzována pomocí nástroje GraphMyRun. Na obrázku č. 33 je viditelná závislost rychlosti běhu na převýšení a nadmořské výšce trasy. Přerušovanými čarami a číslem na obrázku jsou vyznačeny důležité objekty na trase. U všech důležitých objektů na trase jsou znázorněny výrazné změny v převýšení a rychlosti. Vrchní graf s modrou linkou značí nadmořskou výšku na trati. Nižší graf s červenou linkou značí rychlost běhu v metrech za sekundu. Pod číslem 1 se nachází stoupání od kampusu univerzity k mostu přes Labe. Pod číslem 2 a 4 se nachází trasa pod silničním mostem přes

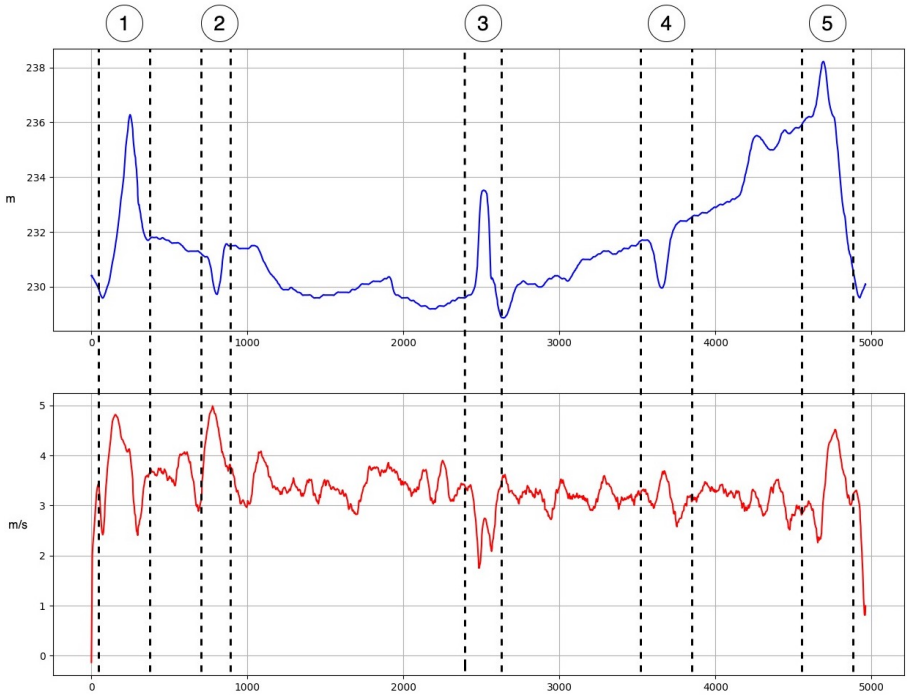
řeku, kde jsou rychlosti větší, jelikož klesá i profil trasy. Nejdůležitějším objektem na trase je most přes Labe pod číslem 3. Na mostě se nachází schody, proto rychlost šla výrazně dolů. Stejně tak šla rychlost dolů i při běhu po schodech dolů na druhé straně mostu. Most je velice dobře vidět na profilu trasy pod číslem 3. A nakonec pod číslem 5 se nachází železný pěší most přes řeku Labe, kde rychlost byla vyšší jelikož se blížil konec trasy. Z aplikace Strava byly vyexportovány data do formátu GPX, a následně analyzovány aplikací GraphMyRun. Porovnání analýz a výsledků je zobrazeno v tabulce č. 1. Výsledky výpočtu aplikací byly srovnatelné a nebyla nalezena výraznější odchylka ve výpočtu.

Tabulka 1: Porovnání výsledků běhu na závodu Akademik run 2022 Zdroj: Vlastní

Analýza	GraphMyRun	Vlastní aplikace
Celková vzdálenost	4.96km	4.961km
Čas	00:26:56	00:26:58
Celkové tempo	5:26 min/km	5:26 min/km
Tempo 1km	5:03 min/km	5:04 min/km
Tempo 2km	5:10 min/km	5:10 min/km
Tempo 3km	5:40 min/km	5:38 min/km
Tempo 4km	5:29 min/km	5:28 min/km
Tempo 5km	5:39 min/km	5:38 min/km



(a) Mapa závodu Academic Run 2022 s důležitými body

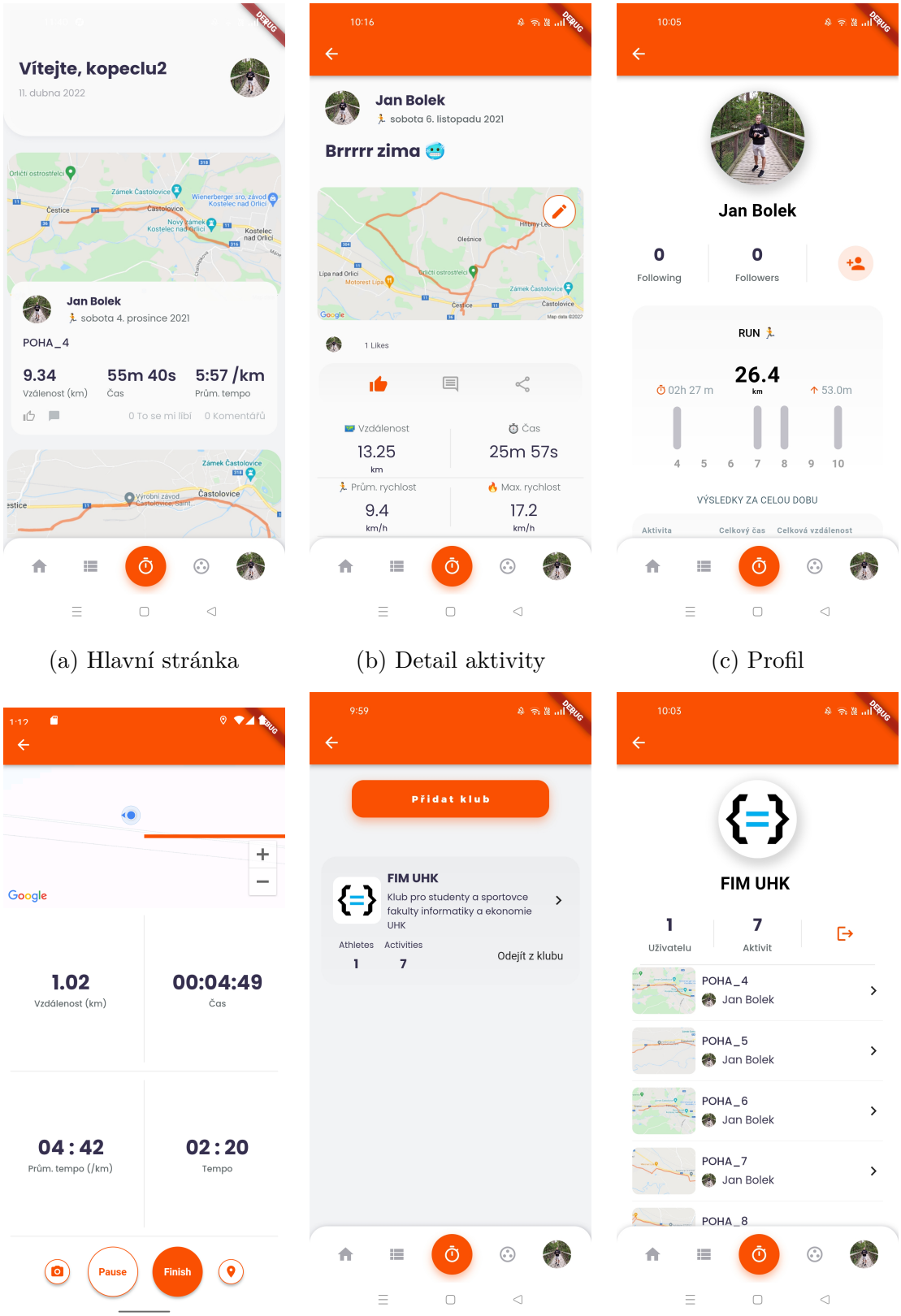


(b) Graf převýšení a rychlost s vyznačenými důležitými body na trase

Obrázek 33: Profil trasy v závislosti na rychlosti běhu na závodě Akademik run. Zdroj: Vlastní

5.3 Výsledný vzhled aplikace

Podle návrhu drátěných modelů (viz. obrázek č. 12) byly implementovány jednotlivé obrazovky aplikace. Cílem bylo vytvořit moderní design aplikace, který je lehký na používání a pochopení funkcionalit aplikace. Design aplikace byl inspirován aplikací Strava. Na obrázku č. 34 jsou vyobrazeny implementace jednotlivých stránek postupně tak, jak byly představeny v sekci návrhu aplikace. Pod písmenem a) je zobrazena hlavní obrazovka s náhledy aktivit, kde se lze přesměrovat na detail aktivity. Detail aktivity je zobrazen pod písmenem b), kde jsou uvedeny jednotlivé statistiky, které byly představeny v předešlé kapitole 5.1. Profil uživatele je vyobrazen pod písmenem c), kde jsou vykresleny různé grafy, které pracují s daty jednotlivého uživatele. Průběh aktivity je vyobrazen pod písmenem d). Výpis a detail klubu jsou vyobrazeny pod písmeny e) a f). V této kapitole jsou zobrazeny pouze nejdůležitější obrazovky.



(a) Hlavní stránka (b) Detail aktivity (c) Profil

(d) Průběh aktivity (e) Přehled klubů (f) Detail klubu

Obrázek 34: Výsledný vzhled obrazovek. Zdroj: Vlastní

6 ZÁVĚR

Cílem práce bylo vytvořit multiplatformní sportovní aplikaci s využitím moderního vývojového frameworku Flutter. Výsledkem práce je podrobná analýza a představení problematiky sportovních aplikací, včetně návrhu architektury vytvořené aplikace. Tímto byly stanovené cíle práce splněny.

Teoretická část představila metody multiplatformního vývoje a problematiku získávání a zpracování geolokačních dat.

Praktická část práce se věnovala samotnému návrhu aplikace, včetně implementace klientské aplikace a serverové části, včetně vysvětlení výpočtu analýz z dostupných dat. Jedná se o demonstrační aplikaci, jak využít moderní multiplatformní frameworky na vývoj komplexní sportovní aplikace, která je srovnatelná například s aplikací Runastic. Design aplikace byl inspirován aplikací Strava. Podařilo se implementovat kompletní analýzu aktivit, kterou většina aplikací nabízí buď v prémiové části, nebo vůbec tyto analýzy nenabízí. Nejdůležitější analýzu tempa na jednotlivé kilometry se podařilo úspěšně otestovat pomocí párového t-testu a bylo zjištěno, že výpočty jsou srovnatelné s výsledky výpočtů komerčního softwaru GraphMyRun. Aplikace byla úspěšně otestována na závodě Akademik Run 2022 pořádaným Univerzitou Hradec Králové. Dále pak byly výstupy z aktivity úspěšně analyzovány.

V rámci případného rozvoje aplikace by mohly být doplněny funkcionality jako audio trenér, který dává informace o dosaženém času předchozího kilometru, a velmi tak zlepšuje výkon sportovce. Zároveň pokud by mohla aplikace být rozšířena o aplikaci na chytré hodinky, které nyní Flutter nepodporuje, a bylo by nutné tuto aplikaci implementovat pomocí nativního vývoje. Jelikož se jednalo o demonstrační aplikaci, nebylo takovéto prvky nutné implementovat. Další možností rozšíření je vývoj webové aplikace, která bude zajišťovat přehled a detail aktivit pro webové rozhraní. Tuto funkčnost nabízí většina sportovních aplikací. Vhodná by byla také samostatná aplikace na analýzu dat z formátu GPX, kde je připraven kompletní výstup ze serveru. Takto by šlo velice jednoduše nahradit například software GraphMyRun, který je po designové stránce již velice zastaralý.

LITERATURA

[1] Strava Launches Apple Watch App With GPS Support For Series 2 Owners.

[2] Cross-platform mobile frameworks used by global developers 2021, July 2021.

[3] Mobile OS market share 2021, February 2021.

[4] Achal Agrawal, Amit Agrawal, Rahul Arya, Hardik Jain, and Jyoti Manoorkar. Comparison of Flutter with Other Development Platforms. *IJCRT2102147*, 2021(203111):1160–1164, February 2021.

[5] Cyntara Adwinda and Satrio Suryodiningrat. DEVELOPING AN ANDROID-BASED RUNNING APPLICATION. June 2020.

[6] Mohamed Ali and Ali Mesbah. Mining and characterizing hybrid apps. pages 50–56, November 2016.

[7] Abeer AlJarrah and Mohamed Shehab. The Demon is in the Configuration: Revisiting Hybrid Mobile Apps Configuration Model. In *Proceedings of the 12th International Conference on Availability, Reliability and Security (ares 2017)*, page A57, New York, 2017. Assoc Computing Machinery. WOS:000426964900057.

[8] Dan Anderson. More on Defining Waypoints, Tracks, and Routes, 2003.

[9] António Valente. Flutter Navigator 2.0 Made Easy with Auto Router, January 2022.

[10] Chaimae Asaad and Karim Baïna. NoSQL Databases – Seek for a Design Methodology: 8th International Conference, MEDI 2018, Marrakesh, Morocco, October 24–26, 2018, Proceedings. pages 25–40. January 2018.

[11] Paolo Atzeni, Francesca Bugiotti, Luca Cabibbo, and Riccardo Torlone. Data modeling in the NoSQL world. *Computer Standards & Interfaces*, 67:103149, January 2020. Place: Amsterdam Publisher: Elsevier WOS:000497888700002.

[12] Andrea Babic, Danijela Jaksic, and Patrizia Poscic. Querying Data in Nosql Databases. *Zbornik Veleucilista U Rijeci-Journal of the Polytechnics of Rijeka*, 7(1):257–270, May 2019. Place: Rijeka Publisher: Polytechnic Rijeka WOS:000471074500016.

[13] Christine Bauer. On the (In-)Accuracy of GPS Measures of Smartphones: A Study of Running Tracking Applications. December 2013.

[14] J. Berndsen, A. Lawlor, and Barry Smyth. Running with Recommendation. In *HealthRecSys@RecSys*, 2017.

[15] Alessandro Biessek. *Flutter for Beginners: An introductory guide to building cross-platform mobile applications with Flutter and Dart 2*. Packt Publishing Ltd, September 2019. Google-Books-ID: pF6vDwAAQBAJ.

[16] Andreas Biørn-Hansen, Tor-Morten Grønli, Gheorghita Ghinea, and Sahel Alouneh. An Empirical Study of Cross-Platform Mobile Development in Industry. *Wireless Communications and Mobile Computing*, 2019:1–12, January 2019.

[17] Guiran Chang, Chunguang Tan, Guanhua Li, and Chuan Zhu. Developing Mobile Applications on the Android Platform. In Xiaoyi Jiang, Matthew Y. Ma, and Chang Wen Chen, editors, *Mobile Multimedia Processing: Fundamentals, Methods, and Applications*, Lecture Notes in Computer Science, pages 264–286. Springer, Berlin, Heidelberg, 2010.

[18] Stephanie Hui-Wen Chuah, Philipp A. Rauschnabel, Nina Krey, Bang Nguyen, Thurasamy Ramayah, and Shwetak Lade. Wearable technologies: The role of usefulness and visibility in smartwatch adoption. *Computers in Human Behavior*, 65:276–284, December 2016.

[19] Rudyar Cortes, Xavier Bonnaire, Olivier Marin, and Pierre Sens. Sport Trackers and Big Data: Studying user traces to identify opportunities and challenges. November 2014.

[20] Nicolas Couvrat. Wait… What Happens When my React Native Application Starts? — An In-depth Look Inside React Native, March 2018.

[21] Dinu Covaciu, Daniela Florea, Ion Preda, and Janos Timar. Using GPS Devices For Collecting Traffic Data. 2008.

[22] Dario Radečić. Data Science for Cycling - How to Visualize Gradient Ranges of a GPX Route, March 2022.

[23] Dashrath Mane, Namrata Ojha, and Ketaki Chitnis. The Spring Framework: An Open Source Java Platform for Developing Robust Java Applications. July 2013(Volume-3 Issue-2):137–143, July 2013.

[24] Chris Davies. Endomondo Is Shutting Down – Here's How To Export Workout Data, November 2020.

[25] G.M. Djuknic and R.E. Richton. Geolocation and assisted GPS. *Computer*, 34(2):123–125, 2001. Conference Name: Computer.

[26] Bakwa Dunka, Edim Emmanuel, and Yinka Oyerinde. HYBRID MOBILE APPLICATION BASED ON IONIC FRAMEWORK TECHNOLOGIES. *International Journal of Recent Advances in Multidisciplinary Research*, 04:3121–3130, December 2017.

[27] Nimesh Ekanayake. Android Operating System. May 2018.

[28] Przemysław Falkowski-Gilski and Jacek Stefański. Android OS: A Review. *Tem Journal*, 4:116–120, 2015.

[29] Iztok Fister, Iztok Fister, Duan Fister, and Simon Fong. Data Mining in Sporting Activities Created by Sports Trackers. In *2013 International Symposium on Computational and Business Intelligence*, pages 88–91, August 2013.

[30] Mark Goadrich and Michael Rogers. Smart smartphone development: IOS versus Android. pages 607–612, January 2011.

[31] Ekrem GÜLCÜOĞLU, Ahmet USTUN, and Neşet Seyhan. Comparison of Flutter and React Native Platforms. *Journal of Internet Applications and Management*, December 2021.

[32] Heliza Rahmania Hatta, Muhammad Suroso, Indah Astuti, Dyna Khairina, and Septya Maharani. Application of Haversine Formula in Education Game "Landmark Nusantara". January 2021.

[33] Ying Hu. Research and Development on e-book Apps Based on iOS Development Platform. pages 395–400. January 2016.

[34] Ng Hui, Liu Chieng, Wen Ting, Hasimah Mohamed, and Muhammad Mohd Arshad. Cross-platform mobile applications for android and iOS. pages 1–4, April 2013.

[35] Ghusoon Idan and Kadhum Al-Majdi. A Freights Status Management System Based on Dart and Flutter Programming Language. *Journal of Physics: Conference Series*, 1530:012020, May 2020.

[36] Mohd Javaid, Abid Haleem, Shanay Rab, Ravi Pratap Singh, and Rajiv Suman. Sensors for daily life: A review. *Sensors International*, 2:100121, January 2021.

[37] K. Divya and S. Venkata KrishnaKumar. Comparative Analysis Of Smart Phone Operating Systems Android, Apple Ios And Windows. *IJSEAS - International Journal of Scientific Engineering and Applied Science*, 2016(2):432–438.

[38] Bryan Kayfitz. Getting Started with the BLoC Pattern, August 2019.

[39] Khanh Nguyen, Brian Tobin, and Luke Cheng. Flutter architectural overview.

[40] Deb Kingsbury. How to Use a GPS: Tracks and Routes, 2018.

[41] Ming Liu. A Study of Mobile Sensing Using Smartphones. *International Journal of Distributed Sensor Networks*, 9(3):272916, March 2013. Publisher: SAGE Publications.

[42] Hagar Mahmoud and Nadine Akkari. Shortest Path Calculation: A Comparative Study for Location-Based Recommender System. pages 1–5, March 2016.

[43] Antonio Martinez-Nicolas, Adrià Muntaner-Mas, and Francisco Ortega. Runkeeper: A complete app for monitoring outdoor sports. *British Journal of Sports Medicine*, 51:bjsports–2016, August 2016.

[44] Sergio Mascetti, Mattia Ducci, Niccolò Cantù, Paolo Pecis, and Dragan Ahmetovic. *Developing Accessible Mobile Applications with Cross-Platform Development Frameworks*. May 2020.

[45] Matthew Mayo. Linear Regression, Least Squares & Matrix Multiplication: A Concise Technical Overview, November 2016. Section: 2016 Nov Tutorials, Overviews.

[46] Wes McKinney. pandas: a Foundational Python Library for Data Analysis and Statistics. *undefined*, 2011.

[47] Wes McKinney. *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. "O'Reilly Media, Inc.", October 2012. Google-Books-ID: v3n4_AK8vu0C.

[48] Krista Merry and Pete Bettinger. Smartphone GPS accuracy study in an urban environment. *PLoS ONE*, 14:e0219890, 2019.

[49] Alberto Miola. *Flutter Complete Reference: Create Beautiful, Fast and Native Apps for Any Device*. Independently Published, 2020. Google-Books-ID: y372zQEACAAJ.

[50] Anna Mironova and Alexander Minakov. *KML cookbook (early version)*. 2018.

[51] Marian Mohr, Christopher Edwards, and Ben McCarthy. A study of LBS accuracy in the UK and a novel approach to inferring the positioning technology employed. *Computer Communications*, 31:1148–1159, April 2008.

[52] Antonio Mozas. Accuracy assessment of speed values calculated from GNSS tracks of roads obtained from VGI. *Survey Review*, 51:1–10, April 2018.

[53] Mukesh Prajapati, Dhananjay Phadake, and Archit Poddar. ISSN 2320-8163 (Online) | International journal of technical research and applications | The ISSN Portal. *International Journal of Technical Research and Applications*, 2016(4).

[54] Ovidiu Constantin Novac, Mihaela Novac, Cornelia Gordan, Tamas Berczes, and Gyöngyi Bujdosó. Comparative study of Google Android, Apple iOS and Microsoft Windows Phone mobile operating systems. In *2017 14th International Conference on Engineering of Modern Electric Systems (EMES)*, pages 154–159, June 2017.

[55] Rostislav Nétek and Jaroslav Burian. Analysis of elevation data with time aspects for athletes. page 10, January 2012.

[56] Onuoha ifeanyi. Dependency Injection In Flutter Using Get_it, April 2021.

[57] Kunwoo Park, Ingmar Weber, Meeyoung Cha, and Chul Lee. Persistent Sharing of Fitness App Status on Twitter. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*, CSCW '16, pages 184–194, New York, NY, USA, 2016. Association for Computing Machinery.

[58] Stefania Pizza, Barry Brown, Donald McMillan, and Airi Lampinen. Smartwatch in *vivo*. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pages 5456–5469, New York, NY, USA, May 2016. Association for Computing Machinery.

[59] Ana Polo. Flutter bloc for beginners, January 2022.

[60] Nandhini Abirami R, Seifedine Kadry, Amir H. Gandomi, and Balamurugan Balusamy. *Big Data: Concepts, Technology, and Architecture*. John Wiley & Sons, March 2021. Google-Books-ID: tI0kEAAAQBAJ.

[61] RNDr. Marie Budíková, Dr. Statistika a pravděpodobnost | Přírodovědecká fakulta Masarykovy univerzity, 2016.

[62] Gustavo Romanillos, Martin Zaltz Austwick, Dick Ettema, and Joost De Kruijf. Big Data and Cycling. *Transport Reviews*, 36(1):114–133, January 2016. Publisher: Routledge _eprint: https://doi.org/10.1080/01441647.2015.1084067.

[63] Angela Schirck-Matthews, Hartwig Hochmair, Gernot Paulus, and Dariia Strelnikova. Comparison of cycling path characteristics in South Florida and North Holland among three GPS fitness tracker apps. *International Journal of Sustainable Transportation*, 2021.

[64] Andysah Putera Utama Siahaan. *Haversine Method in Looking for the Nearest Masjid*. September 2017.

[65] Manav Singhal and Anupam Shukla. Implementation of Location based Services in Android using GPS and Web Services. *International Journal of Computer Science Issues*, pages 16940784–16940814.

[66] Xie Songshi. The Program Construction Method of Navigation Format Files GPX and KML Based on Geological Exploration Point Information. Jun 2021(Čís. 3):83–86, June 2021.

[67] Xing Su, Hanghang Tong, and Ping Ji. Activity recognition with smartphone sensors. *Tsinghua Science and Technology*, 19(3):235–249, June 2014. Conference Name: Tsinghua Science and Technology.

[68] Alycia N. Sullivan and Margie E. Lachman. Behavior Change with Fitness Technology in Sedentary Adults: A Review of the Evidence for Increasing Physical Activity. *Frontiers in Public Health*, 4:289, January 2017.

[69] Carsten Sørensen, Mark de Reuver, and Rahul Basole. Mobile Platforms and Ecosystems. *Journal of Information Technology*, 30:195–197, September 2015.

[70] Aakanksha Tashildar*, Rushabh Gala*, Nisha Shah, Trishul Giri, and Pranali Chavhan. APPLICATION DEVELOPMENT USING FLUTTER. 2020(31), August 2020.

[71] Thomas Burkhart. get_it | Dart Package, July 2022.

[72] Kumar Vishal and Ajay Kushwaha. Mobile Application Development Research Based on Xamarin Platform. pages 115–118, August 2018.

[73] Robert-Andrei Voicu, Ciprian Dobre, Lidia Bajenaru, and Radu-Ioan Ciobanu. Human Physical Activity Recognition Using Smartphone Sensors. *Sensors (Basel, Switzerland)*, 19(3):E458, January 2019.

[74] Liam Richard West. Strava: challenge yourself to greater heights in physical activity/cycling and running. *British Journal of Sports Medicine*, 49(15):1024–1024, August 2015. Publisher: BMJ Publishing Group Ltd and British Association of Sport and Exercise Medicine Section: Mobile App User Guides.

[75] Edy Winarno, Wiwien Hadikurniawati, and Rendy Rosso. Location based service for presence system using haversine method. pages 1–4, November 2017.

[76] Wouter Nieuwerth. How to find the fastest section within a GPX file with Python & Jupyter Notebooks | Data, Analytics & Home Automation, February 2020.

[77] Jun Yang. Toward Physical Activity Diary: Motion Recognition Using Simple Acceleration Features with Mobile Phones. *ACM International Workshop on Interactive Multimedia for Consumer Electronics*, 39:1–10, January 2009.

[78] Frank Zammetti. React Native: A Gentle Introduction. In Frank Zammetti, editor, *Practical React Native: Build Two Full Projects and One Full Game using React Native*, pages 1–32. Apress, Berkeley, CA, 2018.

[79] Paul A Zandbergen. Accuracy of iPhone Locations: A Comparison of Assisted GPS, WiFi and Cellular Positioning. *Transactions in GIS*, 13(s1):5–25, 2009. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-9671.2009.01152.x.

[80] Marianna Zichar. Optimized design of customized KML files *. January 2014.

SEZNAM ZKRATEK

SDK	Software Development Kit
REST	Representational state transfer
CPT	Cross Platform Tools
DI	Dependency Inejction
UWP	Universal Windows Platform
XAML	Extensible Application Markup Language
IDE	Integrated Development Environment
MVC	Model-View-Controller
MVVM	Model-View-View-Model
ARM	Advanced RISC Machines
IL	Intermediate Language
UI	User Inteface
JSX	JavaScript XML
NPM	Node Package Manager
DOM	Document Object Model
API	Application Programming Interface
VM	Virtual Machine
ASOP	Android Open Source Project
ART	Android Runtime
CPU	Contorll Process Unit
GPU	Graphics Process Unit
SQL	Structured Query Language
AAPT	Android Asset Packaging Tool

ADB	Android Debug Bridge
apk	Android application package
jar	Java archive
AIDL	Android Interface Definition Language
DDMS	Dalvik Debug Monitor Service
DX	Dalvik cross-assembler
IPC	InterProcess Communication
SMS	Short Message Service
VPN	Virtual Private Network
JIT	Just In Time
NFC	Near Field Communications
USB	Universal Serial Bus
MAP	Message Access Profile
AOT	Ahead-Of-Time
AI	Artifical Integlligence
MIDI	Musical Instrument Digital Interface
OTA	Over-the-Air
ALAC	Apple Lossless Audio Codec
AAC	Advanced Audio Codec
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
CLR	Common Language Runtime
WinJS	Windows Library for JavaScript

RAM	Random Access Memory
NDK	Native Development Kit
OEM	Original Equipment Manufacturer
LLVM	Low Level Virtual Machine
OOP	Objected Oriented Programming
JVM	Java Virtual Machine
DVM	Dart Virtual Machine
MVP	Model View Presenter
NoSQL	Not Only Structured Query Language
SQL	Structured Query Language
RDBMS	Relational Database Management System
BLOB	Binary Large Object
JSON	JavaScript Object Notation
XML	Extensible Markup Language
BSON	Binary JavaScript Object Notation
RSS	Receive Signal Strenght
GSM	Global System for Mobile Communications
GLONASS	Global Navigation Satellite System
KLM	Keyhole Markup Language
OGC	Open Geospatial Consortium
GPX	GPS eXchange Format
UTC	Coordinated Universal Time
HR	Hearth Rate
GPS	Global Positioning System

A-GPS	Assisted GPS
WLAN	Wireless Local Area Network
TTFF	Time To First Fix
AP	Access Point
E-CID	Enhanced Cell ID
UML	Unified Modeling Language
UML	Unified Modeling Language
ORM	Object Relation Mapping
HTTP	Hypertext Transfer Protocol
CRUD	Create Read Update Delete
ACID	Atomicity, Consistency, Isolation, Durability
GIF	Graphics Interchange Format
PNG	Portable Network Graphics
JPEG	Joint Photographic Experts Group
URL	Uniform Resource Locator
SSe	Error Sum of Squares
SSt	Total Sums of Squared
GNSS	Global Navigation Satellite System

SEZNAM PŘÍLOH

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání diplomové práce

Autor: **Bc. Lukáš Kopecký**

Studium: I2000046

Studijní program: N1802 Aplikovaná informatika

Studijní obor: Aplikovaná informatika

Název diplomové práce: **Smart aplikace pro sportovní aktivity**

Název diplomové práce AJ: Smart Application for Sport Activities

Cíl, metody, literatura, předpoklady:

Cíl práce: Vytvořit aplikaci zaměřenou na záznam sportovních výsledků a aktivit sprotovců. Aplikace bude využívát geoinformační prvky s průběžným vykreslováním a propočítáváním do mobilní aplikace a chytrých hodinek.

1. Úvod
2. Cíl práce a metodika
3. Návrh a implementace
4. Mobilní aplikace
5. Chytré hodinky
6. Shrnutí výsledků
7. Závěr

Programming Flutter: Native, Cross-Platform Apps the Easy Way (The Pragmatic Programmers) ISBN-13: 978-1680506952 Flutter Complete Reference: Create Beautiful, Fast and Native Apps for Any Device ISBN: 9798691939952

Garantující pracoviště: Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu

Vedoucí práce: doc. Mgr. Tomáš Kozel, Ph.D.

Datum zadání závěrečné práce: 15.10.2021