# Nativní Android aplikace PhotonRun pro účastníky běžeckých závodů

Bc. Lukáš Velecký

Diplomová práce
2021

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2020/2021

# ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

| | |
|---|---|
| Jméno a příjmení: | **Bc. Lukáš Velecký** |
| Osobní číslo: | **A18280** |
| Studijní program: | **N3902 Inženýrská informatika** |
| Studijní obor: | **Informační technologie** |
| Forma studia: | **Kombinovaná** |
| Téma práce: | **Nativní Android aplikace PhotonRun pro účastníky běžeckých závodů** |
| Téma práce anglicky: | **A Native Android Application – PhotonRun for Race Running Participants** |

## Zásady pro vypracování

1. Seznamte se technologiemi tvorby mobilních aplikací na platformě .NET. a Xamarin.
2. Seznamte se s tvorbou nativních aplikací pro operační systém Android.
3. Navrhněte aplikaci, která bude sloužit účastníkům vybraných běžeckých závodů
4. Realizujte napojení na již existující systém PhotonHero.
5. Navrženou aplikaci implementujte a ověřte funkčnost vytvořené aplikace.
6. Zhodnoťte vytvořenou aplikaci a navrhněte další možná rozšíření aplikace.

Forma zpracování diplomové práce: **Tištěná/elektronická**

Seznam doporučené literatury:

1. SHACKLES, Greg. Mobile development with C#: Building Native iOS, Android, and Windows Phone Applications. 1. Sebastopol, CA: O'Reilly, 2012. ISBN 978-1449320232.
2. SHARP, John. Microsoft Visual C# 2010: krok za krokem. 3. Brno: Computer Press, 2010. Krok za krokem (Computer Press). ISBN 978-80-251-3147-3.
3. C# Guide | MIcrosoft Docs [online]. [cit. 2019-09-23]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/
4. Documentation | Android Developers [online]. [cit. 2019-09-22]. Dostupné z: https://developer.android.com/docs
5. Xamarin.Android – Xamarin [online]. [cit. 2019-09-22]. Dostupné z: https://docs.microsoft.com/en-us/xamarin/android/

Vedoucí diplomové práce: **Ing. Bc. Pavel Vařacha, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **15. ledna 2021**
Termín odevzdání diplomové práce: **17. května 2021**

**doc. Mgr. Milan Adámek, Ph.D.** v.r.
děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

**prof. Mgr. Roman Jašek, Ph.D.** v.r.
ředitel ústavu

Ve Zlíně dne 15. ledna 2021

**Prohlašuji, že**

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomové práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky. Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má Univerzita Tomáše Bati ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

**Prohlašuji,**

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 17.5.2021 Bc. Lukáš Velecký, v.r.

**ABSTRAKT**

Diplomová práce se zabývá vývojem nativní Android aplikace pro projekt PhotonHero. Cílem aplikace je navrhnout a implementovat aplikaci sloužící pro účastníky běžeckých závodů. Pro tuto podporu aplikace využívá zaznamenávaní polohy uživatele do aktivit ve formátu GPX, které je možné následně analyzovat pomocí připojené služby. Práce řeší specifika vývoje nativní Android aplikace, především omezení v souvislosti se získáváním lokace v souvislosti s verzí Androidu. Výsledná aplikace je vytvořena na platformě .NET s využitím Mono.

Klíčová slova: Android, .NET, Mono, Xamarin, Android Services

**ABSTRACT**

The thesis deals with the development of a native application for Android for the PhotonHero project. The aim of the application is to design and implement an application for participants in footraces. For this purpose, application records the user's location into activities in GPX format, which can then be analyzed using connected services. The work addresses the specifics of developing native application for Android, especially the limitations of obtaining locations in connection with the Android version. The resulting application is created on the .NET platform using Mono.

Keywords: Android, .NET, Mono, Xamarin, Android Services

Chtěl bych poděkovat Ing. Pavlu Vařachovi, Ph.D. za pomoc a umožnění vypracování diplomové práce na vlastní téma.

Dále bych chtěl poděkovat členům projektu PhotonHero za možnost stát se součástí tohoto projektu, za cenné rady v průběhu vývoje a za pomoc při testování vyvíjené aplikace.

## OBSAH

# ÚVOD

Cílem diplomové práce je navrhnout a vytvořit nativní Android aplikaci pro projekt PhotonHero. Původní zaměření projektu PhotonHero se soustředilo na fotografování především běžeckých závodů pro veřejnost. Nejen kvůli omezením pořádání těchto závodů v souvislostí s pandemií koronaviru, ale také rozšíření aktivit se upravilo zaměření projektu i na zaznamenávaní sportovních aktivit a s tím souvisejícími funkcionality.

Práce se v první části zaměří na seznámení se s mobilním vývojem v rámci platforem, které jsou součástí .NET. Práce postupně uvede .NET Framework, .NET Core (s nástupcem v podobě .NET 5) a .NET Standard, a jejich význam v rámci celého systému. Následně představí multiplatformní implementaci v podobě platformy Mono, její význam v rámci celého systému .NET a především její implementace Xamarin.Android a Xamarin.iOS pro multiplatformní vývoj. V souvislosti s touto platformou se práce dále zabývá frameworkem Xamarin.Forms spolu s knihovnou Xamarin.Essentials. Dále zmíní alternativy k Monu a Xamarin.Forms pro vývoj mobilních aplikací na platformě .NET.

V další části se práce věnuje specifikům vývoje aplikací na operační systém Android. Z tohoto pohledu se zaměří především na problematiku získávání polohy zařízení a vytváření úloh, které umí tuto polohu získávat i na pozadí operačního systému. V tomto ohledu práce prozkoumá možnosti tzv. *Services*. K tomu probere vliv jednotlivých verzích Androidu a také, jaký vliv mohou mít jednotlivý výrobci mobilních zařízení.

V praktické části se nejprve práce věnuje samotnému návrhu vyvíjené aplikace. Při návrhu jsou zohledněny požadavky jak ze strany projektu, tak především z pohledu uživatele. Kromě toho se řeší také propojení s již existujícím systémem projektu a návrh přenosu aktivit mezi zařízením a databází. Dále praktická část řeší samotnou implementaci jak webové služby, tak především navrhované aplikace. U obou projektů vysvětluje důvod zvolených technologií a těm se následně i věnuje, včetně daných úloh v rámci projektu. U aplikace je nutné vyřešit otázky související s úlohou pro získávání polohy s ohledem na verzi operačního systému a výrobce daného zařízení.

Poslední část práce představí zvolený způsob testování, především jednotlivé fáze testování, jejich cíle a význam. Práce dále zmíní zvolený způsob distribuce samotné aplikace, především pomocí jakých nástrojů toto bude probíhat. Také uvede možnou budoucnost aplikace, její další možné směrování a rozšiřování.

# I. TEORETICKÁ ČÁST

# 1 CÍLE PRÁCE

Prvním cílem práce, kterému se práce věnuje v kapitole 4, je představit možnosti tvorby mobilních aplikací na platformě .NET s využitím platformy Xamarin a Mono. Práce se postupně věnuje platformám Mono a Xamarin, především jedné z implementací Xamarin.Android. Kromě platformě specifických implementací se práce věnuje také multiplatformnímu vývoji aplikací s využitím frameworku Xamarin.Forms.

Následujícím cílem je prozkoumat možnosti tvorby mobilních aplikací se zaměřením na operační systém Android. V kapitole 5 se prozkoumávají specifické záležitosti jak samotné tvorby mobilních technologií, ale především specifika tohoto operačního systému. Z těchto lze zmínit např. životní cyklus aplikace, systémová oprávněni nebo služeb (Android Services).

Kromě samotného vývoje Android aplikací na platformě .NET se práce zmiňuje o vývoji Android aplikací na jiných programovacích platformách, a to alternativami čistě pro Android (jazyky fungující primárně na Java Virtual Machine) tak multiplatformními např. v podobě React.Native, QT a další.

Hlavním cíle celkové práce je navrhnout a implementovat aplikaci k projektu PhotonHero. Tyto body jsou řešeny v rámci kapitol 7 a 8. Výsledným stavem aplikace je jak napojení se již k existujícímu systému PhotonHero, tak tento systém rozšířit o další funkcionality. Napojení aplikace proběhne vytvořením webové služby připojující se k databázi a přímým přístupem k Azure Storage. Mezi hlavní výsledné funkcionality je zobrazování údajů z databáze, především informace o závodech, závodních sériích a uživatelích.

Kapitola 8 se věnuje způsobu, jakým způsobem proběhne testování a distribuce. Aplikace se jak při testování, tak v produkci bude distribuovat přes Google Play. Analýza a diagnostika funkcionality aplikace bude zjišťována pomocí služby Visual Studio AppCenter.

# 2 MOTIVACE

Hlavní motivací této práce byla nabídka k vytvoření mobilní aplikace k projektu PhotonHero. Kromě samotné nabídky spolupráce na tomto projektu bylo další výraznou motivací také seznámení se s vytvářením mobilních aplikací. Díky autorovým znalostem technologií využívajících .NET Framework, který je využit i v rámci existující infrastruktury PhotonHero, byla pro tvorbu mobilní aplikace zvolena platforma Xamarin, resp. Mono. Právě naučení se vytváření mobilních bylo silnou motivací pro vytvoření této aplikace.

Z pohledu projektu PhotonHero bylo motivací pro vytvoření mobilní aplikace další možné využití již existující platformy a využití funkcionality, kterou je možné poskytnou na mobilních aplikacích, především využití GPS modulů webových pro ukládání polohy uživatele do aktivit. Toto umožní projektu PhotonHero rozšíření zaměření, kdy původně platforma působila primárně jako platforma pro sdílení fotografií ze závodů.

Výraznou motivací, především v období epidemie koronaviru v roce 2020 a minimálně v první polovině roku 2021, se stala podpora pořádání virtuálních závodů. Díky nemožnosti organizovat klasické běžecké závody v období epidemie, někteří z organizátorů zvolili pořádání virtuálních závodů namísto klasických závodů. Právě podpora těchto závodů změnila prioritu aplikace směrem k zaznamenávání polohy uživatelů a jejich následnému zpracování.

# 3 SOUČASNÝ STAV ŘEŠENÉ PROBLEMATIKY

V současné době existuje několik řešení, které řeší problematiku zaznamenávání polohy uživatele. Mezi nejznámější řešení této problematiky je služba Strava. Strava poskytuje kromě samotného zaznamenávání polohy uživatelů do aktivity také následnou analýzu těchto aktivit. Kromě toho v základu také funguje jako sociální síť pro sportovce (sdílení aktivity, fotky aktivit, sledování uživatelů), umožňuje účastnit se výzev a další funkcionality. Umožňuje využití nejen chytrých telefonů, ale také nositelné elektroniky (chytré hodinky a náramky). [1]

Další obdobnou službou, která poskytuje sledování polohy uživatele a zaznamenávání do cvičení je skupina služeb MapMyWalk, MapMyRun, MapMyFitness a MapMyRide. Jednotlivé služby poskytují jako základ zaznamenávání cvičení, základní statistiky či osobní cíle, kromě toho poskytuje také placené rozšíření, které umožňují lepší analýzy cvičeni, tréninkové programy či exporty cvičení. Nevýhodou tohoto řešení je, že každý sport (chůze, běh, cyklistika a fitness) má svou vlastní aplikaci a službu, nefungují tedy jako celek. [2] [3] [4] [5]

Dříve do této skupiny služeb patřilo i Endomondo, které ale k 31.12.2020 ukončilo svou činnost. Při přechodu nabídlo a doporučilo uživatelům přechod na zmíněnou službu MapMyRun. [6]

Mezi další služby či aplikace se řadí také mnoho "platformních"aplikací a webových služeb. Tyto aplikace jsou často přímo propojené s výrobky a příslušenstvím či chytrou elektronikou výrobce. Do této skupiny lze zařadit firmu Garmin s její službou Garmin Connect. Služba umožňuje sledování nejen aktivit, ale také kroků či tělesných vlastností, jako je srdeční tep, úroveň stresu či ukazatel VO2 max (ukazatel aerobní vytrvalosti). Garmin poskytuje kromě služby také svou vlastní nositelnou elektroniku, které se s touto službou automaticky synchronizují. [7]

Kromě Garmin k této skupině služeb řadí také další výrobci nejen nositelné elektroniky, v některých případech i třeba výrobci chytrých mobilních telefonu, např. Huawei se svou službou Huawei Health, Xiaomi s Mi Fit, Apple s Apple Fitness+ nebo služba a hodinky FitBit vlastněné firmou Google.

Mezi služby fungující jako seznam běžeckých závodů lze zařadit službu Termínovka českého portálu Svět běhu. Z pohledu seznamu závodů ale služba funguje pouze jako základní přehled s odkazy na webové stránky konkrétních závodů. Z pohledu uživatelů má přidanou hodnotu v podobě diskuse se zkušenostmi s daným závodem. [8]

Obdobnou službou se stejným názvem poskytuje také portál Běhej, která navíc poskytuje výsledky z proběhlých závodů. [9]

# 4 TVORBA MOBILNÍCH APLIKACÍ NA PLATFORMĚ .NET

Mobilní aplikace na platformě .NET lze vyvíjet dvěma způsoby. Prvním způsobem jsou aplikace naspané v technologii UWP (Universal Windows Platform) pro cílení aplikací na dnes již nevyvíjený operační systém Windows 10 Mobile. Druhou variantou, která je pro tuto diplomovou práci podstatnou a věnuje se ji tato kapitola, je cílení aplikací na operační systémy Android, resp. iOS. Pro tyto účely je využívaná platforma Mono, resp. její implementace Xamarin.Android právě pro Android, resp. Xamarin.iOS pro iOS.

## 4.1 Platforma .NET

.NET je platforma vyvinutá Microsoftem zahrnující knihovny, nástroje a programovací jazyky sloužící pro vývoj např. desktopových či právě mobilních aplikací. Aktuálně je .NET implementován 3 různými implementacemi: .NET Framework, .NET Core a Mono. Navíc existuje specifikace .NET Standart, která formuluje specifikace rozhraní, které jsou shodné pro všechny zmíněné implementace. [10]

V roce 2019 představil přímého nástupce .NET Core s pojmenováním .NET 5. [11] Kromě toho, že tento .NET 5 bude nahrazovat .NET Core ve verzi 3.0, je také nástupcem .NET Frameworku ve verzi 4.8.0.

Mezi programovací jazyky, které jsou podporovány na platformě .NET jsou především jazyky C#, Visual Basic .NET a F#. Jazyk C# byl představen spolu s .NET Framework 1.0 s veřejnou premiérou v roce 2001. [12] Spolu s novým jazykem C# byl také představena nová generace jazyka VB.NET jako nástupce Visual Basic právě pro platformu .NET. Později byl k těmto jazykům přidán i jazyk F# jako zástupce funkcionálních jazyků.

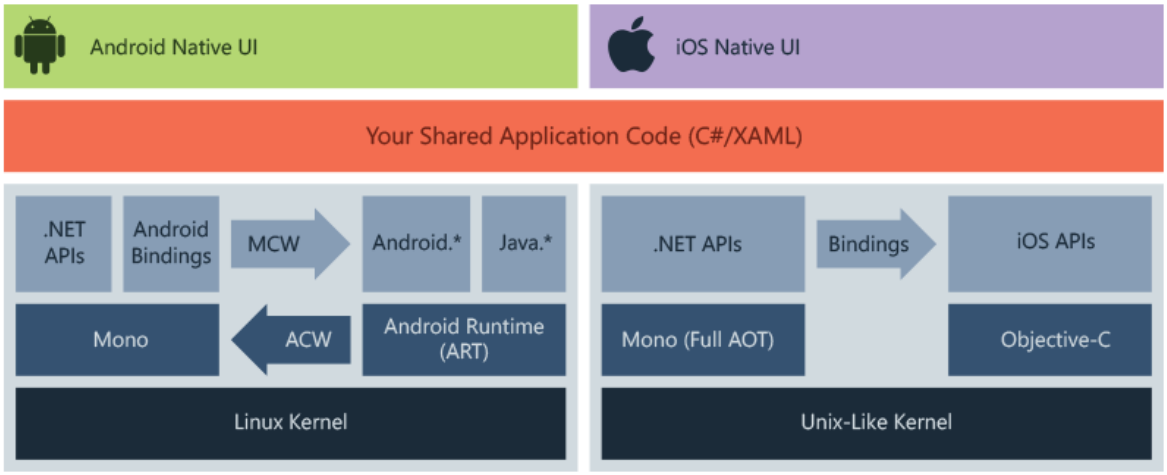
Základem překladu a spouštění aplikací či knihoven v .NET Framework je Common Language Infrastructure (CLI). Tato specifikace umožňuje zdrojové kódy psané v různých jazycích přeložit do kódu konkrétní cílové platformy. K tomuto využívá Common Intermidiate Language (CIL), který představuje mezikrok sjednocující právě zdrojové kódy do instrukcí, a Common Language Runtime (CLR), který zajišťuje samotné převedení CIL do bitového kódu konkrétní platformy, resp. CPU. Mezi implementátory této infrastruktury je především právě .NET Framework nebo Mono. [13]

## 4.2 Platformy Xamarin a Mono

Mono je open source projekt, který byl představen v roce 2001, první release verze 1.0 byla představena o 3 roky později v roce 2004. [14]. Úkolem této platformy, za kterou původně stojí Miguel de Icaza s firmou Ximian (později odkoupena firmou

Novell) [15], bylo vytvořit implementaci .NET Framework pro linuxové jádro. Celá platforma se v roce 2016 stala součástí organizace .NET Foundation a Microsoft se prostřednictvím této organizace stal hlavním sponzorem tohoto projektu. [14]. Vývoj Mona přešel pod nově vytvořenou společnost Xamarin, založenou Microsoftem. Po této změně došlo k úpravě vývoje směrem k mobilním operačním systémům. [16]

Platforma Xamarin je zastřešující platformou pro jednotlivé implementace platformy Mono a k tomu přidává možnosti pro vývoj na Windows. Oproti Monu, který je platformě specifický (každá implementace Mona je pro konkrétní platformu), Xamarin umožňuje mezi platformní architekturu vývoje aplikací. Schéma multiplatformní architektury lze vidět na obrázku 4.1. Při této architektuře lze využít znovupoužitelnosti části kódu, především business logiky, kdy lze pro jednotlivé platformy použit některé části kódu stejné. Nad společnou vrstvou leží uživatelské rozhraní jednotlivých platforem a případná platformě specifická logika aplikace. Obě tyto části jsou psány s ohledem na konkrétní platformu. Pod společnou vrstvou, leží samotné Mono, které se stará o překlad .NET konstrukcí do knihoven a rozhraní konkrétní platformy a vytváří výslednou aplikaci spustitelnou na jádře (kernelu) daného systému. [17]




Obrázek 4.1 Diagram architektury Xamarin[17]

Xamarin.Android je implementací Mona pro operační systém Android a umožňuje vývoj nativních aplikací pro Android. Zpřístupňuje Android SDK pro vývoj na Xamarinu, resp. na .NET technologiích. Kromě zpřístupnění SDK se stará také o převod .NET knihoven na knihovny Javy, resp. Androidu. Výstupem je archiv Android package (APK), která obsahuje vše potřebné pro instalaci aplikace. Obdobným způsobem funguje také Xamarin.iOS, který zpřístupňuje SDK operačního systému iOS a převádí rozhraní .NET iOS rozhraní.

Uživatelské rozhraní je při tomto způsoby vývoje vyvíjeno závisle na zvolené platformě. Tedy v případě Xamarin.Android se jedná o nativní uživatelské rozhraní Androidu. Toto rozhraní, založené na XML, je specifické pro Android a nesouvisí s ostatními

.NET knihovnami pro uživatelské rozhraní.

## 4.3 Xamarin.Forms

Xamarin.Forms je framework nejen pro tvorbu mobilních aplikací. Xamarin.Forms využívá implementace Mona pro jednotlivé platformy spolu s platformou UWP. Toto umožňuje tvorbu mobilních aplikací nejen pro Android a iOS, ale také pro aplikace využívající UWP, tedy Windows 10, Xbox One nebo dříve také Windows 10 Mobile. Kromě mobilních platforem a platforem od Microsofu Xamarin.Forms umožňuje také vývoj na platformu macOS využívající implementaci Mona Xamarin.Mac. Dále také podporuje vývoj aplikací pomocí GTK# nebo pro operační systém Tizen od Samsungu. [18]

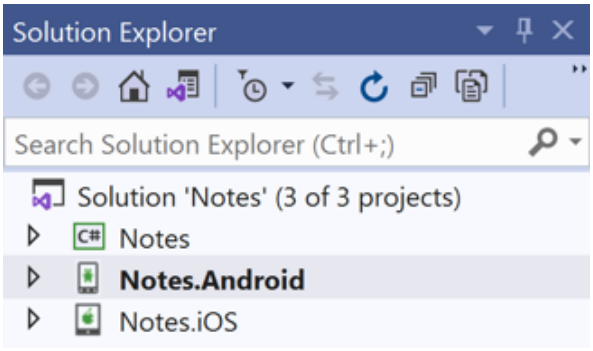
Při vývoji aplikací pomocí jednotlivých platformních implementací Mona je nutné pro každou z těchto aplikací vytvořit vlastní uživatelské rozhraní. V případě vývoje aplikace na více platforem toto může prodloužit dobu vývoje, jelikož některé části je třeba vytvořit ke každé platformě zvlášť. Právě pro usnadnění mezi platformního vývoje slouží Xamarin.Forms. Tento umožňuje vytvářet jak uživatelské rozhraní, ale také velkou část samotného kódu pro všechny platformy současně.

Vytvoření uživatelského rozhraní lze vytvořit pomocí Xamarin dialektu jazyka XAML nebo přes kód psaného v C#. XAML (Extensible Application Markup Language) je značkovací jazyk založený na XML vytvořený Microsoftem. Tento postup tvorby uživatelského rozhraní kromě usnadnění samotného vývoje, také umožňuje konzistentní design aplikace na všech platformách. Jazyk byl vytvořen jako alternativa k programovacím jazykům a stará se jak o vytváření, tak i inicializaci těchto objektů. XAML přímo také podporuje návrhový vzor Model-View-ViewModel (MVVM), sloužící k propojení kódu uživatelského rozhraní s backend modelem. Kromě Xamarin dialektu existují také verze XAMLu pro Windows Presentation Foundation (WPF), UWP a Silverlight. Xamarin.Forms zpřístupňuje také platformě specifické vlastnosti uživatelských rozhraní jednotlivých platforem. [19]

Kromě usnadnění vývoje uživatelského rozhraní vývojářům poskytuje Xamarin.Forms také knihovny ke kompletnímu mezi platformnímu vývoji. K těmto patří mimo jiné řešení navigace v rámci aplikace, knihovny pro animace, vkládání závislostí přes *Dependency services* nebo předávaní zpráv pomoci *Messaging Center*. [19]

### 4.3.1 Struktura projektu s Xamarin.Forms a Xamarin.Android

Základní struktura *solution* s využitím Xamarin.Forms, která tvoří výslednou Android aplikaci, je Xamarin.Forms knihovna a další knihovny pro konkrétní nativní aplikace. Příklad struktury projektu obsahující knihovny pro Android a iOS je zobrazena na 4.2.




Obrázek 4.2 Struktura aplikace s Xamarin.Forms[20]

Prvním vytvořením projektem je .NET Standard knihovna. Tato knihovna obsahuje právě společnou logiku a uživatelské rozhraní. Jedná se právě o samotný Xamarin.Forms projekt. Hlavní třídou tohoto projektu je třída, která dědí třídu Application. Třída, ve výchozím stavu pojmenovaná jako App, se stará o základní inicializaci aplikace a logiku, která je pro celý projekt. K třídě existuje také připojeny XAML soubor. Tento soubor definuje tzv. *ResourceDictionary* Tyto obsahují např. definice stylů vzhledu, definice používání barev či konvertory, které jsou použity napříč aplikací. [20]

Druhým projektem je Android knihovna, využívající implementaci Mona Xamarin.Android. Tato knihovna produkuje výslednou nativní aplikaci se zacílením na operační systém Android. Ve výchozím stavu má tento projekt název složen7 z názvu Xamarin.Forms projektu s příponou *.Android*. Tento projekt, resp. kterýkoliv jiný nativní, slouží k provolání právě Xamarin.Forms projektu skrze výchozí aktivitu aplikace. Kromě provolání Xamarin.Forms projektu slouží tento projekt také implementaci částí logiky, která je specifická pro danou Android. [21]

Základní struktura tohoto projektu je oproti společné Xamarin.Forms částí o poznání složitější. První složkou je složka Properties. Tato složka obsahuje soubor *AndroidManifest*, který obsahuje mimo jiné jméno a verzi aplikace, nebo oprávnění, která aplikace může v průběhu práce s ní vyžadovat. Druhou významnou složkou je složka Assests. Tato složka může obsahovat vlastní fonty, lokální datové, čí textové soubory. Tyto soubory jsou přístupné přes třídu Assets. Poslední složkou je složka Resources, která obsahuje vizuální soubory využité v aplikací, především obrázky či rozvržení aplikace, ale také zdroje jako texty či barvy. Tyto jsou dostupné přes třídu Resources. Více o prostředcích v kap. 5.8. [21]

## 4.4 Xamarin.Essentials

Kromě Xamarin.Forms v ekosystému Xamarinu existuje k mezi platformnímu vývoji také knihovna Xamarin.Essentials. Oproti Xamarin.Forms, který vytváří nadstavbu

nad jednotlivými implementacemi Mona, Xamarin.Essentials tyto knihovny doplňuje. Využití těchto knihoven není cíleno pouze na aplikace napsané pomocí Xamarin.Forms, ale také na ty psané v Xamarin.Android, resp. Xamarin.iOS.

Cílem Xamarin.Essentials je mezi platformní přístup k funkcionalitám, které jsou společné pro jednotlivé platformy, přestože na každé platformě je tato funkcionalita řešena odlišně. Do skupiny těchto funkcionalit se řadí např. oprávnění, geolokace, přístup k údajům baterie, nebo bezpečné ukládání dat. [22].

### 4.4.1 Přesnost geolokace dle platformy

Právě geolokace je příkladem, jak Xamarin.Essentials sjednocuje přístup k rozhraní nezávisle na platformě, přestože každá z nich k této funkcionalitě přistupuje odlišně. Každá z platforem má řešení lokace zařízení řešeno odlišně. V případě řešení získávání by tedy bylo potřeba pro každou z platforem upravit volání. Především ale má každá z platforem řešenou jinak přesnost.

V případě Androidu SDK existují 3 definované hodnoty přesnosti pro požadavek, konkrétně hodnoty pro přesnost na úrovni města, bloku a nejpřesnější. K těmto se přidává poslední hodnota, při které požadavek vrací nejpřesnější možnou polohu, ale pouze tak aby nedošlo ke dodatečné spotřebě energie z baterie. Poloha zařízení v tomto případě je aktualizovaná, pokud jiný klient požádá o aktualizaci polohy zařízení. [23]

Naopak v případě iOS SDK existuje definovaných hodnot přesnosti 7. Tyto hodnoty postupně zmenšují přesnost, případně využívají možnosti dalších senzorů zařízení pro zpřesnění polohy. Nejpřesnější hodnota nevyužívá pouze samotnou GPS, ale zpřesňuje přesnost právě pomocí dalších senzorů. Další hodnoty postupně znázorňují nejlepší možnou přesnost, přesnost do 10 metrů, do 100 metrů, do 1 kilometrů a do 3 kilometrů. Poslední možnost přesnosti je využita, pokud aplikace není autorizována k větší přesnosti.

Jak je vidět v předchozích odstavcích, rozdíly v řešení přesnosti polohy zařízení jsou na zmíněných platformách velmi rozdílné. Právě modul Geolocation Xamarin.Essentials se snaží tento problém vývojářům usnadnit, v 4.1 lze vidět, jaké hodnoty nadefinoval tento modul pro přesnosti a jaké reálné hodnoty mohou zařízení vracet s ohledem na konkrétní platformě.

Tabulka 4.1 Tabulka přesností geolokace (hodnoty uvedeny v metrech) [24]

| | Lowest | Low | Medium (Default) | High | Best |
|---|---|---|---|---|---|
| *Android* | 500 | 500 | 100-500 | 0-100 | 0-100 |
| *iOS* | 3000 | 1000 | 100 | 10 | $\sim 0$ |
| *UWP* | 1000-5000 | 300-3000 | 30-500 | $<= 10$ | $<= 10$ |

### 4.5 Budoucnost Xamarin v .NET a platforma MAUI

V roce 2020 uvedl Microsoft nástupce platformy Xamarin.Forms s názvem *Multi-platform App UI* (MAUI), který bude součástí .NET od verze .NET 6. Kromě tohoto také bylo oznámeno, že od .NET 5 budou jednotlivé implementace Mona integrovány společně s původním .NET Core do základních bázových knihoven a vývojových balíčku. [25]

Oproti Xamarin.Forms by MAUI mělo zjednodušit vývoj multiplatformních aplikací. Zatímco v Xamarin.Forms musel být pro každou platformu vytvořen nový projekt, který využíval Xamarin.Forms, MAUI toto zjednodušuje. Aplikace vytvořena pomocí na této platformě bude vyžadovat pouze jeden projekt a ten bude obsahovat vše potřebné i pro konkrétní platformy, tedy platformě specifické implementace. Všechny zdroje (obrázky, fonty, a další) jsou součástí jednoho hlavního projektu. [25]

Kromě toho, že MAUI je nástupcem Xamarin.Forms, tedy především mobilního vývoje, do budoucna se plánuje také podpora desktopových aplikací na operační systémy Windows a macOS. MAUI tedy není pouze nástupcem Xamarin.Forms, ale také technologií WPF a UWP. Díky faktu, že nová platforma MAUI vychází z Xamarin.Forms a navazuje také jeho XAML dialekt, přechod mezi těmito platformami by neměl vývojářům způsobovat větší problémy. Oproti tomu přechod z technologií WPF a UWP bude díky odlišným XAML dialektům složitější.

### 4.6 Alternativy mobilního vývoje v .NET

Jak už bylo řečeno v rámci předchozích kapitol, v .NET existuje několik variant vývoje mobilních aplikací. Ať už se jedná o nativní řešení v podobě Xamarin.Android nebo Xamarin.iOS, nebo o multiplatformní v podobě Xamarin.Forms, všechny jsou závisle na Monu a jeho implementacích. Kromě těchto technologií se v .NET na mobilní vývoj soustředí platforma UWP. Tato platforma ale není přímo multiplatformní a podporuje pouze operační systémy založené na Windows, tedy kromě samotných Windows také třeba platformu Xbox nebo právě mobilní systém Windows 10 Mobile. Podpora tohoto mobilního systému je ale už ukončena a Microsoft tento systém nadále ani neudržuje.

Přestože se může zdát, že UWP díky ukončení podpory systému Windows 10 Mobile nelze zařadit do mobilního vývoje, není toto tvrzení zcela pravdou. Na základech UWP technologie vzniklo několik platforem, které umožňují stále mobilní vývoj za pomoci UWP. Mezi tyto platformy lze zařadit například UNO. Oproti Xamarin.Forms platforma UNO využívá právě dialekt UWP, na pozadí ale stále využívá jednotlivé implementace Mona. Kromě tohoto je projekt Una sponzorem .NET Foundatioun, organizací, která stojí za vývojem .NET, a díky tomuto se může podílet více na vývoji .NET. [26]

# 5 TVORBA NATIVNÍCH APLIKACÍ PRO ANDROID

Zatímco předcházející kapitola se zabývala obecným vývojem mobilních aplikací na platformě .NET, především s využitím platformy Xamarin a implementací Mona, tato kapitola se bude věnovat vývoji a problémům pro aplikace cílené právě na mobilní systém Android. Postupně se bude věnovat nejen aktivitám či službám, ale také roztříštěnosti Androidu, oprávněním, či dalším specifikům vývoje aplikací na operační systém Android.

## 5.1 Verze Androidu

Verze Androidů jsou rozlišovány buď podle čísla úrovně API nebo podle verze, kdy některé úrovně API jsou zařazeny do jedné verze Androidu. K první polovině roku 2021 existuje celkem 11 verzí Androidu, která podporují dohromady 30 úrovní API. Nejaktuálnější verze, jsou zobrazeny v tabulce 5.1, která obsahuje také jejich kódová označení. Například u verzí 7 a 7.1 je vidět, že kódová označení jsou shodná. Verze 7.1 není novou verzi systému, pouze větší aktualizací verze 7. Tohoto systému bylo využíváno i u jiných verzí Androidu. Přesto tyto aktualizace upravují API úroveň systému, nejedná se tedy o stejnou verzi. Zatímco kódové označení může být duplicitní, v případě úrovně API musí mít každá nová verze novou úroveň, protože ta identifikuje pravé použité SDK.

Tabulka 5.1 Tabulka verzí Androidu [27]

| Kodové označení | Verze | Úroveň API |
|---|---|---|
| Android11 | 11 | API level 30 |
| Android10 | 10 | API level 29 |
| Pie | 9 | API level 28 |
| Oreo | 8.1.0 | API level 27 |
| Oreo | 8.0.0 | API level 26 |
| Nougat | 7.1 | API level 25 |
| Nougat | 7.0 | API level 24 |
| ⋮ | | |

Z pohledu vývoje aplikace jsou především podstatně právě verze úrovně API, které aplikace podporuje. Aplikace napsaná pro Android má definované tři nastavení, které jsou určeny právě úrovní API. Všechny úrovně se nastavují v rámci souboru AndroidManifest, viz kapitola 5.7.

První nastavení je Target Framework. Target Framework, také označený jako *compileSdkVersion*, určuje, jakou úrovní se bude aplikace kompilovat. Toto nastavení neurčuje přímo, které rozhraní knihoven SDK budou dostupné, nemění tedy chování při běhu

aplikace. Naopak ale upozorňuje uživatele, které rozhraní již mohou být například zastaralé. Toto nastavení je ale typické pro Xamarin, v rámci jazyků využívající JVM se nastavují pouze následující. [28]

Druhé nastavení, které je v rámci aplikace nastavené, je Minimum Android Version, také označené jako *minSdkVersion*. Tato úroveň určuje, na kterém nejstarším Androidu půjde aplikace spustit. Verze ale přímo neurčuje, které rozhraní má vývojář k dispozici, vývojář musí zohlednit, které API jsou podporovány v rámci verze a toto zohlednit při tvorbě aplikace.[28] Příkladem tohoto je například vertikální přesnost lokace zařízení, která byla přidána až v úrovni API 26. Pokud tedy aplikaci má minimální úroveň API 24 a spustí se na zařízení s Android 7.0, při pokusu o přístup k hodnotě vertikální přesnosti dojde k runtime chybě, případně i k pádu aplikace.

Posledním nastavením s ohledem na úroveň API je Target Android Version (*targetSdkVersion*). Jedná se o verzi, na které se očekává, že aplikace bude fungovat. Kromě toho tak určuje, které rozhraní budou v rámci aplikace k dispozici. [28]

Kromě toho, že při vývoji musí aplikace brát v potaz jednotlivé úrovně API Androidu jako takového, musí také zohlednit nadstavby uživatelského rozhraní čistého operačního systému u některých výrobců. Mezi tyto patří třeba nadstavby patří např. EMUI od Huawei, resp. jeho alternativa Magic UI pro Honor, MIUI od Xiaomi, One UI od Samsungu nebo Pixel UI přímo od Google. Tyto nadstavby ale v některých případech nezasahují jen do samotného uživatelského rozhraní, ale také například do optimalizace baterie.

Některé nadstavby obchází optimalizaci baterie Androidu a toto si řeší ve vlastní režii. Touhle problematikou se mimo jiné zabývá také stránka DontKillMyApp. [29] Na stránce jsou uvedeny řešení, jak obcházet optimalizaci baterie podle nadstavby operačního systému, případně v kombinaci s verzí Androidu. Tato stránka v době psaní diplomové práce jako nejproblémovější nástavby uvádí především Samsung, OnePlus a Huawei na prvních místech. Nejen u těchto tedy nestačí řešit pouze optimalizaci baterie v rámci Androidu, ale právě i různé kombinace u samotných nadstaveb.

Kromě tohoto musí vývojář zohlednit také verze operačního systému, které nejsou, nemusí nebo nemohou být vybaveny Google Mobile Services. S tímto má problémy především výrobce zařízení Huawei, který tyto služby nemůže od května 2019 využívat. [30]

## 5.2 Aktivity

Aktivita je ve světě Androidu základní komponentou aplikačního modelu. Zatímco základním paradigmatem desktopových aplikací je, že aplikace je spouštěna pomocí hlavní metody *Main()*, případně její alternativy podle zvolené platformy, toto neplatí ve světě

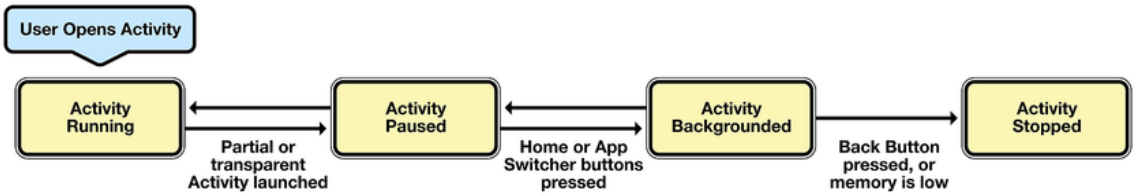
vývoje aplikací na Android. Aktivitu lze v základním stavu chápat jako jednu stránku aplikace. V případě přechodu mezi stránkami aplikace se spustí příslušná aktivita. Aplikace lze tedy chápat jako množinu aktivit, ke kterým se lze dostat. Aplikace jako taková je spuštěna právě pomocí aktivity. V aplikaci obvykle existuje jedna aktivita, která je spouštěna při otvírání aplikace v čistém stavu. Ostatní aktivity mohou být spuštěny jak jinou aktivitou aplikace, jak bylo zmíněno dříve, tak i jinou aplikací nainstalovanou v zařízení. Výchozí aktivita aplikace je nastavena v rámci souboru AndroidManifest, viz kapitola 5.7. [31]

Příkladem tohoto chování lze zmínit např. emailového klienta nebo aplikace pro mapy. V případě emailového klienta může být výchozí obrazovkou seznam doručených emailů, pokud ale přecházím do aplikace z jiné aktivity, může se aplikace spustit pomocí aktivity pro psaní nového emailu. V případě aplikace pro mapy může být výchozí scénou mapa zobrazující aktuální polohu zařízení nebo ta poslední zobrazená. Pokud ale přicházím z jiné aplikace, může se aplikace spustit např. v režimu navigace nebo zobrazovat specifickou lokalitu, kterou si zvolila původní aplikace.

Problémem aktivit je jejich životní cyklus. Důvodem tohoto problému jsou zásahy jak uživatelem (minimalizace aplikace, přepínání aplikací nebo také otočení zařízení), ale také systémem (pozastavení či zastavení aplikace běžící na pozadí). Z tohoto důvodu existuje několik stavů, ve kterých se aplikace, resp. aktivita může nacházet a k tomu několik metod, které jsou vyvolávány při přechodu mezi stavy.

### 5.2.1 Životní cyklus

Na obrázku 5.1 jsou vidět právě jednotlivé stavy, ve kterých se může aktivita nacházet a jak může aktivita mezi jednotlivými stavy přecházet. Tyto stavy mají vliv na to, jak s nimi zachází systém z pohledu jejich ukončování.




Obrázek 5.1 Stavy životního cyklu aktivity[32]

První stav, ve kterém se aktivita může nacházet je Aktivní stav (*Active* nebo *Running*). Aktivní aktivitou se rozumí ta, která je na popředí, tedy je to aktivita, se kterou uživatel aktivně pracuje. Taková aktivita je systémem vedena s nejvyšší prioritou a aktivita může být ukončena systémem až v extrémních situacích, např. pokud vyžaduje více paměti, než má zařízení k dispozici. [32]

Pozastaveným (*Paused*) stavem aktivity se rozumí stav, kdy se uspí zařízení nebo je aktivita překryta jinou aktivitou, která je transparentní či nepřekrývá celou plochu původní aktivity. Pozastavená aktivita je brána jako živá, udržuje si tedy stav a informace. Takové aktivity mají druhou nejvyšší prioritu pro operační systém a jsou ukončovány pouze ve chvíli, kdy systém nemá dostatek prostředků pro zajištění stability a odezvy aktivní aktivity a prostředky, které má aktivita přiřazené jsou potřebné pro zajištění těchto požadavků.

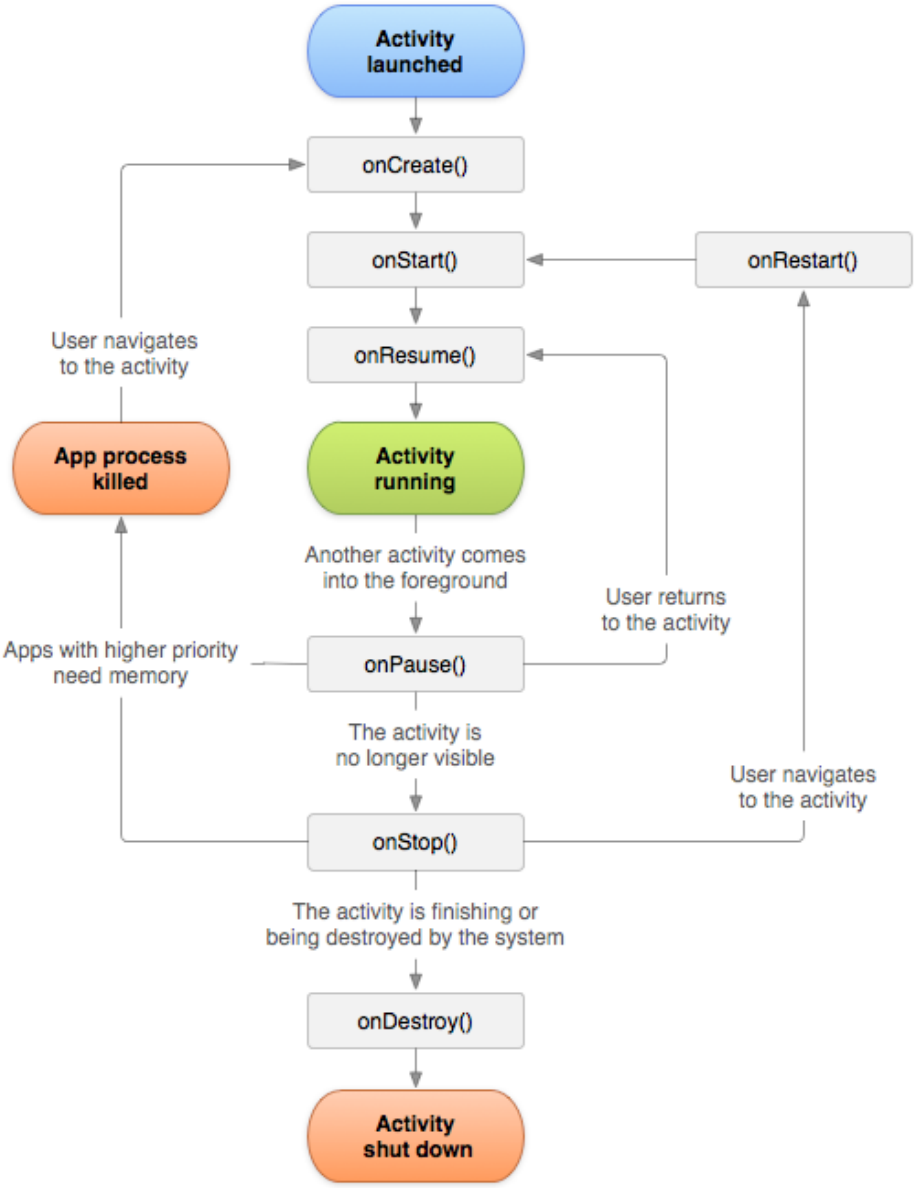
Posledním stavem je aktivita na pozadí (*Backgrounded* nebo *Stopped*). V tomto stavu jsou všechny aplikace, které nejsou v daný okamžik viditelné, ani přímo používané uživatele. Tyto aktivity si sice drží informace o stavu, mají ale nejnižší prioritu, co se týče ukončování, a mohou být kdykoliv ukončen, pokud systém potřebuje prostředky, které aktivita drží. Pokud dojde k ukončení aktivity, aktivitu je nutné znovu spustit.

Kromě samotných stavů SDK také definuje rozhraní metod pro přechod mezi jednotlivými stavy. Celý životní cyklus aktivity včetně právě metod a jak na sebe tyto metody navazují jsou znázornění na obrázku 5.2. V základu existuje 6 metod, které jsou postupně vyvolávány v průběhu života aplikace. Jedná se postupně o metody *OnCreate*, *OnStart*, *OnResume*, *OnPause*, *OnStop* a *OnDestroy*.[1)] K těmto základním existuje navíc metoda *OnRestart*. [31]

První metodou posloupnosti volání při spouštění aplikace je metoda *OnCreate*. Tato metoda slouží k inicializaci po spuštění, inicializuje se tedy především uživatelské prostředí nebo inicializace proměnných. Tato metoda také umožní pomocí parametru typu *Bundle* obnovit stav po znovuspuštění aplikace, a díky tomu je možné nastavit uložený stav právě z instance *Bundle*. Druhou metodou v pořadí při spouštění aplikace je metoda *OnStart*, která vývojáři poskytuje možnost provést specifické úkoly těsně předtím, než se aktivita stane viditelnou. Poslední v hierarchii volání je metoda *OnResume*. Tato slouží k provedení úkolů, které se musí uskutečnit ve chvíli, kdy je zbytek aktivity již inicializovaný, obvykle tyto souvisí s uživatelem nebo interakce s ním. Mezi tyto úkony lze zařadit zahájení animací, odposlech GPS aktualizací či zobrazení dialogů nebo podobného.[32]

Druhou sekvencí jsou metody volané při ukončování aplikace. První v této sekvenci je *OnPause*. Podle názvu je tato metoda volána ve chvíli, kdy systém aktivitu přepíná do pozastaveného stavu. V tuhle chvíli se mohou např. ukládat data do paměti telefonu. Další v sekvenci je metoda *OnStop*, která je volána při přechodu do stavu na pozadí. Poslední metodou v této posloupnosti *OnDestroy*. Mimo obě sekvence je ještě volána metoda *OnRestart*.[32]

[1)] V diagramu životního cyklu jsou metody pojmenovány podle konvence jazyka Java, tedy s prvním písmenem malým, v textu jsou dle konvence jazyka C#.




Obrázek 5.2 Životní cyklus aktivity[31]

Jak je vidět na obrázku 5.2, obě sekvence nemusí být volány vždy kompletně. Tedy po metoda *OnPause* nemusí být nutně volána *OnStop*, ale v případě že uživatel se k aktivitě vrátí tak se vyvolá metoda *OnResume*. Stejně tak po metodě *OnStop* může být volána funkce *OnRestart*, pokud se uživatel vrací k aktivitě, která byla ve stavu na pozadí. Po této metodě se vždy volá funkce *OnStart*. V případě, že je aktivita ukončena systémem kvůli uvolnění prostředků, dochází vždy k zavolání sekvence metod už od metody *OnCreate*.[32]

### 5.3 Services

Zatímco aktivity jsou vázané na uživatelské rozhraní a jsou aktivní pouze ve chvíli, při vývoji Android aplikací existují ještě tzv. *Services* (dále v textu jako služby). Pokud aplikace potřebuje vykonávat i mimo běh nějaké její aktivity, je nutné právě využít služby. K úkonům, které jsou vhodné pro využití služeb patří především stahování souborů, přehrávání hudby nebo i pro komunikaci mezi aplikacemi. Také kromě toho, aby úkol běžel i mimo aktivitu je druhým důvodem pro využití služeb fakt, že systém může aktivitu ukončit nebo restartovat např. při otočení displeje. Pravě služby takové úkony přežívají. [33]

Ačkoliv by se mohlo zdát, že pro tyto úkoly by mohlo stačit vytvoření vlákna, na kterém ty úkoly poběží, není tomu tak právě z důvodu práce s aktivitami z pohledu systému. U podobných úkolů aplikace vyžaduje, aby tyto úkoly neskončili, tedy aby otočení displeje nezrušilo stahování souboru a tento se nemusel celý stahovat znovu. Při rozmyslu, jestli zvolit službu nebo vlákno je potřeba vzít v potaz právě takovéto parametry a podle toho zvolit.

Úkoly, pro které se služby využívají, se dělí na dvě skupiny podle toho, jak probíhají. První skupina jsou *Long Running Task*. Jedná se o úkoly, které trvají delší dobu, běží do chvíle, než jsou explicitně zastavené a musí běžet i když je aktivita na pozadí. Mezi takové se řadí např. přehrávání hudby. Oproti tomu existují *Periodic Tasks*. Jedná se o krátké úkoly, které se periodicky opakují, např. každých 60 sekund. Do takových úkolů lze zařadit získávání dat ze senzorů zařízení, např. GPS. [33]

Z pohledu služeb jako takových existují 3 různé druhy, podle toho, jak se tyto chovají a jaké je jejich použití. První typ služby je *Foreground* služba, která vykonává činnost tak, že o ni uživatel ví pomocí notifikace systému. Druhým typem jsou *Background* služby. Tyto zpravidla nemusí být uživatelem pozorovány. *Bound* služby jsou posledním typem. Tyto služby běží do doby, pokud je k ní vázána alespoň jedna jiná komponenta v systému, jakmile je se poslední vazba uvolní, dojde k zrušení služby. [34]

Z pohledů dlouhodobých úkolů je potřebný především typ služby *Foreground*. Jak již bylo řečeno, tyto služby jsou vázané k nějaké notifikace, která musí mít alespoň nízkou prioritu (*PRIORITY_LOW*) (viz kapitola 5.6). Díky tomu je uživatel upozorněn na to že, nějaká služba běží a že něco spotřebovává jak systémové prostředky, tak i baterii. Kromě příslušející notifikace je potřeba také mít oprávnění k používání těchto služeb ve formě oprávnění *FOREGROUND_SERVICE* definované v Android Manifestu (viz. kapitola 5.7). Pokud aplikace cílí na API úrovně 29 (Android 10) a požaduje přístup k lokaci zařízení, nebo při cílení na API úrovně 30 (Android 11) požaduje přístup ke kameře či mikrofonu, je nutné nadefinovat v Android Manifestu také element *<service>*, který obsahuje kromě názvu služby také jakého je typu. [35]

Přesto s vytvářením takových služeb mohou nastat problémy, především snahou vývojářů operačního systému a jeho nadstaveb vylepšit optimalizaci baterie. Android sice pro tyto účely představil pokyny, kterých by se výrobci zařízení měli držet, někteří se ale těchto pokynů nedrží, proto je potřeba pro některé výrobce tohle ošetřit ať už cestou kódu, pokud to umožňuje API, přesměrovat uživatele na správná nastavení, nebo v nejhorším případě uživatele, když dvě předchozí cesty nejsou možné na tato nastavení uživatele nasměrovat. Tyto důsledky mohou vést k tomu, že služba bude zrušena systémem a uživatel ani aplikace se o tomto nedozví. [29]

## 5.4 Přístup k poloze zařízení

Přístup k poloze zařízení na zařízeních s Androidem lze získat pomocí 3 poskytovatelů. Nejpřesnější zdroj lokace pochází od poskytovatele *GPS Provider*, který kromě GPS používá také aGPS (*Assisted GPS*), který kombinuje data z GPS s vysílači mobilního signálu pro zpřesnění polohy. Méně přesný je zdroj *Network Provider* získavající polohu právě z vysílačů WiFi a mobilních dat, případně také z dat aGPS získaných z těchto vysílačů. Tento poskytovatel je energeticky úspornější, ale může vracet méně přesné výsledky. Poslední zdroj dat je *Passive Provider*, tento zdroj ale poskytuje pouze data, která byla vyžádána jinou aplikací nebo službou. Tento poskytovatel je vhodný, pokud aplikace nepotřebuje aktuální přesnou polohu, díky tomu je ale nejúspornější.[36]

Kromě těchto zdrojů lokace na zařízeních s *Google Mobile Service* existuje tzv. *Fused Location Provider*. Nejedná se ale o zdroj dat jako takový. Tento zdroj využívá *GPS Provider* a *Network Provider* dle vhodnosti, který z nich je vhodnější v danou chvíli a podle požadované přesnosti. Díky tomu je aplikace úspornější na baterii a může dosahovat lepších výsledků než jen při použití *GPS Provider*. Důvodem je fakt, že zatímco ve venkovních otevřených prostorách je výhodnější použít GPS, ve vnitřních prostorách nebo v blízkosti větších budov může být přesnější lokalizace pomocí WiFi nebo mobilního signálu. [36]

Pro využití polohy zařízení je potřeba přes Android Manifestu (viz kapitola 5.7) získat oprávnění. K tomu slouží buď oprávnění *ACCESS_FINE_LOCATION* nebo *ACCESS_COARSE_LOCATION*. První zmíněné slouží k využití *GPS Provider*, druhé oprávnění k využití *Network Provider*. Pokud se využívá *Passive Provider*, je nutné mít alespoň jedno z oprávnění, aby *Passive Provider* měl přístup k příslušnému zdroji. [36] Více o oprávněních v kapitole 5.5.

## 5.5 Oprávnění

Oprávnění slouží v zařízeních Android jako nástroj pro bezpečí zařízení, ale především uživatele. Zabraňuje aplikacím, aby bez vědomí a povolení uživatele přistupovali

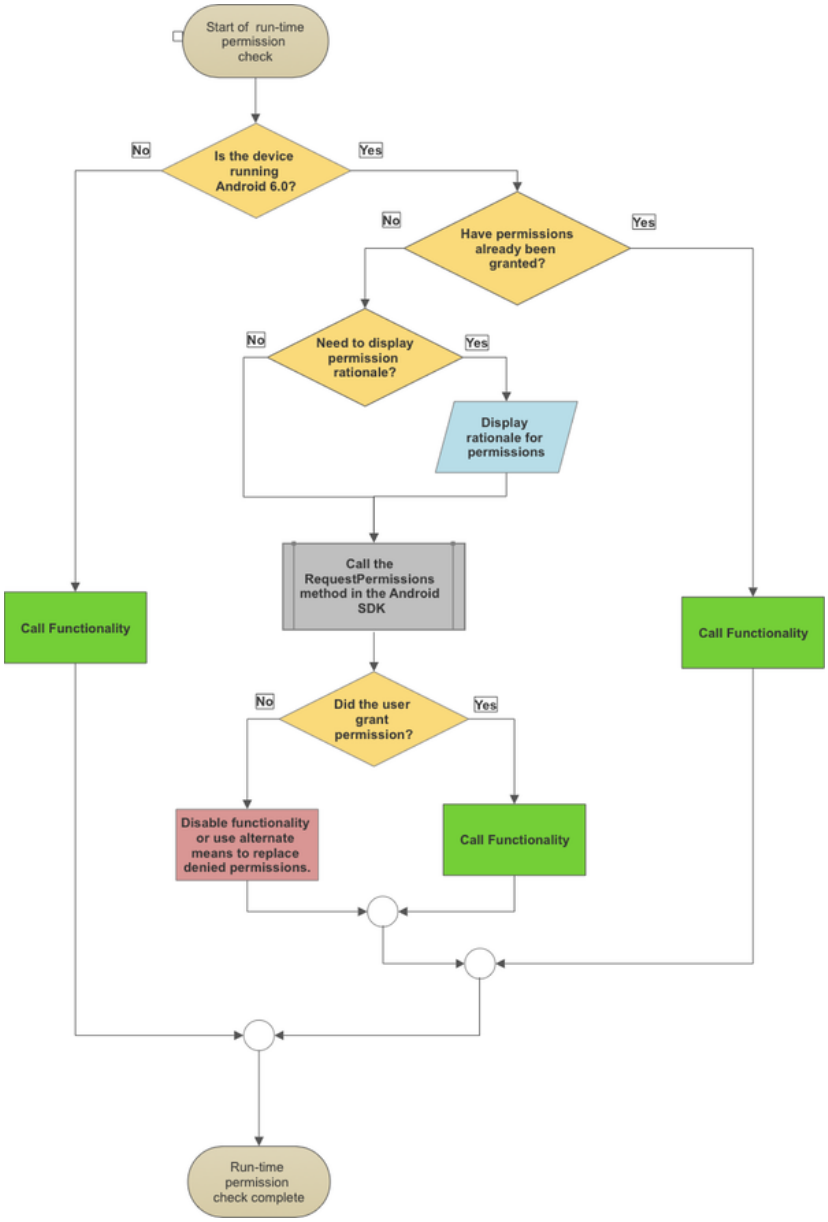
k citlivým datům (jako jsou soubory a kontakty na zařízení) či vykonávali akce (např. získávání polohy zařízení či natáčeli video pomocí kamery). Všechny požadované oprávnění jsou nadefinovány v rámci souboru Android Manifest (viz kapitola 5.7). [37]

Oprávnění se dělí na dvě hlavní kategorie. První kategorií jsou míněny oprávnění *Install-time permissions*, které jsou povoleny aplikaci ve okamžiku instalace aplikace do zařízení, tyto oprávnění nezasahují nějak výrazně do soukromí uživatele. Mezi tyto oprávnění ně patří např. přístup k internetu. Tato oprávnění se navíc dělí na dvě menší podskupiny. První podskupinou jsou tzv. *Normal permissions*, jedná se o základní množinu oprávnění. Druhou skupinou jsou oprávnění *Signature permissions*, které jsou uděleny aplikaci, pokud existuje jiná aplikace, která je podepsána stejným certifikátem a tyto oprávnění ji byly povoleny. [37]

Druhá skupina oprávnění, *Runtime permissions*, také označená jako *Dangerous permissions*, zahrnuje oprávnění, která požadují potvrzení od uživatele. Tyto oprávnění obvykle přistupují k citlivým datům nebo jiným způsobem mohou ovlivňovat soukromí či bezpečí uživatele. Proto aplikace potřebuje dostat povolení k využívání příslušných funkcionalit, jako je přístup ke kameře, kontaktům či poloze zařízení. [37]

Toto oprávnění aplikace získává 2 různými cesty podle úrovně API na cílovém zařízení. U zařízení s Androidem API úrovně 22 a nižší (Android 5.1 a starší) jsou tyto oprávnění získána při instalaci aplikace a není třeba při běhu aplikace vykonávat další kód. Naopak od API úrovně 23 (Android 6 a novější) je nutné uživatele o oprávnění požádat. K tomu existuje několik základních principů, které by měla aplikace dodržovat, pokud to lze. Žádost o oprávnění by měla být zobrazena až ve chvílí, kdy toto oprávnění skutečně vyžaduje nějaká funkcionalita, ne dříve. Žádost by neměla blokovat uživatele, měl by mít možnost tuto žádost stornovat. Ve chvíli, kdy uživatel odmítne oprávnění, měla by se aplikace degradovat na stav, kdy aplikaci bude možné používat bez příslušné funkcionality. [38]

Základní postup žádosti o oprávnění vypadá je vidět na obrázku 5.3. V prvním kroku je třeba zjistit API úroveň zařízení, pokud je menší než 23, je možné rovnou pokračovat funkcionalitou. Stejně tomu tak bude, pokud sice je úroveň API vetší, ale práva již byla udělena dříve. Pokud je potřeba oprávnění získat, je potřeba zjistit, jestli je pro dané oprávnění zobrazit zdůvodnění žádosti a toto případně zobrazit. Následně probíhá volání Android SDK s žádostí o oprávnění a kontrola, jestli uživatel oprávnění udělil. Pokud ano, může aplikace pokračovat funkcionalitou, kvůli které se oprávnění žádalo. [38]




Obrázek 5.3 Postup žádosti o oprávnění[39]

## 5.6 Notifikace

Notifikace v systému Androidu slouží k zobrazování informací uživateli. Android poskytuje dvě systémově kontrolovaná místa, ve kterých tyto zobrazuje. Pokud je notifikace poprvé zobrazena, je zobrazena ve formě ikony v notifikační oblasti. Pro zobrazení detailu notifikací slouží tzv. *Notification drawer*, který se otevře po roztažení notifikační oblasti. [40]

Notifikace jsou zobrazeny pomocí dvou rozložení. Základní rozložení je stručným zobrazením notifikace, obsahuje pouze minimální informace, tedy ikonku, titulek, zprávu a časovou známku. Od Androidu 5 se navíc toto rozložení zobrazuje i na obrazovce zamčeného telefonu. Oproti tomu rozšířené rozložení může zobrazovat více textu nebo

větší obrázky, ne pouze ikony. [40]

Pro různé způsoby zobrazení existují 3 druhy metadat, které upravují prioritu, viditelnost a kategorii. Priorita ovlivňuje, kdy a za jakých okolností se bude notifikace zobrazovat, viditelnost ovlivní, kolik obsahu notifikací se má zobrazit a kategorie, která systém informuje, jak s notifikací zacházet za určitých okolností, např. s ohledem na režim *Do Not Disturb*. [40]

Od API úrovně 26 (Android 8.0) převzala zodpovědnost za metadata notifikací Notifikační kanál. Notifikační kanál slouží jako skupina notifikací a je definovaný jednoznačným ID kanálu, zobrazitelným názvem kanálu a důležitostí kanálu. Od úrovně API 26 musí mít všechny notifikace přiřazený kanál. [40]

Z pohledu nastavení, ať už přímo nebo přes notifikační kanál, lze nastavit kromě popisků, ikon a priority také zvuky a vibrace. Z pohledu priority se rozlišuje 5 stavů priorit, s největší prioritou pro např. příchozí hovory až po nejnižší, která slouží jen pro zobrazení základních informací jako počasí. Viditelnost nastavuje to, v jaké podobě je notifikace zobrazena na zamčené obrazovce mobilu, jestli je vidět kompletně, pouze základní údaje nebo případně není vidět vůbec. [40]

## 5.7 AndroidManifest

Pro popis funkcionalit, vlastností a požadavků aplikace slouží v systému Android existuje soubor AndroidManifest. Mezi základní obsah, který soubor obsahuje je především název balíčku aplikace; zaregistrované komponenty (aktivity, služby, a další); oprávnění, které aplikace vyžaduje; informace o minimální a cílové úrovni APK úrovně a požadavky na hardware nebo software, které ovlivňují, na která zařízení lze aplikaci nainstalovat. [41]

Kořenový element Android Manifest, jehož ukázka je zobrazena ve Fragmentu kódu 1, se jmenuje `<manifest>` a obsahuje informace o balíčku, tedy jeho jméno, kód verze a jméno verze. Jméno balíčku je unikátní ID, které identifikuje aplikaci v rámci zařízení i obchodu. Při vytváření aktualizací musí zůstat název balíčku stejný, aby obchod i zařízení dokázalo přiřadit aktualizaci k aplikaci. Kód verze je číslování jednotlivých verzí balíčku, kdy zařízení nebo obchod se podle tohoto čísla rozhoduje, jestli má danou aplikaci aktualizovat. Pro aktualizaci je nutné, aby toto číslo bylo větší než aktuálně nainstalovaná verze. Jméno verze je pouze prezentovatelnou formou kódu verze. [41]

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.example.myapp"
     android:versionCode="1"
     android:versionName="1.0" >
</manifest>
```

Fragment kódu 1 Obsah kořenového elementu Android Manifestu [41]

Jedním z význačných elementů kořenového elementu je `<uses-sdk>`, který pomocí atributů `minSdkVersion` a `targetSdkVersion` určuje minimální a cílovou úroveň API aplikace. Tento element spolu s elementem `<uses-feature>`, který pomocí dvojice atributů `android:name` a `android:required` určuje hardwarové či softwarové požadavky, omezují, na které zařízení lze aplikaci nainstalovat. [41]

Další z elementů v kořenovém elementu je `<application>`, který mimo jiné přes atributy nastavuje ikonky aplikace nebo prezentovaný název aplikace. V tomto elementu se také definují komponenty aplikace, mimo jiné aktivity pomocí elementů `<activity>` a služby pomocí `<services>`. [41]

Jak už bylo zmíněno v kap. 5.5, v manifestu se také deklarují oprávnění, které aplikace bude využívat. Tyto se nastavují pomocí elementu `<uses-permissions>` s atributem `android:name`. [41]

## 5.8 Prostředky

Prostředky (*Resources*) se rozumí dodatečné soubory a statický obsah v podobě bitmapy, obrázky, texty pro uživatelské rozhraní, a další. Tyto prostředky by měli být nezávislé na kódu, pouze kódem využívány. Prostředky mohou být vytvářeny s ohledem na různé konfigurace zařízení a systému, především podle lokalizace a DPI (dots per inch, počet bodů na palec) displeje. [42]

Prostředky jsou seskupovány do několika kategorií, kdy každá z kategorií má svou složku v adresářově hierarchii aplikace. Částečný seznam těchto složek a jejich vysvětlení lze vidět v tabulce 5.2.

Tabulka 5.2 Částečný seznam kategorií prostředků [42]

| Složka | Typ prostředku |
|---|---|
| *drawable/* | bitmapy (.png, gif, atd.) nebo XML soubory obsahující vykreslitelný obsah |
| *layout/* | XML definice uživatelského rozhraní |
| *mipmap/* | soubory tzv. *launcher* ikonek, jedná se o obrázky zobrazující se jako ikonka aplikace v zařízení |
| *values/* | XML soubory obsahující definice jednoduchých typů jako textové řetězce, konstanty, barvy a další |

Všechny prostředky lze alternovat pomocí konfigurace kvalifikačních (*Qualifier*) hodnot. Tyto mohou ovlivnit kdy a za jakých okolností se některé z prostředku zobrazí s ohledem na konfiguraci. Jejich stručný přehled lze vidět v tabulce 5.3.[42]

Prostředky pro konkrétní konfiguraci se vždy dávají do složky podle jejího typu, případně doplněné o kvalifikátory. Příkladem složky pro bitmapy mohou být názvy složek *drawable-hdpi* pro bitmapy s hdpi hustotou, a *drawable-cs-hdpi* pro bitmapy s českou

lokalizací s hdpi hustotou. Pomocí těchto kvalifikátorů lze vytvořit také překlady textů do různých jazykových mutací. [42]

Tabulka 5.3 Stručný přehled kvalifikátorů [42]

| Konfigurace | Příklad hodnot | Popis |
|---|---|---|
| *Jazyk a region* | en, cs-CZ | jazyk je definovaný podle 2-znakové ISO 639-1 normy, volitelná oblast podle ISO 3166-1-alpha-2 |
| *Velikost obrazovky* | small, normal, large, xlarge | velikost obrazovky zařízení přibližně odpovídá postupně QVGA rozlišení (320x426 dp), HVGA rozlišení (320x470 dp), VGA rozlišení (480x640 dp) a rozlišení vetší (minimální hodnota je 720x960 dp). |
| *Orientace displeje* | port, land | |
| *Mód uživatelského rozhraní* | car, desk, television, appliance, watch, vrheadset | *car* - zařízení je dokováno v automobilu<br>*desk* - zařízení je dokováno na stole<br>*television* - zařízení zobrazuje obraz na televizi<br>*appliance* - zařízení nemá displej, může se jednat se o spotřebič<br>*watch* - zařízení má displej a nosí se na zápěstí<br>*vrheadset* - zařízení zobrazuje obraz ve VR |
| *Hustota displeje* | ldpi, mdpi, hdpi, xhdpi, xxhdpi, xxxhdpi, nodpi, tvdpi, anydpi, nnndpi | hodnoty ldpi až xxxhdpi postupně vyjadřují hustoty displeje odpovídajícím 120, 160, 240, 320, 480 a 640 dpi; nodpi je pro prostředky, které nechceme, aby se škálovaly s ohledem na DPI; tvdpi vyjadřuje typickou hustotu bodů televizí okolo 213 dpi; anydpi vyjadřuje prostředky nezávsile na DPI; a nnndpi vyjadřuje nestandartní hustoty vyjádřené kladným nenulovým číslem |

# II. PROJEKTOVÁ ČÁST

# 6 NÁVRH APLIKACE

Při návrhu samotné aplikace, tedy především uživatelského rozhraní bylo třeba vzít v potaz nejen současnou funkcionalitu a uživatelské rozhraní projektu Photon Hero, ale také jak rozumně zapojit funkcionalitu novou a tyto propojit se stávajícím systémem. Aplikaci je potřeba připravit nejen na zobrazování informací o závodech či závodních sériích, ale také na nové funkcionality, především zaznamenávaní a zobrazování aktivit. Další podstatné hledisko při návrhu aplikace, které bylo nutné zohlednit, byla vizualizace prostorových (GPS) dat do mapového podkladu.

Z důvodu snadné rozšiřitelnosti aplikace z pohledu nových funkcionalit bylo pro navigaci klasické menu, vysouvané z postranního okraje obrazovky, případně pomocí tlačítka v záhlaví aplikace. V rámci menu musí být obsaženy nejen zmíněné aktivity, tak také původní funkcionalita projektu Photon Hero zaměřená na závody, tedy i na zážitkové závody. Těmito závody se rozumí nejen běžecké, ale i ty v jiných sportovních kategoriích, které jsou určeny pro širokou veřejnost. Tyto závody navíc mohou být řazeny do závodních sérii, které shlukují závody stejné skupiny a ve stejném kalendářním roce, případně může mít uživatel závody, kterých se chce účastnit či své oblíbené závody. Další funkcionalitou v rámci webové stránky projektu jsou i uživatelské profily. V rámci návrhu menu je tedy podstatné zvážit zobrazení profilu, ale také možnosti přihlášení či odhlášení.

Z pohledu závodu jako takového jsou podstatné především informace jako je název, místo, kategorie sportu a čas konání závodu. Tyto byly označeny při vývoji jako hlavní údaje a při zobrazení informací o závodě jsou tyto uvedeny prioritně. Jako vedlejší informace byl označen především popis závodu. Jako doplňkové informace jsou zvoleny webová stránka a k jaké závodní sérii závod náleží, případně mapa trasy závodu, pokud taková existuje. Z těchto důvodů byl zvolen obsah položky závodu v seznamu zobrazením obsahu hlavních položek spolu se zkráceným popiskem závodu, zatímco v detailu jsou uvedeny všechny informace, včetně mapy trasy závodu.

Kromě toho detail závodu musí umožnit navigovat uživatele na start závodu, případně polohu zobrazit, pokud v databázi existují příslušné informace o této poloze. Kromě toho lze využít také vlastnosti platformy ve formě notifikací a umožnit uživateli zvolit si notifikaci na blížící se závod v čase, který si uživatel zvolí.

K seznamu závodů je navíc potřeba vytvořit filtr, který umožní zobrazit jen závody podle zadaných parametrů. Kromě základních parametrů, jako je maska názvu či datum konání, je vhodné umožnit uživateli filtrovat závody podle aktuální polohy zařízení, na kterém uživatel vyhledává závody. Pro tyto účely byla rozšířena již existující tabulka pro závody o nový sloupec typu `geography`. Typ `geography` je jedním z typů pro prostorová data Microsoft SQL Serveru. Pomocí prostorových dat není nutné im-

plementovat vlastní algoritmy pro řešení vzdáleností mezi body, ale je možné využít vlastností jazyka SQL a tím dotazy optimalizovat i pro zmíněny scénář.

Z pohledu závodních sérií jsou informace stručné, jsou podstatné pouze název série, její ročník a seznam závodů, které série obsahuje. Z toho důvodu může být i filtr velmi stručný a lze filtrovat pouze přes zadání roku, pro který chci uživatel hledat série.

Nejpodstatnější funkcionalitou, kterou je v rámci návrhu rozmyslet, se týká aktivit. Zde bylo potřeba zvážit několik oblastí. První, nejpodstatnější, je způsob ukládání aktivit, kdy je na výběr několik formátů či způsobů. První variantou je ukládat data přímo do SQL databáze v podobě prostorových dat. Zde by ale mohl být problém s množstvím dat, které by se ukládali, především počtem bodů na aktivitu. Proto se zvolilo ukládání dat do souboru, mezi zvažovanými formáty byly GPX a TCX, oba založené na XML. Z důvodů snadné rozšiřitelnosti byl zvolen GPX, u kterého lze nadefinovat vlastní rozšíření formátu. Druhým důvodem bylo využití tohoto formátu v rámci projektu pro zobrazení trasy závodu.

Formát GPX slouží pro ukládání a výměnu GPS dat mezi aplikacemi a službami. Tento formát obsahuje struktury pro ukládání význačných bodů (*waypoints*), drah (*tracks*) a tras (*routes*). Zatímco seznam význačných bodů je pouze seznamem bodů, dráhy a trasy jsou již složitější struktury. Trasy jsou seznam bodů, které znázorňují definovanou cestu k cíli, a kromě základních parametrů jako je název či pořadí trasy v rámci souboru, tak obsahují především právě seznam bodů. Dráha (*track* na druhou stranu sice obsahují podobné parametry jako trasy, ale jedná se o zaznamenanou trasu z aplikace či zařízení. Tato dráha ale nemusí být návazná, může obsahovat tzv. segmenty, které od sebe mohou být odsazené jak časově, tak polohou. Teprve právě tyto segmenty obsahují samotné seznamy bodů. Dráha je tedy seznamem segmentů. Z pohledu samotných bodů jsou významné atributy bodu zeměpisná šířka (*latitude*) a délka (*longitude*) spolu elementy pro nadmořskou výšku a časové razítko příslušného bodu. Kromě toho lze k většině struktur v rámci formátu GPX nadefinovat vlastní rozšíření. Výsledný soubor následně bude nahráván na Azure Storage.

Další část návrhu se zabývala prezentací získaných dat o aktivitě. Jako hlavní data byla vybrána název aktivity, kategorie sportu a zahájení aktivity. K tomu jsou doplněny vedlejší data v podobě vzdálenosti, času trvání a průměrného tempa. Tyto údaje je potřeba zobrazit v rámci seznamu, v detailu jsou tyto údaje doplněny o další údaje, vypočtené z příslušného GPX, jedná se například nejpomalejší tempo či maximální nadmořskou výšku v rámci aktivity. Tyto data budou získány z databáze, kde budou uloženy při vkládání nové aktivity. O analýzu se bude starat knihovna vytvořená pro tyto účely skrze webovou službu.

Posledním podstatným rozhodnutím s ohledem na aktivity je rozhodnutí, jak se

aktivita bude zahajovat, jak bude probíhat a jak se bude ukládat, tedy samotné zaznamenávání polohy uživatele. Všechny potřebné systémová nastavení, jako je zapnutá lokalizace zařízení či udělená oprávnění budou kontrolována na prvním kroku průvodce, spolu s vybráním aktivity. Druhý krok bude mít na starost zaznamenávání polohy jako takové. Kromě toho se v rámci tohoto kroku budou zobrazovat průběžné výsledky jako trvání aktivity, aktuální tempo nebo aktuální vzdálenost. K tomu musí jít aktivitu pozastavit s možností znovuspuštění. Poslední krok před uložením umožní uživateli zvolit název a viditelnost aktivity vůči ostatním uživatelům. Výsledkem těchto kroků je nový záznam aktivity v databázi obsahující analýzu dat spolu s příslušným souborem GPX v Azure Storage.

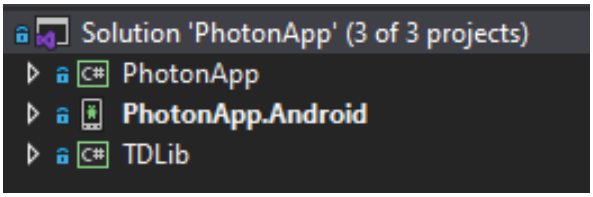
U všech funkcionalit je potřeba rozmyslet, které z těchto budou zobrazitelné pro jen pro přihlášené uživatele. Kromě výběru samotných funkcionalit, je potřeba také umožnit uživateli samotné přihlášení či odhlášení, ale k tomu zobrazit také jeho informace, jako jméno či oblíbené sporty. K viditelnosti příkazů jsou jako funkcionality pro nepřihlášené uživatele vybrány pouze seznam závodů a závodních sérii, ostatní funkcionality jsou již vázané přímo na samotného uživatele, proto jsou pouze pro přihlášené uživatele, včetně přidávání nové aktivity.

Cílovým stavem je distribuce produkční i testovacích verzí s využitím obchodu Google Play, především kanály pro testování v podobě Uzavřeného testování a Otevřeného testování.

# 7 IMPLEMENTACE APLIKACE

Při implementaci aplikace bylo nutné zohlednit nejen požadavky na aplikaci, ale také již existující systém projektu Photon Hero. Současná architektura projektu je postavená na technologiích Microsoftu. Webová stránka projektu je napsána pomocí technologie ASP.NET MVC a k běhu využívá prostředí Azure App Service. Jako databázový server je využíván Microsoft SQL Server, ten stejně jako webová stránka běží na cloudové platformě Azure. Pro ukládání souborů, které nejsou přímou součástí projektu, se využívá Azure Cloud Storage, opět platformy Azure.

Z tohoto důvodu byly pro další vývoj tohoto projektu vybrány technologie využívající .NET, tedy pro mobilní vývoj Xamarin. Podstatnějším rozhodnutím pohledu vývoje bylo, zdali pro vývoj zvolit přímo nativní implementace vývoje pro Android, tedy Xamarin.Android, nebo multiplatformní vývoj v podobě Xamarin.Forms. Jak bylo řečeno již v kapitole 4, přestože je Xamarin.Android nativní implementací Android SDK, lze i v této formě využít znovupoužitelný kód, ale pouze těch částí, které tvoří pouze datovou či aplikační vrstvu. Přestože je v rámci práce a blízké budoucnosti v rámci projektu jako takového plánována pouze Android aplikace, do budoucna je v plánu i její iOS alternativa. Především tedy větší znovupoužitelnost byla hlavním důvodem pro zvolení Xamarin.Forms pro vývoj dané aplikace. Celá aplikace je díky tomu postavená na .NET Standard pro projekt se sdíleným kódem s využitím Xamarin.Forms a na Monu, resp. jeho implementaci Xamarin.Android, pro nativní aplikaci pro Android. V případě rozšíření na iOS bude aplikace využívat implementaci Mona Xamarin.iOS. Tento výběr navíc umožňuje také přejít na novou platformu .NET 5 a do budoucna také nahradit Xamarin.Forms novou platformou MAUI pro uživatelské rozhraní. Výslednou strukturu projektů aplikace je možné vidět na obrázku 7.1




Obrázek 7.1 Struktura projektů aplikace

Kromě výběru platformy bylo třeba zvolit i způsob napojení na již existující databázi, postavenou, jak již bylo řečeno, na Microsoft SQL Serveru s využitím cloudové platformy Azure. Pro tyto účely byla vytvořena webová služba. Tato služba byla kvůli kompatibilitě s webovou stránkou projektu PhotonHero vytvořena na platformě .NET Framework. Z důvodu znalosti a zkušeností nejen autora práce, ale také ostatních členů podílejících se na vývoji celého projektu, byla pro webovou službu vybrána technologie WCF (Windows Communication Foundation).

### 7.1 Webová služba pro napojení aplikace k databázi

Jak již bylo řečeno, webová služba, kterou bude aplikace využívat k propojení s databází, je vytvořena za pomoci technologie WCF. Stejně jako ostatní použité technologie byla tato technologie vyvíjena Microsoftem pro architektury orientované na služby (SOA, *Service Oriented Architecture*). [43]

Základními pojmy WCF služby jsou tzv. zprávy (*Messages*) a koncové body *Endpoints*. Message je v pohledu WCF komunikační jednotkou, pomocí které komunikuje klient služby se samotnou WCF službou. Tyto zprávy obsahují hlavičku a data, která jsem přenášena či vyměňovaná. Koncovými body se rozumí trojice, která danou službu popisuje, tedy kde může klient posílat data a odkud je může přijímat. První částí koncového bodu je adresa služby, která specifikuje, kam se budou zprávy zasílat, a je specifikována jako identifikátor URI (*Uniform Resource Identifier*). Druhou částí koncového bodu je tzv. *Binding*, který určuje, jakým způsobem se přenáší data mezi klientem a službou, lze použít např. protokoly HTTPS nebo TCP. Poslední část, kontrakt (*Contract*), určuje, jakou funkcionalitu a operace umožňuje, jedná se tedy o rozhraní tohoto koncového bodu. [43]

Kontraktů existuje v rámci WCF služby více. Hlavním kontraktem je tzv. *Service contract*, a sjednocuje práce jednotlivé operace do jednoho funkčního celku, s daným *namespace*. Z pohledu implementace se jedná o *interface*, které je označené atributem `ServiceContract`. Pro označení operací v *Service Contract* slouží pojmenování *Operation contract*. Operační kontrakt určuje jejich parametry a návratové hodnoty. Pro označení metod, které jsou součástí *interface* a mají být součástí *Service Contract* lze využít atribut `OperationContract`. Operaci si lze představit jako přijetí jedné zprávy službou od klienta a odeslání druhé zprávy ze služby ke klientovi jako odpověď. Každá operace navíc může nadefinované také tzv. *Fault Contracty*. Tyto kontrakty k jednotlivým operacím asociují pomocí atributu `FaultContract`, jaké chybové stavy může operace vracet, a tyto jsou potom na straně klienta vráceny jako výjimka. Operace může těchto kontraktů mít více, ale nemusí mít žádné. Popis datových typů, které používá služba, resp. její operace, jsou označeny jako datové kontrakty (*Data contracts*), resp. pomocí atributu *DataContract*. [43]

Pro propojení aplikace se službou proto byl vytvořen *Service Contract* v rámci projektu *PhotonDataService*, obsahujících několik operačních kontraktů. Tyto slouží k získávání dat z databáze, pro službu. Tento *Service Contract* je ukázán ve fragmentu kódu 2.

```
[ServiceContract]
public interface IDataService
{
    [OperationContract]
    [FaultContract(typeof(UserFault))]
    User LoginUser(string guid, string email, string
       password);
    [OperationContract]
    [FaultContract(typeof(UserFault))]
    User GetUser(string guid, string email, string
       securityStamp);
    [OperationContract]
    Race GetRace(string guid, int id);
    [OperationContract]
    List<Race> GetRaceList(string guid, RaceFilterModel
       filter);
    [OperationContract]
    RaceSeries GetRaceSeries(string guid, int id);
    [OperationContract]
    List<RaceSeries> GetRaceSeriesList(string guid,
       RaceSeriesFilter filter);
    [OperationContract]
    List<short> GetRaceSeriesYears(string guid);
    [OperationContract]
    List<Workout> GetWorkoutList(string guid, WorkoutFilter
       filter);
    [OperationContract]
    Workout GetWorkout(string guid, int id);
    [Obsolete]
    [OperationContract]
    [FaultContract(typeof(WorkoutFault))]
    void PostWorkoutTraining(string guid,
                             string name,
                             string filePathUrl,
                             WORKOUT_ACCESSIBILITY
                                accessibility,
                             SportCategories activitySport,
                             string userId);
    [OperationContract]
    [FaultContract(typeof(WorkoutFault))]
    int PostWorkout(string guid,
                    string name,
                    string filePathUrl,
                    WORKOUT_ACCESSIBILITY accessibility,
                    SportCategories activitySport,
                    string userId);
    [OperationContract]
    [FaultContract(typeof(WorkoutFault))]
    int PostWorkoutWithoutGps(string guid,
                              string name,
                              WORKOUT_ACCESSIBILITY
                                 accessibility,
                              SportCategories activitySport,
                              string userId,
                              DateTime dateStart,
                              int duration,
                              int? distance);
}
```

Fragment kódu 2 *Service Contrakt* pro propojení aplikace s databází

Všechny operace jsou spojeny společným parametrem jménem `guid`, který obsahuje předem nadefinovanou hodnotu typu Univerzálního unikátního klíče (GUID - *Globally Unique Identifier*), který slouží k ověření, že se službou komunikuje známá aplikace. V případě, že tento klíč nesouhlasí, služba danému požadavku odpoví vyhozením výjimky.

Jako první skupinu lze označit dvojici operace `LoginUser` a `GetUser`. Tyto operace slouží k přihlášeni uživatele, resp. opětovnému načtení jeho dat. Jako klíčové je u obou operací brán email přihlašovaného uživatele, podle kterého se uživatele vyhledává v databázi a k tomu se posílá buď heslo v případě `LoginUser` nebo bezpečnostní razítko, se kterými je uživatel autorizován. Obě tyto operace mají asociován *Fault contract*, v tomto případě typu `UserFault`, který pomocí výčtu značí, z jakého důvodu se nepovedlo uživatele autentizovat či autorizovat. Mezi důvody se řadí především špatné zadané údaje, již nevalidní kontrolní bezpečnostní razítko, ale také, že uživatel nepotvrdil svou registraci. Na jednotlivé důvody pak správně musí reagovat klient. Obě operace jako návratovou hodnotu vrací datový kontrakt typu `User`, který obsahuje právě informace u přihlašovaném uživateli. Tento datový kontrakt je zobrazen v ukázce kódu 3.

```
[DataContract]
public class User
{
    [DataMember]
    public string Id { get; set; }
    [DataMember]
    public string Email { get; set; }
    [DataMember]
    public string SecurityStamp { get; set; }
    [DataMember]
    public string UserName { get; set; }
    [DataMember]
    public string UserNameApp { get; set; }
    [DataMember]
    public string PhoneNumber { get; set; }
    [DataMember]
    public string ProfileImg { get; set; }
    [DataMember]
    public string Location { get; set; }
    [DataMember]
    public string Description { get; set; }
    [DataMember]
    public USER_FLAGS Flags { get; set; }
    [DataMember]
    public List<SportCategories> FavoriteSports { get; set; }
}
```

Fragment kódu 3 Ukázka datového kontraktu pro uživatele

Další dvě dvojice operací se týkají závodů, resp závodních sérií. Příslušné operace vrací na základě vstupních parametrů datové kontrakty typu `Race`, resp. `RaceSeries`. Parametry operací, které slouží k získání detailu, je pouze identifikační číslo (ID), podle kterého se vyhledává v databázi. V případě operací, které vrací seznam příslušných objektů, jako vstupní parametry slouží příslušné datové kontrakty pro filtry. Kromě klasických filtrů, jako je jméno, oba filtry obsahují parametry pro *stránkování*. Klient si zadá, kolik objektů chce vrátit, a díky tomu také kolik objektů má přeskočit. Pro závodní série navíc existuje také operace pro získání ročníků, které jsou v databázi zadané. Tento výsledek slouží k správnému filtrování závodních sérií právě podle ročníků.

Obdobně v páru fungují operace `GetWorkout` a `GetWorkoutList`, pouze jejich ná-

vratovou hodnotou je jiný datový kontrakt, v tomto případě typu `Workout`. K těmto operacím existují operace `PostWorkout` spolu s operací `PostWorkoutWithoutGps`. Tyto operace slouží k nahrávání uživatelských aktivit právě z aplikace.

Obě funkce mají na vstupu základní údaje jako ID uživatele, zvolený název aktivity, kategorii sportu či viditelnost, jakou si uživatel u této aktivity uživatel zvolil. Viditelnost je definována výčtem `WORKOUT_ACCESSIBILITY` s hodnotami pro aktivity, které uvidí jen autor aktivity (privátní aktivity), aktivity pro přátele, a aktivity viditelné všem (veřejné).

V případě `PostWorkout` se jedná o operaci, která slouží k nahrání dat s GPS daty, reprezentující zaznamenanou trasu, včetně časových známek jednotlivých bodu. Jedním ze vstupů operace je adresa k souboru uloženého na Azure Storage, obsahující tato data, konkrétně ve jednom z podporovaných formátů. Tato data jsou následně službou analyzovány, a výsledky analýzy následně uloženy do databáze. Mezi výsledky analýzy je vzdálenost, průměrné tempo a další hodnoty. Druhou variantou je `PostWorkoutWithoutGps`, která se stará o ukládání aktivit, u kterých je nevhodné zaznamenávat polohu souřadnic. Jedná se např. o aktivity typu běh na běžeckém pásu nebo tenis. V případě těchto aktivit se na vstupu očekává pouze celkový čas aktivity spolu se vzdáleností. Pomocí těchto jsou ve službě vypočítány hodnoty pro průměrné tempo a rychlost. Výstupem těchto operací z pohledu klienta je identifikátor nové aktivity. Kromě toho může ukládání aktivit také vracet chybové stavy, které jsou definovány výčtem `WorkoutFaultReason`, který má hodnoty pro chyby při ukládání dat, analýze dat, nebo chybným vygenerováním náhledu mapy.

Služba pro připojení k databázi využívá objektově relační mapování (ORM) framework Entity Framework (EF). Ten se stará jak o samostatné připojení k databázi, tak i o mapování SQL tabulek na datové třídy služby. Pro dotazování tabulek využívá EF třídy dědící třídu `DbContext`, která obsahuje kolekce datových tříd odpovídající databázové struktuře, resp. jejím tabulkám. Pomocí této třídy lze tvořit dotazy, které následně vytvoří dotazy na připojené databázi. Tyto dotazy jsou tvořeny pomocí dotazovacího jazyka LINQ (*Language Integrated Query*), který je součástí .NET platformy a jsou transformovány pomocí převodníku *LINQ to SQL* právě do jazyka SQL.

Pro získávání konkrétních položek, především podle identifikačního čísla, se využívá funkce `FirstOrDefault`, která vyhledá první položku, a jako parametr může mít tzv. lambda funkci, která definuje případné vyhledávací podmínky pro vyhledání. Pokud nenajde žádnou položku odpovídající kritériu nebo je seznam prázdny, funkce vrací `null`. Tato funkce se transformuje do SQL konstrukce `TOP 1`. K filtrování položek lze využívat další funkci LINQ využívající Lamda funkce, a to `Where`. Výstupem této funkce je kolekce odpovídající daným kriteriím. Transformace převede tuto funkci do

konstrukce `WHERE`. Pro řazení položek lze využít funkci *OrderBy* pro řazení vzestupně, resp. *OrderByDesc* pro sestupné řazení.

Kód 4 zobrazuje zjednodušený příklad využití jazyka LINQ pro vytvoření dotazu nad kolekcí, v tomto případě se jedná o kolekci typu `DbSet` obsahující data o závodních sérii. Při vytváření dotazu se nejprve vyfiltrují záznamy podle daného roku zadaného ve filtru pomocí funkce `Where` se vstupní lambda funkcí `rs => rs.Year == filter.Years`. Následně jsou tyto funkce seřazeny vzestupně podle jména závodních sérií. Následně pomocí funkcí `Skip` přeskočí počet záznamů podle stránkování a vezme následujících 5 položek.

```
private static IQueryable<RaceSeries>
   GetFilterQuery(RaceDbContext db, RaceSeriesFilter filter)
{
    IQueryable<RaceSeries> query = db.RaceSeriesSet;
    if (filter.Years > 0)
    {
        query = query.Where(rs => rs.Year == filter.Years);
    }
    var skipRows = 0;
    if (filter.Page > 0)
    {
        skipRows = filter.Page * 5;
    }
    query = query.OrderBy(rs => rs.Name);
    query = query.Skip(skipRows).Take(5);
    return query;
}
```

Fragment kódu 4 Ukázka využití LINQ pro filtrování kolekcí

### 7.2 Cloudové úložiště Azure Blob storage

Jak již bylo řečeno dříve, větší soubory jsou v rámci projektu ukládány do cloudového úložiště Azure Storage. Pro ukládání dat do Azure Storage existuje více způsobů ukládání dat, mezi nimi např. *Azure Tables* sloužící jako úložiště NoSQL databází. V rámci projektu se ovšem využívá úložiště typu *Azure Blobs*, které slouží k ukládání textových a binárních dat. Toto úložiště lze škálovat podle případné potřeby. V rámci *Blob Storage* existují 3 druhy zdrojů, které jsou postupně propojené. Na nejvyšší úrovní se nachází tzv. *storage account*, který slouží jako unikátní pracovní prostor. Každý tento prostor může obsahovat tzv. kontejnery, které obsahují jednotlivé bloby a fungují obdobně jako složky v souborovém systému. Kontejnery potom obsahují samotné bloby, které již obsahují konkrétní data. Všechny 3 úrovně lze adresovat i za pomoci URI ve formátu *http://*mystorageaccount*.blob.core.windows.net/*mycontainer*/*myblob**. [44]

### 7.3 Projekt společné části aplikace

Základem aplikace je třída `App`, která je potomkem třídy `Application`. Tato třída se stará o základní inicializaci celého projektu, např. nastavení výchozí stránky. V rámci

implementace aplikace slouží tato třída také k vytvoření kontejneru pro vkládání závislostí (*Dependency Injection*) a k držení informací o přihlášeném uživateli. Třída mimo jiné přepisuje metodu `OnStart`, která se stará právě o přihlašování uživatele. Aby se uživatel pokaždé nemusel přihlašovat, důležité informace o uživateli jsou ukládány do zabezpečené paměti aplikace, a při spuštění aplikace jsou právě v metodě `OnStart` tyto informace získány a ověřeny vůči webové službě.

Kromě souboru s příponou *.cs* obsahující část napsanou v jazyce C#, existuje také *XAML* verze této třídy, která obsahuje globální definice prostředků pro *XAML*, tím se rozumí konstanty, styly, ale také konvertory.

Hlavní stránkou aplikace je třída `MainPage`. Tato třída implementuje třídu `Shell`, která umožňuje na jednom místě definovat vizuální hierarchii aplikace, ale také umožňuje s využitím URI navigovat na definované stránky v aplikaci. `Shell` umožňuje vizualizovat navigaci několika způsoby pomocí tzv. *Flyout* a *Tabs*. *Flyout* slouží k vytváření menu, které se vysunuje ze strany zařízení, případně po stisku příslušného tlačítka. Oproti tomu *Tabs*, jak už název vypovídá, využívá pro zobrazení navigace spodní část displeje, kde zobrazuje jednotlivé záložky sloužící k přepínání obsahu. Zatímco bez využití `Shell` je možné v základu využit jen jeden z přístupů, v případě potřeby obou přístupů je nutné si navigaci mezi jednotlivými přístupy implementovat vlastní. `Shell` toto usnadňuje, a umožňuje kombinaci těchto přístupů. V rámci aplikace ale byl zvolen přístup přes vysouvací menu, proto se dále práce věnuje jen části *Flyout*.

Stejně jako třída `App`, tak i `MainPage` je složena jak z kódu na pozadí (*code-behind*), tak i z XAML části. `Shell` je definován v této části pomocí elementu `<Shell>`, a může obsahovat právě definice *Flyout* menu. To se může skládat z několika částí, konkrétně se jedná o *Header*, položky *Flyout*, položky *Menu* a případně *Footer*. Hlavička (*Header*) v menu lze deklarovat pomocí elementu `<Shell.FlyoutHeader>`, kdy do tohoto elementu lze vkládat libovolné prvky. Oproti hlavičce jsou prvky pro položky *Flyout* výrazně omezeny. Tyto jsou definovány pomocí elementu `<FlyoutItem>`, a mohou jen elementy pro záložky elementem `<Tab>` nebo samotný obsah pomocí elementu `<ShellContent>`. `<ShellContent>` může být i potomkem elementu `<Tab>`.

Zjednodušeným příkladem definice elementu `Shell` včetně vnořených elementů pro hlavičku a *Flyout* v XAMLu je zobrazen v ukázce 5. U elementu `<FlyoutItem>` s atributem `FlyoutDisplayOptions=‘‘AsMultipleItems‘‘` je demonstrováno, jak více položek položek vizuálně v menu seskupit více položek menu k sobě. Tyto dva obsahy budou zobrazeny i jako položky záložek, pokud se uživatel nachází na jedné ze stránek. Tomuto se dá zabránit případně pomocí atributu `Shell.TabBarIsVisible="False"` buď obecně pro všechny stránky v elementu `<Shell>` nebo na konkrétní stránce. Na obrázku 7.2 je zobrazeno menu pro přihlášeného uživatele vytvořené pomocí `Shell`, kdy lze v záhlaví

vidět informace o přihlášeném uživateli. U položek je navíc vidět seskupení více položek vizuálně k sobě.

```
<Shell    >
    <Shell.FlyoutHeader>
        <StackLayout>
                <Button Text="Login" />
        </StackLayout>
    </Shell.FlyoutHeader>
    <FlyoutItem Title="Home">
        <ShellContent ContentTemplate="..."
                      Route="Home"/>
    </FlyoutItem>
    <FlyoutItem FlyoutDisplayOptions="AsMultipleItems">
        <ShellContent Title="Activity List"
                      Route="ActivityList"
                      ContentTemplate="..." />
        <ShellContent Title="Activity Stats"
                      Route="ActivityStats"
                      ContentTemplate="..." />
    </FlyoutItem>
</Shell>
```

Fragment kódu 5 Ukázka definice Shell pomocí Shell

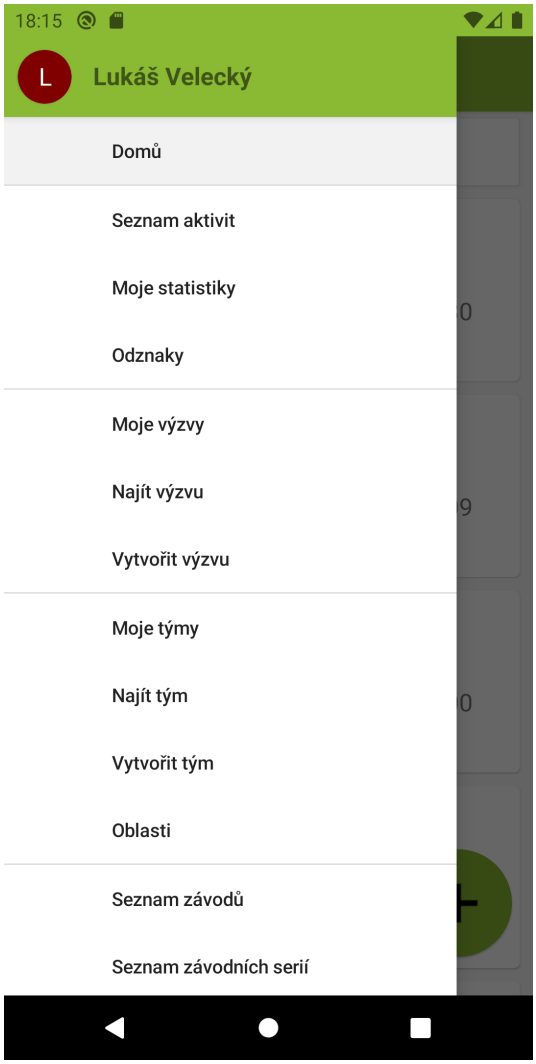
Kromě definice menu lze v ukázce vidět definice cest (*Route*) používaných pro navigování v rámci *Shell*. Navigace probíhá pomocí funkce `GoToAsync` třídy `Shell`, do které se posílá název cesty (*Route*), na kterou se má navigovat. Kromě definice v XAMLu, lze cesty definovat i v kódu na pozadí, tohoto se využívá především pro situace, kdy je potřeba registrovat cesty k stránkám, které nejsou vizuální součástí *Flyout*. Toto se děje pomocí funkce `RegisterRoute` třídy `Routing`. Hlavička menu je využita pro zobrazení informací o uživateli, případně akce pro přihlášení.

V projektu je využito vkládání závislostí (*Dependency Injection*). Využívá se tzv. *Construction Injection*, tedy parametry jsou vkládány v konstruktoru. Tyto závislosti jsou registrované do kontejneru, který je přístupný přes instanci třídy `App`. Vkládání závislostí se využívá např. pro *ViewModely* nebo *Commandy* (viz. kapitola 7.3.1)

### 7.3.1 Vytváření stránek

Jako návrhový vzor pro tvorbu stránek jednotlivých stránek byl zvolen *Model-View-ViewModel* (MVVM), který je v technologii Xamarin podpořený. Jak už vypovídá název, při tomto vzoru je aplikace rozdělena na 3 skupiny. První skupinou jsou Modely, které reprezentují třídy obsahující samotné data. Tyto třídy jsou následně využity ve třídách reprezentující *ViewModel*, který působí jako prostředník mezi *View* a Modely. Návic přidává potřebnou logiku, kterou je potřeba pro správné zobrazení modelu ve *View*. *View* slouží k samotné prezentaci potřebných dat.

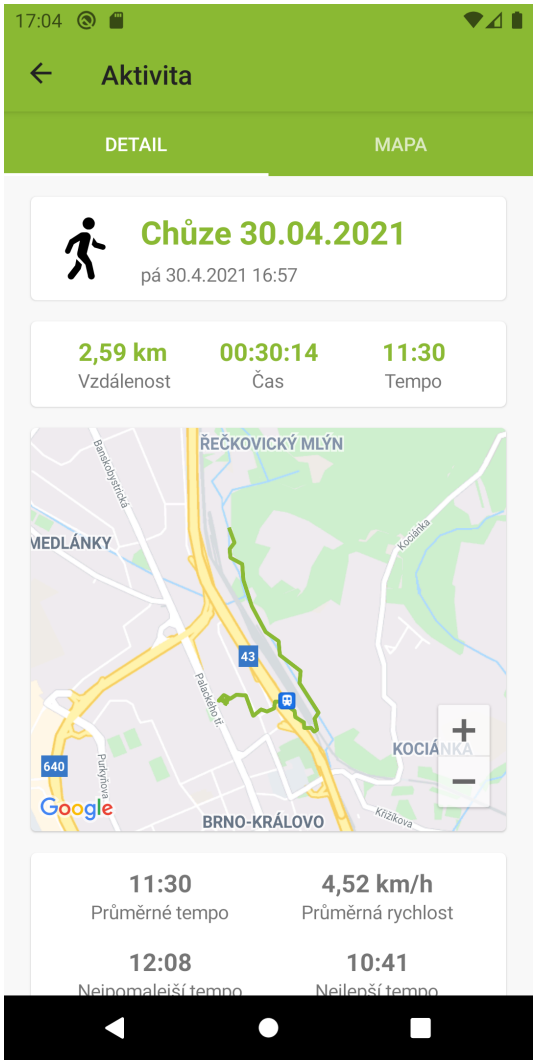
*View* lze definovat dvěma způsoby, buď pomocí vybraného programovacího jazyka, nebo pomocí Xamarin dialektu jazyka XAML. Xamarin umožňuje definovat stránku pomoci více druhů třídy pro stránky. Jelikož se o základ navigace v rámci navigace stará dříve zmiňovaný *Shell*, stránky jsou o tuto logiku zjednodušené. Proto se v apli-




Obrázek 7.2 Výsledný vzhled menu aplikace

kace primárně používá jako základ stránek typ `ContentPage`. Jedná se o nejjednodušší variantu stránek, která neobsahuje složitější logiku a v podstatě slouží jen jako kontejner pro jednotlivé prvky na stránce. Pro některé stránky, které potřebují složitější strukturování obsahu byl jako základ zvolen `TabbedPage`, který pomocí záložek umožňuje přepínání obsahu stránky. Tohoto by sice šlo docílit i pomocí *Shell*, ale protože toto přepínání je mimo navigaci, kterou *Shell* umožňuje, byla zvolena tato varianta. Toto lze vidět na obrázcích 7.3 a 7.4 je vidět výsledná podoba detailu aktivity s využitím právě záložek, kdy jedna záložka slouží pro samotné detaily a druhá pro detail mapy.

Mezi jeden ze základních prvků, který je v aplikaci využit, patří `ListView`, který umožňuje zobrazování kolekce. V aplikaci je větší množství míst, u kterých je tento prvek využit a probíhá průběžné načítání nových položek. Z tohoto důvodu vzniklo rozšíření chování (`Behavior`) `InfiniteScroll`, který umožní vykonat definovaný pří-



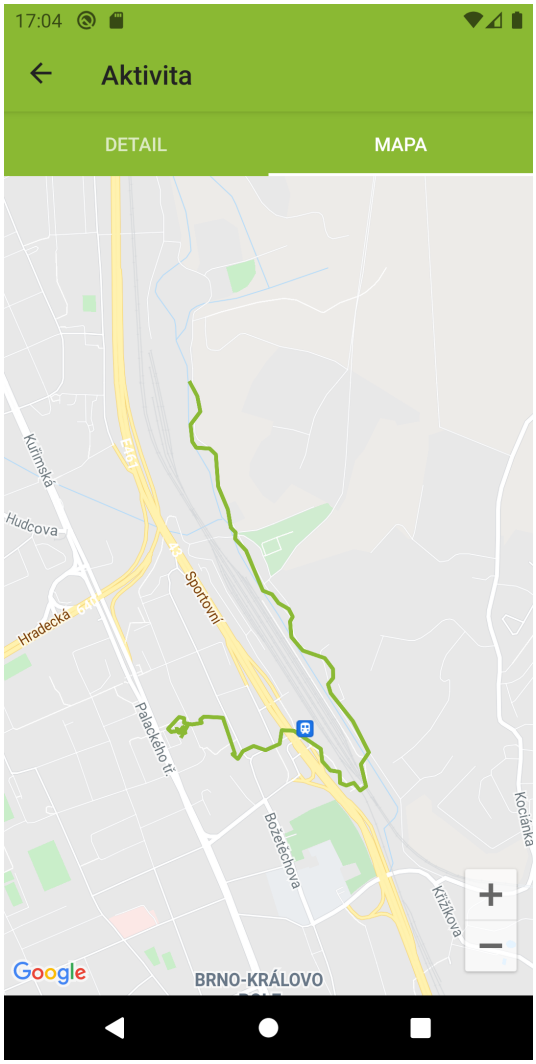

Obrázek 7.3 Výsledný vzhled detailu aktivity pomocí záložek

kaz pokaždé, kdy se dostane do viditelné oblasti poslední aktuálně zobrazená položka seznamu. Využití této třídy je vidět v ukázce 6. Kromě toho nelze na základní `ListView` připojit příkaz pomocí *Binding*, proto pro projekt byl vytvořen potomek `CustomListView`, který toto umožňuje. Výsledné použití tohoto prvku lze vidět na obrázku 7.5, který zobrazuje seznam aktivit.

```
<ListView.Behaviors>
    <listView:InfiniteScroll Command="{Binding ...}" />
</ListView.Behaviors>
```

Fragment kódu 6 Ukázka zapojení `InfiniteScroll`

Kromě samotných stránek existuje v aplikaci několik `ResourceDictionary`, které slouží k definování prostředků. Mezi tyto patří především barvy a styly, které jsou použity napříč aplikací pro zachování shodného vzhledu. K tomu jsou navíc definovaný tzv. *Template* a *TemplateSelectory*. Tyto se používají např. pro zobrazování různých objektů v `ListView`. Tohoto lze využít ve chvíli, kdy potřebujeme v seznamu zobra-



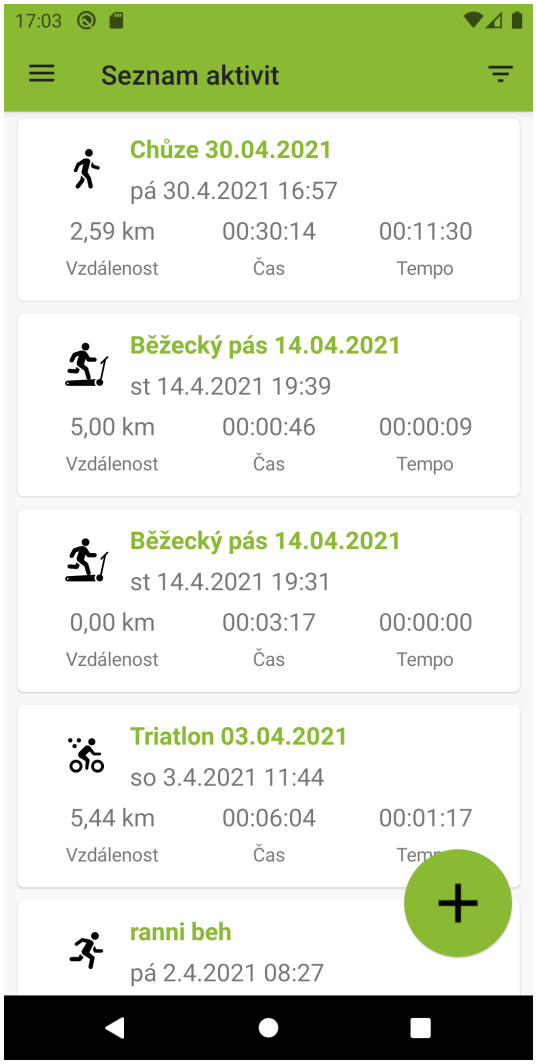

Obrázek 7.4 Výsledný vzhled mapy aktivity pomocí záložek

zovat více vizuálních stylů, v případě aplikace se kromě hlavního typu objektu (např. v případě seznamu aktivit se jedná o objekt aktivity) zobrazují další buňky zobrazující stav načítání, např. že je uživatel bez internetu či se žádné položky nepodařilo najít. Tyto selektory jsou použity potom v jednotlivých `ListView`. Příklad definice takového selektoru je ukázán ve fragmentu 7.

```
<ActivityCellTemplateSelector
    ActivityTemplate="{StaticResource ActivityCell}"
    LoadingTemplate="{StaticResource LoadingCell}"
    EmptyListTemplate="{StaticResource EmptyListCell}"
    NoInternetConnectionTemplate="{StaticResource
       NoInternetConnectionCell}" />
```

Fragment kódu 7 Definice selektoru pro seznam aktivit

Pro *ViewModely* a *Modely* je podstatné, aby pole, které jsou využité v *View*, vyvolávali událost (*Event*) `PropertyChanged`. Proto všechny modely a *ViewModely* dědí bázové třídy `BaseModel`, resp `BaseViewModel`, které usnadní vyvolávaní této události.




Obrázek 7.5 Výsledný vzhled seznamu aktivit

Na tuto událost potom reaguje *View*, které prvky propojené s příslušným polem znovu překreslí.

Pro vykonávání akcí na stránce lze vytvářet příkazy (*Commandy*). Jedná se o funkce, které jsou připojeny s využitím *Bindingu* k akcím prvků uživatelského rozhraní. Mezi takové akce lze zařadit stisknutí tlačítka. Příkazy se implementují pomocí implementace rozhraní `ICommand`, které zveřejňuje funkce pro spuštění příkazu (`Execute`), a jestli příkaz může být spuštěn (`CanExecute`). Obě funkce mají jako vstupní parametr objekt. V ukázce 8 je vidět implementace takového příkazu, který slouží k otevření stránky ve webovém prohlížeči. Příkaz lze vykonat, pouze pokud vstupní parametr je instancí třídy `URI`.

```
public class OpenBrowserCommand : BaseCommand
{
    public override bool CanExecute(object parameter)
    {
        return parameter is Uri;
    }
    public override async void Execute(object parameter)
    {
        await BrowserWrapper.OpenBrowser((Uri) parameter);
    }
}
```

Fragment kódu 8 Ukázka příkazu pro otevření prohlížeče

### 7.3.2 Připojení ke službě

Aplikace je klientem webové služby, přes kterou komunikuje s databází. Pro komunikaci je vygenerovaný klient s rozhraním, které odpovídá *Service Contract* služby. Obsahuje tedy jednotlivé funkce definované ve službě za pomocí *Operation Contract*, stejně tak všechny třídy odpovídající datovým kontraktům. V aplikaci je potom ke klientovi přistupováno přes adaptér `PhotonDataService`, který se kromě provolávání funkcí služby stará také o inicializaci samotného klienta a zachytávání výjimek, které služba vrací.

Kromě samotného přístupu je nutné řešit tvorbu filtrů, které jsou posílány jako parametry operací pro získávání seznamů, ať už aktivit, závodů nebo závodních sérii. Ze stránek Hlavním problémem filtrů je stránkování, kdy je nutné při získávání dalších stránek při načítání. K tomuto slouží příslušné `ModelFactory` třídy. Tyto třídy pomocí funkcí vytváří upravené filtry s ohledem především na stránkování, příkladem takové funkce ve třídě `WorkoutFilterModelFactory` je vidět v ukázce 9. Vizuálním příkladem stránky pro zadávání údajů, podle kterých dojde k filtraci aktivit, je zobrazen na obrázku 7.6.

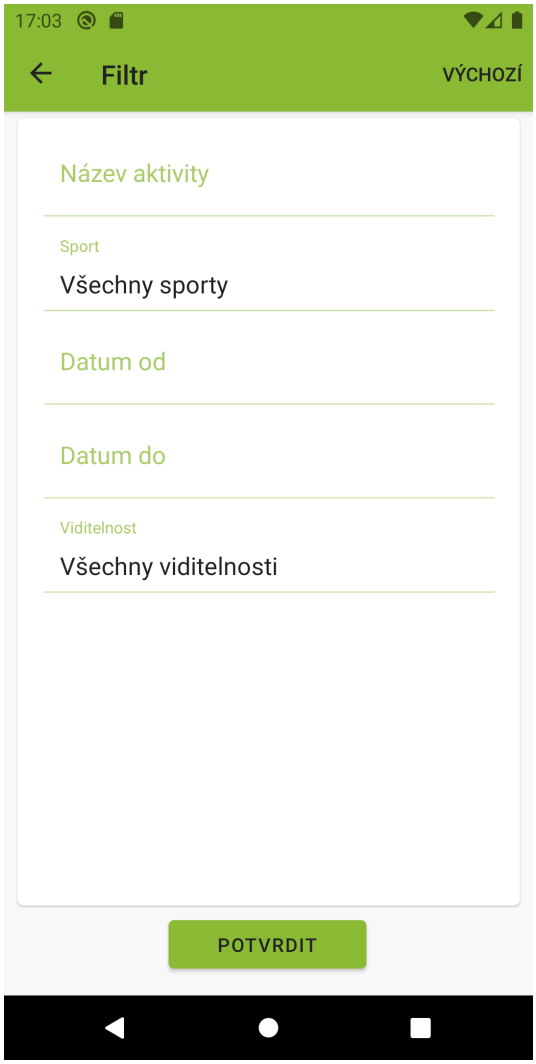
```
internal static WorkoutFilter
    CreateNextPageFilter(WorkoutFilter filter)
{
    var result = CreateCopy(filter);
    result.Page++;
    return result;
}
```

Fragment kódu 9 Funkce sloužící k vytvoření filtru následující stránky

## 7.4 Nativní část aplikace

Přestože velká část logiky aplikace, především sloužící k zobrazení dat a komunikace se službou, je řešena multiplatformně, část logiky musela být vyřešena za pomoci nativní části aplikace. Tato problematika se týká především získávání polohy uživatele pro delší časový úsek.

Základem této části je třída `MainActivity`. Jedná se o výchozí a jedinou aktivitu Android aplikace. Tato aktivita se stará ve funkci `OnCreate` o inicializaci případných




Obrázek 7.6 Výsledný vzhled filtru aktivit

použitých komponent, které jsou v aplikaci použity. Jako hlavní inicializovaná komponenta je *Xamarin.Forms* projekt s multiplatformní částí kódů, kromě toho ale také např. *Xamarin.Essentials* nebo *FormsMaps* pro zobrazování map v aplikaci.

Kromě toho také tato aktivita řeší *Launcher* při spouštění aktivity, který je v této aplikaci řešen jako samostatný styl. Při načtení aplikace, které se děje právě metodou `OnCreate`, tak se změní styl na design aplikace jako takové. Oba styly jsou definovány v ukázce 10. Zatímco styl `MainTheme.Launcher` se stará jen o obrazovku při spouštění, která je statická a definovaná pomoci XML definice obsahující primárně logo aplikace, styl pro samotnou aplikaci `MainTheme.Base` už obsahuje definice barev používaných jak aplikací (především *Shell*), ale také pro systémové lišty při zobrazené aplikaci.

```
<style name="MainTheme.Base"
   parent="Theme.AppCompat.Light.DarkActionBar">
        <item name="windowNoTitle">true</item>
        <item name="windowActionBar">false</item>
        <item name="colorPrimary">#8AB933</item>
        <item name="colorPrimaryDark">#8AB933</item>
        <item name="colorAccent">#8AB933</item>
        <item
           name="android:actionMenuTextColor">#222222</item>
        <item name="android:windowLightStatusBar">true</item>
        <item name="windowActionModeOverlay">true</item>
</style>
<style name="MainTheme.Launcher">
    <item name="android:windowBackground">
        @drawable/launch_screen
    </item>
</style>
```

Fragment kódu 10 Definice stylů aplikace

Další významnou a podstatnou částí nativní aplikace je soubor `AndroidManifest.xml`. Jak již bylo řečeno v kapitole 5.7, tento soubor definuje nastavení a požadavky aplikace. Cílový stav aplikace cílí (*Target SDK)* na Android SDK úrovně 29, s tím, že minimální podporovanou verzí je úroveň 24. Část tohoto souboru z verze s kódovým označením 42 je možné vidět v ukázce 11. Obsah definuje také název balíčku či samotné aplikace z pohledu prezentace, doplněný ikonkou, ale také právě výchozí styl, který je nastaven na styl `MainTheme.Launcher`. K tomu obsahuje definici služeb či oprávnění, které aplikace vyžaduje. Tyto jsou v ukázce 11 promazány, a jsou zobrazena pouze ta potřebná pro získávání polohy zařízení.

```
<manifest
   xmlns:android="http://schemas.android.com/apk/res/android"
          android:versionCode="42"
          android:versionName="r0.3.20"
          package="com.photonrun.photonapp">
    <uses-sdk android:minSdkVersion="24"
              android:targetSdkVersion="29" />
    <application android:icon="@mipmap/ic_launcher"
                 android:label="Photon Hero"
                 android:theme="@style/MainTheme.Launcher">
    </application>
    <uses-permission android:name =
       "android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name =
       "android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name =
       "android.permission.ACCESS_BACKGROUND_LOCATION" />
</manifest>
```

Fragment kódu 11 Část obsahu souboru AndroidManifest.xml

### 7.4.1 Optimalizace baterie kvůli službě pro záznam lokality

Problémem dlouho trvajících úkolů, které aplikace potřebuje vykonávat i na pozadí, je chování systému. Protože zařízení běžící s operačním systémem Android jsou zařízení s baterií, systém se snaží výdrž baterie co nejvíce prodloužit optimalizací aplikací. Z toho důvodu jsou často aplikace ukončeny dříve, než se o ukončení rozhodne uživatel.

Přestože některým optimalizacím nemůže aplikace sama zamezit, lze alespoň uži-

vatele na tato nastavení navést. Pro tyto účely slouží v nativní části aplikace třída `BatteryOptimalization`. Tato třída implementuje rozhraní `IBatteryOptimalization`, přes které je tato třída volána ze společné části projektu.

Funkce `IsBatteryOptimizationTurnedOff` a `OpenBatteryOptimizationSettings` se týkají samotné optimalizace baterie, tedy jestli systém ignoruje u aplikace optimalizování baterie. První funkce toto nastavení pouze kontroluje, nijak tato nastavení neovlivňuje. Při získávání se využívá instance `PowerManager` aktuálního aplikačního prostředí. Ten umí pro danou aplikaci zjistit, jestli už příslušné nastavení není zapnuté. Pokud zapnuté není, aplikace pomocí funkce `OpenBatteryOptimizationSettings` naviguje uživatele do příslušné stránky nastavení, kde si uživatel pro danou aplikaci vypne optimalizaci baterie.

Druhá dvojice funkcí už ale není podstatná pro všechny nadstavby operačního systému. Zatímco optimalizace baterie bude dostačující pro část výrobců mobilních zařízení, které spoléhají na optimalizaci baterie poskytnutou samotným systémem. První funkce `IsOneOfAutoStartCriticalOS` kontroluje, je-li výrobce mobilního zařízení veden jako jeden z těch, pro které je potřeba vykonat i druhou optimalizaci samotné nadstavby. Mezi uživatelské prostředí řešené aplikací jsou ty od firem Xiaomi, Oppo, Vivo a Huawei, resp. Honor. U ostatních výrobců mobilních telefonů nebyly nalezeny problémy vyžadující obdobné řešení. Pro tyto je pomocí funkce `OpenAutoStartSettings` zobrazena příslušná stránka nastavení, ve které uživatel musí především zapnout povolení aplikaci běžet na pozadí.

### 7.4.2 Služba pro záznam lokality

Hlavní částí, o kterou se stará právě nativní část aplikace, kromě samotné inicializace Android aplikace je získávání polohy uživatele. Právě pro bezproblémový běh zaznamenávání aktivity jsou potřeba i kroky zmíněné v kapitole 7.4.1. Tyto kroky ale nejsou jediné, které aplikace musí splňovat, aby zaznamenávání lokace mohlo běžet déle. K tomuto slouží právě služby. V případě aplikace vyvíjené v rámci práce se jedná o službu `LocationUpdatesService`. Tato služba využívá *FusedLocationClient* pro získávání samotné polohy, jedná se o klienta využívající 3 způsoby získávání polohy a využívá ten nejvhodnější pro danou chvíli. Jako požadavek, určující vlastnosti získávání, je vytvořena instance třídy `LocationRequest`, který má nastavené časové intervaly ve kterých bude lokalitu získávat na 1 sekundu, resp. nejrychleji polovinu tohoto času, s tím, že se snaží získat lokalitu s nejvyšší přesností (`LocationRequest.PriorityHighAccuracy`). Inicializace tohoto požadavku je v ukázce 12.

```
private void CreateLocationRequest()
{
    _locationRequest = new LocationRequest();
    _locationRequest.SetInterval(1000);
    _locationRequest.SetFastestInterval(500);
    _locationRequest.SetPriority(
       LocationRequest.PriorityHighAccuracy);
}
```

Fragment kódu 12 Kód pro vytvoření požadavku na lokaci

Ve chvíli, kdy se aplikace přepíná na pozadí, služba musí stále pokračovat. Toto se děje s použitím funkce `StartForeground`, kdy služba je vázaná na notifikaci, která uživatele upozorňuje, že služba stále běží. Kvůli frekvenci obnovování, notifikace má potlačené veškeré zvuky a vibrace, případně jiné projevy. Do API úrovně 25 včetně toto řeší nastavení notifikace jako takové, od této verze má tyto nastavení na starosti příslušný kanál notifikace. Funkce pro vytvoření takového kanálu je vidět v ukázce 13.

```
private void CreateNotificationChannel()
{
    if (Build.VERSION.SdkInt >= BuildVersionCodes.O)
    {
        var name = new String(ChannelName);
        var channel = new NotificationChannel(ChannelId,
           name, NotificationImportance.Low)
        {
            Description = ChannelDescription,
        };
        channel.SetSound(null, null);
        channel.EnableVibration(false);
        channel.EnableLights(false);
        var manager = (NotificationManager)
           GetSystemService(NotificationService);
        manager.CreateNotificationChannel(channel);
    }
}
```

Fragment kódu 13 Kód pro vytvoření notifikačního kanálu

Služba musí být registrována v Android Manifestu, v tomto případě je registrovaná pomocí elementu zobrazeného v ukázce 14. Pomocí atributu `foregroundServiceType ="location"` služba deklaruje, že je využívána pro získávání polohy zařízení. Toto nastavení je nutné od API úrovně 29. Kromě registrace této služby je v souboru *AndroidManifest.xml* také nutné deklarovat oprávnění pro využívání (nejen) této služby. To se děje za pomoci elementu `<uses-permission android:name = "android.permission .FOREGROUND_SERVICE"/>`.

```
<service android:name =
    "com.photonrun.photonapp.LocationUpdatesService"
          android:enabled = "true"
          android:exported = "true"
          android:foregroundServiceType = "location" />
```

Fragment kódu 14 Registrace služby v Android Manifestu

Výstupem této funkce je instance třídy `Location` (z *namespace* `Android.Locations`) obsahující informace o poloze. Protože se získaná data zpracovávají až ve společné části, jsou tato data převedena na instanci třídy `Location`, tentokrát ale z knihovny

`Xamarin.Essentials`. Z pohledu aplikace jsou především podstatné údaje jako poloha daného bodu spolu s časovou známkou, k tomu navíc elevace a jak horizontální, tak vertikální přesnost. Třída *Location* knihovny `Xamarin.Essentials` poskytuje funkci `CalculateDistance`, která slouží pro výpočet vzdálenosti mezi dvěma body. Tohoto se využívá pro výpočet vzdálenosti, která je zobrazena při samotném zaznamenávání polohy v aplikaci.

Zpracování výstupů ze služby se děje ve společné části aplikace, kdy se data průběžně ukládají do instance třídy `ActivityContainer`. Tato třída obsahuje všechna potřebná data o aktivitě, která jsou potřebná při ukládání aktivity do výstupu v podobě souboru GPX a pro ukládání do databáze na službu. Kromě toho se také stará o potřebná data, která se zobrazují uživateli v při aktivním sledování, tedy aktuální tempo, vzdálenost a aktuální čas. Zatímco pro čas a vzdálenost se využívají systémové knihovny a mohou tyto data být aktualizovány pravidelně, pro tempo bylo potřeba vyřešit specifický výpočet a způsob zobrazování. Důvodem k tomuto kroku je fakt, že při aktualizaci tempa každou sekundu dochází k výrazným výkyvům vypočítané hodnoty. Z toho důvodu se tempo aktualizace méně často (konkrétně každých 5 sekund) a pro výpočet se průměruje hodnota vypočítaných temp vycházejících z posledních 6 bodů. Obrázek 7.7 obsahuje stránku aplikace, která zobrazuje průběžně vypočítané hodnoty aktivity zobrazují.

Při vkládání bodů do `ActivityContainer` se vkládají body s horizontální přesností do 200 metrů, aby nedošlo ke zkreslování dat aktivity. K tomuto číslu se došlo při analýze testovacích aktivit z různých zařízení, kdy body s hodnotou horizontální přesností nad tímto prahem tvořili méně než 1 procento dat, díky tomu nedochází k výraznému umělému zkreslení trasy aktivity. Horizontální přesnosti se navíc také využívá k upozorněni uživatele na slabý GPS signál.

Aplikace umožňuje uživateli aktivity pozastavovat. Při pozastavené aktivitě nejsou údaje o poloze zjišťovány, a je toto potřeba zohlednit především při přepočítávání časových údajů, ale také vzdálenosti zobrazených při aktivním sledování.

Výsledkem celé aktivity je především soubor GPX, kromě toho ale také informace zadané uživatelem, jako název, viditelnost či kategorie aktivity. Všechna tato data se odesílají službě k uložení. Při ukládání se ale neodesílají data o vzdálenosti či tempu, tato data jsou potom znovu přepočtena službou.


Obrázek 7.7 Výsledný vzhled stránky pro zaznamenávání polohy

# 8 TESTOVÁNÍ A DISTRIBUCE APLIKACE

Již od začátku aplikace byla testována a distribuována na fyzická zařízení. Testování v rámci vývoje při diplomové práci probíhalo ve 3 fázích. První fáze testování byla vyhrazena pro určení konceptu aplikace, především jak se aplikace chová na různých zařízení, ověření funkcionality služby pro zjišťování polohy, ale také schválené směrování uživatelského prostředí aplikace. Této testovací fáze se zúčastnili především členové projektu PhotonHero a testy probíhaly na asi 8 různých konfiguracích mobilních zařízení, především s ohledem na verzi a nadstavbu operačního systému. Výstupem tohoto testování bylo ověření zvoleného konceptu a přibližné funkčnosti pro zjišťování polohy.

Druhou fázi testování lze označit jako *uzavřené testování* nebo *Alpha testování*, která následovalo ve chvíli, kdy se především služba pro zjišťování polohy zařízení zdála dostatečně stabilní na dosavadních zařízeních. Toto testování probíhalo u většího počtu uživatelů než první fáze. Stále se ale jednalo o úzký kruh pozvaných uživatelů, především osoby blízké členům projektu. Cílem této fáze testování bylo ověřit funkcionalitu zaznamenávání polohy na větším počtu zařízení, ale také již získat první připomínky k uživatelskému rozhraní a ovládání aplikace od dalších osob.

Aktuálně poslední fází testování je tzv. *otevřené testování* nebo *Beta testování*, kdy je aplikace dostupná všem zájemcům. Cílem této fáze je především získání uživatelských připomínek k dosavadním částem aplikace, především k ovládání a uživatelskému prostředí. Vedlejším cílem je také ověřit chování služby pro zaznamenávání na konfiguracích, které nebyly k dispozici při vývoji aplikace.

Zatímco první fáze v podstatě byla jen dočasná, fáze *uzavřeného testování* i *otevřeného testování* se bude využívat nadále pro testování nových funkcionalit. Fáze *uzavřeného testování* bude sloužit především k testování nových funkcionalit, které se mohou, ale nemusí v té konkrétní podobě dostat do otevřeného testování, či dalších úrovní distribuce. *Otevřené testování* bude i nadále slouží k testování funkcí užší veřejností, jedná se primárně o stabilní verze testované v rámci uzavřeného testování.

Z pohledu distribuce byly v rámci vývoje využívány 2 nástroje. Prvním nástrojem, který se využíval pouze v první fázi testování byl *Visual Studio AppCenter* od firmy Microsoft. Jedná se o on-line nástroj určený pro diagnostiku, analýzu, ale právě i distribuci aplikací. Uživatelé si potom mohou aplikaci stáhnout ze stránek, které jsou nástrojem zpřístupněny. Nejedná se tedy o automatizovaný způsob distribuce, uživatel si musí aplikaci aktualizovat ručně. Kromě toho nástroj umožňuje diagnostikovat problémy či analyzovat data o používání aplikace, pokud je nástroj inicializován aplikací.

Druhým způsobem distribuce, který byl využit ve všech 3 fázích testování, byl samotný Google Play, resp. jeho kanály. V rámci první fáze testování byl využit kanál

interního testování, který slouží primárně jen na distribuci APK souborů. Uživatel při tomto kanálu musí aplikaci vždy instalovat ručně přes stránku aplikace v obchodu. Tato stránka ale není přímo dostupná a je nutné se na ni dostat odkazem, i z tohoto důvodu není pro tento kanál plánováno výrazné použití a jeho úlohu převezme nadále *uzavřené testování*. Pro tento kanál už existuje stránka v obchodě, na kterou se lze dostat přes např. detail aplikace v nastavení systému. Uživatele se do tohoto testování nemohou připojit samovolně, je nutné uživatele přidat v nastavení tohoto kanálu. Pro veřejnost byla aplikace distribuována přes kanál *Otevřeného testování*. V tomto kanálu už je aplikace vyhledatelná přes obchod Google Play, a není nutné udělat specifické kroky, aby si uživatel mohl aplikaci stáhnout.

# 9 BUDOUCNOST APLIKACE A MOŽNÁ ROZŠÍŘENÍ

Hlavním cílem nejbližšího dalšího vývoje bude doladit veškeré nalezené chyby či nedostatky v rámci vývoje, především týkajících se uživatelského rozhraní či neošetřených pádů aplikace. Tedy jde především o stabilizaci aktuální funkčnosti. Mezi jednu z nedořešených věcí je využití aplikace Google Maps pro navigování uživatele na závod, není ale zohledněno, že uživatel na zařízení má tuto aplikaci systémově vypnutou. Další nedořešenou, přesto podstatnou funkcionalitou je průběžné ukládání aktivity při aktivním zaznamenávání, aby uživatel při nečekaném pádu aplikace aktivitu neztratil, ale mohl na ni navázat.

Aplikace se v rámci diplomové práce zaměřila především na vývoj nativní Android aplikace, přesto se ale při návrhu a výběru technologií již zohledňovala možnost rozšíření na další operační systémy, především na iOS. Stejně jako u Android verze, i iOS verze bude muset zohlednit omezení platformy tykající se získávání polohy na pozadí, aby bylo možné získávat potřebné informace k aktivitě. Mezi další možné zařízení, na které by se mohla aplikace rozšířit jsou zařízení s operačním systémem *Tizen* od Samsungu, tedy především chytré hodinky. V rámci dalšího vývoje se aplikace také může věnovat případnému rozšíření na operační systém HarmonyOS společnosti Huawei.

Další možná rozšíření z pohledu funkcionality už jsou dána v rámci webové části projektu, jedná se především o přímou podporu funkcionalit týkajících se výzev, čí týmu, ale také detailnější statistiky a informace o konkrétních aktivitách. Z pohledu distribuce je cílem aplikaci zveřejnit do produkčního kanálu v rámci obchodu Google Play.

# ZÁVĚR

Cílem diplomové práce bylo navrhnout a implementovat nativní Android aplikaci pro projekt PhotonHero. Při návrhu a vývoji aplikace se především zohledňovali požadavky dané samotným projektem pro jeho další směrování.

V teoretické části práce nejprve představila platformu .NET a možnosti vývoje mobilních aplikací nejen na Android. V souvislosti s tímto tématem byly nejprve představeny technologie Mono, resp. jeho implementace Xamarin.Android a Xamarin.iOS. Na tyto technologie navázala platforma Xamarin.Forms umožňující vývoj multiplatformních aplikací. Spolu s tím práce zmínila také knihovnu Xamarin.Essentials a především budoucnost těchto technologií s příchodem platformy .NET 5 a platformy MAUI.

V druhé polovině teoretické částí se zmiňují specifika vývoje nativních Android aplikací. V těchto se věnuje aktivitám a jejich životnímu cyklu, jak tento může ovlivnit jednotlivé částí aplikace. Dále také službám (*Services*), které mohou sloužit mimo jiné právě k získávání polohy. Nejen v souvislosti se službami práce dále probírá oprávnění či roztříštěnost Androidu v podobě jeho verzí, ale nadstaveb od jednotlivých výrobců mobilních zařízení.

V praktické části práce navrhla základní vlastnosti jednotlivých funkcionalit aplikace a její napojení na již existující systém PhotonHero. Na základě tohoto návrhu a poznatků z teoretické části došlo k implementaci samotné aplikace. Pro implementaci v souvislosti s rozšířením aplikace na další mobilní platformy byla zvolena platforma Xamarin.Forms. Kromě toho se také praktická část věnovala propojení s webovou službou, která byla vyvíjena současně s mobilní aplikací. U nativní části aplikace se řešila zmíněná specifika se získáváním polohy, její omezení, a především nutné zásahy uživatele před plnohodnotnou funkčností této funkcionality.

V poslední části se řešil způsob testování a distribuce. U testování se vzali v potaz postupné úrovně testování, které byly označený jako interní, uzavřené a otevřené testování, kdy v poslední části již do testování byla zapojena i užší veřejnost. S ohledem na distribuci byl v první fázi zvolen nástroj Visual Studio AppCenter, které v pozdější části již trvale nahradil Google Play.

## SEZNAM POUŽITÉ LITERATURY

[1] Strava | Run and Cycling Tracking on the Social Network for Athletes. [Online; navštíveno 14.2.2021].
URL `https://www.strava.com/features`

[2] MapMyWalk. [Online; navštíveno 16.2.2021].
URL `https://www.mapmywalk.com/`

[3] MapMyRun. [Online; navštíveno 16.2.2021].
URL `https://www.mapmyrun.com/`

[4] MapMyFitness. [Online; navštíveno 16.2.2021].
URL `https://www.mapmyfitness.com/`

[5] MapMyRide. [Online; navštíveno 16.2.2021].
URL `https://www.mapmyride.com/`

[6] Endomondo Is Retired. [Online; navštíveno 16.2.2021].
URL `https://support.endomondo.com/hc/en-us/articles/360016251359-Endomondo-Is-Retired`

[7] International, G.: Garmin Connect | Free Online Fitness Community. [Online; navštíveno 16.2.2021].
URL `https://connect.garmin.com/`

[8] Termínovka běžeckých závodů. [Online; navštíveno 16.2.2021].
URL `https://www.svetbehu.cz/terminovka/`

[9] behej.com: BĚH, MARATON, BĚHÁNÍ. [Online; navštíveno 16.2.2021].
URL `https://www.behej.com/terminovka`

[10] What is .NET Framework? A software development framework. [Online; navštíveno 28.2.2021].
URL `https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet-framework`

[11] Introducing .NET 5. Květen 2019, [Online; navštíveno 28.2.2021].
URL `https://devblogs.microsoft.com/dotnet/introducing-net-5/`

[12] Sharp, J.: *Microsoft Visual C♯ 2010: krok za krokem*. Brno: Computer Press, 2010, ISBN 978-80-251-3147-3, oCLC: 697268072.

[13] What is managed code? [Online; navštíveno 28.2.2021].
URL https://docs.microsoft.com/en-us/dotnet/standard/managed-code

[14] History | Mono. [Online; navštíveno 28.2.2021].
URL https://www.mono-project.com/docs/about-mono/history/

[15] [Mono-list] Mono early history. Červen 2003, [Online; navštíveno 28.2.2021].
URL https://web.archive.org/web/20110606210557/http://lists.ximian.com/archives/public/mono-list/2003-October/016345.html

[16] Xamarin tool aims to show the ease wth which .NET apps can become mobile. Červen 2013, [Online; navštíveno 28.2.2021].
URL https://www.pcworld.com/article/2042950/xamarin-tool-aims-to-show-the-ease-wth-which-net-apps-can-become-mobile.html

[17] profexorgeek: What is Xamarin? - Xamarin. [Online; navštíveno 28.2.2021].
URL https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin

[18] Xamarin.Forms | .NET. [Online; navštíveno 7.3.2021].
URL https://dotnet.microsoft.com/apps/xamarin/xamarin-forms

[19] davidbritch: Xamarin.Forms XAML Basics - Xamarin. [Online; navštíveno 7.3.2021].
URL https://docs.microsoft.com/en-us/xamarin/xamarin-forms/xaml/xaml-basics/

[20] davidbritch: Xamarin.Forms Quickstart Deep Dive - Xamarin.
URL https://docs.microsoft.com/en-us/xamarin/get-started/quickstarts/deepdive

[21] davidortinau: Hello, Android: Deep Dive - Xamarin. [Online; navštíveno 20.3.2021].
URL https://docs.microsoft.com/en-us/xamarin/android/get-started/hello-android/hello-android-deepdive

[22] jamesmontemagno: Xamarin.Essentials - Xamarin. [Online; navštíveno 7.3.2021].
URL https://docs.microsoft.com/en-us/xamarin/essentials/

[23] LocationRequest | Google APIs for Android. [Online; navštíveno 7.3.2021].
URL https://developers.google.com/android/reference/com/google/android/gms/location/LocationRequest

[24] jamesmontemagno: Xamarin.Essentials: Geolocation - Xamarin. [Online; navštíveno 7.3.2021].
URL https://docs.microsoft.com/en-us/xamarin/essentials/geolocation

[25] Introducing .NET Multi-platform App UI | .NET Blog. [Online; navštíveno 20.3.2021].
URL https://devblogs.microsoft.com/dotnet/introducing-net-multi-platform-app-ui/

[26] Uno Platform Uno Platform sponsors .NET Foundation. Únor 2021, [Online; navštíveno 21.3.2021].
URL https://platform.uno/blog/uno-platform-sponsors-net-foundation/

[27] Codenames, Tags, and Build Numbers | Android Open Source Project. [Online; navštíveno 29.3.2021].
URL https://source.android.com/setup/start/build-numbers

[28] davidortinau: Understanding Android API Levels - Xamarin. [Online; navštíveno 29.3.2021].
URL https://docs.microsoft.com/en-us/xamarin/android/app-fundamentals/android-api-levels

[29] Don't kill my app! [Online; navštíveno 6.4.2021].
URL https://dontkillmyapp.com/

[30] The Huawei ban explained: A complete timeline and everything you need to know. Únor 2021, [Online; navštíveno 29.3.2021].
URL https://www.androidauthority.com/huawei-google-android-ban-988382/

[31] Introduction to Activities | Android Developers. [Online; navštíveno 3.4.2021].
URL https://developer.android.com/guide/components/activities/intro-activities

[32] davidortinau: Activity Lifecycle - Xamarin. [Online; navštíveno 3.4.2021].
URL https://docs.microsoft.com/en-us/xamarin/android/app-fundamentals/activity-lifecycle/

[33] davidortinau: Creating Android Services - Xamarin. [Online; navštíveno 5.4.2021].
URL https://docs.microsoft.com/en-us/xamarin/android/app-fundamentals/services/

[34] Services overview | Android Developers. [Online; navštíveno 6.4.2021].
URL https://developer.android.com/guide/components/services

[35] Foreground services | Android Developers. [Online; navštíveno 6.4.2021].
URL https://developer.android.com/guide/components/foreground-services

[36] davidortinau: Location services on Android - Xamarin. [Online; navštíveno 6.4.2021].
URL https://docs.microsoft.com/en-us/xamarin/android/platform/maps-and-location/location

[37] Permissions on Android | Android Developers. [Online; navštíveno 6.4.2021].
URL https://developer.android.com/guide/topics/permissions/overview

[38] Request app permissions | Android Developers. [Online; navštíveno 7.4.2021].
URL https://developer.android.com/training/permissions/requesting

[39] davidortinau: Permissions In Xamarin.Android - Xamarin. [Online; navštíveno 7.4.2021].
URL https://docs.microsoft.com/en-us/xamarin/android/app-fundamentals/permissions

[40] davidortinau: Local Notifications on Android - Xamarin. [Online; navštíveno 7.4.2021].
URL https://docs.microsoft.com/en-us/xamarin/android/app-fundamentals/notifications/local-notifications

[41] App Manifest Overview. [Online; navštíveno 8.4.2021].
URL https://developer.android.com/guide/topics/manifest/manifest-intro

[42] App resources overview. [Online; navštíveno 8.4.2021].
URL https://developer.android.com/guide/topics/resources/providing-resources

[43] HongGit: Fundamental Windows Communication Foundation Concepts - WCF. [Online; navštíveno 18.4.2021].
URL https://docs.microsoft.com/en-us/dotnet/framework/wcf/fundamental-concepts

[44] tamram: Introduction to Blob (object) storage - Azure Storage. [Online; navštíveno 24.4.2021].

URL https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction

# SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

| | |
|---|---|
| aGPS | Assisted GPS |
| APK | Android package |
| CIL | Common Intermidiate Language |
| CLI | Common Language Infrastructure |
| CLR | Common Language Runtime |
| DPI | Dots Per Inch |
| DTO | Data transfer object |
| EF | Entity Framework |
| GMS | Google Mobile Service |
| GPS | Global Positioning System |
| GPX | GPS Exchange Format |
| GUID | Globally Unique Identifier |
| LINQ | Language Integrated Query |
| MAUI | Multi-platform App UI |
| MVVM | Model-View-ViewModel |
| ORM | Object-relational Mapping |
| SDK | Software Development Kit |
| SOA | Service Oriented Architecture |
| URI | Uniform Resource Identifier |
| UWP | Universal Windows Platform |
| VB.NET | Visual Basic .NET |
| WPF | Windows Presentation Foundation |
| XAML | Extensible Application Markup Language |
| XML | Extensible Markup Language |

## SEZNAM OBRÁZKŮ

## SEZNAM TABULEK

## SEZNAM FRAGMENTŮ KÓDU