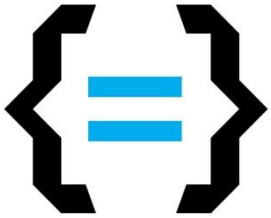
Univerzita Hradec Králové
Fakulta informatiky a managementu

Diplomová Práce

Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod



Srovnání multiplatformního nástroje Xamarin s platformně zaměřeným vývojem

Diplomová Práce

Autor: Bc. Ondřej Vondra
Studijní obor: Aplikovaná Informatika

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

Hradec Králové duben 2020

Prohlášení:

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

V Hradci Králové dne 30. 4. 2020

podpis
Ondřej Vondra

Poděkování:

Děkuji vedoucímu diplomové práce doc. Mgr. Tomáši Kozlovi, Ph.D. za metodické vedení práce a cenné rady. Rovněž děkuji své rodině a přátelům za podporu během studia.

Anotace

V současné době je multiplatformní vývoj diskutován stále častěji. Pro multiplatformní vývoj lze využít širokou škálu technologií, které se na první pohled mohou zdát velmi podobné, nicméně při bližším zkoumání mají značné rozdíly. Vývojářské společnosti zaměřené na mobilní vývoj se často dostávají do situace, kdy se musí rozhodnout, zda udržovat více kódových základen, či zvolit multiplatformní variantu vývoje. Následující práce se zaobírá multiplatformním vývojem se zvláštním zaměřením na nástroj Xamarin. Popisuje jeho principy, výhody, nevýhody a vytváří srovnání s nativním vývojem.

Klíčová slova:

Multiplatformní vývoj, Xamarin, C#, Android, Java, Porovnání vývoje

Annotation

Title:

Comparison of multiplatform tool Xamarin with platform-oriented development

Multiplatform development is being discussed more and more often. There is a wide range of technologies that can be used for a multiplatform development, although at the first glance seem to be the same. The closer look can show the major differences. Companies focused on mobile development often find themself having to decide whether to maintain multiple code bases or to choose a multiplatform development option. Following thesis deals with cross-platform development with special focus on Xamarin. It describes its principles, advantages, disadvantages and makes a comparsion with native development.

Keywords:

Multiplatform development, Xamarin, C#, Android, Java, Development comparison

Obsah

Seznam zkratek

ACW - *Android Callable Wrappers*

ADT - *Android Development Tools*

ALPR - *Automatic License Plate Recognition*

AOSP - *Android Open Source Project*

AOT - *Ahead-of-Time*

API - *Application Programming Interface*

APK - *Android Application Package*

ARC - *Automatic Reference Counting*

ART - *Android Runtime*

ASCII - *American Standard Code for Information Interchange*

BSD - *Berkeley Software Distribution*

CSS - *Cascading Style Sheets*

ES - *Embedded Systems*

GB - *Gigabyte*

GPS - *Global Positioning System*

HAL - *Hardware Abstraction Layer*

HTML - *HyperText Markup Language*

IDE - *Integrated Development Environment*

IL - *Intermediate Language*

IoT - *Internet of Things*

IPC - *Inter Process Communication*

JSON - *JavaScript Object Notation*

JSX - *JavaScript XML*

MB - *Megabyte*

MCW - *Managed Callable Wrappers*

OpenAL - *Open Audio Library*

OpenGL - *Open Graphics Library*

PWA - *Progressive Web Apps*

REST - *Representational State Transfer*

SDK - *Software Development Kit*

SPA - *Single Page Application*

STT - *Speech to Text*

TTS - *Text to Speech*

UI - *User Interface*

VB - *Visual Basic*

XAML - *Extensible Application Markup Language*

XML - *Extensible Markup Language*

Seznam obrázků

Seznam tabulek

Seznam úryvků

1 Úvod

Využití multiplatformních nástrojů pro vývoj, či volba udržování kódových základen pro obě hlavní platformy je v současné době důležitým rozhodnutím, před kterým stojí mnoho společností začínajících s vývojem mobilní aplikace. Nástrojů, prostředí pro multiplatformní vývoj se nabízí hned několik. Mezi nejoblíbenější z nich patří právě testovaný Xamarin, React Native, Ionic a Flutter, přičemž rozdíly mezi těmito nástroji jsou v některých případech drobné a jindy velmi výrazné.

Cílem předkládané práce je vytvořit srovnání platformně zaměřeného vývoje s vývojem v multiplatformním prostředí Xamarin. V době vzniku práce byla nově představena varianta Xamarin.Forms využívající Blazor. Tento projekt se v jeho první verzi nazývá „Microsoft Mobile Blazor Bindings“. I když se jedná o pouhý náhled na možný rozvoj Xamarinu, jde o zajímavý přístup, který je v této práci představen.

Odborná část práce je rozdělena do čtyř hlavních částí. V první části jsou představeny technologie a nástroje, které lze používat při multiplatformním, či nativním vývoji. Spolu s nástroji jsou zde představeny hlavní platformy, na které se dnešní společnosti při vývoji zaměřují. Hlavní důraz je zde kladen na Xamarin a možnosti nativního vývoje. Každý nástroj a přístup je stručně představen a jsou shrnuty jeho nejčastější uváděné klady a zápory.

Druhá část se soustředí na využití srovnávaných vývojových prostředí s důrazem na konkrétní, v nich dostupné nástroje. Kapitola popisuje rozdíly mezi srovnávanými prostředími. Dále obsahuje návrhy testů, na jejichž základě je prováděno hodnocení a vzájemné srovnání. Použité testy jsou kvantitativního a kvalitativního typu. Výstupními parametry jsou tvorba první aplikace, velikost nainstalovaného prostředí, rychlost sestavení projektu, rychlost spouštění aplikace a zhodnocení autora.

Třetí část definuje referenční srovnávací rámec, kterým je již dříve, pro edukační účely vytvořená aplikace. Srovnávanými ukazateli jsou způsob implementace vybraných metod a způsob zápisu programového kódu, délka a složitost kódu nutného pro vývoj komplexní mobilní aplikace. Vybranými metodami a programovými bloky jsou volání a konzumace REST API, obrazovka aplikace z pohledu markupu a funkčního kódu, volání základních hardwarových prvků: mikrofon, reproduktor, kamera a GPS modul.

V poslední části práce je provedeno výkonnostní testování aplikace, které srovnává většinu metod z předchozí části. Srovnání je provedeno porovnáním doby potřebné k vykonání dané operace. Nedílnou součástí je srovnání zabíraného místa na koncovém zařízení a velikostí generovaného APK souboru určeného k instalaci.

Tato práce by měla pomoci začínajícím vývojářům při rozhodování, zda k vývoji využít multiplatformní nástroj Xamarin, nebo jít cestou nativního vývoje a udržovat kód pro obě hlavní platformy.

2 Představení technologií a nástrojů

2.1 Xamarin

2.1.1 Obecně

Jedná se o open-source platformu pro vytváření aplikací především pro iOS a Android. Jde o abstraktní vrstvu, zprostředkovávající komunikaci mezi sdíleným kódem a kódem základní platformy. Vývojářům je takto umožněno sdílet průměrně až 90 % svých aplikací napříč platformami. Veškerá business logika je díky tomu napsána v jednom jazyce, při zachování nativního výkonu, chování a vzhledu aplikace na jednotlivých platformách. (1)

2.1.2 Historie

Xamarin vznikl začátkem roku 2000 jako experimentální pokus o vytvoření verze .NET pro Linux. Nejprve byl známý pod názvem Mono. Později byl projekt vyvíjen a podporován společností s názvem Xamarin, která byla vytvořena prvními Mono vývojáři a byla také známa pod jménem MonoTouch a Mono pro Android. Xamarin je nyní pod křídly společnosti Microsoft a je dodáván jako součást jejího produktu pro vývojáře, Visual Studio. Má i vlastní vývojový nástroj s názvem Xamarin Studio, jenž je dostupný jak pro Windows, tak pro Mac. (2)

2.1.3 Funkcionalita

Xamarin dovoluje tvorbu nativního uživatelského prostředí pro každou cílovou platformu, při zachování business logiky psané v jednotném jazyce, a to v C#, napříč všemi platformami. Ve většině případů se jedná o 80 % veškerého kódu potřebného k běhu aplikace, který je sdílen pomocí Xamarinu.

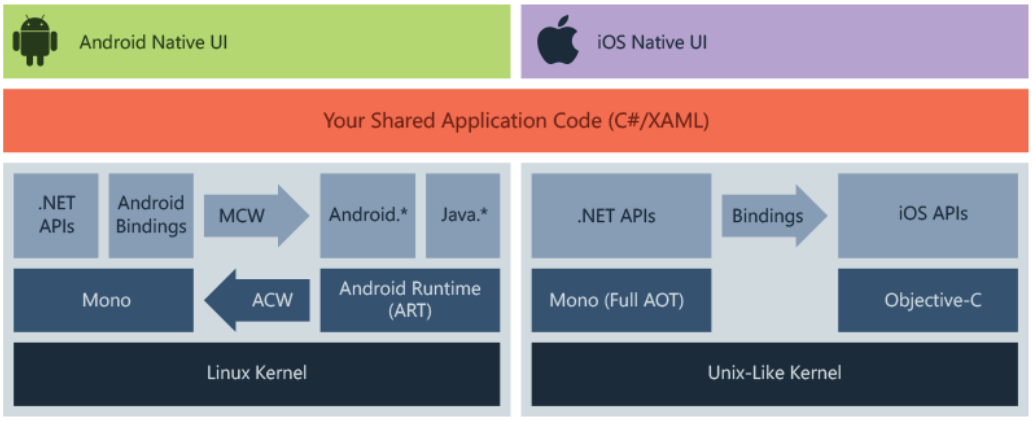
Je postaven na prostředí Mono (dále jen Mono), což je open-source verze .NET frameworku založená na .NET ECMA standardech[1]. Detailní informace k EMCA standardům jsou dostupné na webu Emca International®. (4) Mono existuje téměř stejně dlouho jako .NET Framework a funguje na většině platforem, včetně Linuxu UNIXU, FreeBSD[2] a macOS. Detaily projektu FreeBSD jsou dostupné na webu Freebsd.org. (5)

[1] **ECMA Standardy** – Normalizační standardy pro informační a komunikační systémy.

[2] **FreeBSD** – Operační systém využívaný převážně pro servery a embedované platformy.

Mono automaticky řídí úlohy jako je alokace paměti, uvolňování paměti a spolupracuje spolu se základními platformami. (3)

Na obrázku 1. je vidět diagram celkové architektury Xamarinu pro různé platformy. Jednotlivé části jsou rozebrány níže.



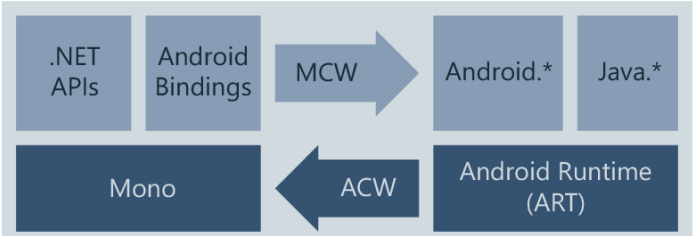
Obrázek 1 - Xamarin architektura (1)

2.1.4 Architektura

Dvěma hlavními částmi, ze kterých se skládá architektura Xamarinu, jsou Xamarin.Android a Xamarin.iOS. Vývoj v Xamarinu je podobný vývoji nativnímu, s použitím nativních nástrojů. Díky tomu, že používá pro vývoj jazyk C#, může být logická vrstva snadno sdílena mezi platformami a vývojář je tak ušetřen od psaní duplicitního kódu. Jedná se o runtime prostředí, jenž přidává mezi balíčky knihoven a zdrojový a strojový kód další vrstvu. (1)

2.1.4.1 Xamarin.Android

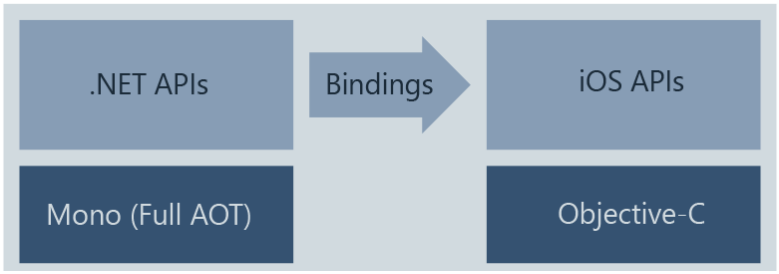
Aplikace Xamarin.Android je kompilována z jazyka C# do jazyka IL (Intermediate Language), jenž je při spuštění aplikace za běhu kompilován do nativního sestavení. Běží v prostředí Mono spolu s ART (Android runtime) virtuálním strojem. Xamarin poskytuje vazby na Android.* a Java.* knihovny, které Mono zprostředkovává za pomoci MCW (Managed Callable Wrappers) a poskytuje ACW (Android Callable Wrappers) pro ART. Toto dovoluje oběma prostředím vzájemné vyvolání kódu, jak je přehledně znázorněno na obrázku 2. (1)



Obrázek 2 - Xamarin.Android struktura (1)

2.1.4.2 Xamarin.iOS

Aplikace Xamarin.iOS jsou plně AOT (Ahead-of-Time) kompilovány z C# do nativního ARM sestavení. Zde Xamarin využívá Selektory k vystavení objektového C pro řízení C# a Registrátory k vystavení kódu psaného v C# do objektového C. Selektory a Registrátory jsou společně nazývány Vazbami a dovolují vzájemnou komunikaci mezi objektovým C a C#. Tyto vazby (Bindings) jsou znázorněny na obrázku 3. (1)



Obrázek 3 - Xamarin.iOS struktura (1)

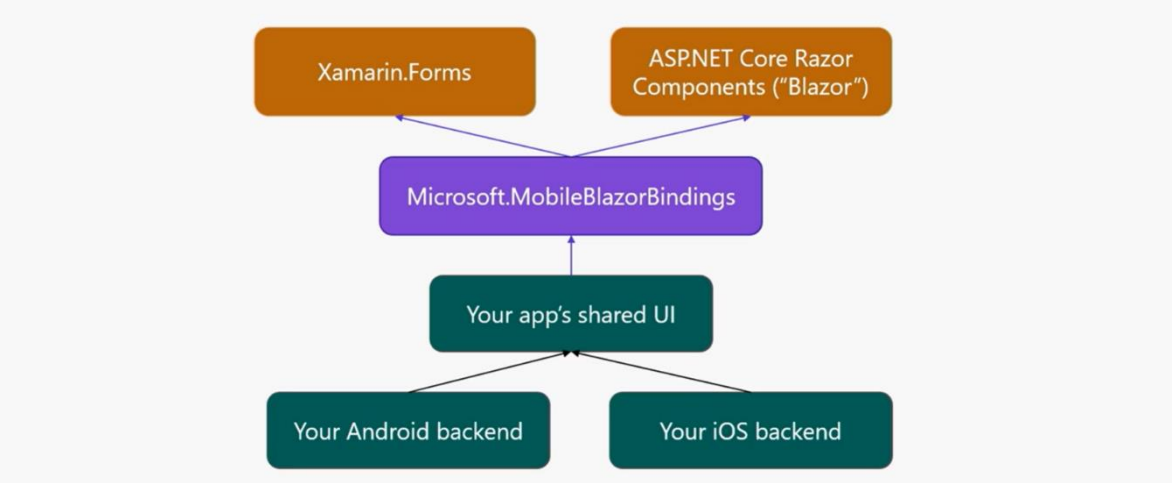
2.1.4.3 Xamarin.Forms

Jedná se o open-source UI Framework, jenž dovoluje vývojářům stavbu aplikací spolu s uživatelským prostředím pro Android, iOS a Windows z jednoho sdíleného kódu. Vývojář vytváří uživatelské prostředí za pomoci XAML markupu spolu s C# kódem běžícím na pozadí. Takovéto uživatelské prostředí je zobrazeno jako prostředí s nativními ovládacími prvky pro každou platformu. (1)

2.1.4.4 Xamarin Blazor

Na konferenci .NET Conf 2020 Microsoft představil experimentální projekt nazvaný „Microsoft Mobile Blazor Bindings“, který umožňuje vývojářům vytvářet nativní mobilní aplikace za použití Blazor a Xamarin. Blazor je framework sloužící pro tvorbu interaktivního uživatelského prostředí na straně klienta v .NETu. Mobile Blazor Bindings je ve své podstatě další způsob tvorby Xamarin.Forms uživatelského prostředí za použití Razor syntaxe místo XAML. (6)

Obrázek 4 popisuje architekturu knihovny Mobile Blazor Bindings. Ta propojuje Xamarin.Forms a Balzor. Díky tomu lze vytvářet uživatelské prostředí sdílené napříč platformami.



Obrázek 4 - Architektura Mobile Blazor Bindings (6)

2.1.5 Výhody

- Jedna technologická základna – Díky tomu, že Xamarin využívá C# spolu s .NET frameworkem, je tímto zredukována většina zdrojového kódu, čímž se urychlí vývojový cyklus a sníží se náklady na více vývojových prostředí.
- Výkon blízký nativnímu – Na rozdíl od hybridních řešení založených na webových technologiích, lze platformovou aplikaci vytvořenou za pomoci Xamarinu klasifikovat jako nativní.
- Využití nativních zkušeností vývojářů – Xamarin.Forms umožňuje uživatelům vytvářet aplikace, které jsou kompilovány do nativních platforem, nicméně vývojář si i tak může zvolit platformní cestu a provádět vývoj, především grafického prostředí nativně.
- Plná hardwarová podpora – Díky funkčnosti aplikace na nativní úrovni, jsou eliminovány problémy s hardwarovou kompatibilitou způsobené využitím pluginů a nejrůznějších API.
- Podpora Microsoftu – Díky podpoře takto velké společnosti, je Xamarin dobrá volba i pro dlouhodobější projekty. Neustálý rozvoj je vidět i na posledním počinu Microsoftu v podobě „Mobile Blazor Bindings“ projektu. (7)

2.1.6 Nevýhody

- Opožděná podpora nejnovějších aktualizací platformy – Je prakticky nemožné, reagovat na změny v platformě bez sebemenšího zpoždění. Avšak i přesto je tato podpora zpožděná v rámci dnů.
- Omezený přístup k open-source knihovnám – I když není multiplatformní vývoj novinkou posledních let, stále se najdou společnosti preferující platformě zaměřený vývoj. Ten je podpořen komunitou a open-source řešeními, které usnadňují vývoj.
- Vysoké náklady na profesionální a firemní využití – Přestože je Xamarin bezplatnou platformou s otevřeným zdrojovým kódem pro jednotlivé vývojáře, může se jeho cena pro společnost velmi zvýšit zakoupením některé z licencí Visual Studia.
- Menší komunita – I přes veškeré výhody multiplatformního vývoje za použití Xamarinu, je stále komunita nativních vývojářů rozsáhlejší a přívětivější. Nicméně i komunita Xamarinu je dostatečně obsáhlá a úspěšná. (7)

2.2 Platformně zaměřený vývoj

2.2.1 Platformy

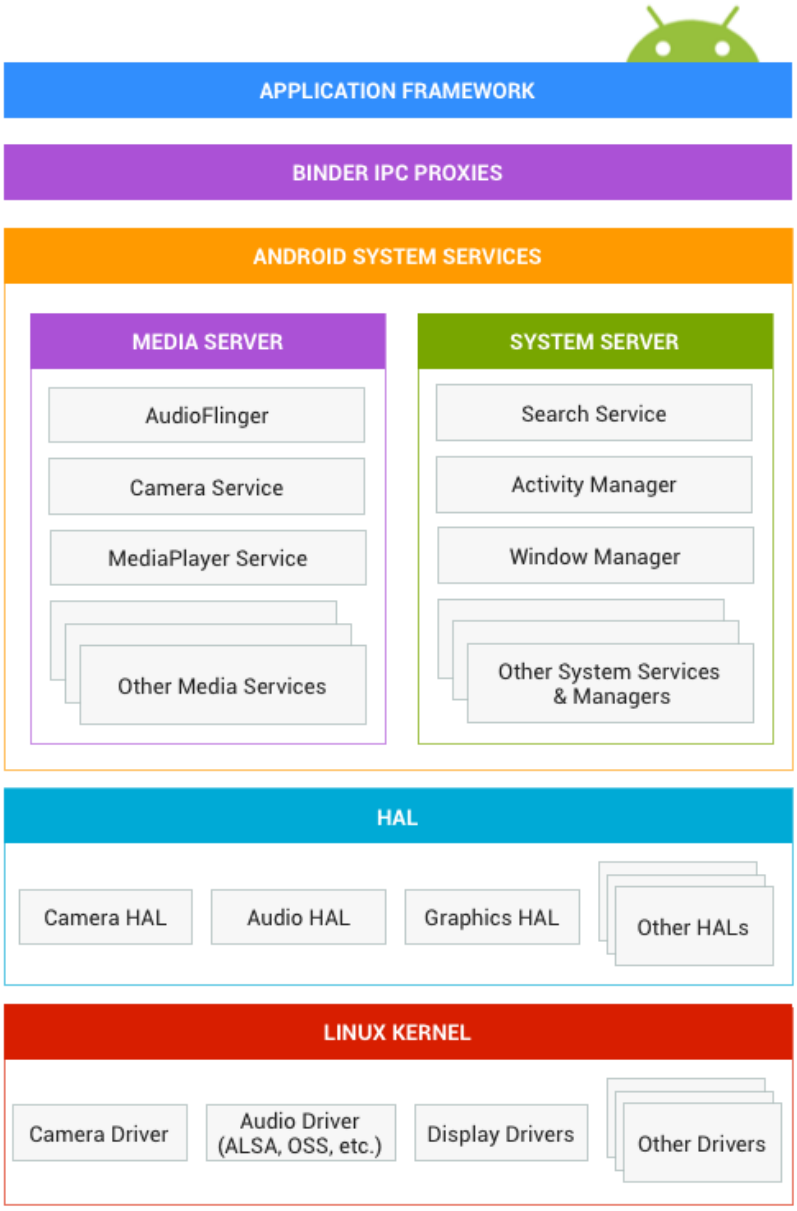
2.2.1.1 Android

Jedná se o mobilní operační systém vytvořený společností Google. Je založen na linuxovém jádře a dalších open-source softwarech. Byl navržen primárně pro dotyková mobilní zařízení, jako jsou mobilní telefony a tablety. Jelikož lze nalézt chytrá zařízení v prakticky jakékoli formě, Google se rozhodl zpracovat formu Androidu ve formě Android TV pro využití v televizorech. Další využití si Android našel v nositelných zařízeních ve formě WearOS nebo v automobilovém průmyslu v podobě Android Auto. Každá z těchto konkrétních implementací je doplněná specifickým uživatelským prostředím. Variace Androidu jsou taktéž využity pro IoT (Internet of Things) a další hardware.

Android byl vyvíjen společností Android Inc., která byla zakoupena Googlem v roce 2005 a později, v roce 2007 byla představena první verze Android. První zařízení pro komerční účely se světu ukázala v září 2008. Od té doby Android prošel značnými změnami a v aktuální době se nachází na verzi 10 Q, spuštěné v září 2019. Zdrojový kód Androidu je

známý jako AOSP (Android Open Source Project) a je distribuován pod licencí Apache License[3]. (8), (9)

2.2.1.1.1 Architektura



Obrázek 5 - Architektura Android (10)

- Aplikační Framework (na obrázku 5 Application Framework) – Zastřešuje nativní knihovny a android runtime. Aplikační framework zahrnuje různá Android API, jako jsou uživatelské prostředí, poskytovatelé dat, zdroje a další. Poskytuje nejrůznější třídy a rozhraní pro aplikační vývoj. (10)

[3] **Apache License** – Licence pro svobodný software vytvořená společností Apache Software Foundation.

- Binder IPC (na obrázku 5 Binder IPC Proxies) – Mechanismus mezi-procesorové komunikace dovoluje aplikačnímu frameworku vytvářet přímá volání systémových služeb Androidu. Tímto je umožněno vysokoúrovňovým API rozhraním pracovat spolu se systémovými službami Androidu. Na úrovni aplikačního frameworku je tato komunikace odstíněna před vývojářem.
- Systémové služby (na obrázku 5 Android System Services) – Jedná se o modulárně zaměřené komponenty, jako je manažer oken, služba vyhledávání nebo manažer notifikací. Funkcionalita vystavená rozhraním aplikačního frameworku, komunikuje se systémovými službami z důvodu přístupu k základnímu hardwaru.
- Hardwarová abstrakční vrstva (na obrázku 5 HAL - Hardware abstraction layer) – Definuje standardní rozhraní pro výrobce hardware a umožňuje tak Androidu jistou nevědomost ohledně nízkoúrovňové implementaci ovladačů. Využití HAL umožnuje implementaci funkcionality, bez nutnosti úpravy systému. Implementace HAL je zabalena v modulech, které jsou načítány systémem v odpovídající chvíli.
- Linuxové jádro (na obrázku 5 Linux Kernel) – Android využívá verzi Linuxového jádra s několika málo speciálními dodatky, jako je například „Low Memory Killer“[4]. Tyto dodatky jsou zde přítomny zejména z důvodu systémové funkcionality a nemají vliv na vývoj ovladačů. (11)

2.2.1.1.2 Vývojové prostředí

Android Studio je oficiálním IDE (integrovaným vývojovým prostředím) pro vývoj aplikací založených na platformě Android. Je založeno na IntelliJ IDEA[5] a na integrovaném vývojovém prostředí Java a obsahuje nástroje pro úpravy kódu a mnoho vývojářských utilit. Více informací k IntelliJ IDEA je dostupných na webu jetbrains.com (12). Pro podporu vývoje aplikací operačního systému Android, používá Android Studio systém sestavování

4 **Low Memory Killer** – systém řízení paměti, který je mnohem agresivnější v zachování paměti.

5 **IntelliJ IDEA** – IDE napsané v Javě pro vývoj software.

založený na Gradle[6], emulátor zařízení s platformou Android, šablony kódů a pro podporu kooperativního vývoje integraci GitHub. Podrobnosti na webu Gradle User Manual. (16)

Android Studio používá funkci Instant Push, díky které jsou změny kódu, nebo zdrojů přímo zavedeny do běžící aplikace. Editor kódu pomáhá vývojáři vytvářet aplikaci za pomoci napovídání doplňování kódu, refaktorací nebo aktivní analýzy kódu. Aplikace vytvořené za použití Android Studia jsou po dokončení zkompilovány do formátu APK, a následně mohou být publikovány do obchodu Google Play.

Poprvé bylo Android Studio představeno Googlem v květnu 2013, ale první stabilní verze byla vydána více než rok poté, a to v prosinci 2014. Android Studio je k dispozici ve verzích pro Linux, Windows i Mac. Jeho předchůdcem bylo vývojové prostředí Eclipce ADT (Android Development Tools), jež bylo plně nahrazeno právě Android Studiem. Toto prostředí je volně dostupné přímo z oficiálních stránek Google. (13)

2.2.1.2 iOS

Systém iOS je známý tím, že slouží jako základní software, který uživatelům mobilních zařízení značky Apple, jako jsou iPhone, iPad, iWatch a další, umožnuje interakci pomocí dotykových gest. Ta jsou ve většině případů prováděna na kapacitních dotykových obrazovkách s možností provádění gest více prsty poskytujících rychlou odezvu. iOS dominuje severoamerickému trhu s podílem okolo 60 % celkových počtů uživatelů. (14)

Historie iOS sahá do roku 2007, kdy se tato rozsáhlá platforma nazývala iPhone OS a vypadala naprosto rudimentálně. V té době neobsahoval funkce ani nástroje, jako jsou App Store, multitasking, či prostou funkci složek. V průběhu let se platforma transformovala v bohatý operační systém ve verzi iOS 13, který je aktuální v době publikace této práce. (15)

Vývojáři mohou pomocí sady pro vývoj softwaru iOS (SDK) vytvářet aplikace pro mobilní zařízení Apple. Tato sada obsahuje nástroje a rozhraní pro vývoj, instalaci, provoz a testování aplikací. Nativní aplikace lze psát za použití systémových frameworků s pomocí programovacího jazyka Objective-C. Součástí sady iOS SDK jsou nástroje Xcode, které zahrnují integrované vývojové prostředí (IDE) pro správu aplikačních projektů, grafický nástroj pro vytvoření uživatelského rozhraní a nástroj pro ladění a analýzu výkonu. Obsahuje také emulátor prostředí iOS, který umožňuje vývojářům testovat aplikace bez fyzických zařízení přímo na svých vývojářských strojích Mac. Samozřejmostí je také vývojářská

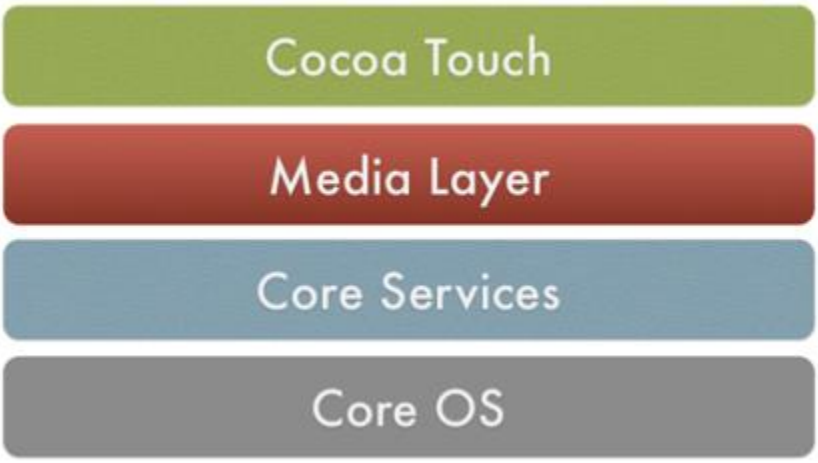
6 **Gradle** – Automatizovaný systém pro sestavování aplikací distribuovaný pod open-source licencí.

knihovna, poskytující veškeré informace potřebné při vývoji, jako je dokumentace a užitečný referenční materiál. (14)

2.2.1.2.1 Architektura

iOS disponuje typem architektury, jenž je označována jako vrstvená. Na nejvyšší úrovni funguje iOS jako prostředník mezi základním hardwarem a aplikacemi, vytvořenými vývojáři. Aplikace jako takové spolu se základním hardwarem napřímo nijak nekomunikují. Uvedená komunikace se děje prostřednictvím kolekce velmi pečlivě nadefinovaných systémových rozhraní. Tato rozhraní velmi usnadňují tvorbu aplikací, které neustále pracují na zařízeních s nejrůznějšími hardwarovými schopnostmi.

Spodní vrstvy poskytují základní služby, na které všechny aplikace spoléhají. Horní vrstvy poskytují sofistikované služby související s grafikou a uživatelským rozhraním. (17)



Obrázek 6 - Architektura iOS (17)

- Cocoa Touch – Tato vrstva je nejvyšší vrstvou architektury iOS. Obsahuje některé klíčové frameworky, na kterých je závislá většina nativních aplikací, přičemž nejvýznamnější je UIKit framework[7]. Informace o využití UIKitu jsou dostupné z vývojářské dokumentace společnosti Apple. (18)

  Poskytuje vývojářům velké množství vysokoúrovňových funkcí, jako je automatické rozložení prvků aplikace, rozpoznávání gest a další. Kromě UIKitu obsahuje mimo jiné Map Kit, Event Kit a Message UI frameworku.

- Media Layer – Grafika, zvuk a video jsou zpracovávány vrstvou media layer. Obsahuje řadu klíčových technologií, jako například Core Graphics, OpenGL

[7] **UIKit framework** – Vytváří a spravuje událostmi řízené grafické uživatelské rozhraní.

ES (Embedded Systems), OpenAL a další. Mimo jiné obsahuje také velké množství frameworků, včetně Assets Library frameworku sloužícím pro přístup k fotografiím a videím uložených v paměti zařízení.

- Core Services – Vrstva Core services má na starosti správu základních systémových služeb, které nativní aplikace iOS používají. Vrstva Cocoa Touch závisí na některých službách této vrstvy. Dále poskytuje řadu nezbytných funkcí, jako jsou blokové objekty, In-App Purchase a uložiště iCloud.

  Jednou z velmi užitečných funkcí Core Services je ARC (Automatické počítání referencí), jenž je funkcí kompilátoru, která zjednodušuje proces správy paměti v Objective-C.

  Dalším důležitým frameworkem, který je úzce spjatý s Foundation frameworkem je Foundation framework na bázi C, nebo zkráceně Core Foundation framework. Ten umožňuje různým knihovnám a frameworkům navzájem sdílet data a kód. Jednou z nejznámějších funkcí je toll-free bridging neboli přemostění bez poplatků, jenž umožňuje nahrazení objektů z Cocoa vrstvy za objekty Core Foundation.

- Core OS – Většina funkcí poskytovaných třemi vrstvami vyšší úrovně je postavena na vrstvě Core OS a jejích nízko úrovňových funkcích. Vrstva Core OS poskytuje několik frameworků, které aplikace využívají napřímo, jako jsou například Accelerate a Security frameworky. Tato vrstva také zapouzdřuje jádro a nízkoúrovňové UNIXové rozhraní, ke kterému aplikace z bezpečnostních důvodů nemají přístup. Nicméně, za použití knihovny libSystem, lze využívat některé nízkoúrovňové funkce. (17)

2.2.1.2.2 Vývojové prostředí

Xcode je integrované vývojové prostředí (IDE) společnosti Apple, které je využíváno k vytváření aplikací pro produkty společnosti Apple, včetně iPadů, Iphonů, Apple watch a Mac. Xcode poskytuje nástroje pro správu celého pracovního postupu vývoje – od vytvoření, testování, optimalizace až po odeslání finální aplikace do obchodu App Store. První verze byla poprvé vydána v roce 2003, od té doby se mnohé změnilo a aktuální verze 11.3 je prostřednictvím Mac App Store zdarma pro uživatele MacOS Catalina.

Rozhraní Xcode integruje úpravy kódu, návrh uživatelského rozhraní, správu aktivit, testování a debuggování v jednom pracovním prostoru. Prostředí během práce samostatně překonfiguruje jeho obsah. Uživatelské prostředí je přizpůsobitelné v mnoha ohledech.

Tvorba kódu, ať už za použití jazyka Swift, Objektového C, C++ nebo kombinace, je doprovázena kontrolou chyb, ať už syntaktických, či funkčních. Takto zaznamenané chyby jsou v kódu zvýrazněny, a pokud je to možné editor nabídne způsob opravy. Díky inteligentnímu dokončování kódu je psaní velmi rychlé a efektivní. Samozřejmostí jsou operace jako je vyhledávání a refaktorace kódu, který umožňuje provádět rozsáhlé a rychlé změny.

Většinu času věnovaného tvorbě aplikace vývojář stráví na kódovacích úkolech. Nicméně při vývoji aplikace pro App Store je nutné provádět řadu administrativních úkolů. Kromě Xcode je třeba použít webový nástroj pro správu smluv, nastavení daňových a bankovních informací a pro získávání nejrůznějších finančních zpráv ohledně tvořených aplikací. Tento nástroj se jmenuje iTunes Connect. Odeslání aplikace na App Store je vícestupňový proces, který začíná právě v iTunes Connect registrací aplikace a poskytnutím potřebných informací pro publikaci. V prostředí Xcode se vytvoří archiv s projektem, který se odešle do App Store, kde proběhne proces schválení a po tomto schválení může být aplikace uvolněna, zadáním data uvolnění v iTunes Connect. (19)

2.2.2 Výhody

- Nativní aplikace jsou tvořeny pro konkrétní operační systém, díky čemuž plně využívá procesovací rychlosti zařízení, což zajišťuje hladkou práci, vysokou provozní rychlost a bezchybný výkon.
- Velké možnosti designu a „user experiences“. Klasickým případem toho je implementace sofistikovaného uživatelského rozhraní s propracovanou grafikou a animacemi.
- Jednoduchá integrace s hardwarovými prostředky jako jsou GPS, kamera nebo dotykový display.
- SDK pro vývojáře poskytuje polo připravená, pravidelně aktualizovaná řešení a knihovny, které správně spolupracují s daným operačním systémem. (20)

2.2.3 Nevýhody

- Vysoké počáteční investice, které jsou způsobeny zdvojeným vývojem jedné aplikace. Tento problém se ve většině případů, alespoň ze začátku řeší odříznutím jedné platformy, a tedy části trhu.
- Náklady na údržbu a aktualizace rostou úměrně s řadou podporovaných operačních systémů.
- Obsah aplikace je například oproti webové aplikaci přístupný pouze nativními prostředky, nebo dedikovanou webovou aplikací, což by ale znamenalo další vývoj a potencionální údržbu. (20)
- Množství udržovaného kódu. Jelikož je při nativním vývoji prakticky nemožné sdílení kódu napříč vývojovými základnami, je množství redundantního kódu velmi velké.

2.3 Alternativy

2.3.1 React Native

2.3.1.1 Obecně

React Native je framework pro vývoj aplikací, ve kterém lze používat standartní webové technologie pro vytvoření kýžené aplikace. To znamená, že využívá HTML, JavaScript a CSS. Je založen na React frameworku vytvořeném společností Facebook, jenž je oblíbeným webovým frameworkem pro vývoj aplikací. Hlavním rozdílem mezi React Native a původním Reactem je skutečnost, že React je určený a cílí převážně na webové prohlížeče, zatímco React Native nikoli.

React Native umožňuje vytváření mobilních aplikací, které vypadají a pocitově se chovají mnohem více jako nativně vyvinuté než klasické webové aplikace. To je díky skutečnosti, že technologie na pozadí je nativní, což umožnuje vývojáři tvořit aplikace, při využití zkušeností a znalostí z webového vývoje. (21)

2.3.1.2 Historie

Počátky React Native se datují od roku 2011, kdy byl vedlejším produktem společnosti Facebook. Byl vytvořen vývojářem Facebooku Jordanem Walkem, který ho poprvé využil pro Facebookové novinky. Později byl využit pro následný vývoj Instagramu a byla mu vytvořena open-source licence v květnu 2013.

Netrvalo dlouho, aby si ve vývojovém oddělení Facebooku uvědomili, že React jako takový může také vyřešit problémy mobilního vývoje. S vývojovou komunitou podporující React, byl React Native přirozeným vývojem. V lednu 2015 bylo poprvé představeno veřejné preview React Native. Chvíli na to byl představen jako veřejně dostupný vývojový nástroj na GitHubu. (21)

2.3.1.3 Výhody

- Na první pohled jsou aplikace typicky více podobné těm napsaným v nativním kódu.
- Protože je založen na Reactu, je zde mnoho vývojářských materiálů a obecné znalosti, která jsou velice přínosné.
- Jak u Google, tak u Apple při nasazování aplikací na store, prochází aplikace procesem schválení. Nicméně obě společnosti umožnují aplikacím načítat změněné JavaScriptové knihovny. Tím pádem jakékoli změny v aplikaci nepodléhají schvalovacímu procesu. (21)

2.3.1.4 Nevýhody

- Jelikož se React Native kód jen nerenderuje do WebView[8] a je více svázaný se základním API operačního systému, mohou zde nastávat jisté prodlevy, během kterých nepodporuje nejnovější verzi aktuální platformy.
- Debuggování může být velice obtížné, jelikož React Native obsahuje více vrstev abstrakce.
- Nutnost naučit se pracovat s JSX (JavaScript XML). (21)

2.3.2 Ionic

2.3.2.1 Obecně

Ionic je framework, jenž umožňuje vývojářům vytvářet hybridní aplikace[9] za použití webových technologií, jako jsou HTML5, CSS a JavaScript. Ionic Framework je snadno integrovatelný s Angular, React či Vue. Dále poskytuje komponenty, které je možné využít k tvorbě funkcí, jež jsou na první pohled nativní. Na Ionic je možné v první řadě nahlížet

[8] **WebView** – UI komponenta sloužící pro zobrazení webového obsahu uvnitř aplikace.

[9] **Hybridní aplikace** – Mobilní Aplikace běžící v prostředí WebView.

jako na knihovnu umožňující tvorbu hybridních mobilních aplikací. Jsou dostupné všechny základní funkcionality jako modální okna, gesta a pop-upy a tuto základní sadu lze rozšířit o nové funkce anebo ty stávající přizpůsobit vlastním potřebám.

Ionic jako takový neposkytuje možnosti pro komunikaci s vlastnostmi zařízení, jako jsou GPS nebo kamera, místo toho pracuje bok po boku s Cordovou, která toto umožňuje.

Tento framework je dodáván s vestavěným JavaScriptovým řešením pro stavbu uživatelského prostředí s názvem AngularJS. Díky právě Angularu je možné vytvářet mobilní aplikace, při použití techniky SPA (Single Page Application). (22)

2.3.2.2 Historie

Společnost Drifty Co. Představila Ionic v roce 2013. V listopadu 2013 vydala první alfa verzi tohoto frameworku. Poslední aktuální verze Ionic je verze v4+, která byla vypuštěna v roce 2019. Historicky byl Ionic 1 vyvinut pro tvorbu mobilních aplikací za pomoci Angularu.

Ionic 1 byl představen na konci roku 2013. Zpočátku nepoužíval webovou komponentu. Namísto toho používal Angular jako komponentu, což umožňovalo využití pouze uvnitř Angularové aplikace.

Ionic 2, který byl představen v roce 2016, navazuje na svou předchozí verzi v podpoře Angular 2. Tato verze Angularu je zásadně odlišná od své předchůdkyně. Stejně jako verze jedna přinesl Ionic 2 pouze podporu Angularových aplikací a nikoli webových komponent.

Ionic 3 nepřinesl prakticky žádný výrazný posun a pouze vylepšil některé, již existující funkce. Tvůrci si však začali uvědomovat, že pouhá podpora Angularu nestačí. Tato verze byla vydána v roce 2017.

Převratem ve verzích Ionic frameworku byla verze 4, jenž je založena na webových komponentách, díky čemuž přináší vyšší podporu běžných frameworků. Verze Ionic nyní vycházejí v šestiměsíčních cyklech a jsou založeny na technologii PWA (Progressive Web Apps), díky které podporuje všechny frameworky. (23)

2.3.2.3 Výhody

- Rychlý vývoj a rychlost uvedení na trh v porovnání s nativním vývojem.
- Lze vyvíjet hlavně v prohlížeči, s výjimkou nativních funkcí telefonu, které je nutné odladit přímo na zařízení.
- Není nutná znalost Java, Swift nebo Objective-C, stejně jako u všech ostatních hybridních přístupů.
- Je k dispozici spousta pluginů umožňujících využívání vlastností zařízení, spolu se spoustou komponent pro tvorbu uživatelského prostředí. (24)
- Volně vázané komponenty. Vývojář se může sám rozhodnout, jestli použije vlastní komponenty, při tvorbě aplikace.

2.3.2.4 Nevýhody

- Nativní pluginy nejsou stabilní a mohou být ve vzájemném konfliktu. Nicméně, jsou nezbytné k vyřešení nedostatku funkcí, které chybí v základu Ionic.
- Debuggování aplikace postavené na Ionic je občas velmi náročné a vyžaduje více času kvůli nejasným chybovým hláškám.
- Občasné problémy při sestavování finální aplikace bez specifických důvodů. (24)

2.3.3 Flutter

2.3.3.1 Obecně

Jedná se o SDK sadu pro vývoj mobilních aplikací, vytvořenou společností Google, díky níž lze současně vyvíjet jak pro Android, tak pro iOS bez potřeby udržovat obě kódové základny. Kromě toho mohou být aplikace kompilovány pro nadcházející operační systém Fuchsia od stejné společnosti.

Skládá se ze dvou hlavních částí. Jedná se tedy o SDK, což je kolekce vývojářských nástrojů, které výrazně pomáhají s vývojem aplikací. Mezi tyto nástroje patří například nástroj pro kompilaci do nativního strojového kódu.

Druhou částí je UI framework, tedy kolekce prvků uživatelského rozhraní, které lze přizpůsobit podle vlastní potřeby. Mezi tyto prvky patří například tlačítka, textové vstupy a další.

Pro vývoj pomocí Flutteru je využíván programovací jazyk nazvaný Dart. Ten byl vytvořen společností Google v říjnu 2011 a v posledních letech prošel výrazným zlepšením. Dart se zaměřuje na front-end vývoj a může být použit k vytváření mobilních a webových aplikací. (25)

2.3.3.2 Historie

Flutter byl poprvé oznámen v roce 2015 na Dart Developer summitu, pod názvem „SKY“. Alfa verze byla uvedena na trh v květnu 2017. Později v září 2018 se Google rozhodl vypustit druhou preview verzi Flutteru.

Verze Flutteru 1.0 byla poprvé spuštěna 5. prosince 2018, po velkém úspěchu některých pilotních aplikací jako jsou Alibaba nebo Reflecty.

Při spuštění verze 1.0 společnost Google vydala aplikaci s názvem „Historie všeho“, což je vlastně časová osa od Velkého třesku až do současnosti. Tato aplikace měla dokázat, že Fluttrerové aplikace jsou schopny spouštět animace s rychlostí 120 snímků za sekundu a také svou použitelnost ve světě vývoje multiplatformních aplikací. (26)

2.3.3.3 Výhody

- Opětovné načtení kompilovaného kódu za provozu aplikace. Této funkci se říká „Hot reload“ a umožňuje především úpravy v UI znovu načítat okamžitě. Existují však i jisté limitace kdy je potřeba celou aplikaci sestavit znovu. Jedná se především o úpravy datových typů a dalších.
- Jelikož Flutter není závislý na platformě, díky svým vlastním widgetům a designovým prvkům, aplikace vypadá na obou platformách stejně s jen jednou kódovou základnou.
- Pokud je aplikace stejná pro obě platformy, znamená to méně testování. Proces zajištění kvality může být rychlejší. Díky jedné kódové základně vývojáři vytvářejí automatické testy pouze jednou. Navíc mají specialisté na zajištění kvality méně práce, protože mají pouze jednu aplikaci ke kontrole. Pokud se však aplikace liší, je samozřejmě nutné je otestovat na obou platformách.

- Vzhled aplikace je stejný na nových i na starších systémech Android nebo iOS. Podpora starších zařízení není zpoplatněna. (27)

2.3.3.4 Nevýhody

- Flutter je relativně nový, a tak nepodporuje zdaleka všechny funkce, které jsou dostupné v nativních knihovnách. Vývojáři si musejí komponenty sestavit sami což je časově, a tudíž finančně náročné. (27)
- Jelikož je Dart vcelku nový jazyk, je pro začínající vývojáře složitější najít pomoc online oproti zaběhlým jazykům s obsáhlou developerskou komunitou.
- Největší výhoda Flutteru je zároveň i jeho velkou nevýhodou. Jedná se o fakt, že aplikace využívá univerzální multiplatformní komponenty pro vzhled. To má za následek to, že aplikace ztrácí na nativním vzhledu a působení. Jde ale spíše o individuální případy.
- Protože aplikace využívají vestavěné widgety a nikoli widgety platformy, jejich velikost je obvykle výrazně větší. I když Google neustále pracuje na optimalizaci, aktuální nejmenší aplikace se velikostně nikdy nedostane pod 4 MB. (28)

3 Srovnání a vyhodnocení nástrojů

V úvodu této kapitoly je připomenuta hlavní myšlenka, či téma předkládané práce. Předkladatelovým (autorovým) záměrem je provést srovnání vývojového nástroje Xamarin s platformě zaměřeným vývojem. Obsahovou náplní kapitoly 3 je zamýšlené srovnání.

Jako cílová platforma pro provedení srovnání byl zvolen Android. Důvodem této volby je větší dostupnost platformy. Testování iOS aplikací je díky uzavřenosti platformy komplikovanější. S ohledem na skutečnost, že se hodnoty srovnávaných ukazatelů pro platformy Android a iOS zásadně neliší, považuje předkladatel (autor) omezení na platformu Android za akceptovatelné a plně podporující záměr práce.

3.1 Popis nástrojů

Pro vývoj nativních aplikací na platformě Android se nabízí několik možných alternativ. Tato práce je zaměřena na nejpoužívanější z nich. Tím je prostředí nazývající se Android Studio. Jeho podrobnější popis byl proveden v kapitole 2.2.1.1.2 („Vývojové prostředí“).

Protože je Xamarin rozšířením .NETu, není jistě překvapením, že jej lze nalézt jako volitelnou část Microsoft Visual Studia (dále jen Visual Studio).

Visual Studio je integrované vývojové prostředí (často se také používá označení zkratkou. IDE podle anglického Integrated Development Environment) vyvinuté společností Microsoft, za účelem vývoje různých aplikací. Tyto aplikace mohou být webové, konzolové, mobilní nebo desktopové. Jelikož se nejedná o IDE specifické pro jeden daný programovací jazyk lze aplikace psát například v C#, C++, VB (Visual Basic), Pythonu nebo JavaScriptu. Poskytuje podporu 36 různých programovacích jazyků a je k dispozici jak pro Windows, tak pro MacOS.

První verze byla vydána v roce 1997 a byla pojmenována Visual Studio 97. Aktuální vydaná verze je Visual Studio 2019. Dělí se na tři varianty, dle licenčních podmínek. Tyto verze jsou:

- **Comunity** – Varianta, která je zdarma pro nekomerční účely.
- **Professional** – Jedná se o komerční vydání, které je ideální pro menší společnosti a jednotlivce.

- **Enterprise** – Řešení pro společnosti s velkým vývojovým týmem. Toto řešení poskytuje vysokou škálovatelnost. (29)

3.2 Instalace prostředí

Tato kapitola vysvětluje úplný začátek vývoje, a to samotnou instalaci vývojového prostředí. Je zde sledováno:

1. Vytvoření jednoduché „Hello World“ aplikace.
2. Rychlost sestavení vytvořené aplikace.
3. První načtení prostředí s vytvořeným projektem takzvaný „Studený start“.
4. Velikost nainstalovaného prostředí.
5. Subjektivní pohled autora na průběh instalace, a to v polaritě jednoduchost – složitost. Jedná se o kvalitativní ukazatel a v celkovém hodnocení má informativní. Je k němu přihlíženo pouze v případě rovnosti všech testovaných kritérií.

Testování je prováděno ve stejných podmínkách a na stejném zařízení. Veškerá kritéria, která nelze kvantitativně vyjádřit (například v časových jednotkách) jsou hodnocena bodově. Autor měl k dispozici pět bodů, které dle výsledků rozdělil mezi testovaná prostředí. Počet bodů byl zvolen z důvodu vyřešení rovnosti při hodnocení. Všechny výsledky jsou zpracovány v kapitole 3.2.1 („Vyhodnocení instalace“).

Prvním krokem při provádění srovnání bylo vytvoření jednoduché aplikace „Hello World“ pro ověření funkčnosti prostředí. Protože Visual Studio i Android Studio sdílí nástroje pro emulaci zařízení, byl čas potřebný pro spuštění emulátoru z výsledného hodnocení vynechán. Hodnocení tohoto kroku rozděleno do dvou částí, z nichž jedna popisuje kvantitativní část a druhá část subjektivní. V kvantitativní části je měřena rychlost vytvoření projektu, s předpokladem předchozího obeznámení se autora s tímto procesem. Subjektivní část hodnotí právě předešlé obeznámení autora s daným procesem a jeho pocity ze složitosti vytvoření ukázkové aplikace. Hodnocení subjektivní části je vyjádřeno bodovým ohodnocením, jak bylo popsáno výše.

Druhým sledovaným ukazatelem byla rychlost sestavení vytvořené aplikace. Pro tento test byla použita aplikace vytvořená v prvním kroku. Rychlost, s jakou lze vyvíjenou aplikaci sestavit do spustitelné podoby, je pro většinu vývojářů klíčová. Čím více času vývojář stráví čekáním na kompilaci aplikace, tím méně času stráví potřebným vývojem.

Ve třetím kroku srovnání byl měřen tzv. „studený start“ vývojového prostředí. Tento test byl proveden měřením času potřebného ke spuštění prostředí s aplikací vytvořenou v prvním kroku. Test byl pro každé prostředí proveden na čerstvě spuštěném zařízení a měření bylo opakováno 5krát. Výsledná hodnota je založena na prostém aritmetickém průměru výsledků dílčích měření.

Čtvrtý krok je zaměřen na porovnání velikosti instalovaného prostředí. Tento ukazatel lze jednoduše měřit množstvím MB, které zabírají nainstalovaná prostředí. Do výsledku měření se promítá přítomnost dodatečných online komponent, nezbytných pro funkčnost prostředí v testovaném rozsahu. Obě prostředí lze ovšem provozovat offline, a není tedy nutné přihlížet k dodatečným online zdrojům. Veškeré soubory potřebné pro provoz prostředí jsou uloženy na konkrétním zařízení spolu s prostředím.

Pátý a zároveň poslední krok srovnání byl zaměřen na výše popsané kvalitativního hodnocení samotného testování a průběhu instalací. Pro tento bod je nutné definovat zkušenosti autora. Autorem je vývojář s několikaletými zkušenostmi s vývojem na platformě .NET s využitím jazyka C#. I když je autor především C# vývojář, ani Android a Java mu nejsou cizí.

3.2.1 Vyhodnocení instalace

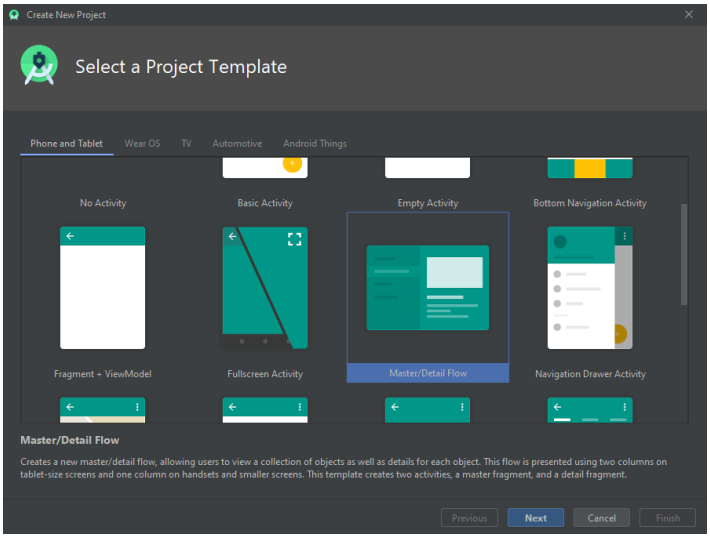
Testování bylo provedeno na nejpoužívanějších vývojových prostředích, a to na Visual Studiu od společnosti Microsoft a na Android Studiu od společnosti Google. Popis konkrétně použitých verzí uvádí následující seznam.

- **Visual Studio 2019 pod licencí Comunity**. Tato verze je zdarma pro nekomerční účely a je dostupná z webových stránek Microsoft: https://visualstudio.microsoft.com.
- **Android Studio 3.6.2**. Android Studio je na rozdíl od Visual Studia zdarma pro komerční i nekomerční účely. Použitá verze je dostupná z webových stránek Google: https://developer.android.com/studio.

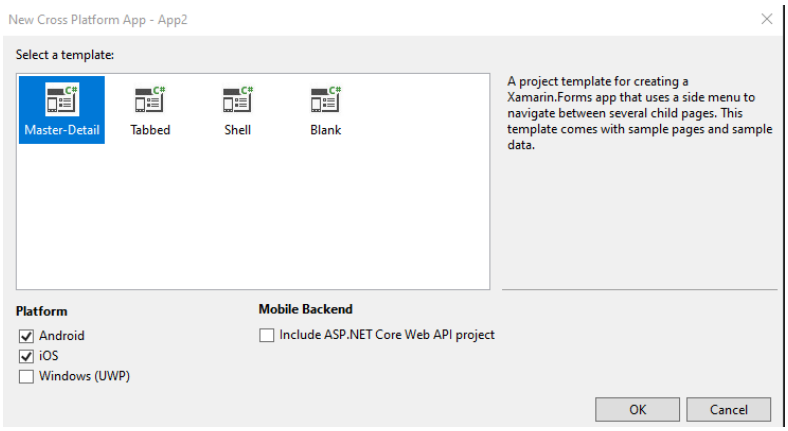
V první části prvního kroku bylo měřeno časové kritérium, rychlost vytvoření projektu. Konečný výsledek je založen na pěti měřeních, na aritmetickém průměru naměřených hodnot. Bylo provedeno pět vytvoření základních projektů neboli „Hello World“ aplikací. Projekt v Android Studiu byl vytvořen skoro dvakrát pomaleji oproti projektu tvořeném v prostředí Visual Studio.

Jelikož síla Xamarin.Forms je především v jeho multiplatformním zaměření, byl pro zajímavost proveden test, kdy byla vytvořena zároveň i aplikace pro iOS. Výsledky tohoto testu vykázaly výrazně delší časy a lze konstatovat, že došlo až k dvojnásobnému zpomalení vytvoření aplikace.

Druhou částí prvního kroku je kvalitativní hodnocení založené na subjektivním pohledu autora na tvorbu aplikací a intuitivnost prostředí. Visual Studio nabízí velmi omezené množství startovacích/výukových projektů. Nicméně tyto „Hello World“ aplikace nabízejí velké množství už implementovaných řešení, což přináší uživateli výhodu v učení se tvorbě složitějších aplikací. Android Studio nabízí rozmanitější škálu předdefinovaných řešení, ale pokud vývojář hledá inspiraci pro vytvoření opravdu komplexní aplikace, bude muset pro výuku použít jiné materiály a zdroje. Jelikož je hodnotící škála od jedné do pěti, autor se rozhodl na základě subjektivních pocitů přidělit body ve prospěch Visual Studia, pro jeho komplexnost projektů. Konkrétní rozdělení bodů viz tabulka 1 na straně 30. Na obrázcích 7 a 8 lze vidět rozdílnost prostředí pro definici projektů v obou studiích.



Obrázek 7 - Vytvoření projektu Android Studio (Vlastní tvorba)



Obrázek 8 - Vytvoření projektu Visual Studio (Vlastní tvorba)

Druhý test spočíval v měření času potřebného pro kompilaci, neboli sestavení vyvíjeného projektu. Test byl proveden měřením času, který konkrétní prostředí spotřebovalo pro sestavení projektu. Po kompilaci projektu je tento údaj v obou prostředích, při správném nastavení dostupný v logu nebo output okně, které zaznamenává výsledky kompilace.

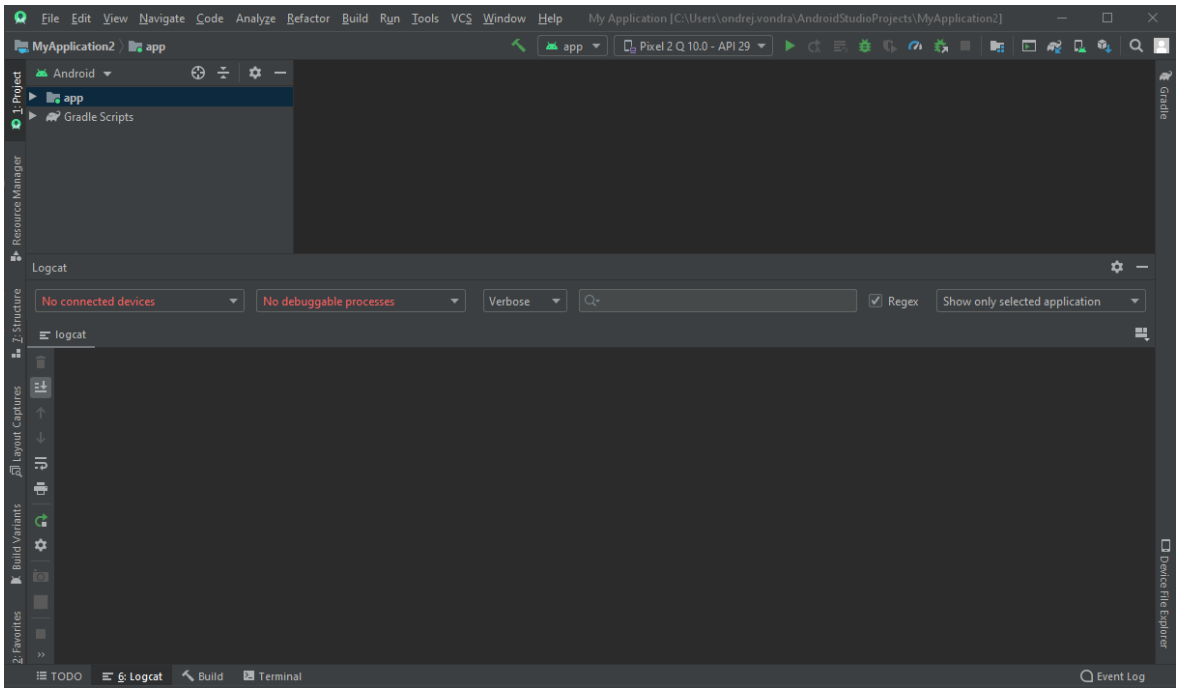
Výsledná hodnota je založena na aritmetickém průměru pěti měření provedených pro každé testované prostředí. Konkrétní hodnoty viz tabulka 1. Pro objektivnost testu byla v obou prostředích provedena rekompilace projektu, čímž byla vynucena kompilace celého projektu a ne, jen změněných částí. Zajímavou skutečností jsou rozdíly mezi prvním a opakovanými sestaveními, kdy sestavení projektu v prostředí Visual Studio trvalo ve všech měřených případech téměř stejně a rozdíly byly v rámci milisekund. Zatímco v prostředí Android Studia se první měření od dalších lišilo v rámci sekund. Podle údajů z logu Android Studio provádělo všechny operace jako při prvním sestavení.

Rychlejší sestavení poskytuje prostředí Android Studio díky větší jednoduchosti projektu. Nutno však podotknout, že při prvním načtení se rychlost sestavení projektu pohybovala kolem 10 sekund, což ve srovnání s Visual Studiem, kde se rychlost pohybuje kolem 13 sekund, není tak velký rozdíl. Rozdíl nastává až ve chvíli, kdy má Android Studio provádět další kompilace po úvodním přednačtení.

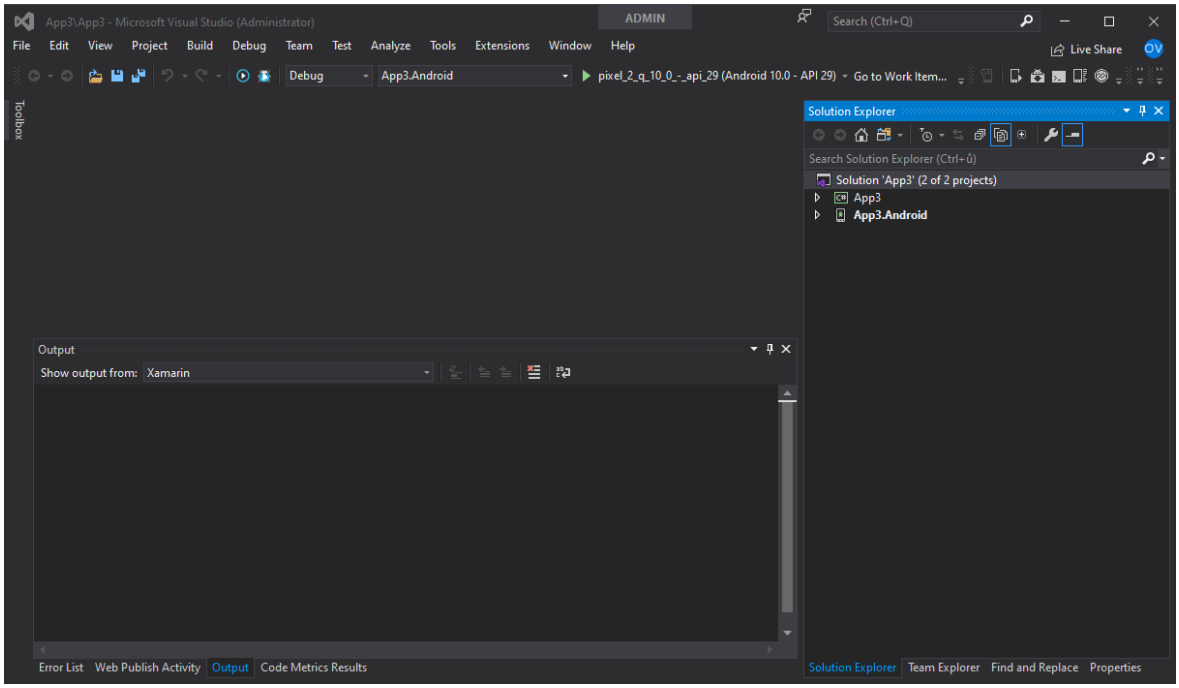
Třetí hodnocený ukazatel je rychlost načítání projektů. Načtením projektu je myšleno spuštění studia i s projektem do stavu, kdy s ním vývojář může bez jakýchkoli prodlev začít pracovat. Jelikož Visual Studio i Android Studio nabízejí načtení projektu přímo při startu prostředí, je tento test snadněji proveditelný. Opět byl měřen čas spuštění, ale tentokrát je Visual Studio o pár sekund rychlejší než Android Studio. Tyto časy jsou velmi důležité z pohledu vývojáře, který často přepíná mezi projekty a potřebuje mít svižné prostředí.

Hodnota ukazatele, kterým byl měřen čtvrtý krok srovnávání, byla sycena údajem o velikosti místa zabíraného nainstalovanými prostředími. Test byl proveden sledováním obsazenosti disku po instalaci jednotlivých studií. Ve výsledku tohoto testu se odráží složitost obou prostředí, jelikož Visual Studio je komplexní nástroj pro vývoj nejrůznějších aplikací, jeho velikost nepatří k nejmenším. Naopak zaměření Android Studia na jednu platformu způsobuje, že velikost nainstalovaného prostředí je velmi malá. Nároky Android Studia na dostupné místo jsou dokonce až pětkrát menší než nároky Visual Studia. Obě prostředí byla nainstalována v nejmenší možné variantě, pouze pro spuštění základního projektu, aby byly výsledky srovnatelné. Do instalací nebyly zahrnuty emulační prostředky pro zařízení Android. Zajímavý je fakt, že i když pro svou instalaci požadovalo Visual Studio 6,5 GB volného místa, jeho výsledná velikost byla následně o 800 MB větší.

Pátým a posledním krokem srovnání je pohled autora na průběh instalace a testování. Ani jednomu z prostředí nelze vytýkat velké nedostatky. Obě instalace probíhaly bez sebemenších problémů a jejich průběh byl příjemně intuitivní. Jak Visual Studio, tak Android Studio nabízí v rámci prvního nastavení možnosti přizpůsobení vzhledu a základního ovládání. Na obrázku 9 je zachyceno Android Studio při svém prvním načtení testované aplikace. Při pohledu na obrázek 10 je vidět, že obě prostředí ve svém tmavém módu jsou velmi podobná. Hodnocení vychází již dříve popsaného způsobu hodnocení pomocí bodového rozdělení. Jelikož autor oceňuje možnosti a rozsáhlost Visual Studia a dále je mu jakožto C# vývojáři bližší, rozhodl se hodnotit třemi body. Android Studio je však z hlediska intuitivnosti obstojným protivníkem, co se intuitivity týče. Jeho nastavení je velmi jednoduché a rychlé. Proto bylo hodnoceno zbývajícími dvěma body.



Obrázek 9 - Prostředí Android Studio (Vlastní tvorba)



Obrázek 10 - Prostředí Visual Studio (Vlastní tvorba)

Protože testy založené na měření času byly provedeny pomocí stopování, mohly být jejich výsledky zkresleny vlivem lidského faktoru. Proto bylo pro minimalizaci tohoto zkreslení vždy provedeno několik měření, konkrétně 5 pro každý test a prostředí. Výsledné hodnoty t_v použité pro závěrečné hodnocení je tak založeny na aritmetických průměrech dílčích měření dle (3.1).

$$t_v = \frac{1}{n}\sum_{i=1}^{n} t_i \tag{3.1}$$

Hodnoty získané těmito měřeními jsou pro účely těchto testů více než dostačující, jelikož se zkreslení pohybuje v rámci jednotek sekund.

Výsledky všech provedených testů jsou pro přehlednost zaneseny do tabulky 1.

	Visual Studio	Android Studio
Vytvoření projektu (Kvantitativní)	9,39 s	18,51 s
Vytvoření projektu (Kvalitativní)	3 body	2 body
Rychlost sestavení projektu	13,13 s	4,8 s
Rychlost Prvního načtení	8,77 s	11,60 s
Velikost nainstalovaného prostředí	7,3 GB	1,4 GB
Pohled autora	3 body	2 body

Tabulka 1 - Srovnání prostředí (Vlastní tvorba)

Při pohledu na výsledky v tabulce 1, nelze s jistotou konstatovat, že některé z prostředí je hned na první pohled lepší. Visual Studio je velké, pomalu se instaluje, ale má svižné načítání projektů a rychlejší zpracování tvorby nové aplikace se šablonou. Bohužel svižné načítání projektů kompenzuje delším sestavováním projektu, díky kterému je v konečném důsledku práce v tomto pomalejší. Oproti tomu Android Studio je výrazně menší a rychlejší pro instalaci a kompilaci, avšak jeho rychlost pro načítání projektů a jejich prvotní kompilaci oproti Visual Studiu mírně zaostává. Čas strávený úvodním načítáním projektů je však v prostředí Android Studia kompenzován rychlostí sestavování projektů. Při porovnání rychlosti a svižnosti prostředí jsou tedy misky vah nakloněny spíše Android Studiu. Subjektivní pocity autora se přiklánějí spíše ve prospěch Visual Studia, ale ani zde není zcela zřejmé, že první pocity z jednoho, či druhého prostředí jsou výrazně lepší, nebo horší.

3.3 Dokumentace a zdroje

Dokumentace, ukázky zdrojových kódů, základní knihovny a knihovny třetích stran jsou nedílnou součástí každého vývoje. Dokumentace je pro Xamarin dostupná přímo ze stránek Microsoftu a je velmi přehledná a použitelná pro vývoj. I dokumentace pro Android

je dostupná z oficiálních zdrojů a orientace v ní je velmi rychlá a snadná. Ukázky zdrojových kódů pro základní prvky a interní metody jsou taktéž dostupné ve velmi srovnatelné kvalitě a kvantitě.

Problémy s dostupností se začínají projevovat teprve až při snaze o vytvoření aplikace, která je funkčně, či vzhledově složitější. Pro takové aplikace je velmi důležitá spolupráce komunity a jednoduše dostupná řešení. V opačném případě se vývoj zbytečně prodlužuje díky neefektivnímu způsobu vývoje stylem pokus omyl. Stejně jako velká základna profesionálů a řešení je velmi příjemná i možnost využití hotových knihoven. V obou případech dominuje vývoj pro Android, jelikož oblíbenost a využití Xamarinu není největší. Většinu problémů řešených při vývoji lze snáze nalézt pro Android než pro Xamarin, ať už se jedná o problémy s prostředím, či problémy s funkčním kódem. Android taktéž dominuje v dostupnosti hotových knihoven a řešení. Ačkoli pro Xamarin je velké množství podobných knihoven také dostupných, dokumentace a jednoduchost nasazení je poněkud složitější.

4 Srovnání vývoje

Tato kapitola pojednává o pohledu na vývoj za použití nativního, či multiplatformního přístupu. Veškeré srovnání je popsáno ze dvou pohledů:

1. Kvalitativní – subjektivní pohled autora na danou problematiku.
2. Kvantitativní – například množství kódu potřebného pro vykonání konkrétní operace u nativní a multiplatformní aplikace.

4.1 Popis srovnávané aplikace

Jako referenční rámec pro srovnávání byla použita aplikace od stejného autora, předkladatele práce. Tato aplikace je přepsána do multiplatformní podoby za použití nástroje Xamarin, s jeho aktuálně experimentální variantou „Microsoft Mobile Blazor Bindings“ pro Xamarin.Forms. I když není použití varianta v aktuální chvíli nic víc než alfa verze připravovaného produktu, práce s ní byla překvapivě příjemná.

Aplikace použitá pro srovnání se nazývá „Driver’s Score“ Jedná se o aplikaci vytvořenou primárně pro platformu Android. Tato aplikace byla zvolena, jelikož využívá přístupy, jenž splňují požadavky autora na testování požadavků na hardwarové prostředky a nabízí možnosti testování různých situací.

Základem aplikace je specifický druh rozpoznávání obrazu nazývající se ALPR, jenž pomocí segmentace obrazu detekuje poznávací značky vozidel a převádí tyto značky na strojový text, s nímž lze jednoduše pracovat. Nejjednodušší implementací tohoto způsobu čtení obrazu je implementace s využitím API volání. Tento způsob má svá pro i proti. Mezi jeho hlavní výhody patří nižší nároky na výkon samotného zařízení a jednoduchost nasazení. Nevýhodou je nutnost připojení k internetu, což ale v dnešní době není zas až tak velký problém. Aplikace využívá openALPR cloud API, jež nabízí free variantu API s vysokým počtem volání. Výhodou tohoto poskytovatele je rozsah informací vracených ze serveru. Mezi informace patří kromě poznávacích značek s procentuálními výsledky také barva, výrobce a typ vozidla, jenž obohacuje aplikaci o grafické prvky a celkovou uživatelskou přívětivost.

Další důležitá část systému se dělí na dva moduly. Prvním z těchto modulů je výstup informací pro uživatele, jenž je řešen hlasovými pokyny za použití integrované knihovny TextToSpeech společnosti Google. Tato knihovna umožňuje přímé čtení textových dat z

aplikace. V základním nastavení aplikace využívá hodnocení řidičů jako ve škole (od jedné do pěti). Uživatel má ale možnost si tato hlasová hodnocení změnit podle svého uvážení.

Druhým z těchto modulů je vstup informací od uživatele. Tento modul se od předchozího liší svou složitostí, protože načítání zvukových dat a jejich následná transformace do textové podoby je poněkud složitějším úkonem než obrácená varianta. Naštěstí i v tomto případě společnost Google nabízí integrovanou knihovnu pro převod řeči na text. V tomto případě je implementován algoritmus pro zjišťování nejpodobnějších výsledků textového vstupu. Podle procentuálních výsledků je zvolen nejpravděpodobnější výsledek a hodnocení je následně uloženo.

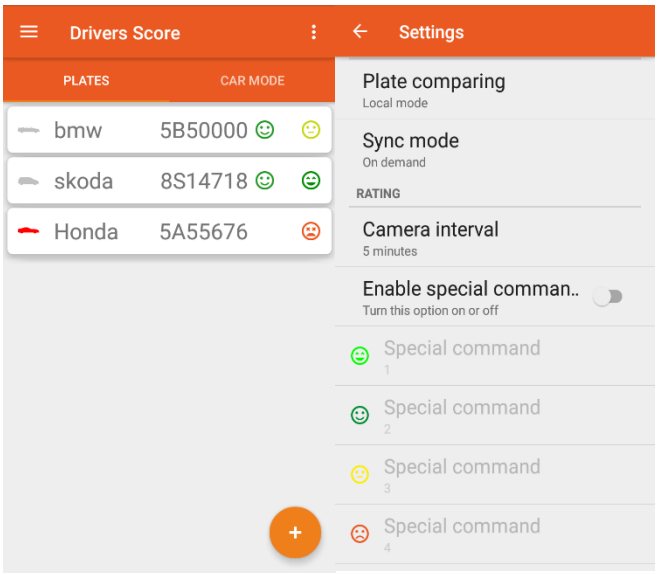
Aplikace je přístupná pouze přihlášeným uživatelům a jejich data jsou ukládána buďto lokálně nebo na centrálním serveru. Pro ukládání dat na centrální server je vytvořeno REST API, jenž obsluhuje data na serveru. Toto API je zpracováno za použití Microsoft ASP.NET serveru s MS SQL databází, jenž udržuje data všech uživatelů pro jejich snadný přistup. Aplikace disponuje i svou vlastní lokální databází. V této databázi jsou uložena data potřebná pro běh aplikace a data, která nejsou synchronizována na serveru.

Aplikace pracuje v několika funkčních módech. Tyto módy jsou následující: Místní, Serverový a Kombinace těchto dvou. Při použití místního módu se nedotazuje na server pro hodnocení vozidla jedoucího před uživatelem a data jsou uložena pouze lokálně do paměti zařízení. Serverový mód naopak neprovádí mezi-ukládání dat v lokální databázi, nýbrž při jakékoli nutnosti získání, či uložení dat sáhne rovnou na server. Kombinovaný mód nejprve zjistí, zda nemá informace o vozidle v lokální databázi a pokud ne zeptá se na serverové hodnocení.

Celkově je aplikace rozdělena do dvou hlavních módů, které oddělují dvě hlavní funkcionality. Tyto módy jsou Mód prohlížení a správy již získaných hodnocení s možností drobných editací a nastavení funkčnosti aplikace. Druhým módem je Mód snímání obrazu takzvaný „Black box“.

V aplikaci je umožněno manuální zadávání hodnocení. Toto hodnocení je poté provedeno pomocí pořízení fotografie z fotoaparátu, či načtení již existující fotografie vozidla z galerie. Další možností je ruční zadání všech vstupů. (30)

Na obrázku 11 lze vidět vzhled aplikace popsané v této kapitole. Aplikace je doplněna o získávání GPS souřadnic zařízení, jelikož v původní verzi aplikace byl pouze připraven návrh této funkcionality.



Obrázek 11 - Vzhled testované aplikace (30)

Nutno podotknout, že aplikace je několik let stará a její opětovné spuštění na nové verzi Android Studia nebyla nejjednodušší záležitost. Jelikož byla aplikace vytvořena v době, kdy byla aktuální verze Gradle verze 2.3.2, spuštění prostředí a zbuildování aplikace do funkčního stavu pro novější verze bylo velmi obtížné. Aplikace obsahovala prvky, které přestaly být podporovány, neboť nesplňovaly moderní standardy. Příkladem takového prvku je například využité CameraView které na aktuální verzi Gradle 3.6.0 nebylo možné použít. Spuštění takovéto aplikace je série pokusů a omylů, kdy je nutné najít správnou kombinaci a nastavení prostředí, která zajistí správný běh aplikace.

4.2 Metriky a zápis Kódu

Hlavní testovanou metrikou byla délka kódu programových bloků. Kromě toho byla porovnávána i složitost a jednoznačnost programového kódu. Níže uvedený výpis vyjmenovává obory měření, na kterých byly metriky aplikovány. Uvedené obory měření odpovídají částem programového kódu provádějícího stejné operace na nativní i multiplatformní aplikaci. Pro srovnání byla použita pouze část multiplatformní aplikace vytvořena pro platformu Android.

1. Volání REST API
2. Přihlašovací obrazovka z pohledu markupu
3. Přihlašovací obrazovka z pohledu funkčního kódu
4. Využití reproduktoru pro TextToSpeech (TTS)
5. Převod mluveného hlasu na text SpeechToText (STT)

6. Pořízení obrázku s použitím fotoaparátu
7. Zjištění aktuální lokace zařízení s využitím GPS

Prvním z oborů měření je volání REST API, které v době vytváření původní aplikace bylo velmi složitou záležitostí. Vývojář si musel velkou část kódu připravit sám, či se spolehnout na varianty třetích stran, kterých ale v době vývoje mnoho nebylo.

Dalším oborem měření je přihlašovací obrazovka, která uživateli umožňuje přímé přihlášení či registraci do aplikace. Měření bylo provedeno na dvou částech, jelikož jak Android, tak Xamarin využívají každý svou ale ve výsledku velmi podobnou formu markup jazyka pro definování layoutu aplikace, za kterou staví logický kód pro ovládání prvků na stránce. V první části bylo srovnáváno množství a syntaxe kódu potřebného k zadefinování layoutu stránky. Na stránce je několik elementů, jako jsou vstupy pro zadání textu ve formě čistého textu, či hesla. Dále zde najdeme ImageView pro zobrazení loga aplikace a dva typy tlačítek. Druhá část se zabývá srovnáním kódu obsluhujícího události a prvky ve stránce. Obsluha událostí a jednoduché operace patřící do třídy, která je úzce spojená s markupem, je velmi podobná ve všech jazycích využívajících takovýto zápis. Obecně platí, že tento kód by měl být co nejkratší a jednoduchý a složitější logika by měla ležet o úroveň výš.

Třetím oborem měření je vyvolání operací, které pracují s hardwarovými prostředky. Bylo srovnáváno volání knihovny pro zaznamenání vstupu z mikrofonu přes dostupné knihovny a předání textu pro přečtení a přehrání zvuku z reproduktoru zařízení. Dále bylo sledováno množství kódu pro práci s lokálním uložištěm a načítání obrazových souborů z galerie. S tím souvisí i další sledovaný prvek, kterým je spuštění kamery zařízení a získání jejího zaznamenaného výstupu. Jelikož byla aplikace doplněna o získávání GPS souřadnic, byl srovnáván i kód pro tuto manipulaci.

Testovaný kód, obory měření, byly vybrány na základě nejčastějších osvědčených postupů „best practices“. Jde především o řešení, které vývojář při běžném průzkumu před samotným vývojem nalezne. Jedná se o nejčastěji využívaný kód pro řešení dané problematiky. Jelikož byla testována varianta Xamarin.Forms, bylo cíleno na její multiplatformní výhody, které spočívají především v zápisu pouze jednoho kódu, a proto byly zvoleny hlavně varianty využívající jednodušší řešení třetí strany. Naopak nejčastější nativní řešení pro Android jsou zaměřena především na přímou obsluhu a ruční volání funkcí. Pro znázornění možností bylo srovnáváno řešení problematiky TTS, které bylo zapsáno ručně jak v Xamarinu tak v Androidu.

4.2.1 Vyhodnocení Metrik kódu

První srovnávanou částí kódu je konzumace REST API response, vytvoření requestu a samotné zavolání koncového API. Pro srovnání byla vybrána metoda „UserAuthentication“ sloužící pro přihlášení uživatele. Jelikož REST API přijímá requesty ve formátu JSON, je nejprve potřeba převést objekty do textové podoby formátu JSON. Pro tyto účely je v androidu napsaná metoda `mapObject()`, která zajišťuje obecný převod a formátování objektů pro následné vložení do objektu JSON. Popisovaná metoda `mapObject()` je vidět na následujícím úryvku kódu 1.

```
private Object mapObject(Object o) {
    Object finalValue = null;
    if (o.getClass() == String.class) {
        finalValue = o;
    } else if (Number.class.isInstance(o)) {
        finalValue = String.valueOf(o);
    } else if (Region.class.isInstance(o)) {
        finalValue = (o).toString();
    } else if (Date.class.isInstance(o)) {
        SimpleDateFormat sdf = new SimpleDateFormat("MM/dd/yyyy hh:mm:ss", new Locale("en","USA"));
        finalValue = sdf.format((Date) o);
    } else if (Collection.class.isInstance(o)) {
        Collection<?> col = (Collection<?>) o;
        JSONArray jarray = new JSONArray();
        for (Object item : col) {
            jarray.put(mapObject(item));
        }
        finalValue = jarray;
    } else {
        Map<String, Object> map = new HashMap<>();
        Method[] methods = o.getClass().getMethods();
        for (Method method : methods) {
            if(!method.getName().contains("Object")) {
                if (method.getDeclaringClass() == o.getClass()
                        && method.getModifiers() == Modifier.PUBLIC
                        && method.getName().startsWith("get")) {
                    String key = method.getName().substring(3);
                    try {
                        Object obj = method.invoke(o, null);
                        Object value = mapObject(obj);
                        map.put(key, value);
                        finalValue = new JSONObject(map);
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            }
        }
    }
    return finalValue;
}
```

Úryvek kódu 1 – Android metoda `mapObject()` pro serializaci

Po této ruční serializaci je předán výsledný objekt v textovém formátu do metody `load()`, která obstarává spojení s koncovým bodem požadovaného API. Metoda provede komunikaci s API a vrátí odpovídající response při úspěšném requestu. V opačném případě vrátí chybu. Metoda `load()` je vidět na úryvku kódu 2.

```
private String load(String contents, String sUrl) throws IOException {
    URL url = new URL(sUrl);
    HttpURLConnection conn = (HttpURLConnection) url.openConnection();
    conn.setRequestMethod("POST");
    conn.setConnectTimeout(60000);
    conn.setDoOutput(true);
    conn.setDoInput(true);
    OutputStreamWriter w = new OutputStreamWriter(conn.getOutputStream());
    w.write(contents);
    w.flush();
    if (conn.getResponseCode() == HttpsURLConnection.HTTP_OK) {
        InputStream iStream = conn.getInputStream();
        return convertStreamToUTF8String(iStream);
    } else {
        return "error";
    }
}
```

Úryvek kódu 2 – Android metoda `Load()` pro provedení API volání

Obě výše zmíněné metody jsou volané z metody `UserAuthentication()`, která celou operaci zajišťuje. Tuto metodu lze opět vidět na úryvku kódu 3.

```
User UserAuthentication(String sLogin, String sPassword) throws Exception {
    JSONObject result = null;
    JSONObject o = new JSONObject();
    JSONObject p = new JSONObject();
    o.put("interface", "RestAPI");
    o.put("method", "UserAuthentication");
    p.put("sLogin", mapObject(sLogin));
    p.put("sPassword", mapObject(sPassword));
    o.put("parameters", p);
    String s = o.toString();
    String r = load(s, DATA_URL);
    result = new JSONObject(r);
    JSONObject object;
    try {
        object = result.getJSONObject("Value");
    } catch (Exception e) {
        return new User(-1);
    }
    return JSONParser.parseAll(object);
}
```

Úryvek kódu 3 – Android metoda pro přihlášení uživatele

Pro srovnání v úryvku z kódu 4 ekvivalentní metoda, která vytváří JSON string pro použití v API requestu, ale napsána pro použití v Xamarinu. Tato metoda obecně příjme objekt a vytvoří z něho finální string.

```
public static string Serialize(TType instance)
{
    var serializer = new DataContractJsonSerializer(typeof(TType));
    using (var stream = new MemoryStream())
    {
        serializer.WriteObject(stream, instance);
        return Encoding.UTF8.GetString(stream.ToArray());
    }
}
```

Úryvek kódu 4 – Xamarin metoda pro serializaci objektu

Tento string je poté předán do metody `DoRequest()`, zobrazené na úryvku kódu 5, která vykoná spojení s API a výsledek navrátí zpět.

```
private static async Task<string> DoRequest(string sRequest)
{
    using (var client = new HttpClient())
    {
        HttpContent content = new StringContent(sRequest, Encoding.UTF8, "application/json");
        HttpResponseMessage response = await client.PostAsync(DATA_URL, content);
        if (response.IsSuccessStatusCode)
            return await response.Content.ReadAsStringAsync();
        else
            return "";
    }
}
```

Úryvek kódu 5 – Xamarin metoda pro vykonání API volání

Celkové propojení obou výše zmíněných metod je zobrazeno na úryvku kódu 6, kde metoda `Login()` Provádí vytvoření requestu ze zadaných parametrů a vrací informace o pokusu o přihlášení.

```
public static async Task<User> Login(string sLogin, string sPassword)
{
    Request<UserLoginParameters> request = new Request<UserLoginParameters>
    {
        method = "UserAuthentication",
        parameters = new UserLoginParameters
        {
            sLogin = sLogin,
            sPassword = sPassword.Encrypt()
        }
    };
    string sRequest = SerializeJSON<Request<UserLoginParameters>>.Serialize(request);
    string sResponse = await DoRequest(sRequest);
    return GetUserFromResponse(sResponse);
}
```

Úryvek kódu 6 – Xamarin metoda pro přihlášení uživatele

Při pohledu na úryvky kódu souvisejících s voláním REST API v Androidu a Xamarinu jsou vidět patrné rozdíly a zjednodušení na straně Xamarinu. Spojení všech tří nejpodstatnějších metod v Xamarinu se při standartním formátování vejde na 37 řádků, přičemž prakticky stejný kód v Androidu vyžaduje řádků 79. Jedná se pouze o úryvky kódu, ale složitost zápisu volání REST API v Androidu je o dost větší než na Xamarinu.

Dalším srovnávaným prvkem je zápis markupu přihlašovací obrazovky. Tento zápis je v obou případech velmi podobný a jeho ukázky jsou zobrazeny na úryvku kódu 7 pro Android a na úryvku kódu 8 pro Xamarin.

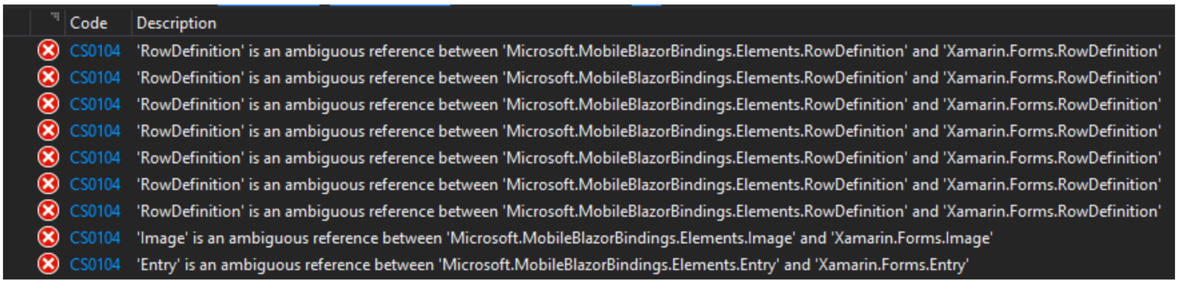
```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@drawable/background"
    android:fitsSystemWindows="true">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:paddingLeft="24dp"
        android:paddingRight="24dp"
        android:paddingTop="56dp">
        <ImageView
            android:contentDescription="@string/app_name"
            android:id="@+id/imageView3"
            android:layout_width="150dp"
            android:layout_height="150dp"
            android:layout_gravity="center_horizontal"
            android:layout_marginBottom="24dp"
            android:src="@drawable/logo" />
        <android.support.design.widget.TextInputLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginBottom="8dp"
            android:layout_marginTop="8dp"
            android:textColor="@color/colorLightText"
            android:textColorHint="@color/colorLightText">
            <EditText
                android:id="@+id/etLogin" android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:hint="@string/prompt_login"
                android:inputType="text"
                android:textColor="@color/colorLightText"
                android:textColorHint="@color/colorLightText" />
        </android.support.design.widget.TextInputLayout>
        <android.support.design.widget.TextInputLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginBottom="8dp"
            android:layout_marginTop="8dp"
            android:textColor="@color/colorLightText"
            android:textColorHint="@color/colorLightText">
            <EditText
                android:id="@+id/etPassword" android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:hint="@string/prompt_password"
                android:inputType="textPassword"
                android:textColor="@color/colorLightText"
                android:textColorHint="@color/colorLightText" />
        </android.support.design.widget.TextInputLayout>
        <android.support.v7.widget.AppCompatButton
            android:id="@+id/btnSignIn"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_marginBottom="24dp"
            android:layout_marginTop="24dp"
            android:background="@color/colorPrimary"
            android:padding="12dp"
            android:text="@string/prompt_login"
            android:textColor="@color/colorLightText" />
        <TextView
            android:id="@+id/tvSignUp"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_marginBottom="24dp" android:gravity="center"
            android:text="@string/action_haveAccount"
            android:textColor="@color/colorLightText"
            android:textColorHint="@color/colorLightText"
            android:textSize="16sp" />
    </LinearLayout>
</ScrollView>
```

Úryvek kódu 7 – Android markup pro přihlašovací obrazovku

```
<ContentPage BackgroundImageSource="@(new FileImageSource { File = "background.jpg" })"
             @ref="contentPage">
    <ScrollView Orientation="ScrollOrientation.Vertical">
        <Grid BackgroundColor="Color.Transparent" RowSpacing="0">
            <Layout>
                <RowDefinition GridUnitType="GridUnitType.Auto" />
                <RowDefinition GridUnitType="GridUnitType.Star" />
                <RowDefinition GridUnitType="GridUnitType.Star" />
                <RowDefinition GridUnitType="GridUnitType.Star" />
                <RowDefinition GridUnitType="GridUnitType.Star" />
                <RowDefinition GridUnitType="GridUnitType.Auto" />
                <RowDefinition GridUnitType="GridUnitType.Auto" />
            </Layout>
            <Contents>
                <GridCell Row="0">
                    <Image Margin="new Thickness(0, 50, 0, 0)"
                           VerticalOptions="LayoutOptions.CenterAndExpand"
                           Source="@(new FileImageSource { File = "logo.png" })" />
                </GridCell>
                <GridCell Row="1">
                    <Entry @bind-Text="sLogin" Placeholder="Login"
                           PlaceholderColor="Color.DarkOrange"
                           TextColor="Color.WhiteSmoke"
                           VerticalTextAlignment="TextAlignment.Start"
                           VerticalOptions="LayoutOptions.CenterAndExpand" />
                </GridCell>
                <GridCell Row="2">
                    <Entry @bind-Text="sPassword" Placeholder="Password"
                           PlaceholderColor="Color.DarkOrange"
                           TextColor="Color.WhiteSmoke" IsPassword="true"
                           VerticalTextAlignment="TextAlignment.Start"
                           VerticalOptions="LayoutOptions.CenterAndExpand" />
                </GridCell>
                <GridCell Row="3">
                    @if (ShowRegister)
                    {
                        <Entry @bind-Text="sFirstName" Placeholder="Name"
                               PlaceholderColor="Color.DarkOrange"
                               TextColor="Color.WhiteSmoke"
                               VerticalTextAlignment="TextAlignment.Start"
                               VerticalOptions="LayoutOptions.CenterAndExpand" />
                    }
                </GridCell>
                <GridCell Row="4">
                    @if (ShowRegister)
                    {
                        <Entry @bind-Text="sLastName" Placeholder="Surname"
                               PlaceholderColor="Color.DarkOrange"
                               TextColor="Color.WhiteSmoke"
                               VerticalTextAlignment="TextAlignment.Start"
                               VerticalOptions="LayoutOptions.CenterAndExpand" />
                    }
                </GridCell>
                <GridCell Row="5">
                    <Button Text="@(ShowRegister ? "SIGN UP" : "LOGIN")"
                            TextColor="Color.WhiteSmoke"
                            BackgroundColor="Color.DarkOrange"
                            VerticalOptions="LayoutOptions.CenterAndExpand"
                            OnClick="LoginRegister" />
                </GridCell>
                <GridCell Row="6">
                    <Button Text="@(ShowRegister ? "Do you already have an account?"
                                    : "Do not have an account?")"
                            TextColor="Color.WhiteSmoke"
                            BackgroundColor="Color.Transparent"
                            OnClick="ShowRegisterForm" />
                </GridCell>
            </Contents>
        </Grid>
    </ScrollView>
</ContentPage>
```

Úryvek kódu 8 - Xamarin markup pro přihlašovací obrazovku

Jelikož byl kód jak v Androidu, tak v Xamarinu automaticky naformátován, jejich výsledek byl podle očekávání velmi podobný. Při pohledu na oba markupy je vidět, že mezi nimi nejsou významné rozdíly. I co se týče délky kódu, je zde dost vyrovnaný výsledek, kdy varianta psaná v Xamarinu zabrala 76 řádků a nativní varianta v Androidu zabrala 74 řádků. U Xamarinu je použit zápis pro Microsoft Mobile Blazor Bindings, a protože se jedná o identický zápis jako pro Xamarin.Forms, přináší obvyklé strasti, jako jsou drobné problémy s kompilací, jelikož se jedná o alfa verzi a oficiálně nebyla vypuštěna ani první stabilní verze. Z důvodu stejných názvů prvků ve stránce je zde problém s referencemi na správné knihovny. Jak je vidět z obrázku 12 kvůli IntelliSense[10] projekt hlásí problémy s referencemi. Podrobnější informace o IntelliSense jsou k dispozici na webu INTELLISENSE IN VISUAL STUDIO. (31) Nicméně tyto problémy nebrání kompilaci aplikace a jsou to tedy jen nepříjemnosti začátků technologie.



Code	Description
CS0104	'RowDefinition' is an ambiguous reference between 'Microsoft.MobileBlazorBindings.Elements.RowDefinition' and 'Xamarin.Forms.RowDefinition'
CS0104	'RowDefinition' is an ambiguous reference between 'Microsoft.MobileBlazorBindings.Elements.RowDefinition' and 'Xamarin.Forms.RowDefinition'
CS0104	'RowDefinition' is an ambiguous reference between 'Microsoft.MobileBlazorBindings.Elements.RowDefinition' and 'Xamarin.Forms.RowDefinition'
CS0104	'RowDefinition' is an ambiguous reference between 'Microsoft.MobileBlazorBindings.Elements.RowDefinition' and 'Xamarin.Forms.RowDefinition'
CS0104	'RowDefinition' is an ambiguous reference between 'Microsoft.MobileBlazorBindings.Elements.RowDefinition' and 'Xamarin.Forms.RowDefinition'
CS0104	'RowDefinition' is an ambiguous reference between 'Microsoft.MobileBlazorBindings.Elements.RowDefinition' and 'Xamarin.Forms.RowDefinition'
CS0104	'RowDefinition' is an ambiguous reference between 'Microsoft.MobileBlazorBindings.Elements.RowDefinition' and 'Xamarin.Forms.RowDefinition'
CS0104	'Image' is an ambiguous reference between 'Microsoft.MobileBlazorBindings.Elements.Image' and 'Xamarin.Forms.Image'
CS0104	'Entry' is an ambiguous reference between 'Microsoft.MobileBlazorBindings.Elements.Entry' and 'Xamarin.Forms.Entry'

Obrázek 12 – Chybové hlášky v prostředí Visual Studio (Vlastní tvorba)

Druhou částí srovnání přihlašovací obrazovky je funkční kód na pozadí, který obsluhuje veškeré události na stránce. Zde musel být kód lehce upraven, aby ho bylo možné alespoň přibližně srovnat. Jelikož Java a C# využívají trochu jiné zápisy pro definice tříd, jako jsou import/using pro Javu a package/namespace pro C#, byly tyto části kódu vyřazeny ze srovnání.

[10] **IntelliSense** – Nástroj integrovaný do prostředí Visual Studio pomáhající v dokončování kódu.

```
public class LoginActivity extends AppCompatActivity {

    private EditText etPassword, etLogin;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        Thread.setDefaultUncaughtExceptionHandler(new ExceptionHandler(this));

        setContentView(R.layout.activity_login);

        etPassword = (EditText) findViewById(R.id.etPassword);
        etLogin = (EditText) findViewById(R.id.etLogin);
        etLogin.requestFocus();

        Button btnSignIn = (Button) findViewById(R.id.btnSignIn);
        btnSignIn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                attemptLogin();
            }
        });

        TextView tvSignUp = (TextView) findViewById(R.id.tvSignUp);
        tvSignUp.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent i = new Intent(LoginActivity.this, RegisterActivity.class);
                startActivity(i);
            }
        });
    }

    @Override
    public void onBackPressed() {
        moveTaskToBack(true);
    }

    private void attemptLogin() {

        String sLogin = etLogin.getText().toString();
        String sPassword = General.encryptPassword(etPassword.getText().toString());

        if (!TextUtils.isEmpty(sLogin) && !TextUtils.isEmpty(sPassword)) {

            User user = new User(sLogin, sPassword);

            AsyncLoader asyncLoader = new AsyncLoader(this);

            asyncLoader.logUser(user);
        }
    }
}
```

Úryvek kódu 9 – Funkční kód přihlašovací obrazovky v Androidu

```
public partial class Login
{
    [Inject]
    private AppState _AppState { get; set; }

    [Inject]
    private GlobalSettings _GlobalSettings { get; set; }

    Microsoft.MobileBlazorBindings.Elements.ContentPage contentPage;

    protected bool ShowRegister { get; set; } = false;

    private string sLogin;
    private string sPassword;
    private string sFirstName;
    private string sLastName;

    protected override void OnInitialized()
    {
        base.OnInitialized();
        _AppState.OnChange += OnChanged;
    }

    protected async Task LoginRegister()
    {
        string sMessage = "";
        if (String.IsNullOrEmpty(sLogin))
            sMessage = "Login is empty";
        else if (String.IsNullOrEmpty(sPassword))
            sMessage = "Password is empty";
        else if (ShowRegister && String.IsNullOrEmpty(sFirstName))
            sMessage = "Name is empty";
        else if (ShowRegister && String.IsNullOrEmpty(sLastName))
            sMessage = "Surname is empty";

        if (!String.IsNullOrEmpty(sMessage))
        {
            await Application.Current.MainPage.DisplayAlert("Error", sMessage, "Back");
            return;
        }
        try
        {
            User user;

            if (!ShowRegister)
                user = await RestService.Login(sLogin, sPassword);
            else
                user = await RestService.Register(sLogin, sPassword, sFirstName, sLastName);

            if (user != null)
            {
                await _AppState.Database.SaveUserAsync(user);
                _GlobalSettings.UserID = user.ID;
                StateHasChanged();
                await _AppState.CallChanged();
                await contentPage.PopModalAsync();
            }
            else
                await Application.Current.MainPage.DisplayAlert("Error"
                      , $"User could not be {(ShowRegister ? "registered." : "logged")}.", "Back");
        }
        catch (Exception e)
        {
            await Application.Current.MainPage.DisplayAlert("Error", e.Message, "Back");
        }
    }

    protected void ShowRegisterForm()
    {
        ShowRegister = !ShowRegister;
    }
}
```

Úryvek kódu 10 – Funkční kód přihlašovací obrazovky v Xamarinu

Jak je z úryvku kódu 9 pro Android a z úryvku kódu 10 pro Xamarin vidět, životní cyklus stránky je velmi podobný, kdy při prvním načtení probíhá inicializace prvků stránky s jejich potřebným nastavením. Stránka v Xamarinu je odlišná v tom, že je do ní zakomponována rovnou i registrace a vyřešeno jednoduché přepínání zobrazených prvků formuláře, jelikož Xamarin ve spojení s Blazorem umožňuje přímé přepínání zobrazených prvků a jejich jednoduchou obsluhu. Po odstranění řádků souvisejících s registrací se počet řádků sníží na 62, což je stále více než počet řádků napsaných v Androidu.

Další srovnávanou ukázkou kódu je přečtení textu a jeho přehrání skrze reproduktor. Z existujícího projektu byl využit kód nezbytně nutný pro tuto operaci a je svým zápisem přínosný pro srovnání. Ostatní části, jako je samotné volání metod, je z těchto ukázek odstraněno.

```
public void getSpeechFromText(final String sText) {
    tts = new TextToSpeech(activity, new TextToSpeech.OnInitListener() {

        @Override
        public void onInit(int status) {
            if (status == TextToSpeech.SUCCESS) {
                int result = tts.setLanguage(Locale.getDefault());
                if (result != TextToSpeech.LANG_MISSING_DATA
                        && result != TextToSpeech.LANG_NOT_SUPPORTED)
                    ConvertTextToSpeech(sText);
            }
        }
    });
}

private void ConvertTextToSpeech(String sText) {
    String[] splitSpeech = sText.split("\\|\\|");

    for (int i = 0; i < splitSpeech.length; i++) {
        if (i == 0) { // Use for the first splited text to flush on audio stream
            tts.speak(splitSpeech[i].trim(), TextToSpeech.QUEUE_FLUSH, null, "id1");
        } else { // add the new test on previous then play the TTS
            tts.speak(splitSpeech[i].trim(), TextToSpeech.QUEUE_ADD, null, "id1");
        }
        tts.playSilentUtterance(100, TextToSpeech.QUEUE_ADD, null);
    }
}
```

Úryvek kódu 11 – TextToSpeech zápis v Androidu

```
public class TextToSpeechService : Object, ITextToSpeechService, TextToSpeech.IOnInitListener
{
    private TextToSpeech speaker;
    private string toSpeak;

    public void Speak(string text)
    {
        watch = Stopwatch.StartNew();

        if (!string.IsNullOrWhiteSpace(text))
        {
            toSpeak = text;
            if (speaker == null)
                speaker = new TextToSpeech(MainActivity.Instance, this);
            else
                speaker.Speak(toSpeak, QueueMode.Flush, null, null);
        }
    }

    #region IOnInitListener implementation
    public void OnInit(OperationResult status)
    {
        if (status.Equals(OperationResult.Success))
        {
            speaker.Speak(toSpeak, QueueMode.Flush, null, null);
        }
    }
    #endregion
}
```

Úryvek kódu 12 – TextToSpeech zápis v Xamarinu

Jedna z možností implementace převodu textu na řeč pomocí Xamarin.Forms je implementace vlastního rozhraní, které je nutno použít jak pro Android, tak iOS verzi podle potřeby. Úryvek kódu 11 zobrazuje implementaci pro Android. Samotné volání je potom řešeno ve sdílené kódové základně. Co se týče srovnání implementace v Androidu a Xamarinu, jsou zde patřičné rozdíly. V Androidu si většinu problematiky vyřeší vývojář většinou sám a v Xamarinu se obvykle spoléhá na knihovny třetích stran, respektive integrované obslužné knihovny. Délkou je kód prakticky stejný, ale díky využití knihovny na Xamarinu je výsledek podstatně jednodušší.

Úryvek kódu 13 a úryvek kódu 14 zobrazuje opačnou situaci, kdy je nutné převést mluvenou řeč do textové podoby a z této textové podoby vytvořit odpovídající hodnoty. V tomto případě hodnocení řidiče, které lze hodnotit na stupnici od jedné do pěti, či speciálně definovanými příkazy. Srovnávání načteného textu s definovanými hodnotami zde není zobrazeno, jelikož se jedná o pouhé porovnávání hodnot.

```
public void promptSpeechInput(Plate plate) {
    Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
    intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL
                                        , RecognizerIntent.LANGUAGE_MODEL_WEB_SEARCH);
    intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE, Locale.getDefault());
    intent.putExtra(RecognizerIntent.EXTRA_PROMPT, activity.getString(R.string.speech_prompt));
    try {
        activity.startActivityForResult(intent, General.REQ_CODE_SPEECH_INPUT);
    } catch (ActivityNotFoundException a) {
        Toast.makeText(activity, activity.getString(R.string.speech_not_supported),
                Toast.LENGTH_SHORT).show();
    }
}

private Score getScoreFromSpeech(int requestCode, int resultCode, Intent data) {
    Score score = new Score(-1);
    if (resultCode == RESULT_OK && null != data) {
        switch (requestCode) {
            case General.REQ_CODE_SPEECH_INPUT:
                ArrayList<String> result =
                                    data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);
                score.setValue(General.parseIntegerFromArray(result, settings));
                break;
        }
    }
    return score;
}
```

Úryvek kódu 13 – SpeechToText zápis v Androidu

```
public static async Task<string> GetScoreAsync(List<string> Keywords)
{
    string retVal = null;

    var permission = await CrossSpeechRecognition.Current.RequestPermission();
    if (permission == SpeechRecognizerStatus.Available)
        retVal = await CrossSpeechRecognition.Current.ListenForFirstKeyword(Keywords.ToArray());

    return retVal;
}
```

Úryvek kódu 14 – SpeechToText zápis v Xamarinu

Implementaci hlasového vstupu, zobrazenou v úryvku kódu 14 autor hodnotí jako nejvíce problematickou, jelikož je velmi málo hotových řešení. Řešení použité pro tento úkol je sice označené jako zastaralé, ale autor volil jeho implementaci, protože jeho novější verze s sebou přináší nepříjemná úskalí a velké změny do struktury samotného projektu. I přes skutečnost, že se jedná o zastaralý plugin, je jeho využití jednoduché a elegantní. V tomto případě, jelikož jde o využití hotového řešení, je zápis implementace oproti Androidu výrazně kratší. Jelikož se jedná o implementaci pluginu pro obě platformy, je jeho implementace umístěna pouze do sdílené knihovny.

V úryvku kódu 15 lze vidět metody pro spuštění kamery a získání pořízeného snímku. Opět byl kód zjednodušen a upraven pro srovnání s verzí napsanou za použití Xamarinu, kterou lze vidět na úryvku kódu 16. Srovnán je pouze nezbytný kód pro obsluhu

kamery a načtení bitmapové informace, která je převedena do formátu Base64[11] a vrácena v textové podobě.

```
intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
if (intent.resolveActivity(activity.getPackageManager()) != null) {
    activity.startActivityForResult(intent, General.REQ_CODE_CAMERA_INPUT);
}

private String getImage(int requestCode, int resultCode, Intent data) {
    String encodedImage = "";
    if (resultCode == RESULT_OK) {
        switch (requestCode) {
            case General.REQ_CODE_CAMERA_INPUT: // Camera
                Bundle extras = data.getExtras();
                encodedImage = General.encodeImage((Bitmap) extras.get("data"));
                break;
        }
    }
    return encodedImage;
}
```

Úryvek kódu 15 – Zachycení obrázku pomocí kamery v Androidu

Více informací o kódování Base64 je dostupných například na webu WHAT IS BASE64. (32)

```
public static async Task<string> GetPictureAsync()
{
    string sEncodedImage = "";
    MediaFile photo = null;
    if (CrossMedia.Current.IsCameraAvailable && CrossMedia.Current.IsTakePhotoSupported)
        photo = await CrossMedia.Current.TakePhotoAsync(new StoreCameraMediaOptions());

    if (photo != null)
        sEncodedImage = photo.GetStream().ConvertToBase64();

    return sEncodedImage;
}
```

Úryvek kódu 16 – Zachycení obrázku pomocí kamery v Xamarinu

V úryvku kódu 16 je předvedena asynchronní metoda pro pořízení snímku a jeho transformace do Base64 stringu pro pozdější použití v API metodách. Pro snazší implementaci bylo použito hotové řešení, které obstará za vývojáře řešení práv a potřebných svolení uživatele pro využití fotoaparátu zařízení. Při srovnání implementace s variantou psanou v Androidu zde nejsou vidět příliš velké nároky, co se délky kódu týče. Nicméně samotná implementace v Xamarinu je velmi snadná právě díky využití obslužných knihoven. Je-li opomenuto volání a řešení práv, je kód, co se týče řádku, identicky dlouhý. Tato implementace je řešena multiplatformním pluginem, který nepotřebuje žádnou implementaci logiky do nativních částí projektu.

[11] **Base64 formát** – Reprezentuje binární data ve formátu řetězce ASCII.

Poslední úryvek kódu 17 zobrazuje metodu „getLocation“, která se kromě získání zeměpisné šířky a délky stará o zabezpečení práv. Jsou zde implementovány oba přístupy pro získání lokace, a to získání pomocí GPS modulu, nebo určení lokace pomocí triangulace síťové polohy zařízení. S takto získanou lokací se potom snadno pracuje a v tomto případě je lokace uložena k lokalizovanému automobilu.

```
public void getLocation(final int nScoreID) {
    if (ActivityCompat.checkSelfPermission(activity, Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED
            || ActivityCompat.checkSelfPermission(activity,
Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED
            && ActivityCompat.checkSelfPermission(activity,
Manifest.permission.ACCESS_NETWORK_STATE) != PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions(activity, new String[]{
                Manifest.permission.ACCESS_FINE_LOCATION,
                Manifest.permission.ACCESS_COARSE_LOCATION,
                Manifest.permission.ACCESS_NETWORK_STATE}, 1);
    }

    final LocationRequest locationRequest = new LocationRequest();
    locationRequest.setInterval(10000);
    locationRequest.setFastestInterval(3000);
    locationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);

    LocationServices.getFusedLocationProviderClient(activity)
            .requestLocationUpdates(locationRequest, new LocationCallback() {
                @Override
                public void onLocationResult(LocationResult locationResult) {
                    super.onLocationResult(locationResult);
                    LocationServices.getFusedLocationProviderClient(activity)
                            .removeLocationUpdates(this);
                    if (locationResult != null && locationResult.getLocations().size() > 0) {
                        General.SetScoreLocation(nScoreID, locationResult.getLastLocation()
                                .getLatitude(), locationResult.getLastLocation().getLongitude());
                    }
                }
            }, Looper.getMainLooper());
}
```

Úryvek kódu 17 – Zápis metody pro získání GPS souřadnic v Androidu

```
public static async Task<(decimal? Latitude, decimal? Longitude)> GetLocationAsync()
{
    (decimal?, decimal?) retVal = (null, null);

    Position position = null;

    if (CrossGeolocator.Current.IsGeolocationAvailable
       && CrossGeolocator.Current.IsGeolocationEnabled)
        position = await CrossGeolocator.Current.GetPositionAsync();

    if (position != null)
        retVal = ((decimal?)position.Latitude, (decimal?)position.Longitude);

    return retVal;
}
```

Úryvek kódu 18 – Zápis metody pro získání GPS souřadnic v Xamarinu

Úryvek kódu 18 ukazuje poslední srovnávanou metodu, a to metodu pro získání zeměpisné délky a šířky v Xamarinu. Stejně jako v úryvku kódu 16 byla použita připravená knihovna třetí strany, která obsluhuje kromě GPS modulu i práva přístupu k těmto informacím. Srovnání s Android implementací je o dost jednodušší právě díky zjednodušení

pomocí využitého pluginu. Android implementace totiž sama řeší situaci využití GPS modulu či síťovou lokaci pro získání polohy. Implementace Xamarin má toto rozhodování vyřešeno uvnitř pluginu a je proto výrazně jednodušší. Stejně jako plugin pro obsluhu fotoaparátu zařízení je i tato implementace sdílena pro obě platformy.

Výsledky srovnání byly pro přehlednost zobrazení zaneseny do tabulky 2.

	Xamarin	Android
Volání REST API	37 řádků	79 řádků
Přihlašovací obrazovka Markup	76 řádků	74 řádků
Přihlašovací obrazovka Funkční kód	77 řádků	54 řádků
Reproduktor (TTS)	27 řádků	26 řádků
Mikrofon (STT)	10 řádků	27 řádků
Kamera	12 řádků	17 řádků
GPS	15 řádků	29 řádků
Celkem	**254**	**306**

Tabulka 2 – Srovnání metrik kódu (Vlastní tvorba)

Při srovnávání zápisu některých hlavních částí Androidu a Xamarinu je nutné přihlížet k několika faktorům. Srovnání ukazuje, že psaní multiplatformní aplikace má v některých částech čistší zápis kódu a je elegantnější, ne vždy je však snazší. Například taková konzumace REST API je díky C# implementaci velmi jednoduchá a rychlá. Naopak využití některých prostředků zařízení může být v některých případech velmi složité a nepříjemné. Implementace STT je díky Google API na Androidu otázka pár minut, ale stejná implementace, díky své multiplatformitě, je na Xamarinu otázka hodin a v některých případech i dní. Jelikož srovnání délky kódu lze vyjádřit kvantitativně, je možné ze sumáře tabulky poznat, že i přes multiplatformní vlastnosti Xamarinu je jeho zápis v některých částech kratší a výsledná délka kódu je taktéž kratší.

5 Testování výsledných aplikací

Poslední kapitola se zabývá srovnáním výsledků vývoje. Byla testována výsledná aplikace napsaná při použití Xamarinu spolu s aplikací již dříve vytvořenou v Androidu. Hlavními testovanými kritérii byly výkon a výsledná velikost aplikace při vytvoření publikovatelného APK souboru a velikost po instalaci aplikace na koncové zařízení. Z pohledu výkonu byly testovány konkrétní nároky na vykonání jednotlivých operací. Spolu s výkonem byla srovnána výsledná velikost aplikace.

5.1 Výkonnostní srovnání

Výkonnostní aspekt je jedním z hlavních vstupů při rozhodování vývojáře přemýšlejícího o využití multiplatformního nástroje. Srovnání výkonu bylo provedeno na základních metodách a operacích, bez kterých by se většina složitějších aplikací neobešla. V kapitole 4 („Srovnání vývoje“) byly popsány bloky kódu (obory měření), které jsou zde výkonnostně testovány.

Z předchozí kapitoly byly vybrány následující nárokované metody pro ověření výkonu multiplatformního a nativního zápisu.

1. Volání a konzumace REST API
2. Převod textu na řeč (STT) s využitím reproduktoru
3. Přijmutí hlasového vstupu a jeho následný převod do textové podoby (TTS)
4. Pořízení obrazového vstupu při použití fotoaparátu
5. Zjištění geografické polohy zařízení

Konkrétně se jedná o Přihlášení uživatele. Jelikož se jedná o přihlášení za pomoci ověřování vůči vzdálenému serveru, je testován čas nutný pro dokončení procesu přihlášení. V případě úspěšného přihlášení je uživateli provedena synchronizace globálních dat. Měřen je průběh volání od startu metody až po vrácení hodnot v patřičných objektech.

Druhým výkonnostním srovnáním bylo přečtení textu skrze reproduktor zařízení. Tento test je proveden měřením času potřebného k předání požadovaného textu zařízení. Zjednodušeně se jedná o čas, který uběhne od zadání příkazu k přehrání textu až po okamžik jeho samotného přehrání.

Třetí test je opakem druhého, kdy byl měřen a porovnáván čas, za který bylo zařízení schopno přijmout zvuk a převést ho na text. Tentokrát byl sledován čas od spuštění metody pro aktivaci mikrofonu, až po převod hlasu na výsledný text. Testované slovo je „test“. U tohoto testu je nutné provést velké množství opakování, jelikož se jedná o test, ve kterém figuruje lidský element.

Čtvrtým testovaným prostředkem byla kamera a pořízení snímku. V tomto testu bylo sledováno vyvolání aplikace pro obsluhu kamery a načtení snímku s jeho následným převedením do textové podoby ve formátu Base64. Tento test byl stejně jako předchozí velmi citlivý na odchylku, jelikož zde figuruje lidský faktor.

Posledním testem byl test načtení aktuální GPS souřadnice zařízení. V tomto testu bylo měřeno spuštění metody pro získání těchto informací, až po výsledné předání informací dále do aplikace ke zpracování.

Veškeré výkonnostní testování bylo provedeno za použití stejného emulátoru, který simuluje dostatečné podmínky pro běh aplikace. Tyto podmínky byly definovány tak, že zařízení má skvělé připojení k síti a výkon aplikace není omezen hardwarovými nedostatky, jako jsou nedostatek výpočetního výkonu zařízení, nebo nedostatek výpočetní paměti. Testování je provedeno s několika opakováními. Pro zajištění validity výsledků, bylo u každého testu provedeno dvacet opakování pro maximální eliminaci odchylek způsobených rušivými vlivy.

Pro testování výkonu je možno použít nejrůznější metody hlídání průběhu vykonávání kódu. Jednou z možností při sledování konkrétních metod je sledování časových úseků pomocí vytvořených timerů, které jsou vloženy na začátek a konec sledované metody.

5.1.1 Vyhodnocení výkonnostního srovnání

První srovnávanou metodou je přihlášení uživatele za použití volání REST API. Tato metoda je znatelně rychlejší při použití Xamarinu, než při použití kódu psaného v Androidu.

Výsledek druhého testu je naopak podle očekávání při využití nativního zápisu rychlejší. Volání TTS v Xamarinu bylo znatelně pomalejší a při průchodu a změření času vykonávání kódu je zřetelně vidět, že výkon při použití Androidu je znatelně lepší.

Jelikož ve třetím testu figuruje lidský faktor, je potřeba brát výsledky s jistým nadhledem. Nicméně i přes skutečnost, že načítání hlasu přes mikrofon a následnou transformaci řeči na text výrazně ovlivňuje zpoždění při zadání lidským faktorem, nativní

kód psaný pro platformu Android v tomto testu opět výkonnostně vedl. Jelikož byla lidská odchylka maximálně minimalizována provedením více testů, dají se testy pro tento účel považovat za dostačující.

Čtvrtý test, stejně jako předchozí, byl ovlivněn lidským zásahem. Byl ale pro obě aplikace ovlivněn stejným uživatelem a dá se tedy tvrdit, že zkreslení je tímto faktem a skutečností, že bylo provedeno více testů, maximálně minimalizováno. Při testování bylo pozorováno výrazné zpomalení při spuštění kamery při použití aplikace psané v Xamarinu oproti té psané v Androidu.

Poslední test spočíval v získání aktuální polohy zařízení. Tato operace proběhla na obou testovaných aplikacích velmi rychle. Aplikace psaná v Androidu, při použití ručního získání lokačních informací, proběhla výrazně rychleji.

Pro všechny testy je provedeno dvacet opakování. Jelikož je každý test vyhodnocen za použití aritmetického průměru, je nutné ověřit, zda je využití daného průměru oprávněné. Ověření oprávněnosti aritmetického průměru je provedeno výpočtem směrodatných odchylek (σ) a variačních koeficientů (v) jednotlivých měření. Je možné použít výpočet pro směrodatnou odchylku základního souboru, jelikož se jedná o celkové výsledky ze všech tesů. Tato odchylka je vypočtena dle vzorce (5.1).

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2} \tag{5.1}$$

Výpočet variačního koeficientu je při známé směrodatné odchylce snadný a je proveden následujícím vzorcem (5.2). Pro lepší čitelnost je použit vzorec pro výpočet procentuální hodnoty.

$$v = \frac{\sigma}{\bar{x}} 100 \tag{5.2}$$

Výsledky dle očekávání ukazují nejvyšší směrodatnou odchylku u měření, kde figuroval lidský faktor. Jde o měření metody pro provedení operace SpeechToText a metodu pro pořízení snímku z kamery. Protože jsou variační koeficienty pro všechny testy menší než 30 %, dají se výsledky aritmetických průměrů označit za oprávněné.

Zprůměrované výsledky všech testů spolu s jejich směrodatnými odchylkami a variačními koeficienty byly pro přehlednost zaneseny do následující tabulky 3.

	Xamarin			Android		
	Průměr	Směrodatná odchylka (σ)	Variační koeficient (v)	Průměr	Směrodatná odchylka (σ)	Variační koeficient (v)
Volání REST API	106,45 ms	7,13	6,70 %	149,25 ms	15,71	10,53 %
Reproduktor (TTS)	2997,30 ms	51,97	1,73 %	2757,95 ms	78,93	2,86 %
Mikrofon (STT)	5246,55 ms	128,14	2,44 %	4287,80 ms	132,02	3,08 %
Kamera	5186,50 ms	133,03	2,56 %	4747,60 ms	146,10	3,08 %
GPS	22,65 ms	6,10	29,95 %	18,95 ms	1,99	10,49 %

Tabulka 3 – Srovnání výkonnostních testů (Vlastní tvorba)

Poslední srovnávanou hodnotou byla velikost výsledného APK souboru generovaného z Androidu a Xamarinu spolu s místem zabíraným v paměti zařízení hned po instalaci aplikace. Velikost APK souboru vytvořeného přes nativní vývoj pro Android byla něco málo přes 2 MB konkrétně 2,084 MB, což je oproti aplikaci vytvořené pro Xamarin výrazně menší. Aplikace vytvořená multiplatformním nástrojem Xamarin přesahuje velikost 24 MB, s přesnou velikostí 24,211 MB. Ani po instalaci se poměry velikosti aplikací nezměnily ve prospěch aplikace psané za použití Xamarinu. Multiplatformní aplikace zabírala po instalaci celkově 39,15 MB, přičemž aplikace psaná nativně pouhých 6,86 MB.

Při pohledu na velikost aplikací a jejich srovnávané výkony je nutno uznat, že výhody multiplatformního vývoje při použití nástroje Xamarin jsou především v ušetření nákladů na udržování více kódových základen a nikoli ve výkonu, či velikosti výsledné aplikace. Je tedy na vývojáři, jakou cestou se rozhodne jít. Nativní vývoj si sice dle výsledků výkonnostních testů vede lépe po stránce výkonu, ale rozdíl v některých metodách není prokazatelný, respektive není zřetelný natolik, aby kompletně zastínil multiplatformní vývoj. Srovnání výsledné velikosti aplikace přidává plusové body především nativnímu vývoji, protože jsou výsledné velikosti podstatně rozdílné už i u takto relativně malé aplikace.

6 Závěr

V této diplomové práci je popsáno srovnání multiplatformního vývoje s platformně zaměřeným přístupem pro vývoj mobilních aplikací. Srovnání je založeno na několika testovacích případech. Aplikované postupy a principy rozdělit do čtyř hlavních částí.

Prvním krokem před každým vývojem je průzkum dostupných možností, které lze k vývoji použít. S tímto krokem koresponduje první část práce, která je zaměřena na představení vývojových možností. Jelikož je multiplatformní vývoj v dnešní době velmi diskutovanou a řešenou záležitostí, existuje mnoho různých nástrojů pro vývoj mobilních aplikací. Jsou zde představeny alternativní nástroje pro multiplatformní vývoj, jako jsou React Native, Ionic a Flutter, jelikož se jedná o nejžádanější a nejznámější alternativy Xamarinu. Kapitola 2 („Představení technologií a nástrojů“) se primárně zaměřuje na představení vývojových možností při použití Xamarinu a pojednává o možnostech nativního vývoje, spolu s představením hlavních platforem jako je iOS a Android. Pro lepší představu jsou zde popsány alternativní možnosti vývoje. U všech možností jsou popsány jejich výhody a nevýhody. Při pohledu na provedené teoretické srovnání nástrojů je vidět, že nativní vývoj, leč má své nevýhody převážně ve formě nákladů na udržování několika kódových základen, je stále výhodnější variantou mobilního vývoje.

Další část je zaměřena na srovnání vývojových prostředí sloužících pro vývoj mobilních aplikací. Srovnání je popsáno v kapitole 3 („Srovnání a vyhodnocení nástrojů“), kde jsou představena prostředí pro vývoj a je provedeno testování jejich vlastností a možností. Testy jsou provedeny jak z pohledu výkonu, tak z pohledu autorových pocitů z přívětivosti prostředí. Nejzajímavějšími ukazateli z testovaných hodnot jsou výkonová srovnání, kdy je měřen čas, který vývojář stráví při vykonávání daných úkonů. Čas strávený čekáním na běžné úkony jako je vytvoření projektu, jeho kompilace, či jeho spuštění jsou velmi důležité pro každodenní vývoj. Pro zajímavost je srovnána i výsledná velikost, kterou prostředí zabírá na disku. Na základě výsledků lze tvrdit, že Visual Studio pro vývoj multiplatformní aplikace s využitím Xamarinu je výrazně větší, díky své robustnosti a možnostem vývoje pro další platformy. I přes svou velikost je Visual Studio svižné při prvních načteních a úkonech prováděných před začátkem samotného vývoje. Jelikož však tyto úkony vývojář provede ve většině případů jen jednou, neměly by být nejdůležitějším faktorem při výběru prostředí. Rychlost sestavení neboli kompilace projektu je ukazatel, ve

kterém dominuje Android Studio, které je menší a svižnější pro běžnou práci. Autor zde zhodnotil i své pocity z instalace a využívání prostředí.

Parametrů pro srovnání nativního vývoje s vývojem multiplatformním je mnoho. V kapitole 4 („Srovnání vývoje“) je nejprve představena referenční aplikace. Ta byla vytvořena pro studijní účely stejným autorem pro platformu Android. Dále jsou zde rozebrány vybrané prvky zápisu kódu při použití Xamarinu a jeho verze „Microsoft Mobile Blazor Bindings“ spolu se zápisem pro Android psaném v Javě. Srovnávána je především složitost a délka zápisu konkrétních operací. Pro srovnání jsou vybrány: volání REST API, kódový zápis obrazovek a obsluha hardwarových prvků zařízení. Při pohledu na celkové součty všech operací se může zdát, že obsluha všech metod je kratší při využití Xamarinu, ale ne všechny zápisy jsou nejjednodušší. Autor jako příklad jednoduchého a elegantního zápisu uvádí konzumaci REST API, která byla v mobilním prostředí vždy problematická. Nicméně pro většinu hardwarových funkcí je vývojář nucen buďto využít knihovny třetích stran, či se vydat cestou implementace vlastního řešení, která je v některých případech velmi problematická. Pro poukázání na oba přístupy jsou použity obě varianty obsluhy hardwarových prostředků zařízení.

Poslední část práce pojednává o výkonnostních možnostech multiplatformní aplikace ve srovnání s aplikací nativní. Srovnávány jsou kódové ukázky z předchozí kapitoly 4 („Srovnání vývoje“). Výkon aplikace je velmi důležitý, stejně tak jako velikost výsledné aplikace. Čím je aplikace menší a rychlejší, tím je pro koncového uživatele zajímavější. Výsledky jsou v tomto testu ve většině případů nakloněny na stranu nativního vývoje, jelikož kromě testu volání REST API, byl nativní vývoj vždy rychlejší. Při pohledu na srovnání výsledných velikostí aplikace nainstalované na koncové zařízení jsou vidět významné rozdíly, kdy aplikace napsaná za použití Xamarinu je výrazně větší. Stejně jako nainstalovaná aplikace i instalační APK soubor vyrobený pro tuto aplikaci je pro Xamarin významně větší.

Cílem této práce bylo srovnat multiplatformní vývoj s použitím Xamarinu s vývojem platformně zaměřeným a poskytnout tak začínajícím vývojářům pomoc při rozhodování, zda k vývoji použít multiplatformní nástroj Xamarin, či zvolit cestu správy dvou kódových základen pro obě hlavní platformy. Pro menší aplikace bez náročných hardwarových funkcí se jeví multiplatformní vývoj s použitím Xamarinu jako vhodná volba. Aplikace, od kterých se očekává náročnější práce s konkrétním hardwarem, je lepší psát nativně. Tento cíl práce byl splněn, avšak jak dlouho a v jakém rozsahu bude tato práce vývojářům schopna

poskytovat validní náhled na multiplatformní vývoj v Xamarinu ve srovnání s nativním vývojem, se ukáže až časem.

7 Použitá literatura

[1] WHAT IS XAMARIN? [online]. Microsoft, 2019 [cit. 2020-04-18]. Dostupné z: https://docs.microsoft.com/en-gb/xamarin/get-started/what-is-xamarin

[2] TAK, Rohin a Jhalak MODI. Mobile DevOps: Deliver continuous integration and deployment within your mobile applications. 1. Birmingham, UK: Packt Publishing, 2018 [cit. 2020-04-18]. ISBN 9781788295741.

[3] MONO [online]. Mono Project, 2020 [cit. 2020-04-18]. Dostupné z: https://www.mono-project.com

[4] HISTORY OF ECMA [online]. ECMA International, nedatováno [cit. 2020-04-18]. Dostupné z: http://www.ecma-international.org/memento/history.htm

[5] ABOUT FREEBSD [online]. The FreeBSD Project, 1995-2020 [cit. 2020-04-18]. Dostupné z: https://www.freebsd.org/about.html

[6] BAHRAMINEZHAD, A. 2020: Blazor and Native Mobile Apps [online]. ITNEXT, 2020 [cit. 2020-04-18]. Dostupné z: https://itnext.io/blazor-and-native-mobile-apps-9177c5a6488b

[7] THE GOOD AND THE BAD OF XAMARIN MOBILE DEVELOPMENT [online]. AltexSoft, 2019 [cit. 2020-04-18]. Dostupné z: https://www.altexsoft.com/blog/mobile/pros-and-cons-of-xamarin-vs-native

[8] KUCHERENKO, Igor a S. M. Mohi Us SUNNAT. Learn Spring for Android Application Development: Build robust Android applications with Kotlin 1.3 and Spring 5. 1. Birmingham, UK: Packt Publishing, 2019 [cit. 2020-04-18]. ISBN 9781789341911.

[9] APACHE LICENSES [online]. The Apache Software Foundation, 2019 [cit. 2020-04-18]. Dostupné z: https://www.apache.org/licenses

[10] ANDROID ARCHITECTURE [online]. JavaTpoint, 2011-2018 [cit. 2020-04-18]. Dostupné z: https://www.javatpoint.com/android-software-stack

[11] ANDROID ARCHITECTURE [online]. Google LLC, 2020 [cit. 2020-04-18]. Dostupné z: https://source.android.com/devices/architecture

[12] INTELLIJ IDEA OVERVIEW [online]. JetBrains s.r.o., 2000-2020 [cit. 2020-04-18]. Dostupné z: https://www.jetbrains.com/help/idea/discover-intellij-idea.html

[13] ROUSE, M. 2018: Android Studio [online]. TechTarget, 2018 [cit. 2020-04-18]. Dostupné z: https://searchmobilecomputing.techtarget.com/definition/Android-Studio

[14] IOS [online]. Techopedia Inc., 2020 [cit. 2020-04-18]. Dostupné z: https://www.techopedia.com/definition/25206/ios

[15] HEIN, B. 2017: The evolution of iOS: From iPhone OS to iOS 11 [online]. Cult of Mac, 2017 [cit. 2020-04-18]. Dostupné z: https://www.cultofmac.com/488454/ios-evolution-iphone-os

[16] GRADLE USER MANUAL [online]. Gradle Inc, 2020 [cit. 2020-04-18]. Dostupné z: https://docs.gradle.org/current/userguide/userguide.html

[17] IOS ARCHITECTURE [online]. IntelliPaat, 2019 [cit. 2020-04-18]. Dostupné z: https://intellipaat.com/blog/tutorial/ios-tutorial/ios-architecture

[18] UIKIT FRAMEWORK [online]. Apple Inc., 2020 [cit. 2020-04-18]. Dostupné z: https://developer.apple.com/documentation/uikit

[19] XCODE OVERVIEW: AT A GLANCE [online]. Apple Inc., 2016 [cit. 2020-04-18]. Dostupné z: https://developer.apple.com/library/archive/documentation/ToolsLanguages/Conceptual/Xcode_Overview/index.html

[20] LASTOVETSKA, A. 2018: Native App Development vs. Hybrid and Web App Building [online]. MLSDev, 2018 [cit. 2020-04-18]. Dostupné z: https://mlsdev.com/blog/native-app-development-vs-web-and-hybrid-app-development

[21] ZAMMETTI, Frank. Practical React Native : build two full projects and one full game using React Native. 1. Berkeley, CA: Apress, 2018 [cit. 2020-04-18]. ISBN 9781484239391.

[22] YUSUF, Sani. Ionic Framework By Example. 1. Birmingham, UK: Packt Publishing, 2016. ISBN 9781785282720.

[23] IONIC HISTORY [online]. JavaTpoint, 2011-2018 [cit. 2020-04-18]. Dostupné z: https://www.javatpoint.com/ionic-history

[24] 6 PROS AND 3 CONS OF IONIC DEVELOPMENT [online]. Ukad Ltd., nedatováno [cit. 2020-04-18]. Dostupné z: https://www.ukad-group.com/blog/6-pros-and-3-cons-of-ionic-development

[25] GAËL, T. 2019: What is Flutter and Why You Should Learn it in 2020 [online]. FreeCodeCamp, 2019 [cit. 2020-04-18]. Dostupné z: https://www.freecodecamp.org/news/what-is-flutter-and-why-you-should-learn-it-in-2020

[26] A BRIEF INTRODUCTION TO FLUTTER [online]. Dream Squad, 2019 [cit. 2020-04-18]. Dostupné z: https://www.dreamsquadgroup.com/whats-flutter-brief-introduction

[27] MROCZKOWSKA, A. 2019: Flutter Pros & Cons for Mobile App Owners [online]. Droids On Roids Sp., 2019 [cit. 2020-04-18]. Dostupné z: https://www.thedroidsonroids.com/blog/flutter-in-mobile-app-development-pros-and-cons-for-app-owners

[28] WHY CHOOSE FLUTTER? PROS & CONS [online]. Iteo, 2019 [cit. 2020-04-18]. Dostupné z: https://medium.com/@iteo/why-choose-flutter-pros-cons-46cd12881d14

[29] AGGARWAL, A. nedatováno: Introduction to Visual Studio [online]. GeeksForGeeks, nedatováno [cit. 2020-04-18]. Dostupné z: https://www.geeksforgeeks.org/introduction-to-visual-studio

[30] VONDRA, Ondřej, Jan DVOŘÁK, Ondřej KREJCAR a Petr BRIDA. Detection of Drivers Plate at Smart Driver's Score Application Controlled by Voice Commands. Intelligent information and database systems. 2019, 2019, 830, 349-361. DOI: 978-3-030-14131-8.

[31] INTELLISENSE IN VISUAL STUDIO [online]. Microsoft, 2018 [cit. 2020-04-18]. Dostupné z: https://docs.microsoft.com/en-gb/visualstudio/ide/using-intellisense

[32] WHAT IS BASE64? [online]. Base64 guru, 2020 [cit. 2020-04-18]. Dostupné z: https://base64.guru/learn/what-is-base64

8 Přílohy

Seznam příloh

Příloha A – Zadání diplomové práce

Zadání diplomové práce

Autor:	**Bc. Ondřej Vondra**
Studium:	I1600891
Studijní program:	N1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název diplomové práce:	**Srovnání multiplatformního nástroje Xamarin s platformně zaměřeným vývojem**
Název diplomové práce AJ:	Comparison of multiplatform tool Xamarin with platform-oriented development

Cíl, metody, literatura, předpoklady:

Cílem práce je zpracovat srovnání vývoje multiplatformních aplikací za použití nástroje Xamarin s platformně zaměřeným vývojem pro platformy Android a iOS. V teoretické části práce autor představí použité technologie, nástroje a způsoby srovnání. V praktické části poté autor vytvoří multiplatformní aplikaci s využitím nástroje Xamarin a porovná možnosti vývoje s téměř identickou aplikací vyvíjenou platfomně závislou cestou. Osnova: Úvod Představení technologií a nástrojů Tvorba aplikace Srovnání vývoje Vyhodnocení výsledků Závěr

TAK, Rohin a Jhalak MODI. Mobile DevOps. 1. Birmingham: Pack Publishing, 2018. ISBN 978-1-78829-624-3.

Garantující pracoviště:	Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu
Vedoucí práce:	doc. Mgr. Tomáš Kozel, Ph.D.

Datum zadání závěrečné práce: 14.1.2015