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

# Porovnání mobilních frameworků

Diplomová práce

Autor: Bc. Jakub Beneš
Studijní obor: Aplikovaná informatika

Vedoucí práce: doc. Ing. Filip Malý, Ph.D.

Odborný konzultant: Ing. Roman Jašek

Hradec Králové Duben 2020

Prohlášení:

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

V Hradci Králové dne 27.4.2020 Bc. Jakub Beneš

Poděkování:

Děkuji vedoucímu diplomové práce doc. Ing. Filipovi Malému, Ph.D. za metodické vedení práce, odborné rady a vstřícný přístup. Dále děkuji Ing. Romanovi Jaškovi za odborné a cenné konzultace při zpracování diplomové práce.

# Anotace

Práce seznamuje čtenáře o způsobech vývoje multiplatformních mobilních aplikací a s frameworky, které se pro jejich vývoj používají. V práci jsou podrobně prozkoumány nejznámější a nejvíce používané frameworky, ale i frameworky známé méně.

Dále se práce zabývá popisem dnes nejrozšířenějších mobilních operačních systémů, na kterých je vývoj aplikací silně závislý. Popsány jsou i možnosti distribuce mobilních aplikací na zmíněných platformách.

V praktické části této práce jsou navrženy testovací scénáře. Na navržených testovacích scénářích jsou porovnány možnosti frameworků z pohledu využitelnosti napříč mobilními operačními systémy. Zkoumána je jejich rychlost, výkon, ale i složitost jejich použití při vývoji multiplatformní mobilní aplikace.

# Annotation

## Title: Comparison of mobile frameworks

The thesis describes the ways of developing multiplatform mobile applications and frameworks that are used for their development. The work examines in detail the best known and most used frameworks, but also frameworks known less.

The second part deals with the description of today's most widespread mobile operating systems, on which the development of applications is heavily dependent. The possibilities of distribution of mobile applications on the mentioned platforms are also described.

In the practical part of this work, test scenarios are proposed. The proposed test scenarios compare the possibilities of frameworks in terms of usability across mobile operating systems. The scenarios examine their speed, performance, but also the complexity of their use in the development of a multiplatform mobile application.

# Obsah

1 Úvod ... 1
1.1 Cíl práce ... 2
2 Vývoj multiplatformních mobilních aplikací ... 3
2.1 Mobilní operační systémy ... 3
2.2 Přístupy k vývoji mobilních aplikací ... 5
2.2.1 Webové aplikace ... 5
2.2.2 Nativní aplikace ... 7
2.2.3 Hybridní aplikace ... 8
2.2.4 Multiplatformní aplikace (Cross-Platform) ... 10
2.2.5 Porovnání ... 12
3 Frameworky pro tvorbu multiplatformních mobilních aplikací ... 14
3.1 React Native ... 15
3.1.1 Architektura frameworku ... 15
3.1.2 Zkušenosti vývojářů ... 16
3.2 Flutter ... 17
3.2.1 Architektura frameworku ... 18
3.2.2 Zkušenosti vývojářů ... 18
3.3 Xamarin ... 19
3.3.1 Architektura frameworku ... 20
3.3.2 Zkušenosti vývojářů ... 21
3.4 NativeScript ... 21
3.5 Qt ... 22
3.6 Fuse Open ... 23
3.7 Weex ... 23
3.8 Herní frameworky ... 24

3.8.1 Unity ... 25
3.8.2 Unreal Engine ... 25
3.8.3 Corona ... 26
3.9 Porovnání multiplatformních mobilních frameworků ... 26
4 Návrh testovacích scénářů ... 28
4.1 Testovací scénář 1 – instalace, vytvoření projektu a ladění aplikace ... 28
4.2 Testovací scénář 2 – tvorba uživatelského rozhraní a navigace ... 29
4.3 Testovací scénář 3 – načítání a zobrazení seznamu objektů z REST API ... 30
4.4 Použité technologie a nástroje ... 30
4.4.1 NPM ... 30
4.4.2 Node.js ... 30
4.4.3 Express ... 31
4.4.4 Android Emulator ... 31
4.4.5 JavaScript ... 31
4.4.6 Dart ... 32
4.4.7 Mono ... 34
5 Implementace testovacích scénářů ve vybraných frameworcích ... 35
5.1 Vytvoření testovacího API ... 35
5.2 React Native ... 38
5.2.1 Testovací scénář 1 – instalace, vytvoření projektu a ladění aplikace ... 39
5.2.2 Testovací scénář 2 - tvorba uživatelského rozhraní a navigace ... 41
5.2.3 Testovací scénář 3 - načítání a zobrazení seznamu objektů z REST API 44
5.3 Flutter ... 48
5.3.1 Testovací scénář 1 - instalace, vytvoření projektu a ladění aplikace ... 49
5.3.2 Testovací scénář 2 - tvorba uživatelského rozhraní a navigace ... 50

5.3.3 Testovací scénář 3 - načítání a zobrazení seznamu objektů z REST API 52

5.4 Xamarin ... 56

5.4.1 Testovací scénář 1 - instalace, vytvoření projektu a ladění aplikace ... 57

5.4.2 Testovací scénář 2 - tvorba uživatelského rozhraní a navigace ... 59

5.4.3 Testovací scénář 3 - načítání a zobrazení seznamu objektů z REST API 61

6 Shrnutí výsledků ... 65

6.1 Testovací scénář 1 – instalace, vytvoření projektu a ladění aplikace ... 65

6.2 Testovací scénář 2 - tvorba uživatelského rozhraní a navigace ... 66

6.3 Testovací scénář 3 - načítání a zobrazení seznamu objektů z REST API ... 68

7 Závěry a doporučení ... 72

8 Seznam použité literatury ... 73

# Seznam obrázků

Obrázek 1: Podíl mobilních OS na celosvětovém trhu. [3] ..... 4
Obrázek 2: Podíl zobrazení webových stránek na mobilních zařízeních od roku 2009 do roku 2018. [9] ..... 6
Obrázek 3: Princip hybridních mobilních aplikací. [14] ..... 9
Obrázek 4: Princip cross-platform mobilních aplikací. [14] ..... 11
Obrázek 5: Graf znázorňující zájem o vybrané multiplatformní mobilní frameworky. [19] ..... 14
Obrázek 6: Znázornění interakce frameworku React Native s nativními komponentami. [21] ..... 16
Obrázek 7: Architektura frameworku Flutter. [21] ..... 18
Obrázek 8: Architektura frameworku Xamarin. [31] ..... 20
Obrázek 9: Architektura frameworku NativeScript. [34] ..... 22
Obrázek 10: Srovnání zastoupení aplikací v obchodě App Store podle typu – listopad 2019. [40] ..... 24
Obrázek 11: Návrh aplikace pomocí nástroje Miro. [Zdroj: Autor práce] ..... 29
Obrázek 12: Možnost zvolení šablony projektu ve frameworku React Native. [Zdroj: Autor práce] ..... 40
Obrázek 13: Struktura projektu React Native a ukázka hlavní komponenty App. [Zdroj: Autor práce] ..... 40
Obrázek 14: Uživatelské rozhraní úvodní obrazovky a obrazovky druhého scénáře – React Native. [Zdroj: Autor práce] ..... 44
Obrázek 15: Využití zdrojů při neaktivitě – React Native. [Zdroj: Autor práce] ..... 46
Obrázek 16: Využití zdrojů při práci s velkým množstvím dat – 1. test React Native. [Zdroj: Autor práce] ..... 47
Obrázek 17: Využití zdrojů při práci s velkým množstvím dat – 2. test React Native. [Zdroj: Autor práce] ..... 47
Obrázek 18: Využití zdrojů při práci s velkým množstvím dat – 3. test React Native. [Zdroj: Autor práce] ..... 47
Obrázek 19: Kontrola správného nainstalování frameworku Flutter. [Zdroj: Autor práce] ..... 49

Obrázek 20: Struktura projektu Flutter a ukázka části hlavního widgetu aplikace. [Zdroj: Autor práce] ..... 50
Obrázek 21: Uživatelské rozhraní úvodní obrazovky a obrazovky druhého scénáře – Flutter. [Zdroj: Autor práce] ..... 52
Obrázek 22: Využití zdrojů při neaktivitě – Flutter. [Zdroj: Autor práce] ..... 54
Obrázek 23: Využití zdrojů při práci s velkým množstvím dat – 1. test Flutter. [Zdroj: Autor práce] ..... 55
Obrázek 24: Využití zdrojů při práci s velkým množstvím dat – 2. test Flutter. [Zdroj: Autor práce] ..... 55
Obrázek 25: Využití zdrojů při práci s velkým množstvím dat – 3. test Flutter. [Zdroj: Autor práce] ..... 55
Obrázek 26: Možnost zvolení šablony projektu ve frameworku Xamarin. [Zdroj: Autor Práce] ..... 58
Obrázek 27: Struktura projektu Xamarin a ukázka třídy App. [Zdroj: Autor práce] ..... 59
Obrázek 28: Uživatelské rozhraní úvodní obrazovky a obrazovky druhého scénáře – Xamarin. [Zdroj: Autor práce] ..... 61
Obrázek 29: Využití zdrojů při neaktivitě – Xamarin. [Zdroj: Autor práce] ..... 62
Obrázek 30: Využití zdrojů při práci s velkým množstvím dat – 1. test Xamarin. [Zdroj: Autor práce] ..... 63
Obrázek 31: Využití zdrojů při práci s velkým množstvím dat – 2. test Xamarin. [Zdroj: Autor práce] ..... 63
Obrázek 32: Využití zdrojů při práci s velkým množstvím dat – 3. test Xamarin. [Zdroj: Autor práce] ..... 63
Obrázek 33: Porovnání vytvořeného uživatelského rozhraní ve vybraných frameworcích. [Zdroj: Autor práce] ..... 67
Obrázek 34: Porovnání vytvořeného uživatelského rozhraní načtených dat ve vybraných frameworcích ..... 69

# Seznam tabulek

Tabulka 1: Porovnání přístupů k vývoji multiplatformním mobilních aplikací. [Zdroj: Autor práce] ..... 12
Tabulka 2: Porovnání hybridního a multiplatformního přístupu. [14] ..... 13
Tabulka 3: Porovnání multiplatformních mobilních frameworků. [Zdroj: Autor práce] ..... 27
Tabulka 4: Přehled naměřených přibližných hodnot – React Native. [Zdroj: Autor práce] ..... 48
Tabulka 5: Přehled naměřených přibližných hodnot – Flutter. [Zdroj: Autor práce] ..... 56
Tabulka 6: Přehled naměřených přibližných hodnot – Xamarin. [Zdroj: Autor práce] ..... 64
Tabulka 7: Porovnání vybraných frameworků – 1. testovací scénář. [Zdroj: Autor práce] ..... 66
Tabulka 8: Porovnání vybraných frameworků – 2. testovací scénář. [Zdroj: Autor práce] ..... 68
Tabulka 9: Porovnání naměřených hodnot při neaktivitě. [Zdroj: Autor práce] ..... 69
Tabulka 10: Porovnání naměřených hodnot při načítání dat. [Zdroj: Autor práce] 70
Tabulka 11: Porovnání naměřených hodnot při neaktivitě po načtení dat. [Zdroj: Autor práce] ..... 70
Tabulka 12: Porovnání naměřených hodnot při procházení dat. [Zdroj: Autor práce] ..... 71

# Seznam ukázek kódů

Ukázka kódu 1: Použití jazyka JavaScript. [Zdroj: Autor práce] ..... 32
Ukázka kódu 2: Ukázka deklarace a volání funkcí v jazyce Dart. [56] ..... 33
Ukázka kódu 3: Vytvoření testovacích dat a uložení do souboru. [Zdroj: Autor práce] ..... 36
Ukázka kódu 4: Obsah vygenerovaného souboru data.json. [Zdroj: Autor práce] .. 37
Ukázka kódu 5: Zdrojový kód testovacího API. [Zdroj: Autor práce] ..... 38
Ukázka kódu 6: Vytvoření jednoduché komponenty – React Native. [60] ..... 38
Ukázka kódu 7: Vytvoření jednoduché komponenty – React. [Zdroj: Autor práce] 39
Ukázka kódu 8: Vytvoření navigace pomocí knihovny react-navigation. [Zdroj: Autor práce] ..... 42
Ukázka kódu 9: Přechod na jinou obrazovku pomocí tlačítka. [Zdroj: Autor práce] ..... 42
Ukázka kódu 10: Použití předdefinovaných komponent – React Native. [Zdroj: Autor práce] ..... 43
Ukázka kódu 11: Načtení dat ze síťového zdroje – React Native. [Zdroj: Autor práce] ..... 45
Ukázka kódu 12: Uživatelské rozhraní pro zobrazení seznamu načtených dat. [Zdroj: Autor práce] ..... 46
Ukázka kódu 13: Použití základních widgetů. [64] ..... 49
Ukázka kódu 14: Metoda, která provádí změnu aktivní obrazovky – Flutter. [Zdroj: Autor práce] ..... 51
Ukázka kódu 15: Část zdrojového kódu uživatelského rozhraní úvodní obrazovky – Flutter. [Zdroj: Autor práce] ..... 52
Ukázka kódu 16: Přidání závislosti balíčku http – Flutter. [Zdroj: Autor práce] ..... 52
Ukázka kódu 17: Import balíčku http v hlavičce souboru – Flutter. [Zdroj: Autor práce] ..... 53
Ukázka kódu 18: Třída určená pro práci s objekty získanými z API – Flutter. [Zdroj: Autor práce] ..... 53
Ukázka kódu 19: Načtení dat ze síťového zdroje – Flutter. [Zdroj: Autor práce] ..... 54

Ukázka kódu 20: Vytvoření uživatelského rozhraní pomocí jazyka XAML - Xamarin. [67] ..... 56
Ukázka kódu 21: Obsluha kliknutí a provedení změny aktivní obrazovky pomocí NavigationPage – Xamarin. [Zdroj: Autor práce] ..... 60
Ukázka kódu 22: Část kódu uživatelského rozhraní úvodní obrazovky v jazyce XAML – Xamarin. [Zdroj: Autor práce] ..... 60
Ukázka kódu 23: Načtení dat ze síťového zdroje – Xamarin. [Zdroj: Autor práce] . 61
Ukázka kódu 24: Třída ScenarioItem – Xamarin. [Zdroj: Autor práce] ..... 62

# Seznam zkratek

PWA ..... Progressive web apps
IDE ..... Integrated Development Environment
SDK ..... Software development kit
UI ..... User interface
JSC ..... JavaScriptCore
NDK ..... Native Development Kit
LLVM ..... Low Level Virtual Machine
AOT ..... Ahead-of-time
IL ..... Intermediate Language
CLI ..... Command Line Interface
QML ..... Qt Modeling Language
XML ..... Extensible Markup Language
XAML ..... Extensible Application Markup Language
API ..... Application Programming Interface
GPS ..... Global Positioning System
RAM ..... Random Access Memory
CPU ..... Central Processing Unit

# 1 Úvod

Mobilní aplikace jsou v dnešní době součástí našeho každodenního života. Ráno nás probouzí a dávají nám informace o kvalitě našeho spánku. Při snídani nás informují o událostech z celého světa a o počasí, které by nás daný den mohlo překvapit. Cestou do práce nahrazují navigace, hudební přehrávače, diáře či knihy. V práci jsou využívány jako běžné pracovní nástroje, běžně nahrazují stolní počítače a notebooky. Využíváme je i při sportovních aktivitách, kde měří naše výkony, počítají spálené kalorie a motivují nás do dalších výkonů.

Mobilní aplikace dnes již nenajdeme pouze v našich telefonech či tabletech. Objevila se i další zařízení, na která jsou aplikace vyvíjeny či jednoduše přenášeny z mobilních verzí. Při zmíněných sportovních aktivitách lidé využívají např. chytré hodinky, které mohou díky speciálnímu hardwaru měřit aktivitu mnohem přesněji než mobilní telefony. Takovýto speciální hardware neměří pouze uběhnuté kilometry či vystoupaná podlaží, ale dává uživatelům informace o jejich zdraví. Běžné je měření srdečního tepu či hluku v okolí, který by mohl mít nežádoucí dopad na sluch uživatele. Naměřené údaje zastřešují aplikace, které jejich uživatelům poskytují podrobná aktuální i historická data. Aplikace může naměřené hodnoty poskytovat dále např. zdravotnímu zařízení, které může data analyzovat a rychle reagovat na případné zdravotní komplikace.

Z dnešního pohledu původně jednoduché aplikace se postupem času vyvinuly ve velmi komplexní software, který umožňuje provázání a synchronizaci s jinými systémy. Při jejich vývoji je nutné myslet nejen na kompatibilitu s různými druhy telefonů a tabletů, ale i s chytrými hodinkami, televizory či automobily. Celý vývoj komplikuje existence více platforem, které z historických a konkurenčních důvodů neumožňovaly vyvíjet aplikace jednotným způsobem. Tento problém časem vyřešily až multiplatformní frameworky, kterými se bude dále zabývat tato diplomová práce.

## 1.1 Cíl práce

Cílem této práce je prozkoumat a seznámit čtenáře s možnostmi vývoje multiplatformních mobilních aplikací a navzájem je porovnat z hlediska výkonu, rychlosti vývoje, distribučních možností či ceny.

První část práce si klade za cíl vysvětlit pojmy mobilní aplikace a mobilní operační systém. Zároveň budou představeny různé přístupy k vývoji mobilních aplikací.

Tímto se práce posune do další části, do oblasti multiplatformních frameworků a vývoje multiplatformních mobilních aplikací. Autor práce si klade za cíl provést rozsáhlý průzkum těchto frameworků a porovnat je podle různých kritérií. Při porovnání bude kladen důraz na frameworky, které se co nejvíce blíží nativnímu vývoji. Záměrně jsou vynechány frameworky webové a hybridní.

Poslední kapitoly teoretické části se budou věnovat návrhu vhodných testovacích scénářů. Cílem je navrhnout takové scénáře, na kterých bude možné vhodně otestovat různé vlastnosti mobilních frameworků. Bude se jednat o samostatné scénáře, které nemají za cíl fungovat jako jedna ucelená aplikace.

Cílem praktické části této diplomové práce je implementovat navrhnuté scénáře ve vybraných multiplatformních frameworcích. Popsat postup vývoje navrhnutých scénářů, v každém z vybraných frameworků a následně je porovnat podle předem definovaných kritérií.

# 2 Vývoj multiplatformních mobilních aplikací

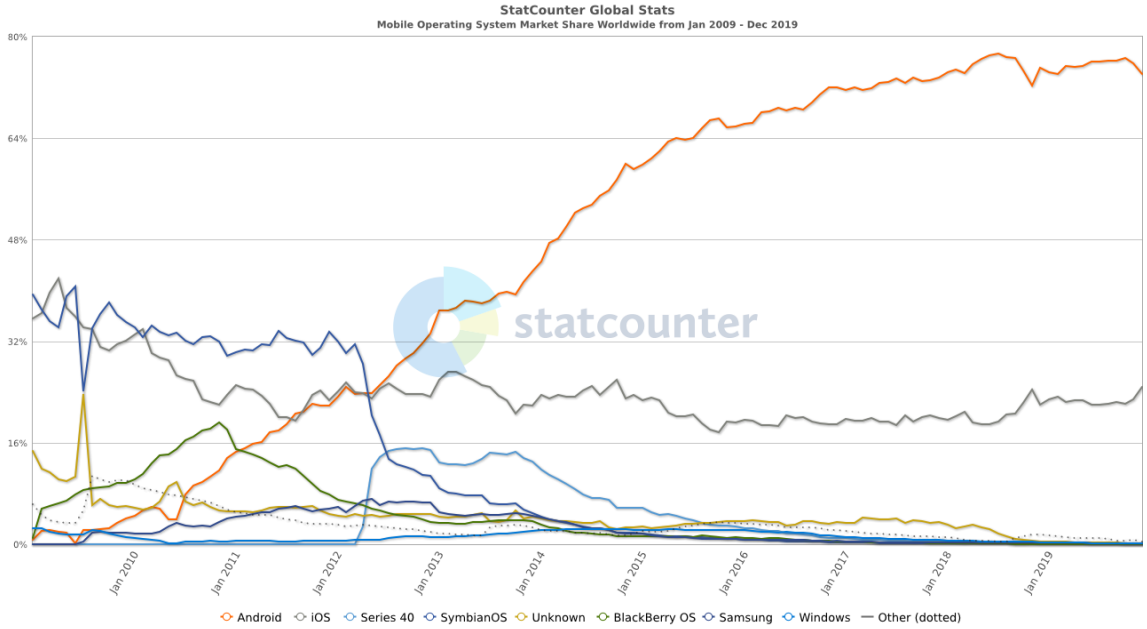
Jak již bylo zmíněno v úvodu této diplomové práce, mobilních aplikací existuje obrovské množství a využíváme je v různých oblastech našeho života. Podle článku Brief History of Mobile Apps [1] se původně jednalo o velmi jednoduché programy např. kalkulačky, kalendáře, editory vyzváněcích tónů či malé arkádové hry. Později se objevily první aplikace, které umožňovaly emailovou komunikaci. Důležité je zmínit, že v době vzniku těchto prvních aplikací neexistovaly žádné distribuční služby, které by umožňovaly uživatelům nakupovat a stahovat aplikace nové. Touto skutečností byli omezeni i vývojáři, kteří neměli možnost svoje aplikace jednoduchým způsobem distribuovat jako dnes.

Z technického pohledu je mobilní aplikace software napsaný v určitém programovacím jazyce. Výsledný kód je pomocí kompilátoru zkompilován do spustitelné podoby. Podoba spustitelného programu a jazyk, ve kterém je možné zdrojový kód aplikace psát, jsou závislé na operačním systému, pro který je mobilní aplikace určena.

## *2.1 Mobilní operační systémy*

Podle webu operating-system.org [2] existuje obrovské množství operačních systémů, přičemž je možné je rozdělit do různých kategorií. Některé systémy jsou určeny pro stolní počítače a notebooky, jiné pro mobilní telefony a tablety, další zase pro chytré hodinky, automobily a jiné elektrické spotřebiče. Často vznikají nové operační systémy na základě jiných operačních systémů. Typickým příkladem mohou být různé Linuxové distribuce. Dalším příkladem mohou být menší systémy, které jsou určeny např. pro již několikrát zmiňované chytré hodinky. Takovéto systémy jsou mnohem menší, mají omezené funkce, ale jsou dostatečné z hlediska jejich účelu.

Na poli mobilních telefonů a tabletů jsou nejvíce zastoupeny dva operační systémy Android a iOS. Na grafu níže je možné pozorovat, že si podíl celosvětového trhu dělí nerovnoměrným dílem. V roce 2019 společnost Google se systémem Android ovládala více jak 74 % celosvětového trhu. Jeho konkurent, společnost Apple, ovládala necelých 25 %. Zbylá procenta patří ostatním operačním systémům.




**Obrázek 1: Podíl mobilních OS na celosvětovém trhu. [3]**

Je třeba zmínit, že se podíly rapidně mění ve chvíli, kdy se nedíváme na celosvětový trh, ale zaměříme se např. jen na Spojené Státy Americké. Na americkém trhu je boj zmíněných dvou společností mnohem více vyrovnaný. Společnost Apple ovládá dokonce větší část trhu [4].

Evropský podíl trhu je v současnosti velmi podobný celosvětovému [5]. Dále může být zajímavý např. trh čínský, kde je podíl systému Android ještě vyšší než podíl celosvětový [6].

Společnost Apple představila svůj první smartphone nazvaný iPhone již v roce 2007 [2]. O několik let dříve, v roce 2003, vznikla ve státu Kalifornie společnost Android Inc. Jejím cílem bylo vyvinout chytrá zařízení, která budou znát svého majitele, a budou tak schopna mnohem lépe vyhovovat jeho potřebám [3]. O několik let později, v roce 2005, byla společnost Android Inc odkoupena společností Google, která ve vývoji tohoto systému pokračuje dodnes.

Krátce po představení zařízení iPhone, společnost Google oznámila, že jejich vizí je vyvinout výkonný systém, který bude pohánět tisíce různých modelů mobilních telefonů [7]. Tuto vizi Google proměnil ve skutečnost, a to je právě jeden z důvodů, proč společnost Google ovládá většinu trhu. Android je distribuován jako open-source a je tedy volně dostupný všem výrobcům telefonů. Tito výrobci vytvářejí vlastní nadstavby systému. Je tedy běžné, že systém vypadá

z uživatelského hlediska, u každého výrobce telefonu trochu jinak. Jedním z dalších důvodů, proč je Android tím více zastoupeným systémem, je cena. Telefony využívající tento systém se prodávají ve všech cenových kategoriích [8]. Oproti tomu jsou chytré telefony iPhone prodávány jako telefony exkluzivní a spadají do vyšší cenové kategorie [8].

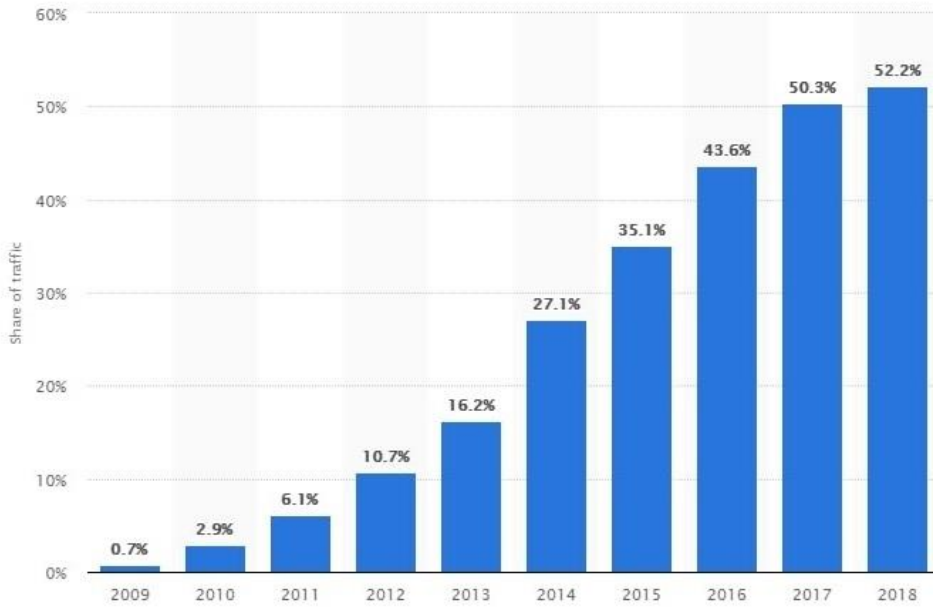
## 2.2 Přístupy k vývoji mobilních aplikací

Před začátkem vývoje mobilní aplikace stojí před vývojáři důležité rozhodnutí. Je nutné si uvědomit, o jaký druh aplikace se jedná, k čemu bude sloužit a jaké jsou její požadavky a cíle. Některé mohou být komplexní a náročné na výkon, jiné mohou poskytovat pouze jednoduchou funkci. Před začátkem vývoje by si měli vývojáři vždy odpovědět na základní otázku, na jakých platformách má mobilní aplikace fungovat. Po ujasnění cílů a požadavků mají vývojáři několik cest, kterými se mohou při vývoji vydat.

### 2.2.1 Webové aplikace

Webové mobilní aplikace fungují na principu zobrazení webové stránky pomocí prohlížeče. Vývojáři vytvoří běžnou webovou aplikaci pomocí technologií jako jsou HTML5, JavaScript či CSS3. Při vývoji je nutné myslet na optimalizaci pro různé druhy prohlížečů. Uživatelé mobilních zařízení pro spuštění aplikace použijí nainstalovaný prohlížeč. Tímto odpadá nutnost instalovat do mobilního zařízení další software. Tato skutečnost by mohla být důvodem tvrdit, že se nejedná o mobilní aplikaci v pravém slova smyslu.

V dnešní době jsou takovéto aplikace více než běžné. Sociální sítě, e-shopy či internetové magazíny bývají optimalizované pro mobilní zařízení. Takováto optimalizace je v dnešní době již standardem a není pro vývojáře žádnou velkou překážkou. Vývojářům pomáhají technologie jako Bootstrap, Bulma či UIKIT. Na grafu níže je znázorněno procento zobrazení webových stránek na mobilních zařízeních za poslední roky.




**Obrázek 2: Podíl zobrazení webových stránek na mobilních zařízeních od roku 2009 do roku 2018. [9]**

Jednoduchý a rychlý vývoj webových mobilní aplikací nejsou jediné výhody tohoto přístupu. Na druhou stranu má webový vývoj i mnoho nevýhod, které by mohly značně ovlivnit rozhodnutí vývojářů o jeho využití.

**Výhody webové mobilní aplikace:**

- Vývoj pomocí běžných webových technologií.
- Pro spuštění stačí webový prohlížeč, není nutné instalovat další aplikace. Odpadá nutnost distribuce aplikace skrze Google Play či AppStore.
- Jednotný vývoj pro všechny platformy.
- Jednoduché aktualizace, které není nutné stahovat do zařízení. Provedou se jednou v hlavní webové aplikaci.

**Nevýhody webové mobilní aplikace:**

- Spouštět aplikaci skrze webový prohlížeč může být nepraktické.
- Přestože je možné aplikaci zobrazit na všech platformách, je nutné myslet na optimalizaci pro různé prohlížeče.
- Výkon webových aplikací je oproti jiným typům řádově nižší.

- Chybí přístup k hardwaru zařízení např. k fotoaparátu.
- Nemožnost využívat aplikaci bez přístupu k internetu.
- Nepřítomnost aplikací v distribučních obchodech může být nevýhodou u aplikací, které tyto obchody chtějí využít jako nástroj pro vlastní propagaci.

#### 2.2.1.1PWA

PWA neboli Progressive Web Apps jsou webové aplikace rozšířeny o moderní nativní API, které těmto aplikacím poskytuje možnost využívat nativní funkce systémů, na kterých jsou spouštěny. Autoři článku What are Progressive Web Apps [10] zmiňují, že PWA byly navrženy tak, aby byly schopné, spolehlivé a aby bylo možné tyto aplikace instalovat na koncová zařízení stejně jako aplikace nativní.

Podle autorů stejného článku [10] je zmíněnou schopností PWA myšleno, že aplikace jsou schopny přistupovat k souborovému systému, notifikacím, či že mají plnou kontrolu nad schránkou sloužící k přenášení dat. Spolehlivosti aplikací je dosaženo jejich rychlostí a schopností fungovat nezávisle na síle síťového připojení. Rychlost je klíčovou vlastností, která uživatelům poskytne nativní pocit z používání dané aplikace. U nativních aplikací uživatelé očekávají, že jejich spouštění bude rychlé a hladké i v případě slabého internetového připojení či dokonce připojení žádného.

Obrovská výhoda PWA přichází s možností dané aplikace instalovat a spouštět např. z domovské obrazovky zařízení. S touto možností přichází i rozšířené možnosti distribuce přes distribuční obchody jako je Google Play či App Store.

### 2.2.2 Nativní aplikace

Nativní mobilní aplikace je software vyvinutý speciálně pro jednu konkrétní platformu. V závislosti na tom, pro kterou platformu je aplikace vyvíjena, je použit konkrétní programovací jazyk a další nástroje, které jsou s danou platformou úzce spojeny. Pokud je tedy aplikace vyvinuta pro platformu Android, nelze ji spustit na zařízeních, která jsou poháněna systémy od společnosti Apple a naopak. Apple i Google poskytují vývojářům veškeré nástroje, které jsou pro vývoj potřebné.

Hlavní výhodou těchto aplikací je využití plného výkonu daného zařízení s možností přístupu ke všem periferiím a senzorům. Nativní vývoj rovněž dává vývojářům přístup k celé sadě funkcí vybraného operačního systému.

Vývoj nativní aplikace pro Android vyžaduje znalost programovacího jazyka Java nebo Kotlin. Společnost Google poskytuje vývojářům speciální IDE nazvané Android Studio, které umožňuje nejen psát samotný zdrojový kód, ale i navrhovat vzhled aplikace, simulovat ji pomocí virtuálního zařízení či udržovat aktuální verzi SDK.

Aplikace pro systém iOS jsou psány v jazyce Swift nebo Objective-C. Apple rovněž poskytuje speciální IDE nazvané Xcode, které disponuje podobnými funkcemi jako Android Studio. Oficiálně je možné vyvíjet nativní aplikace pro iOS pouze na počítačovém operačním systému macOS.

**Výhody nativní mobilní aplikace:**

- Vysoký výkon, schopnost využít potenciál mobilního zařízení.
- Plný přístup k periferiím a senzorům.
- Možnost aplikace distribuovat skrze online obchody daných platforem.

**Nevýhody nativní mobilní aplikace:**

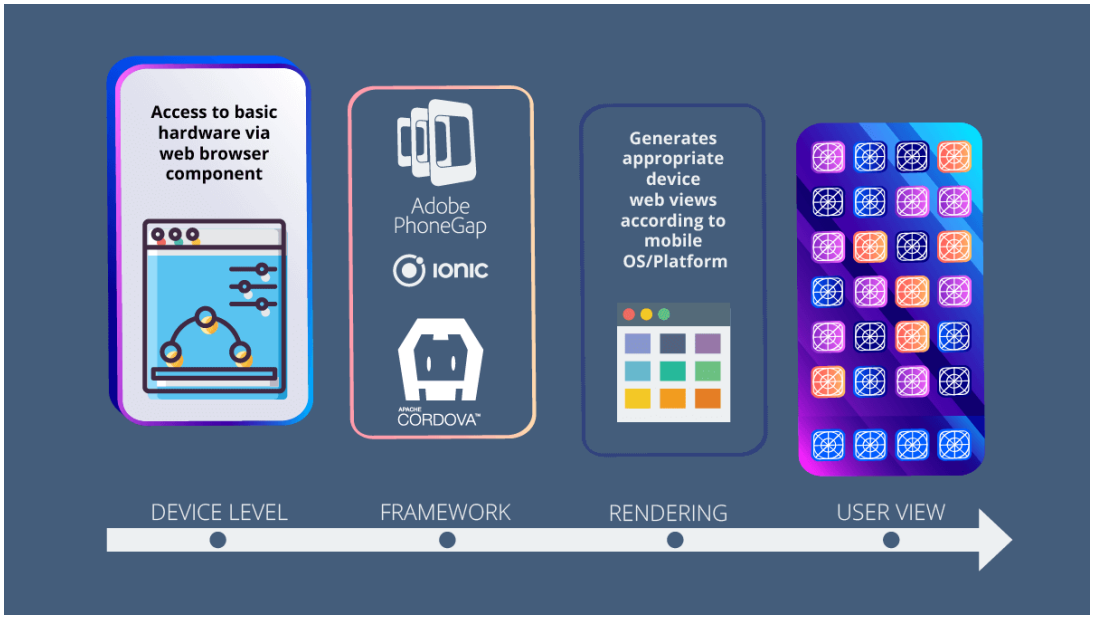
- Nutnost aplikace vyvíjet, pro každou platformu zvlášť.
- Časově náročnější a složitější vývoj.
- Nejsou nejlepší volbou pro velmi jednoduché aplikace z důvodu složitosti vývoje.

### 2.2.3 Hybridní aplikace

Hybridní aplikace kombinují oba předešlé přístupy. Jsou tvořeny pomocí webových technologií, a přesto se částečně chovají jako nativní aplikace. K jejich spouštění není třeba žádný webový prohlížeč, je nutné je do systému nainstalovat a mohou být distribuovány skrze distribuční obchody. Mají přístup k periferiím a senzorům daného zařízení, ale výkonově nedosahují na aplikace nativní.

Obrovskou výhodou hybridních aplikací je jednotný vývoj pro více platforem. Aplikace je vytvořena pomocí webových technologií a zkompilována pomocí

nástrojů třetích stran pro konkrétní platformu. Pod těmito nástroji je možné si představit např. Ionic, PhoneGap či Onsen UI. Autor článku What is Hybrid App Development [11] popisuje princip fungování hybridních aplikací. Jedná se o běžnou webovou stránku, která ale není spouštěna přes uživatelský prohlížeč, ale k jejímu zobrazení je použit prohlížeč vestavěný přímo v daném operačním systému. V systému iOS je používán objekt WKWebView [12], v systému Android objekt WebView [13].




**Obrázek 3: Princip hybridních mobilních aplikací. [14]**

**Výhody hybridní mobilní aplikace**:

- Vývoj pomocí běžných webových technologií.
- Přístup k periferiím a senzorům.
- Multiplatformní vývoj.

**Nevýhody hybridní mobilní aplikace:**

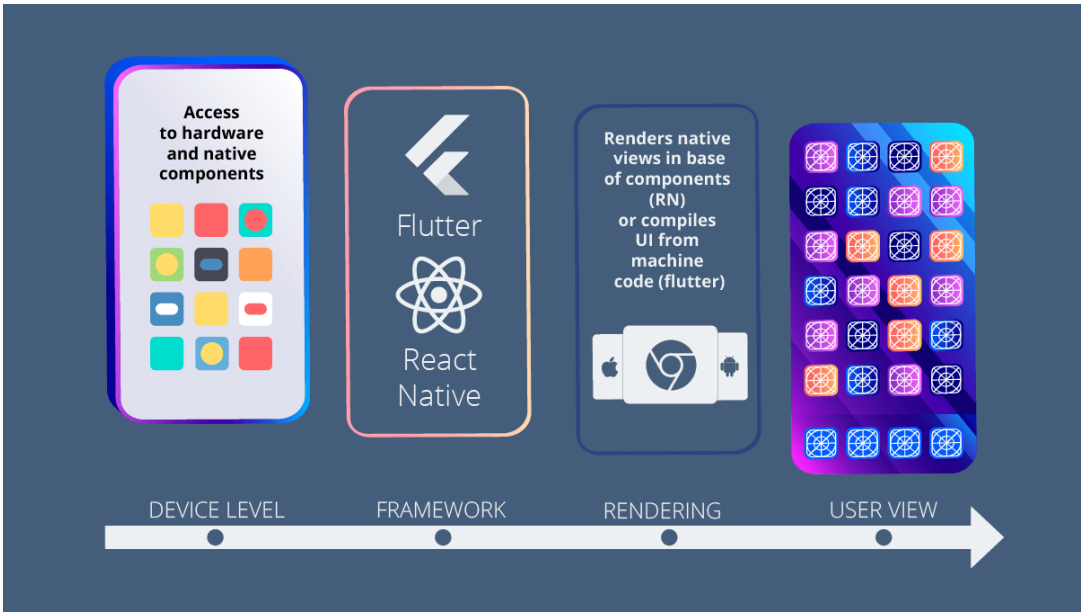
- Výkonově nedosahují na nativní aplikace.
- Pro kompilaci je nutné použít nástroje třetích stran.

### 2.2.4 Multiplatformní aplikace (Cross-Platform)

Multiplatformní vývoj, který se někdy nazývá i cross-platform vývoj [15], se zabývá vývojem multiplatformních mobilních aplikací. Jedná se o aplikace, které jsou vyvinuty v určitých technologiích pouze jednou a následně je možné, je spustit na více platformách. Jak již bylo zmíněno v předešlých kapitolách, mobilní trh si dnes rozdělují převážně dvě platformy, konkrétně Android a iOS. Tímto se stává vývoj multiplatformních aplikací o něco jednodušší.

Hlavním přínosem tohoto způsobu vývoje je bezpochyby nutnost udržovat pouze jeden zdrojový kód. Technologie, které multiplatformní vývoj umožňují, dávají vývojářům přístup ke všem funkcím dané platformy a daného mobilního zařízení. Výsledná aplikace se chová jako aplikace nativní s možností využívat celé hardwarové vybavení zařízení. Takovýto vývoj má i své nevýhody. Multiplatformní aplikace nedosahuje výkonu jako aplikace nativní. Není tedy vhodná na určité druhy aplikací, typicky se může jednat o graficky náročný software [15]. Další nevýhodou mohou být nesrovnalosti v uživatelském rozhraní daných platforem. Obě platformy se chovají jinak z hlediska navigace, tlačítek, seznamů apod. Vývojáři musí při vývoji na tyto nesrovnalosti myslet, aby bylo výsledné řešení uživatelsky přívětivé [15].

Cross-platform vývoji se věnuje např. článek Cross-Platform vs Hybrid – two different stories [14]. Constantin Breahna v článku porovnává hybridní a multiplatformní přístup, a to hned z několika pohledů. Podle autora mají multiplatformní frameworky přístup přímo k nativním komponentám, což má za následek vyšší výkon aplikace. Zmíněnému porovnání se bude práce věnovat v jedné z dalších kapitol.




**Obrázek 4: Princip cross-platform mobilních aplikací. [14]**

**Výhody multiplatformního vývoje:**

- Nutnost udržovat pouze jeden zdrojový kód.
- Výsledné aplikace mají většinu výhod nativních aplikací.
- Rychlejší a levnější vývoj.

**Nevýhody multiplatformního vývoje:**

- Nesrovnalosti v uživatelském rozhraní, které je nutné řešit v jednom zdrojovém kódu.

Multiplatformní vývoj má své výhody a nevýhody. Rychlejší a levnější vývoj, nativní prostředí a jeden zdrojový kód jsou důvody, které zapříčinily vysokou poptávku po takovýchto aplikacích. V posledních letech malé i velké společnosti vyvinuly technologie, které multiplatformní mobilní vývoj umožňují a značně ho ulehčují. Právě těmito technologiemi se budou zabývat další kapitoly této diplomové práce.

### 2.2.5 Porovnání

V předešlých kapitolách byly představeny a popsány různé postupy při vývoji multiplatformních mobilních aplikací. Rovněž byly zmíněny jejich výhody a nevýhody. Z předešlého textu je zřejmé, že každý přístup k vývoji je vhodný pro jiný druh aplikací. Při výběru správného přístupu je nutné zohlednit druh vyvíjené aplikace, výkonovou náročnost, ale např. i cenu, za kterou má být aplikace vytvořena.

Následující tabulka slouží jako srovnání zmíněných přístupů a byla vytvořena na základě několika zdrojů, konkrétně článků The Mobile App Comparison Chart Hybrid vs. Native vs. Mobile Web [16], Native app vs. progressive web app (PWA) Everything you need to know [17] a Progressive Web App vs Native App vs Cross Platform vs Hybrid App [18].

| | **Webová aplikace** | **PWA** | **Nativní aplikace** | **Hybridní aplikace** | **Multiplatformní aplikace** |
|---|---|---|---|---|---|
| **Požadované znalosti** | HTML5, CSS3, JavaScript a další webové technologie. | HTML5, CSS3, JavaScript a další webové technologie. | **iOS** – Swift, Objective-C. **Android** – Java, Kotlin. | HTML5, CSS3, JavaScript + konkrétní framework. | Webové technologie, C#, C++, Dart a další. |
| **Distribuce přes Google Play a App Store / Možnost instalace** | Ne | Ano | Ano | Ano | Ano |
| **Možnost používat v offline režimu** | Ne | Ano | Ano | Ano | Ano |
| **Možnosti využití periferií a senzorů zařízení** | Průměrné | Průměrné | Vysoké | Průměrné | Vysoké |
| **Výkon** | Nízký | Nízký | Vysoký | Nízký | Průměrný |
| **Rychlost vývoje** | Rychlý | Rychlý | Pomalý | Průměrný | Průměrný |
| **Cena vývoje** | Nízká | Nízká | Vysoká | Průměrná | Průměrná |

**Tabulka 1: Porovnání přístupů k vývoji multiplatformním mobilních aplikací. [Zdroj: Autor práce]**

Velmi zajímavé porovnání přináší i již zmíněný článek od Constantina Breahnana [14], který se věnuje porovnání hybridního a multiplatformního přístupu. Autor porovnává výkon, podobnost k nativním aplikacím, cenu a další zajímavé vlastnosti.

| | **Hybridní přístup** | **Multiplatformní přístup** |
|---|---|---|
| **Podobnost k nativním aplikacím** | Pravděpodobně až 40 % ve srovnání s podobnou nativní aplikací. | Pravděpodobně až 80 % ve srovnání s podobnou nativní aplikací. |
| **Odhadovaný přístup k HW a k funkcím OS** | Přibližně 30% až 40% přístup k HW a funkcím OS. | Až 80% přístup k HW a funkcím OS. |
| **Výkon** | Až 2krát pomalejší oproti podobné multiplatformní aplikaci. | Pravděpodobně 1,5krát pomalejší ve srovnání s podobnou nativní aplikací. |
| **Náklady na vývoj** | Velmi malé. | Vyšší ve srovnání s hybridním přístupem. |
| **Příklady frameworků** | Apache Cordova, Ionic, PhoneGap. | React Native, Xamarin, Flutter. |

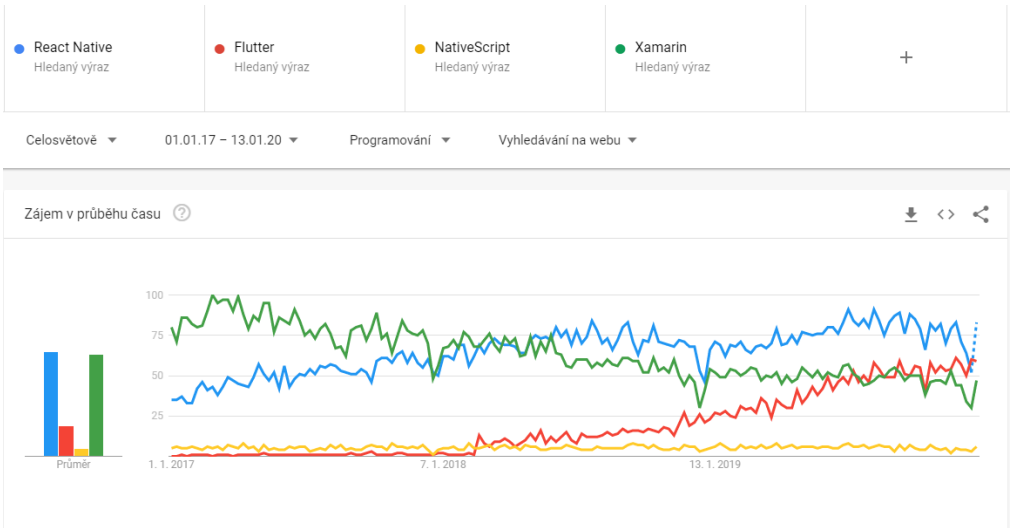
**Tabulka 2: Porovnání hybridního a multiplatformního přístupu. [14]**

# 3 Frameworky pro tvorbu multiplatformních mobilních aplikací

Během posledních několika let se objevují stále nové technologie, které umožňují multiplatformní vývoj. Takovýchto nástrojů existují desítky a jsou poskytovány pod různými licencemi. Najdeme zde frameworky, které jsou poskytovány jako open-source, ale také nástroje, za jejichž využití jsou vývojáři nuceni zaplatit. Licence není to jediné, v čem se frameworky pro tvorbu multiplatformních mobilních aplikací liší. Jednou z prvních vlastností, podle které se vývojáři rozhodují, zda daný framework využijí či nikoliv, je možnost využití konkrétních programovacích jazyků. Některé frameworky umožňují psát aplikace ve webových technologiích jako je JavaScript, HTML5 či CSS3. Jiné frameworky se zaměřují na použití jiných jazyků např. C# či C++.

V této kapitole budou popsány nejznámější a nejvíce používané multiplatformní frameworky. Záměrně jsou vynechány frameworky hybridní a webové. Tato diplomová práce si klade za cíl porovnat čistě multiplatformní frameworky umožňující nativní vývoj. Frameworky budou srovnány z pohledu jejich možností využití, vlastností a licence, se kterou jsou poskytovány.

Podle článku Cross-platform mobile development 2020: trends and frameworks [15] patří mezi nejvýznamnější multiplatformní mobilní frameworky React Native, Flutter, Xamarin a NativeScript.




**Obrázek 5: Graf znázorňující zájem o vybrané multiplatformní mobilní frameworky. [19]**

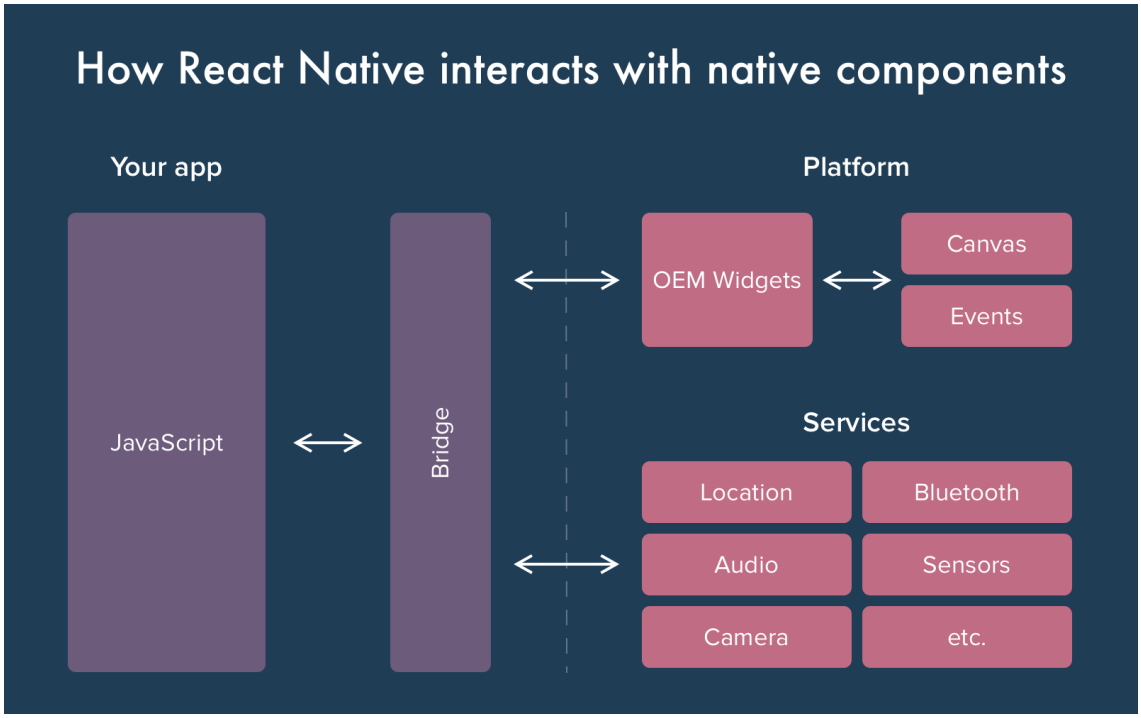
## 3.1 React Native

React Native je open-source multiplatformní mobilní framework vyvinutý společností Facebook. Byl vydán v roce 2015 a od té doby byl použit k vytvoření mnoha velmi populárních aplikací. Jako příklad lze uvést Facebook, Instagram, Skype, Discord či aplikaci Pinterest [20]. V roce 2018 se React Native pochlubil druhým největším počtem přispěvatelů na webové službě GitHub. Jeho vývoj je podporován jednotlivci a společnostmi z celého světa, mezi které můžeme zařadit např. Callstack, Expo či Microsoft [20]. Celkově se může tento framework pochlubit obrovskou komunitou a velkým množstvím knihoven, které je možné při vývoji využívat.

React Native je postavený na technologii React, což je JavaScriptová knihovna, která je určena pro tvorbu uživatelského rozhraní ve webových aplikacích. Pomocí této technologie mohou být vytvářeny malé samostatné komponenty nebo celé aplikace, které jsou nazývány single-page aplikace. Takovéto aplikace se vyznačují tím, že při interakci s nimi není nutné načítat obsah celé stránky. Namísto toho jsou načítány pouze jednotlivé části stránky, které se skutečně mění. Tento přístup má velký vliv na rychlost načítání celé aplikace.

### 3.1.1 Architektura frameworku

Přestože React Native umožňuje vývoj mobilních aplikací pomocí webových technologií, konkrétně JavaScriptu, nejedná se o hybridní multiplatformní framework. Síla frameworku spočívá v nevyužívání objektů jako je např. WebView či WKWebView. Autor článku Flutter vs React Native — Comparing the Features of Each Framework [21] vysvětluje, že architektura React Native aplikace je rozdělena na dvě hlavní části. První část je vytvořená vývojářem aplikace a je napsána v jazyce JavaScript. JavaScriptový kód komunikuje s částí druhou, která je nativní a umožňuje vývojářům využívat např. systémové notifikace. Pro komunikaci je používán tzv. Bridge. Popsaná architektura je vyobrazena na obrázku níže.




**Obrázek 6: Znázornění interakce frameworku React Native s nativními komponentami. [21]**

### 3.1.2 Zkušenosti vývojářů

Některé společnosti, které využily React Native pro vytvoření vlastní aplikace, se podělily o zkušenosti s tímto frameworkem. Společnost Pinterest popisuje v článku Supporting React Native at Pinterest [22] jejich zájem o framework již od roku 2015. Autor článku popisuje vývoj od samého začátku, jeho rozdělení na dvě fáze, vytvoření malého prototypu a následné provedení rozsáhlých testů výkonu. Společnost se zaměřila na různé kritické metriky např. na rychlost spouštění aplikace. Prototyp, který vývojáři vytvořili naznačil, že sdílený kód má obrovský dopad na rychlost vývoje. První implementace pro systém iOS zabraly vývojářům přibližně 10 dní včetně integrace do stávající firemní infrastruktury. Za pouhé další 2 dny byli vývojáři schopni přenést prototyp na platformu Android. Zdrojový kód uživatelského rozhraní byl ze 100 % sdílený.

O své zkušenosti se podělili i další vývojáři. Fanghao (Robin) Chen se v článku Why Discord is Sticking with React Native [23] zaměřuje na zkušenosti s frameworkem React Native při vývoji aplikace Discord. Autor zmiňuje obrovské výhody sdíleného zdrojového kódu napříč platformami. Stejně jako v článku od společnosti Pinterest [22], i zde je vyzdvihnuta skutečnost, že portování ze systému

iOS na systém Android trvalo pouhé dva dny. Článek popisuje i poměrně zásadní problémy, se kterými se vývojáři během vývoje setkali. Byli nuceni ukončit vývoj pro platformu Android, jelikož narazili na problémy s výkonem dotykových událostí a na nedostatek podpory 64 bit knihoven třetích stran. Zmíněné chybě se věnuje vlákno na webové službě GitHub [24]. Chyba byla vyřešena ve verzi React Native 0.59, která byla vydána v březnu 2019. V poznámkách k nové verzi [25] je zmíněno, že JSC (JavaScriptCore)[1] na platformě Android bylo zastaralé, což způsobovalo nedostupnost moderních funkcí JavaScriptu a horší výkon oproti moderní verzi JSC na platformě iOS.

## 3.2 Flutter

Flutter je open-source framework od společnosti Google, který slouží pro tvorbu mobilních, webových a desktopových aplikací. Byl vydán v roce 2017 a podle Google Trends [19] se mu dostává velké popularity. V roce 2019 byl o Flutter větší zájem než o framework Xamarin.

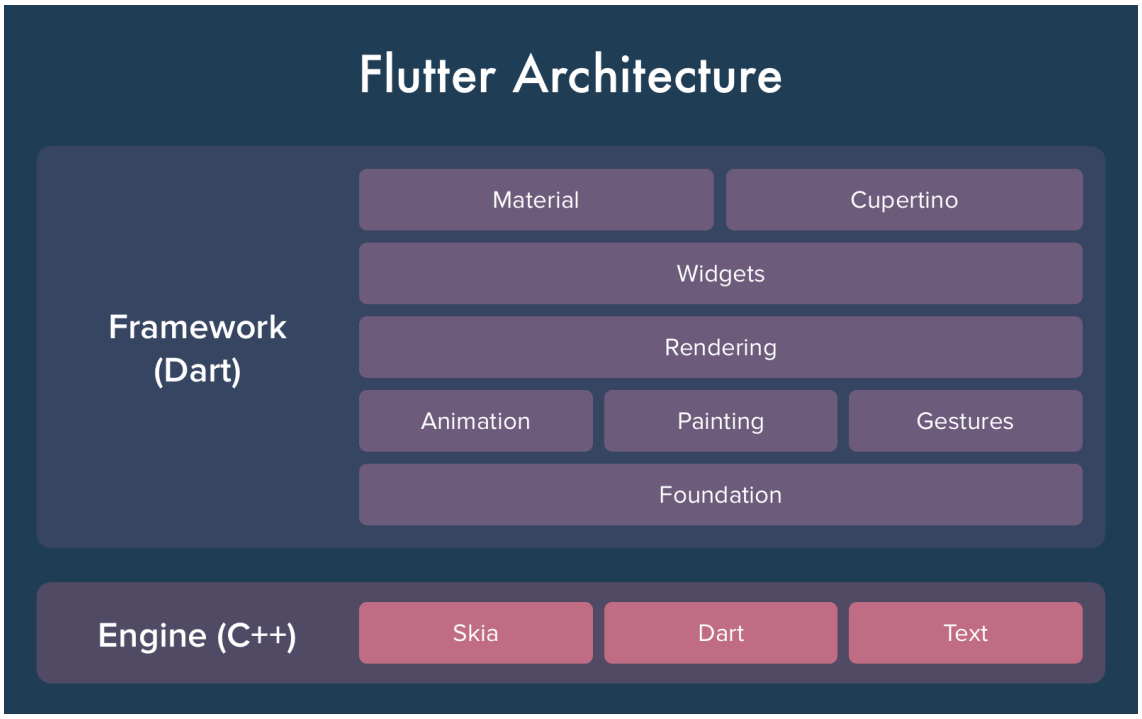
Zmíněnou oblíbenost dokazuje i skutečnost, že tuto technologii využívají velké společnosti jako Alibaba Group, Groupon, Tencent Holdings Limited, eBay Inc. či BMW [26]. V portfoliu mobilních aplikací, které byly vytvořeny pomocí technologie Flutter, je možné nalézt aplikace zmíněných firem, ale i aplikace od samotného Googlu. Příkladem může být aplikace Google Ads nebo mobilní aplikace pro nově vzniklou herní streamovací platformu Stadia [26].

Podle autora článku What is Flutter and Why You Should Learn it in 2020 [27] se Flutter skládá ze souboru nástrojů, které vývojářům pomáhají s vývojem aplikací a starají se o kompilaci zdrojového kódu do kódu nativního. Dále Flutter obsahuje soubor elementů uživatelského rozhraní, které je možné využívat a různě modifikovat. Pomocí těchto elementů vzniká uživatelské rozhraní aplikace. Pro vývoj v technologii Flutter je využíván programovací jazyk Dart.

[1] JavaScriptový engine, který je umístěn v jádře některých webových prohlížečů.

### 3.2.1 Architektura frameworku

Stejně jako ostatní frameworky, které jsou v této práci popisovány, ani Flutter není hybridním frameworkem. Podle již zmiňovaného článku od Kuprenka [21] je Flutter velmi výkonným frameworkem díky schopnosti rychle kompilovat jazyk Dart. Výkonu napomáhá i skutečnost, že framework Flutter je vytvořen v jazycích C a C++. Stejnému tématu se věnuje i článek Flutter, Google's Disruptive Innovation [28], ve kterém autor zmiňuje, že C a C++ kód je v případě platformy Android kompilován pomocí nástroje Android NDK[2]. V případě platformy iOS, je jazyk C a C++ kompilován do nativního kódu pomocí technologie LLVM[3]. Na nejnižší úrovni je k vykreslení komponent uživatelského rozhraní používán vykreslovací engine Skia.




**Obrázek 7: Architektura frameworku Flutter. [21]**

### 3.2.2 Zkušenosti vývojářů

Jednou z aplikací, která byla vytvořena pomocí technologie Flutter, je aplikace Reflectly. Aplikace byla vyvinuta společností Reflectly ApS. Jeden ze zakladatelů této společnosti, Daniel Vestergaard, v článku Reflectly — From React Native to

[2] Sada nástrojů, které umožňují implementovat části nativního kódu aplikace pomocí jazyků C a C++.
[3] LLVM je technologie implementující optimalizační kompilátor.

Flutter [29] popisuje jejich zkušenosti s frameworkem Flutter. Autor nejdříve popisuje vývoj první verze aplikace, která byla vytvořena v létě 2017 pomocí technologie React Native. Stejně jako vývojáři aplikace Discord, jejichž zkušenosti byly popsány v předešlých kapitolách, měli i vývojáři Reflectly problémy s přenesením vytvořené aplikace z platformy iOS na platformu Android. Důvodem byly problémy s výkonem při zpracovávání animací a zvláštní ořezávání elementů při posouvání obrazovky. V průběhu 6 měsíců byli vývojáři nuceni, každou novou verzi upravovat speciálně pro platformu Android. Začátkem roku 2018 padlo rozhodnutí, že bude celá aplikace Reflectly vytvořena znovu a zvolena byla právě technologie Flutter. Jelikož byl v té době framework pouze ve verzi alfa, vývojáři velmi riskovali, což potvrzuje i Daniel Vestergaard. Risk se vyplatil a tvůrci aplikace si nyní pochvalují konzistenci mezi platformami, funkci hot-reload, skvělé nástroje a vysoký výkon.

## 3.3 Xamarin

Xamarin je open-source multiplatformní framework, který umožňuje vyvíjet aplikace pro Android a iOS s využitím platformy .NET a jazyka C#. Xamarin vyvinula stejnojmenná společnost v únoru roku 2013. O 3 roky později, v roce 2016, byla tato firma získána společností Microsoft, která následně uvolnila technologii Xamarin pod licencí open-source. Xamarin se může chlubit velkou popularitou, kterou dokazuje výčet aplikací od známých společnosti [30]. Jako příklad lze uvést aplikace UPS, Microsoft Azure, Alaska Airlines či Aggreko.

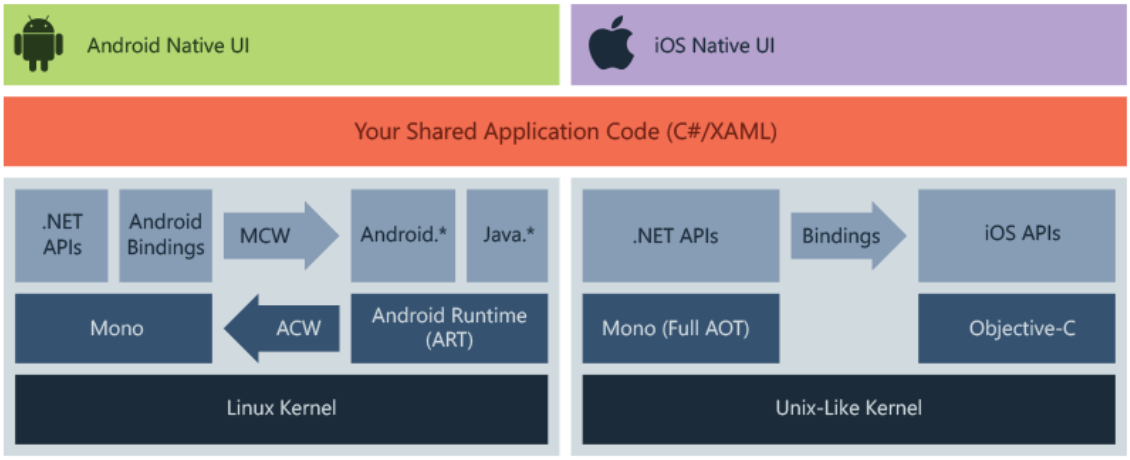
Podle oficiálního článku What is Xamarin [31] framework umožňuje vývojářům sdílet průměrně 80 % zdrojového kódu napříč dostupnými platformami. Tímto je vývojářům umožněno vyvíjet mobilní aplikace pomocí jednoho programovacího jazyka, a přesto dosáhnout výkonu, vzhledu a pocitu jako u běžných nativních aplikací. Podle autorů stejného článku je Xamarin vhodným frameworkem pro vývojáře, kteří cílí na sdílený kód, sdílené možnosti testování a možnost sdílet hlavní funkce aplikace napříč platformami. Dále framework cílí na vývojáře, kteří upřednostňují programovací jazyk C# a nástroj Visual Studio.

### 3.3.1 Architektura frameworku

Na diagramu níže je znázorněna architektura frameworku Xamarin. Podle již několikrát zmíněného článku [31] je Xamarin postaven na open-source verzi frameworku .NET, která se nazývá Mono. Framework Mono existuje téměř stejně dlouho jako samotný framework .NET a je možné ho využívat na většině platformách. Příkladem může být Linux, Unix, FreeBSD či macOS.

Aplikace pro platformu Android jsou z jazyka C# kompilovány do Intermediate Language (IL)[4] [31]. Následně je kód tzv. Just-in-Time (JIT)[5] zkompilován do nativního kódu platformy. Aplikace je spuštěná v prostředí Mono po boku virtuálního stroje Android Runtime (ART)[6]. Xamarin poskytuje přístup do jmenných prostorů platformy Android pomocí tzv. Android Bindings.

Aplikace pro platformu iOS je tzv. Ahead-of-Time (AOT)[7] zkompilována do nativního kódu platformy. Pro komunikaci mezi jazyky C# a Objective-C jsou používány prostředky, které se nazývají Bindings, podobně jako u platformy Android.




**Obrázek 8: Architektura frameworku Xamarin. [31]**

[4] Abstraktní programovací jazyk používaný kompilátorem jako mezikrok při kompilaci do strojového kódu.
[5] Kompilace je prováděna za běhu programu.
[6] Virtuální stroj v systému Android pro aplikace napsané v programovacím jazyce Java.
[7] Kompilace, jejímž výsledkem je spustitelný nativní kód.

### 3.3.2 Zkušenosti vývojářů

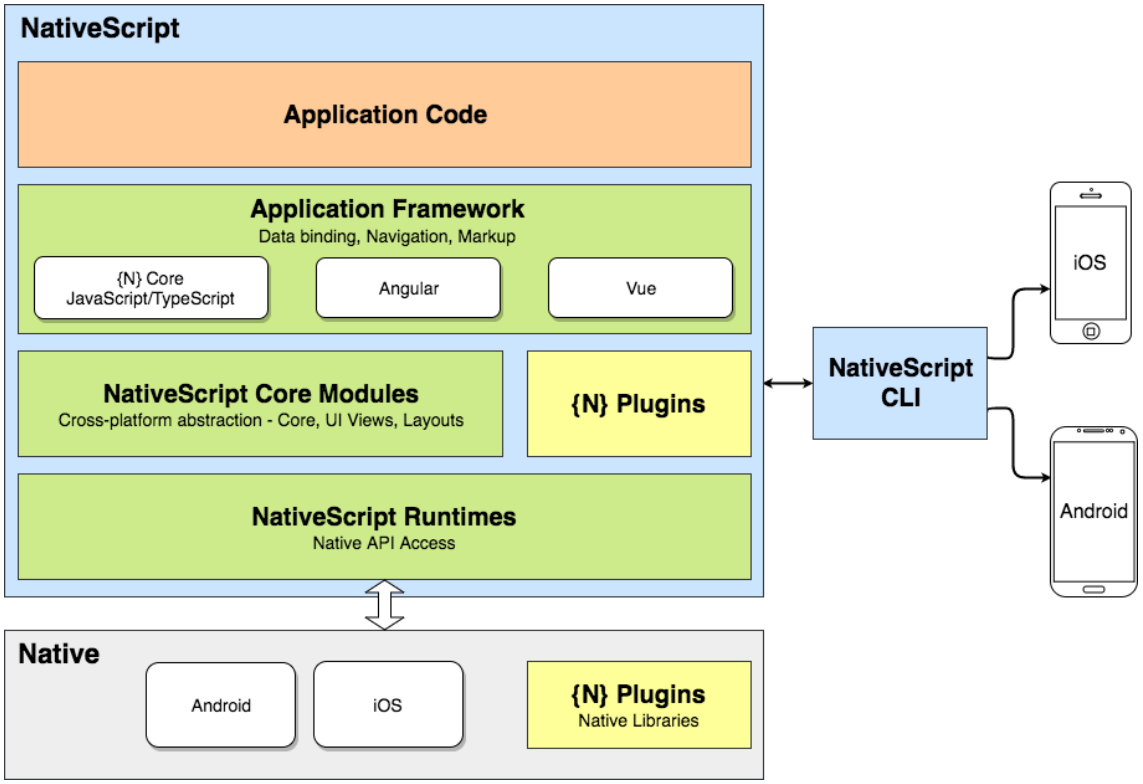
O své zkušenosti s frameworkem Xamarin se pochlubil např. Thomas Smolder, autor článku We worked with Xamarin for almost three years, this is what we learned [32]. Autor popisuje zkušenosti týmu vývojářů ze společnosti In The Pocket. Vývojáře nadchla myšlenka sdíleného kódu a přístup společnosti Microsoft, který začal nově nabytou technologii poskytovat s licencí open-source. V průběhu vývoje tým narazil na nečekaná omezení. Aby bylo možné pomocí nástroje Xamarin.Forms vytvořit uživatelské rozhraní přesně podle představ, byli vývojáři nuceni tvořit vlastní renderery. Xamarin.Forms umožňoval tvořit pouze základní uživatelské rozhraní.

## *3.4 NativeScript*

NativeScript je open-source mobilní multiplatformní framework vytvořený společností Progress Software Corporation a distribuovaný pod licencí Apache 2.0. Jak již název frameworku napovídá, jedná se o technologii, která umožňuje vyvíjet multiplatformní aplikace s cílem dosáhnout jejich nativního vzhledu, výkonu a pocitu z používání. NativeScript nabízí možnost vývoje pomocí technologií Javascript, TypeScript, CSS, Angular a Vue. Vývojáři na oficiální webové stránce [33] prezentují framework jako velmi výkonný, rozšiřitelný a jednoduchý na naučení. Kvalitu a popularitu frameworku podporuje zájem společností jako Samsung, Audi, SAP, Bosch a mnoho dalších [33].

Podle oficiální dokumentace [34] se framework NativeScript skládá z několika hlavních částí. Konkrétně se jedná o Runtimes, Core Modules, CLI a Plugins. Runtime umožňuje volat nativní API daných platforem pomocí jazyka JavaScript. Core Modules poskytují základní abstrakci, která je potřebná pro přístup k nativním platformám. CLI je rozhraní příkazového řádku, které umožňuje vytvářet, sestavovat a spouštět aplikace vytvořené pomocí frameworku NativeScript. Plugins jsou stavební bloky, které zapouzdřují některé funkce a pomáhají vývojářům vytvářet aplikace rychleji. Zmíněnou architekturu je možné pozorovat na následujícím diagramu.

Pomocí frameworku NativeScript byly vytvořeny aplikace jako MyPUMA, Sennheiser Smart Control, California Court Access App či Smart Evaluation.




**Obrázek 9: Architektura frameworku NativeScript. [34]**

## *3.5 Qt*

Qt je multiplatformní framework vytvořený společností Qt Group, který byl podle informací na oficiálních stránkách společnosti [35] představen veřejnosti poprvé již v roce 1995. Autor článku Why You Should Choose Qt For Cross-Platform App Development [36] uvádí, že se jedná o framework vytvořený v jazyce C++, který byl navržen pro vývoj desktopových, mobilních, ale i vestavěných aplikací a systémů.

Pro vývoj hlavních funkcí systému je používán jazyk C++. Pro tvorbu uživatelského rozhraní jazyk QML. Framework poskytuje velké množství vývojářských a designerských nástrojů. Některé z těchto nástrojů umožňují využívat programovací jazyky Python a JavaScript.

Framework Qt je dostupný v komerční, ale i open-source verzi. Společnost Qt Group k placené verzi nabízí podporu obchodních cílů, technickou podporu a blízké obchodní vztahy. Při použití open-source verze se vývojáři zavazují používat framework pouze k vývoji otevřeného softwaru pod licencí GPL.

Mezi aplikace, které byly vytvořeny pomocí frameworku Qt můžeme zařadit např. Night Clock, Atomic Puzzle, Ukemi Ninja či Squaby.

## 3.6 Fuse Open

Další z open-source multiplatformních mobilních frameworků je Fuse Open. Pro tvorbu mobilního nativního uživatelského rozhraní slouží speciální značkovací jazyk UX Markup, který je podle oficiálního webu frameworku [37] jednoduchý na naučení i použití. Pro tvorbu hlavních funkcí aplikací je používán programovací jazyk JavaScript. Fuse Open se skládá z několika komponent, které jsou vytvořeny nezávisle na sobě, aby bylo možné každou z nich používat samostatně. Těmito komponentami jsou Uno, UX, Fuselibs a Fuse Studio.

Uno je programovací jazyk a kompilátor založený na jazyku C#. Generuje C++ kód, který je možné zkompilovat pomocí běžných nativních vývojářských nástrojů, jako je Android Studio či Xcode. Komponenta UX již byla zmíněna. Jedná se o deklarativní jazyk pro tvorbu uživatelského rozhraní, který je založený na XML. Další komponentou je Fuselibs. Jedná se o soubor knihoven, které vývojářům poskytují framework pro tvorbu aplikací. Fuse Studio je grafický editor určený pro práci s frameworkem Fuse Open.

## 3.7 Weex

Weex je dalším frameworkem, který je určený k tvorbě multiplatformních mobilních aplikací, a který umožnuje vývoj pomocí JavaScriptu. Za vývojem stojí The Apache Software Foundation, která technologii Weex poskytuje s licencí Apache Licence 2.0[8].

Struktura frameworku je podle oficiální dokumentace [38] rozdělená, což znamená, že vykreslovací engine je oddělený od vrstvy syntaxe, a tedy že framework není závislý na specifickém programovacím jazyce či frameworku pro tvorbu uživatelské části aplikace. Přesto Weex v současné době podporuje pouze technologie Vue a Rax.

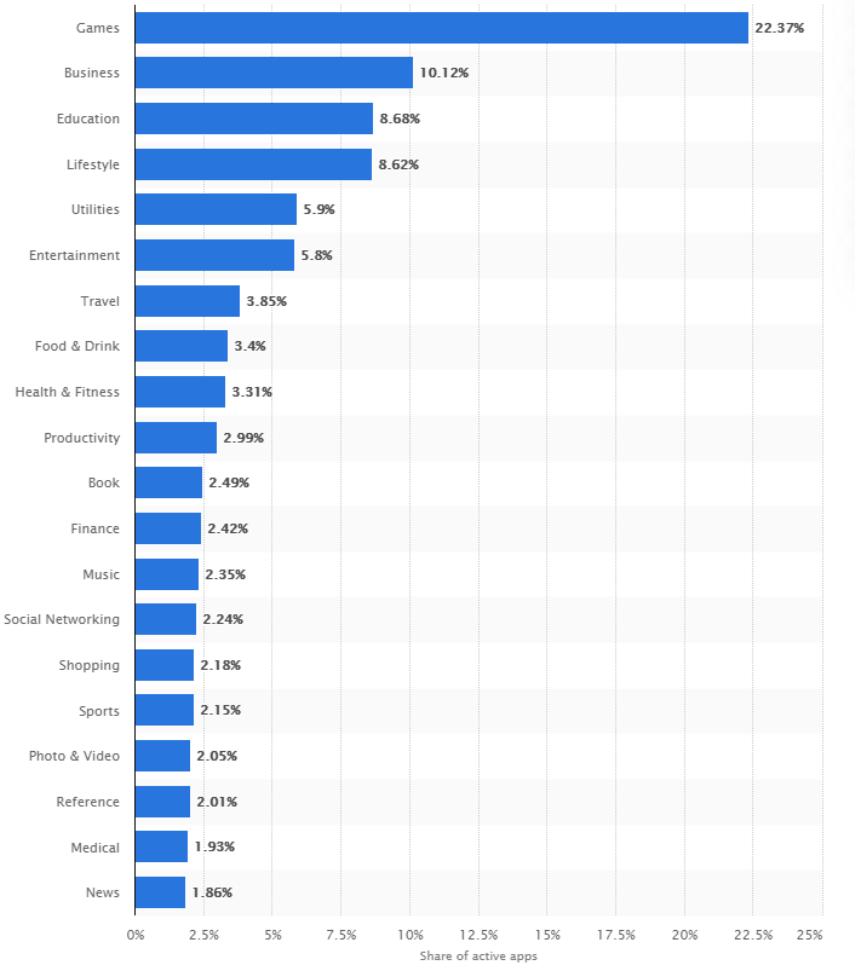
Autorka článku Weex framework - fresh blood in the cross-platform development ring [39] hodnotí framework jako velmi ambiciózní a konkurence

[8] Svobodná softwarová licence, jejímž autorem je Apache Software Foundation. Licence požaduje po uživateli zachování autorství a tzv. disclaimer, tedy zřeknutí se odpovědnosti.

schopný s frameworky jako React Native. Zmiňuje výhody velkého výkonu frameworku, který je zajištěn skutečným využitím nativních komponent daných platforem. Podle autorky článku má frameworky i jisté nevýhody, mezi které patří např. nejistá budoucnost frameworku.

## 3.8 Herní frameworky

Herní frameworky je možné zařadit do speciální kategorie. Jsou vytvořeny za účelem umožnit vývojářům vytvářet graficky náročné aplikace. Příkladem mohou být hry, různé druhy vizualizačních nástrojů či aplikace podporující virtuální a augmentovanou realitu. Tvorba ostatních druhů aplikací může být naopak velmi obtížná, jelikož herní frameworky mají často specifický přístup k tvorbě uživatelského rozhraní. Právě hry jsou nejvíce zastoupeným typem mobilních aplikací v distribučních obchodech. Tuto skutečnost dokazuje následující průzkum.




**Obrázek 10: Srovnání zastoupení aplikací v obchodě App Store podle typu – listopad 2019. [40]**

### 3.8.1 Unity

Unity je multiplatformní engine a vývojářská platforma od společnosti Unity Technologies, zaměřující se na práci s 3D grafikou. Engine je využíván primárně pro vývoj her, ale na oficiální webové prezentaci [41] je prezentován jako užitečný nástroj pro mnoho jiných oborů. Engine Unity je využíván v automobilovém, dopravním a výrobním průmyslu. Dále v oblasti filmů, kde jeho sílu využijí filmaři a animátoři. Unity je velmi vhodným nástrojem i pro architekty a inženýry v průmyslu stavebním. Engine umožňuje vývoj na velmi bohaté množství platforem, mezi které je možné zařadit např. Windows, Linux, macOS, Android, iOS, PS4 či Xbox One. Unity využívá programovací jazyk C#.

Unity se pyšní silnou komunitou a skvělou dokumentací, která byla v nedávné době obohacena o sekci nazvanou Learn [42]. Tato sekce obsahuje velké množství textových materiálů a videí, které novým uživatelům a vývojářům pomohou s prvními kroky vývoje v enginu Unity.

Unity nabízí hned několik verzí, které se liší cenou a funkcemi, které engine poskytuje. Cena začíná na částce 40$ za měsíc. Dostupná je i verze zdarma či verze studentská [41].

Pomocí enginu Unity bylo vytvořeno mnoho mobilních aplikací, mezi které je možné zařadit např. Call of Duty: Mobile, Elder Scrolls: Blades, Assasin's Creed Rebellion či Inside.

### 3.8.2 Unreal Engine

Unreal Engine je dalším multiplatformní engine, který slouží pro práci s 3D grafikou. Je velmi často srovnáván s již právě zmíněným enginem Unity. Unreal Engine byl vytvořen v roce 1998 společností Epic Games, Inc a je oblíbeným nástrojem vývojářů, filmařů, animátorů či architektů z celého světa [43]. Stejně jako jeho konkurent, je i tato technologie rozšířená v různorodých oborech. Příkladem může být architektura, automobilový a dopravní průmysl či filmařství. Využití nalezne i v různých simulačních a tréninkových úlohách. Unreal Engine umožňuje vývoj pro velké množství platforem, mezi které patří Windows, Linux, macOS, Android, iOS, PS4 či Xbox One. Pro vývoj v technologii Unrel Engine je používán jazyk C++ nebo

speciální vizuální programovací jazyk, vyvinutý společností Epic Games, který nese název Blueprints.

Unreal Engine má stejně jako Unity velmi silnou komunitu a kvalitě zpracovanou dokumentaci. Rovněž poskytuje studijní texty a videa pro začátečníky i pokročilé uživatele. Cenová politika této technologie se od konkurence velmi liší. Unreal Engine je poskytován zdarma se všemi funkcemi enginu. Engine je zpoplatněn teprve ve chvíli, kdy vydavatel softwaru dosáhne za daný software určitých příjmů [44].

Pomocí enginu Unreal Engine byly vytvořeny např. mobilní aplikace Fortnite, PUBG MOBILE či Ark Survival.

### 3.8.3 Corona

Corona je open-source multiplatformní framework určený primárně pro tvorbu mobilních, ale i desktopových dvourozměrných her. Byl vytvořen společností Corona Labs Inc a je distribuován kompletně zdarma bez jakýkoliv poplatků. Framework je založený na jazyce Lua. Jedná se o open-source skriptovací jazyk, který se může podle autorů frameworku Corona [45] pochlubit svojí rychlostí, jednoduchostí a výkonem. Jazyk Lua byl využit i během vývoje velmi úspěšných her jako Warcraft, Angry Birds či Civilization [45]. Jak již bylo popsáno, Corona se zaměřuje na širokou škálu zařízení, mezi kterými nalezneme mobilní telefony, tablety, desktopové počítače, ale i chytré televize. Mezi aplikace, které byly pomocí tohoto frameworku vytvořeny, můžeme zařadit např. Tiny Boxes, Ava Airborne, Grow Beets Clicker či Freeze! 2 – Brothers.

## *3.9 Porovnání multiplatformních mobilních frameworků*

V předešlých kapitolách byly postupně prozkoumány a popsány multiplatformní mobilní frameworky. Kapitoly se zabývaly čistě cross-platfrom technologiemi. Hybridní a webové frameworky byly záměrně vynechány. V tabulce níže jsou zmíněné frameworky navzájem porovnány podle různých kritérií. Autor této diplomové práce při vytváření přehledu čerpal z již zmiňovaných oficiálních prezentací frameworků a článků: Flutter vs React Native — Comparing the Features of Each Framework [21], React Native vs. Ionic vs. Flutter: Comparison of Top Cross-

Platform App Development Tools [46], Xamarin vs React Native vs Ionic vs NativeScript: Cross-platform Mobile Frameworks Comparison [47], React Native vs Xamarin vs Ionic [47] a Flutter Versus Other Mobile Development Frameworks: A UI And Performance Experiment [48, s. 2].

| | **Podporované platformy** | **Programovací jazyky a další technologie** | **Společnost** | **Licence/Cena** |
|---|---|---|---|---|
| **React Native** | Android, iOS. | JavaScript, React. | Facebook. | Open-source. |
| **Flutter** | Android, iOS, macOS, web. | Dart. | Google. | Open-source. |
| **Xamarin** | Android, iOS, macOS, Windows. | C#. | Microsoft. | Open-source. |
| **NativeScript** | Android, iOS. | Angular, Vue, JavaScript, TypeScript, CSS. | Progress Software Corporation. | Apache License 2.0 |
| **Qt** | Android, iOS, macOS, Windows, Linux, tvOS, LG webOS a další… | C++, QML, Python, JavaScript. | Qt Group. | Open-source. Možnost zaplatit podporu a další služby. |
| **Fuse Open** | Android, iOS. | JavaScript, UX Markup. | Fuse Open. | Open-source. |
| **Weex** | Android, iOS, web. | JavaScript, Vue, Rax, CSS. | The Apache Software Foundation. | Apache License 2.0. |
| **Unity** | Android, iOS, macOS, Windows, Linux, PS4, Xbox One, tvOS a další… | C# | Unity Technologies. | Několik licencí. Cena začíná na 40$/měsíc.<br>Osobní licence zdarma. |
| **Unreal Engine** | Android, iOS, Windows, Linux, PS4, Xbox One, SteamOS a další… | C++, Blueprints. | Epic Games, Inc. | Zdarma pro nekomerční projekty. Komerční projekty – zdarma do určitých příjmů. |
| **Corona** | Android, iOS, macOS, Windows, Apple TV a další… | Lua. | Corona Labs Inc. | Open-source. |

**Tabulka 3: Porovnání multiplatformních mobilních frameworků. [Zdroj: Autor práce]**

# 4 Návrh testovacích scénářů

V následující kapitole budou navrhnuty testovací scénáře, jejichž cílem je porovnat vybrané frameworky podle několika kritérií. Scénáře budou zaměřeny na rychlost načítání, výkon, využití zdrojů, ale i na možnosti a rychlost vývoje pomocí daných frameworků.

V závislosti na průzkumu a porovnání frameworků v předchozích kapitolách se autor práce rozhodl, v této praktické části, porovnat frameworky React Native v0.62.2, Flutter v1.12.13 a Xamarin v4.5. Důvodem je, že vybrané frameworky patří mezi nejoblíbenější technologie v oblasti vývoje multiplatformních mobilních aplikací [19]. Dalším důvodem jsou i osobní preference autora. Autor s vybranými technologiemi nemá žádné praktické zkušenosti, což napomůže výsledkům při porovnávání frameworků z hlediska pohodlí a rychlosti práce s nimi.

Pro testování vytvořených scénářů bude použito virtuální zařízení Google Pixel 3a s operačním systémem Android ve verzi 10. V určitých scénářích bude aplikace otestována i na zařízeních reálných. Konkrétně se bude jednat o Honor 10 lite s operačním systémem Android 9.

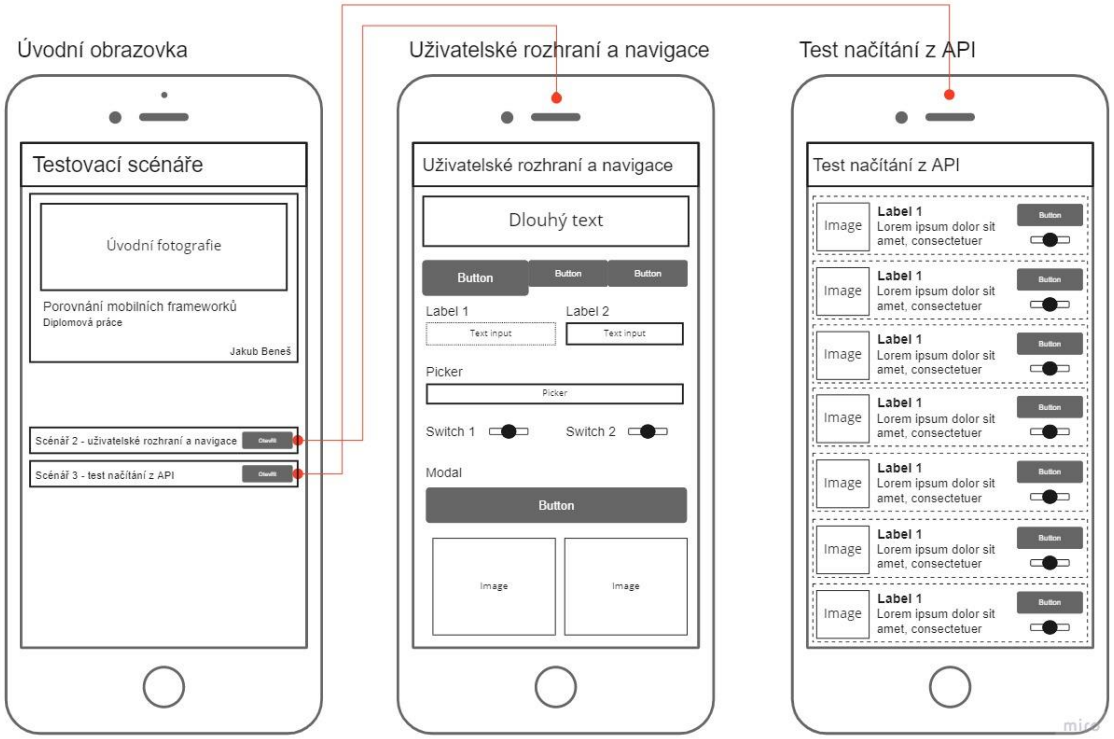
## *4.1 Testovací scénář 1 – instalace, vytvoření projektu a ladění aplikace*

Cílem 1. testovacího scénáře je prověřit framework při jeho prvním použití. Bude provedena instalace frameworku a všech potřebných závislostí. Dále bude pomocí frameworku vytvořena šablona projektu a otestováno první spuštění aplikace. Důraz bude kladen i na možnosti a složitost spuštění aplikace na virtuálním zařízení, ale vyzkoušeno bude i spuštění na zařízení reálném. Autor práce se zaměří na ladící nástroje či schopnost frameworku reagovat na změny v programu bez nutnosti opětovné instalace aplikace.

## 4.2 Testovací scénář 2 – tvorba uživatelského rozhraní a navigace

Další scénář bude zaměřen na tvorbu uživatelského rozhraní. Autor navrhl několik obrazovek, které obsahují různé ovládací prvky, obrázky a další komponenty. Uživatel aplikace bude mít možnost se mezi obrazovkami navigovat pomocí odkazů a tlačítek. Cílem scénáře je otestovat možnosti tvorby uživatelského rozhraní, dostupnost různých ovládacích prvků a složitost implementace navigace v aplikaci.

Na obrázku níže je možné pozorovat autorem navrhnuté obrazovky. Znázorněna je i navigace mezi nimi. Pro návrh aplikace byl použit nástroj Miro [49]. Jedná se o platformu, která umožňuje efektivní spolupráci v týmu pomocí různých diagramů, myšlenkových map či běžných poznámek.




**Obrázek 11: Návrh aplikace pomocí nástroje Miro. [Zdroj: Autor práce]**

## 4.3 Testovací scénář 3 – načítání a zobrazení seznamu objektů z REST API

Třetí testovací scénář je zaměřen na výkon frameworku. Konkrétně se autor zaměří na schopnost frameworku zpracovávat velké množství dat, která budou načítána z externího zdroje. Takto načtená data budou strukturována do autorem vytvořeného uživatelského rozhraní, které bude obsahovat textová pole, tlačítka či obrázky. Pomocí externích nástrojů autor provede testy, které změří rychlost načítání či využití zdrojů jako je operační paměť a procesor zařízení.

Jak již bylo zmíněno, data budou načítána z externího zdroje. Za tímto účelem, autor práce vytvoří REST API, které bude testovaným frameworkům poskytovat velké množství předem připravených dat.

## 4.4 Použité technologie a nástroje

Mimo samotných testovaných frameworků a jazyků, které dané frameworky používají, budou v praktické části této práce použity další technologie a nástroje, které pomohou v implementaci a následném testování navrhnutých testovacích scénářů. V této kapitole budou krátce představeny a bude vysvětlen důvod jejich použití.

### 4.4.1 NPM

Npm neboli Node package manager je balíčkovací systém od společnosti npm, Inc. a je dostupný zdarma [50]. V této diplomové práci bude hojně využíván při instalaci nových knihoven, zakládání projektů či spouštění příkazů různých druhů.

### 4.4.2 Node.js

JavaScriptové prostředí Node.js, které umožňuje spouštět JavaScriptový kód mimo prohlížeč, bylo vyvinuto v roce 2009 [51]. Node.js klade důraz na vysokou škálovatelnost aplikací a možnost manipulovat se zdroji operačního systému. Autor práce zvolil tuto technologii, jelikož umožňuje vývojářům velmi rychle a jednoduše vytvořit testovací REST API.

### 4.4.3 Express

Express je minimalistický a velmi flexibilní Node.js framework pro tvorbu webových aplikací. Poskytuje širokou škálu knihoven pro práci s cookies, relacemi, URL parametry, uživatelskými účtu a mnoho dalších. Autor framework využil z důvodu jeho jednoduchosti a schopnosti umožnit vytvoření REST API ve velmi krátkém čase.

### 4.4.4 Android Emulator

Pomocí technologie Android Emulator je možné vytvořit virtuální zařízení s operačním systémem Android v prostředí desktopových operačních systémů. Na takto vytvořeném zařízení je možné testovat vyvíjené aplikace a simulovat různé stavy zařízení např. měnit parametry senzorů, sílu signálu, aktuální GPS souřadnice apod. Jednou z velkých výhod virtuálních zařízení je možnost testovat aplikace na široké škále typů telefonů a verzí systému Android.

### 4.4.5 JavaScript

JavaScript je programovací jazyk, který byl původně vytvořen jako jazyk určený pouze pro spouštění na straně klienta, typicky přímo v prohlížeči. Podle webu javascript.info [52] byl vytvořen s cílem oživit statické webové stránky. Programy napsané v tomto jazyce se nazývají skripty a pro jejich spuštění není nutné provádět žádnou speciální kompilaci.

JavaScript se měl původně nazývat LiveScript, ale jelikož vznikl v době, kdy byl velmi oblíbený programovací jazyk Java, bylo rozhodnuto, že název JavaScript tomuto jazyku pomůže s jeho rozšířením mezi vývojáři [52].

V dnešní době je možné JavaScript spouštět i mimo webový prohlížeč. Typicky je používán na serveru, ale i dalších zařízeních. K jeho spouštění je nutné používat speciální program známý jako JavaScript engine [52].

```
<!DOCTYPE HTML>
<html>
<body>
  <p>Your name:</p>
  <input type="text" id="name" name="name" title="Your name" />
  <button type="button" id="btnGreet" onclick="greet()">Greet!</button>
  <script>
      function greet(){
         var enteredName = document.getElementById('name').value;
         if(enteredName !== ''){
           alert(`Hello ${enteredName}`);
         }else{
           alert(`Enter name`);
         }
      }
  </script>
</body>
</html>
```

**Ukázka kódu 1: Použití jazyka JavaScript. [Zdroj: Autor práce]**

V ukázce zdrojového kódu je jednoduchá HTML5 stránka, která obsahuje krátký text, formulářový prvek pro zadání textu a tlačítko, kterým je zadání potvrzeno. Po stisknutí tlačítka je zavolána JavaScriptová funkce `greet()`, která převezme text, který byl zadán do formulářového prvku a zobrazí jej uživateli ve vyskakovacím okně. V případě, že nebyl žádný text zadán, uživatel je o tom informován. Ukázkový kód demonstruje možnost využití JavaScriptu při tvorbě webových stránek a jeho kombinaci s HTML5 tagy.

Syntaxe jazyka JavaScript nevyhovuje všem vývojářům a z tohoto důvodu vznikla celá řada nových jazyků, které jsou před samotným spuštěním přeloženy do čistého JavaScriptu. Tyto jazyky vývojářům ulehčují a zpřehledňují práci, dávají jim možnost psát méně kódu, a přesto zachovat funkčnost. Mezi tyto jazyky patří např. Dart, TypeScript, CoffeeScript či Flow. V žebříčku popularity se programovací jazyk JavaScript nachází na 7. místě [53].

### 4.4.6 Dart

Programovací jazyk Dart vytvořila společnost Google. První verze byla odhalena v roce 2013 s cílem nahradit v prohlížečích jazyk JavaScript [54]. Syntaxe jazyka je velmi podobná jazykům Java, JavaScript či C++. David Bolton ve svém článku [54]

uvádí, že snaha nahradit JavaScript byla způsobená frustrací vývojářů společnosti Google. Vývojáři byli nuceni udržovat masivní JavaScriptový kód pro aplikace Gmail a GoogleMaps. I přes velký marketingový nátlak, se Dart nikdy nestal nástupcem JavaScriptu.

Nejednalo se ale o konec jazyka Dart, popularita tohoto jazyka vzrostla z důvodu vzniku technologie Flutter. Autor článku Why Flutter uses Dart [55] zmiňuje důvody, proč Google pro svůj nový framework pro tvorbu multiplatformních mobilních aplikací zvolili právě tento jazyk. Dart funguje na všech platformách a umožňuje rychlý vývoj. Díky Dartu je možné poskytnou vývojářům funkci hot-reload, která zajistí, že se jakákoliv změna ve zdrojovém kódu, do méně než 1 vteřiny, projeví v běžící testovací aplikaci bez nutnosti aplikaci restartovat. Jako další důvod, proč Google zvolil jazyk Dart, je výkon. Dart umožňuje kompilaci do ARM[9] a 64bitového strojového kódu, což má za následek rychlý start aplikace a hladké provádění animací. Programovací jazyk Dart se nachází na 24. místě v indexu TIOBE [53].

```
// A function declaration.
int timesTwo(int x) {
  return x * 2;
}

// Arrow syntax is shorthand for `{ return expr; }`.
int timesFour(int x) => timesTwo(timesTwo(x));

main() {
  print("4 times two is ${timesTwo(4)}");
  print("4 times four is ${timesFour(4)}");
}
```

**Ukázka kódu 2: Ukázka deklarace a volání funkcí v jazyce Dart. [56]**

V ukázce zdrojového kódu je možné vidět dva způsoby deklarace funkce a jejich následné použití ve funkci `main()`, která slouží jako hlavní funkce aplikace. První deklarovaná funkce vynásobí přijaté celé číslo číslem 2 a vrací výsledek rovněž jako celé číslo. Druhá deklarace demonstruje zkrácenou možnost zápisu funkce.

[9] Architektura procesorů s nízkou spotřebou elektrické energie.

### 4.4.7 Mono

Mono je open-source vývojářská platforma založená na platformě .NET, která byla původně vytvořena společností Novell. V současné době je vyvíjena pod organizací .NET Foundation [57]. Mono umožňuje vývojářům vytvořené aplikace spouštět na různých operačních systémech, mezi které patří např. Android, iOS, macOS, Windows a další systémy založené na linuxovém jádře. Prostředí platformy Mono dává vývojářům možnost využívat velké množství programovacích jazyků, mezi které patří například C#, Visual Basic, F#, Java či Python.

C# je objektově orientovaný programovací jazyk, který je podle indexu TIOBE [53] 5. nejoblíbenější programovací jazyk na světě. Byl vytvořen společností Microsoft a jeho první verze byla vydána v roce 2002 [58].

Visual Basic vznikl již v roce 1991 a byl vyvinut společností Microsoft. Jedná se o událostmi řízený programovací jazyk. Podle TIOBE [53] je tento jazyk stále velmi oblíbený. V žebříčku popularity se nachází na 6. místě.

Jazyk F# je stejně jako předchozí jazyky vytvořen společností Microsoft v roce 2005. Jedná se o multiparadigmatický programovací jazyk, který spojuje funkcionální a objektově orientovaný přístup. Oproti dvěma předchozím jazykům je F# mnohem méně populární. Podle indexu TIOBE [53] se nachází až na 28. místě v žebříčku oblíbenosti.

# 5 Implementace testovacích scénářů ve vybraných frameworcích

V následujících kapitolách budou postupně popsány postupy při implementaci navrhnutých scénářů. Snahou autora bylo dosáhnout co nejpodrobnějšího popisu a během samotné implementace se držet návrhu v co největší míře. Kapitoly obsahují zajímavé ukázky zdrojových kódu a různá porovnání uživatelského rozhraní mezi vybranými frameworky. Autor kladl důraz na udržení stejných podmínek pro všechny vybrané frameworky, aby bylo následné porovnání co nejpřesnější.

## 5.1 Vytvoření testovacího API

Ještě před začátkem práce s testovacími scénáři v daných frameworcích, autor vytvořil podle návrhu testovací API. Toto rozhraní bude použito pro všechny vybrané frameworky, aby bylo docíleno stejných podmínek pro testování.

V prvním kroku autor nainstaloval JavaScriptové prostředí Node.js. Instalace byla provedena pomocí instalačního balíčku, který je dostupný na oficiálních stránkách vývojářů [59]. Pomocí příkazu `npm init` byl vytvořen soubor `package.json`[10], který slouží jako konfigurační soubor projektu. Dále autor nainstaloval minimalistický webový framework Express. Instalace byla provedena pomocí příkazu `npm install express –save`. Dále byl autorem práce vytvořen soubor `generateData.js`, který slouží k vygenerování testovacích dat. Zdrojový kód soubor je k nahlédnutí v ukázce níže.

```
const LoremIpsum = require("lorem-ipsum").LoremIpsum;
const fs = require("fs");

const lorem = new LoremIpsum();
// Počet objektů, které budou vytvořeny
const numberOfObjects = 15000;

let objects = [];
// Vytvoření objektů
for (var i = 1; i < numberOfObjects; i++) {
```

[10] Soubor obsahující metadata o projektu např. název projektu, jméno autora nebo informace o závislostech.

```
  objects = [
    ...objects,
    {
      id: i,
      name: lorem.generateWords(1),
      description: lorem.generateSentences(1),
      image: `http://192.168.214.49:5000/static/${Math.floor(
        Math.random() * 10
      ) + 1}.jpg`
    }
  ];
}
// Uložení dat do souboru data.json
fs.writeFile("data.json", JSON.stringify(objects), err => {
  if (err) throw err;
  console.log("The file has been saved!");
});
```

**Ukázka kódu 3: Vytvoření testovacích dat a uložení do souboru. [Zdroj: Autor práce]**

Nejdříve je provedeno naimportování knihoven, které jsou při generování testovacích dat použity. Knihovna `lorem-ipsum` slouží k vytvoření zástupného textu, může se jednat o odstavce, věty a slova. Knihovna `fs` poskytuje API pro práci se systémem souborů. Následuje vytvoření konstanty `numberOfObjects`, která určuje, kolik testovacích objektů bude v dalším kroku vytvořeno. Dále jsou v cyklu vygenerovány objekty. Každému objektu je přiřazen jednoznačný identifikátor. Název a popis je vytvořen pomocí knihovny `lorem-ipsum`. Každý objekt obsahuje informaci o náhledovém obrázku. Do struktury projektu byly autorem přidány obrázky 1-10, kde každý obrázek vytvořil sám autor. Při generování objektů, je objektu náhodně přiřazen jeden z vytvořených obrázků. V posledním kroku je provedeno uložení vygenerovaných objektů do souboru `data.json`. Spuštěním příkazu `node generateData`, bylo provedeno spuštění právě popsaného kódu a vytvoření souboru `data.json`. Pro testování bylo vygenerováno 15000 záznamů.

```
[
  {
    "id": 1,
    "name": "ex",
    "description": "Ut laborum consectetur aliqua minim veniam elit ullam
co deserunt cupidatat.",
    "image": "http://192.168.214.49:5000/static/3.jpg"
  },
  {
    "id": 2,
    "name": "velit",
    "description": "Labore voluptate occaecat tempor sunt eiusmod enim pa
riatur.",
    "image": "http://192.168.214.49:5000/static/7.jpg"
  }
]
```

**Ukázka kódu 4: Obsah vygenerovaného souboru data.json. [Zdroj: Autor práce]**

Autor práce dále vytvořil soubor index.js, který slouží jako hlavní soubor projektu. Stará se o samotné spuštění služby a o obsluhu požadavků. Opět byla použita knihovna `fs`, která se v tomto případě postará o načtení obsahu předem vygenerovaného souboru `data.json`. Důvodem, proč jsou data vygenerována předem a pouze jednou, je skutečnost, že generování dat při každém obsloužení požadavku by bylo časově náročné a mohlo by ovlivnit testování vybraných frameworků.

```
const express = require('express')
const fs = require("fs");

const app = express()
const port = 5000

// Obsluha požadavků
app.get('/', function (req, res) {
    // Načteme obsah souboru data.json
    fs.readFile('./data.json', (err, json) => {
        let obj = JSON.parse(json);
        // Obsah souboru pošleme jako odpověď
        res.json(obj);
    });
})

// Povolení statických souborů např. obrázků
app.use(express.static('files'))
```

```
// Učení adresáře, ze kterého se mají statické soubory načítat
app.use('/static', express.static('public'))

// Spuštění služby a naslouchání na zadaném portu
app.listen(port, () => console.log(`API listening on port ${port}!`))
```

**Ukázka kódu 5: Zdrojový kód testovacího API. [Zdroj: Autor práce]**

Tímto bylo zprovozněno testovací API, které je dostupné na adrese http://localhost:5000 nebo http://192.168.214.49:5000, kde `192.138.214.49` je adresa počítače, na kterém je REST API spuštěno .

## 5.2 React Native

Jak již bylo popsáno v teoretické části této diplomové práce, React Native je postavený na knihovně React. Zatímco React je využíván ve webovém prostředí a pracuje s HTML5 tagy, React Native používá namísto těchto tagů nativní komponenty. Pod těmito komponentami si lze představit např. `Text` (text), `TextInput` (formulářový prvek pro zadání textu), `Button` (tlačítko) nebo `Image` (obrázek). Tyto komponenty jsou při vykreslení zobrazeny tak, aby pro konkrétní platformu vypadaly a chovaly se nativně. Tímto je zajištěna nutnost udržovat pouze jeden zdrojový kód pro více platforem. Na oficiálních stránkách frameworku se nachází podrobná dokumentace [60], která novým zájemcům o tuto technologii dokáže pomoci v začátcích a vysvětlit jednoduché i složitější postupy při implementaci běžných funkcí mobilních aplikací.

```
import React, { Component } from 'react';
import { Text, View } from 'react-native';

export default class HelloWorldApp extends Component {
  render() {
    return (
      <View style={{ flex: 1, justifyContent: "center",
alignItems: "center" }}>
        <Text>Hello, world!</Text>
      </View>
    );
  }
}
```

**Ukázka kódu 6: Vytvoření jednoduché komponenty – React Native. [60]**

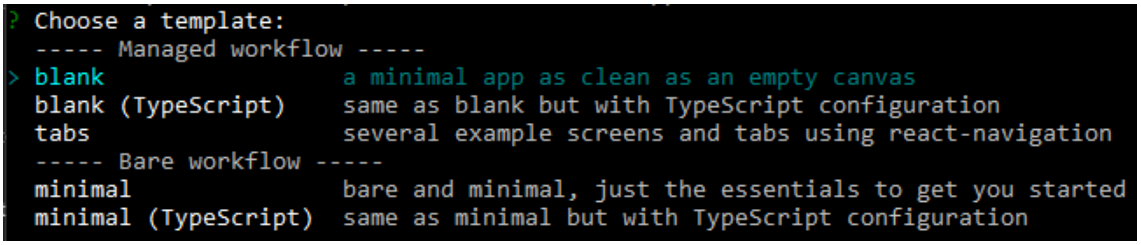
```
import React from "react";
import ReactDOM from "react-dom";

class HelloWorld extends React.Component {
    render() {
        return(
            <div className="greetings" style={{color: 'green'}}>
               Hello, world!
            </div>
        )
    }
}

ReactDOM.render(
    <HelloWorld />,
    document.getElementById("app")
);
```

**Ukázka kódu 7: Vytvoření jednoduché komponenty – React. [Zdroj: Autor práce]**

V ukázkách zdrojových kódů je znázorněn rozdíl mezi technologiemi React a React Native. Ve webovém prostředí je text vkládán do HTML5 tagů, v tomto případě do párového tagu `div`. Elementu je přiřazena třída a CSS3 styl, který se postará o změnu barvy textu v tagu na zelenou. React Native používá namísto HTML5 tagů nativní komponenty. V této ukázce jsou použity komponenty `View` a `Text`. Pomocí `View` je obalena obrazovka aplikace a pomocí atributu `style`, je text na obrazovce zarovnán na střed. Komponenta `Text` vykreslí textový řetězec.

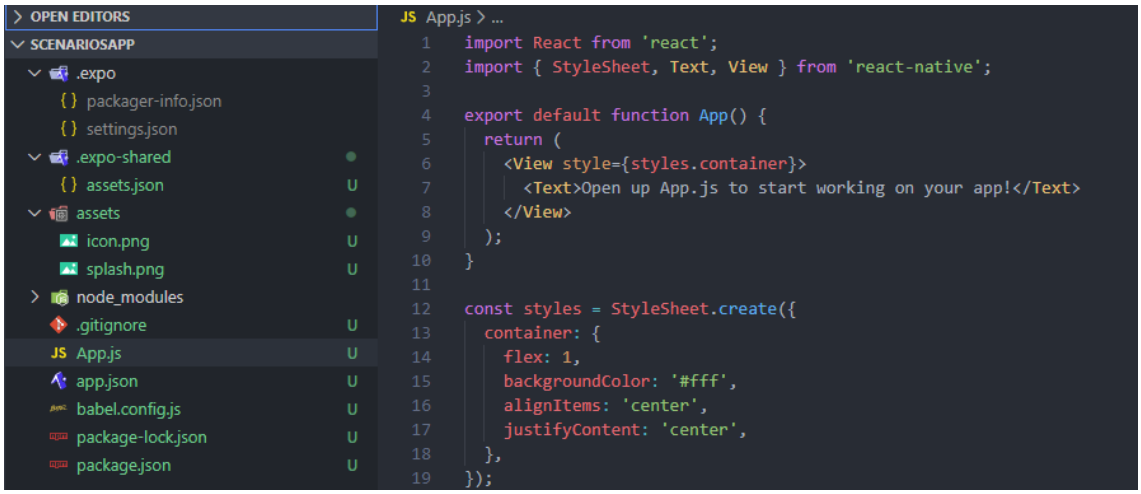
### 5.2.1 Testovací scénář 1 – instalace, vytvoření projektu a ladění aplikace

Instalace frameworku je velmi jednoduchá a zabere pouze pár minut. Nejjednodušším způsobem je použití nástroje Expo CLI. Jedná se o sadu nástrojů, která poskytuje mnoho funkcí a jednou z nich je právě možnost vytvořit projekt aplikace React Native. Pro instalaci nástroje Expo CLI byl použit balíčkovací nástroj npm: `npm install -g expo-cli`. Spuštěním následujícího příkazu byl vytvořen projekt aplikace React Native: `expo init ScenariosApp`. Při vytváření projektu má uživatel možnost zvolit typ šablony. Pro tuto diplomovou práci byl zvolen typ šablony `blank`.




**Obrázek 12: Možnost zvolení šablony projektu ve frameworku React Native. [Zdroj: Autor práce]**

Autor práce pomocí nástroje Expo CLI vytvořil projekt, jehož strukturu souborů je možné pozorovat na obrázku níže. Projekt obsahuje konfigurační soubory jako `package.json` či `babel.config.js`[11]. Za zmínění stojí i soubor `App.js`, ve kterém je definována hlavní komponenta aplikace nazvaná `App`.




**Obrázek 13: Struktura projektu React Native a ukázka hlavní komponenty App. [Zdroj: Autor práce]**

Pomocí příkazu `npm start` byla aplikace prvně spuštěna. Před samotným spuštěním jsou vývojářům nabídnuta další nastavení, která se týkají zvolení prostředí pro spuštění či možnost přihlášení pomocí účtu Expo. Autor práce otestoval aplikaci na virtuálním zařízení Pixel 3a a reálném zařízení iPhone 8.

Při změně zdrojového kódu spuštěná aplikace okamžitě reagovala a nebylo nutné ji žádným způsobem restartovat či znovu instalovat. Změny se správně projevovaly na virtuálním i reálném zařízení.

[11] Konfigurační soubor JavaScriptového kompilátoru Babel.

O ladících nástrojích se zmiňuje oficiální dokumentace frameworku [61]. Framework poskytuje poměrně bohaté možnosti ladění aplikací. Autor práce otestoval možnost ladění aplikace přímo v prohlížeči, kde mu bylo umožněno využívat rozšíření prohlížeče nazvané `React DevTools`[12].

### 5.2.2 Testovací scénář 2 - tvorba uživatelského rozhraní a navigace

Autor práce v prvním kroku implementoval navigaci aplikace. K dosáhnutí tohoto cíle byla použita knihovna `react-navigation`. Autor při vytváření projetu zvolil typ šablony `blank`. Z tohoto důvodu bylo nutné knihovnu doinstalovat, což bylo provedeno spuštěním příkazu `npm install @react-navigation/native`. Následovalo upravení hlavní komponenty aplikace. Byly naimportovány potřebné knihovny a vytvořen navigační kontejner `NavigationContainer`, což je komponenta, která má informaci o všech dostupných obrazovkách a o aktuálním stavu navigace. Funkce `createStackNavigator()` vrací objekt, který má dvě vlastnosti, konkrétně vlastnost `Screen` a `Navigator`. V obou případech se jedná o komponenty, které se používají pro konfiguraci navigace. Vše zmíněné je možné vidět v ukázce kódu níže. Autor práce v tomto kroku vytvořil dvě obrazovky. `HomeScreen` slouží jako úvodní obrazovka, která bude fungovat jako rozcestník pro další testovací scénáře. Obrazovka `Scenario2Screen` bude použita pro otestování možností tvorby uživatelského rozhraní.

```
import 'react-native-gesture-handler';
import React from 'react';
// Import navigačního kontejneru
import { NavigationContainer } from '@react-navigation/native';
// Import navigační funkce
import { createStackNavigator } from '@react-navigation/stack';
// Import autorem vytvořených obrazovek
import HomeScreen from './screens/HomeScreen';
import Scenario2Screen from './screens/Scenario2Screen';

export default function App() {
  // Zavolání funkce a získání potřebných komponent
  const Stack = createStackNavigator();
```

[12] Rozšíření pro prohlížeče Chrome a Firefox, které vývojářům umožňuje ladit aplikace postavené na knihovně React.

```
  return (
    <NavigationContainer>
      <Stack.Navigator
        initialRouteName="Home"
        screenOptions={{
          headerTintColor: 'white',
          headerStyle: { backgroundColor: '#03adfc' },
        }}
      >
        {/* Úvodní obrazovka */}
        <Stack.Screen
          name="Home"
          component={HomeScreen}
          options={{
            title: 'Testovací scénáře',
          }} />
        {/* Obrazovka 2. scénáře */}
        <Stack.Screen
          name="Scenario2"
          component={Scenario2Screen}
          options={{
            title: 'Uživatelské rozhraní a navigace',
          }} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

**Ukázka kódu 8: Vytvoření navigace pomocí knihovny react-navigation. [Zdroj: Autor práce]**

Důležité je i zmínit, jakým způsobem je proveden samotný přechod na jinou obrazovku. Použita je komponenta `Button`, u které je možné nastavit atribut `onPress`. Funkce, která je nastavena v tomto atributu se provede po stisknutí tlačítka.

```
<Button
    title="Otevřít"
    onPress={() => this.props.navigation.navigate("Scenario2")}
/>
```

**Ukázka kódu 9: Přechod na jinou obrazovku pomocí tlačítka. [Zdroj: Autor práce]**

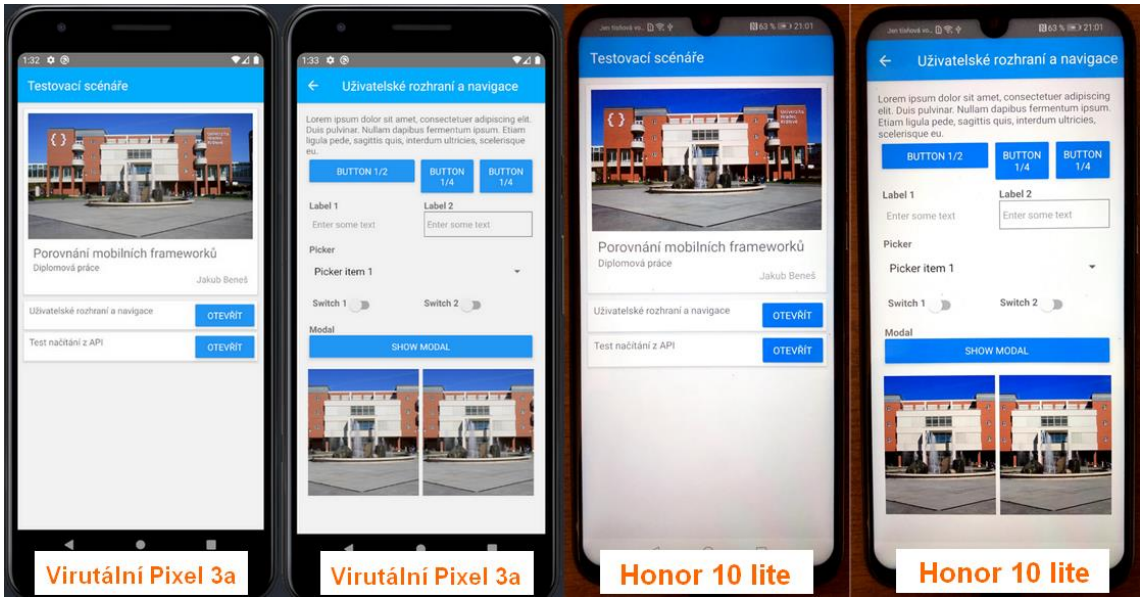
V dalším kroku byl autorem vytvořen obsah obrazovky `Scenario2Screen`. Podle návrhu měla obrazovka obsahovat různé prvky např. text, tlačítka, textové vstupy atd. Framework autorovi článku umožnil vytvořit obrazovku přesně podle návrhu, bez nutnosti používat externí knihovny. Níže se nachází vybrané ukázky zdrojových kódů.

```
{/* Zobrazení tlačítek, tvorba layoutu pomocí flexDirection a flex  */}
<View style={{ flexDirection: 'row' }}>
    <View style={{ flex: 2, padding: 5 }}>
        <Button title="Button 1/2" />
    </View>
    <View style={{ flex: 1, padding: 5 }}>
        <Button title="Button 1/4" />
    </View>
    <View style={{ flex: 1, padding: 5 }}>
        <Button title="Button 1/4" />
    </View>
</View>
{/* Label a textový vstup  */}
<View style={{ flex: 2, padding: 5 }}>
    <Text style={{ fontWeight: 'bold' }}>Label 1</Text>
    <TextInput
        style={{ height: 40, padding: 5 }}
        placeholder="Enter some text"
    />
</View>
```

**Ukázka kódu 10: Použití předdefinovaných komponent – React Native. [Zdroj: Autor práce]**

Autor práce se věnoval i tvorbě uživatelského rozhraní na úvodní obrazovce. Snahou bylo otestovat, jestli framework poskytuje dostatek možností při tvorbě složitějšího uživatelského rozhraní. Je třeba zdůraznit, že autor má základní zkušenosti s technologií React a jisté principy si tak mohl přenést i do vývoje pomocí frameworku React Native. Navrhnuté uživatelské rozhraní bylo vytvořeno pomocí autorem definovaných komponent `Card`, `ScenarioList` a `ScenarioListItem`.

Na obrázcích níže je možné pozorovat výsledné uživatelské rozhraní úvodní obrazovky a obrazovky druhého scénáře. Rozhraní bylo otestováno na virtuálním zařízení Pixel 3a a reálném zařízení Honor 10 lite.




**Obrázek 14: Uživatelské rozhraní úvodní obrazovky a obrazovky druhého scénáře – React Native. [Zdroj: Autor práce]**

### 5.2.3 Testovací scénář 3 - načítání a zobrazení seznamu objektů z REST API

Pro práci se seznamem objektů poskytuje React Native dvě komponenty nazvané `FlatList` a `SectionList`. Komponenta `FlatList` zobrazí rolovací seznam podobně strukturovaných dat, která se mohou měnit v čase. Komponenta vykresluje pouze prvky, které jsou aktuálně viditelné na obrazovce, čímž dochází k optimalizaci aplikace [62]. V případě, že je nutné rozdělit data do určitých logických sekcí, je možné využít právě zmíněnou komponentu `SectionList`, která tuto a další funkce podporuje.

React Native poskytuje metody, které umožňují síťovou komunikaci. Využití těchto metod bude nezbytné pro dosažení cíle 3. testovacího scénáře. Pro načtení dat byla použita metoda `fetch()`, kterou je podle oficiální dokumentace [63] vhodné použít uvnitř metody `componentDidMount()`. Po načtení jsou data uložena do stavu komponenty, čímž dojde k jejímu překreslení. Popsaný kód je k nahlédnutí níže.

```
componentDidMount() {
    return fetch("http://192.168.214.49:5000/")
      .then(response => response.json())
      .then(responseJson => {
        this.setState(
```

```
          {
            isLoading: false,
            dataSource: responseJson
          },
          function() {}
        );
      })
      .catch(error => {
        console.error(error);
      });
  }
```

**Ukázka kódu 11: Načtení dat ze síťového zdroje – React Native. [Zdroj: Autor práce]**

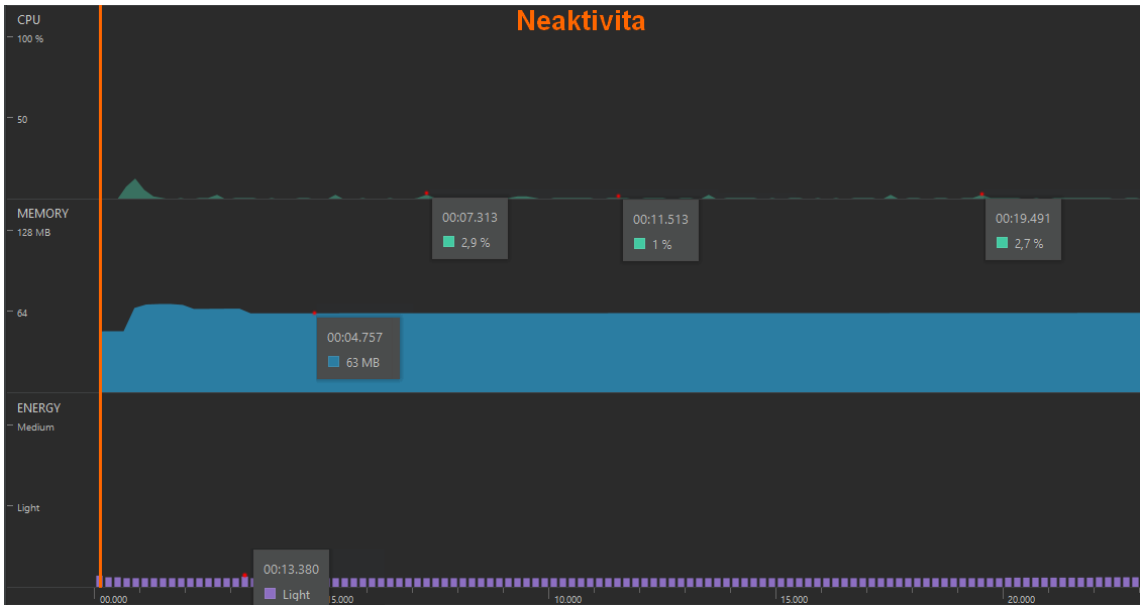
Dále bylo autorem vytvořeno uživatelské rozhraní, které vhodně zobrazí načtená data. K zobrazení dat byla použita komponenta `FlatList`, která dále obsahuje komponenty `Image`, `Text`, `Button` a `Switch`. Některé komponenty jsou naplněny daty, která byla načtena ze vzdáleného zdroje. Jiné byly použity pouze z důvodu donutit framework vykreslit více prvků a tím ho více zatížit. Obrázek je do komponenty `Image` načítán rovněž ze vzdáleného zdroje, čímž jsou opět testovány schopnosti frameworku.

```
<View style={{ flex: 1, padding: 10 }}>
  <FlatList
    data={this.state.dataSource}
    renderItem={({ item }) => {
      return (
        <View style={styles.item}>
          <View style={{ flex: 1 }}>
            <Image
              style={{ width: "100%", height: 50 }}
              source={{ uri: item.image }}
            />
          </View>
          <View style={{ flex: 3, paddingRight: 10, paddingLeft: 10 }}>
            <Text style={{ fontWeight: "bold" }}>{item.name}</Text>
            <Text>{item.description}</Text>
          </View>
          <View style={{ flex: 0.5 }}>
            <Button title="OK" />
            <Switch />
          </View>
        </View>
      );
```

```
    }}
    keyExtractor={({ id }, index) => id}
  />
</View>
```

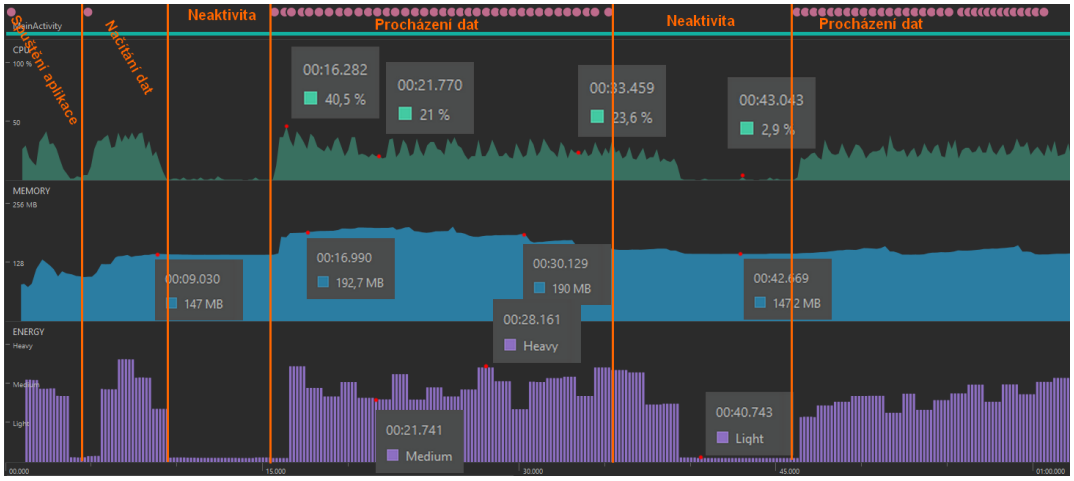
**Ukázka kódu 12: Uživatelské rozhraní pro zobrazení seznamu načtených dat. [Zdroj: Autor práce]**

Po dokončení uživatelského rozhraní byla vytvořená obrazovka podrobena testu, jehož cílem bylo změřit schopnosti frameworku a jeho náročnost na využití zdrojů mobilního zařízení. K měření byl použit profiler, který je poskytnut prostřednictvím vývojářského prostředí Android Studio. Autor práce spustil měření a po krátké chvíli otevřel testovací obrazovku. Po načtení dat začal s jejich procházením. Během procházení byly profilerem zaznamenávány důležité údaje, konkrétně vytížení CPU, využití paměti RAM a náročnost na spotřebu energie. Tento postup byl opakován třikrát. Na obrázcích níže se nachází naměřené hodnoty při všech třech opakováních. Dále je zde k nahlédnutí měření aplikace při neaktivitě.

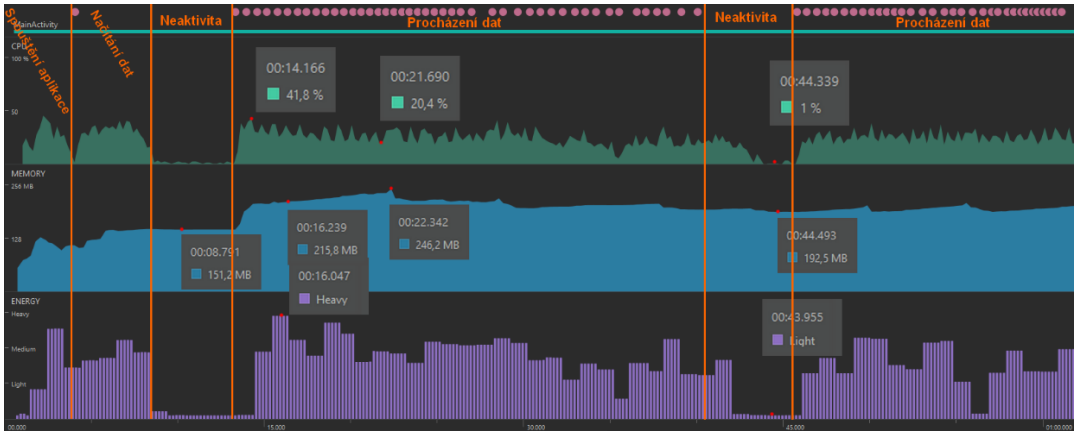



**Obrázek 15: Využití zdrojů při neaktivitě – React Native. [Zdroj: Autor práce]**

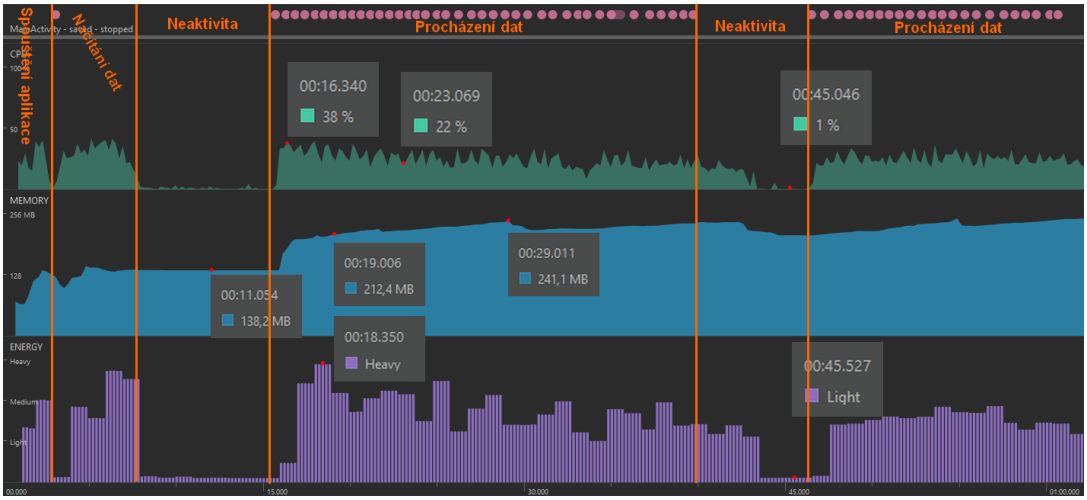
Při neaktivitě je aplikací využíváno přibližně 1 % až 3 % procesoru. Využití paměti RAM je rovněž konstantní. Aplikace využívá při neaktivitě přibližně 63 MB paměti. Po celou dobu je spotřeba energie označena jako nízká.




**Obrázek 16: Využití zdrojů při práci s velkým množstvím dat – 1. test React Native. [Zdroj: Autor práce]**




**Obrázek 17: Využití zdrojů při práci s velkým množstvím dat – 2. test React Native. [Zdroj: Autor práce]**




**Obrázek 18: Využití zdrojů při práci s velkým množstvím dat – 3. test React Native. [Zdroj: Autor práce]**

Při všech třech pokusech byly naměřené hodnoty velmi podobné. Při načítání testovacích dat se využití procesoru pohybovalo okolo 35 %, bylo využito přibližně 145 MB RAM a využití energie bylo označeno jako střední až vysoké. Po načtení dat kleslo vytížení procesoru na 1 %, využití paměti zůstalo na stejné hodnotě a využití energie bylo označeno jako nízké. V tuto chvíli neprobíhala s aplikací žádná interakce ze strany autora práce. Následovalo procházení načtených dat. Během tohoto procesu se vytížení procesoru pohybovalo přibližně mezi 22 % až 40 %, bylo využíváno 190 MB až 250 MB paměti RAM a využití energie bylo označeno jako střední až vysoké.

| | CPU (%) | RAM (MB) | Využití energie |
|---|---|---|---|
| **Neaktivita** | 1 - 3 | 63 | Nízké |
| **Načítání dat** | 35 | 145 | Střední – vysoké |
| **Neaktivita po načtení dat** | 1 | 145 | Nízké |
| **Procházení dat** | 22 - 40 | 190 - 250 | Střední - vysoké |

**Tabulka 4: Přehled naměřených přibližných hodnot – React Native. [Zdroj: Autor práce]**

## 5.3 Flutter

V prostředí frameworku Flutter jsou využívány speciální elementy, pomocí kterých je vytvářeno uživatelské rozhraní. Každý takovýto element se nazývá widget. Google v dokumentaci [64] zmiňuje, že se při vývoji systému widgetů inspiroval technologií React. Základní myšlenkou je, že se celé uživatelské rozhraní skládá z předdefinovaných widgetů. Každý widget si uchovává informaci o svém aktuálním nastavení, tedy vzhledu a stavu. V případě změny stavu widgetu dojde k jeho překreslení. Jako příklad widgetu je možné uvést `Text`, `Icon`, `Image`, `Column` či `Container`.

```
import 'package:flutter/material.dart';

void main() {
  runApp(
    Center(
      child: Text(
        'Hello, world!',
        textDirection: TextDirection.ltr,
```

```
      ),
    ),
  );
}
```

**Ukázka kódu 13: Použití základních widgetů. [64]**

V ukázkovém kódu jsou použity dva widgety. Widget `Center` je kořenový widget a slouží k zarovnání všech svým potomků na střed. Jeho potomkem je widget `Text`, který slouží k zobrazení krátkého textu. Jelikož je `Text` potomkem widgetu `Center`, bude widget `Text` příslušně zarovnán, tedy na střed. Mimo samotného textu je widgetu `Text` předána informace o směru psaní textu, v tomto případě zleva doprava.

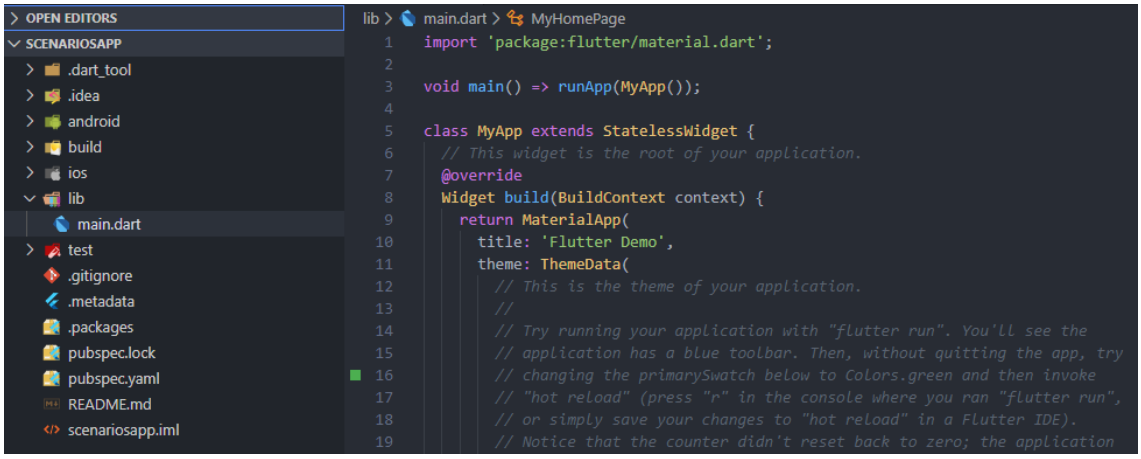
### 5.3.1 Testovací scénář 1 - instalace, vytvoření projektu a ladění aplikace

Prvním krokem při instalaci frameworku je stažení aktuální verze SDK, které je doporučeno stahovat z oficiálního zdroje frameworku [65]. Aby bylo možné spouštět příkazy frameworku Flutter z příkazové řádky systému Windows, je nutné přidat Flutter do systémové proměnné prostředí `PATH`, o čemž rovněž informuje oficiální dokumentace [65]. Spuštěním příkazu `flutter doctor`, autor provedl kontrolu správného nainstalování vývojového prostředí Flutter.

```
C:\Users\Jakub>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, v1.12.13+hotfix.8, on Microsoft Windows [Version 10.0.18362.720], locale cs-CZ)
[√] Android toolchain - develop for Android devices (Android SDK version 29.0.2)
[√] Android Studio (version 3.5)
[√] Connected device (1 available)

• No issues found!
```

**Obrázek 19: Kontrola správného nainstalování frameworku Flutter. [Zdroj: Autor práce]**

Dále autor vytvořil projekt aplikace pomocí příkazu `flutter create scenariosapp`. Strukturu souborů vytvořeného projektu je možné pozorovat na obrázku níže. Důležitým souborem je `main.dart`, který obsahuje hlavní widget aplikace.




**Obrázek 20: Struktura projektu Flutter a ukázka části hlavního widgetu aplikace. [Zdroj: Autor práce]**

Následným provedením příkazu `flutter run` byla vytvořená aplikace prvně spuštěna. Spuštění bylo rychlé a povedlo se bez jakýkoliv problémů. Při úpravě části zdrojového kódu hlavní komponenty se změna ve spuštěné aplikaci neprojevila ihned. Změny se projevily až po stisknutí příslušné klávesy, o čemž informuje i oficiální dokumentace frameworku [65]. Po stisknutí klávesy je aktualizace aplikace okamžitá bez nutnosti opětovné instalace či restartu.

Autor dále otestoval ladící nástroje, které jsou bohaté a intuitivní. Samotný framework umožňuje ladění aplikací přímo v internetovém prohlížeči, ale zůstává možnost využívat nástroje vývojových prostředí jako je např. Android Studio či Visual Studio Code.

### 5.3.2 Testovací scénář 2 - tvorba uživatelského rozhraní a navigace

Během implementace 2. scénáře byla autorem nejdříve vytvořena navigace mezi obrazovkami. Autor vytvořil obrazovky `scenario2Screen.dart` a `scenario3Screen.dart`. Následně v hlavním widgetu aplikace byla autorem vytvořena metoda `navigateToScreen()`, která podle zaslaného parametru provádí samotnou změnu aktivní obrazovky. Tato metoda je volána po stisknutí příslušného tlačítka. Pro dokončení navigační funkce nebylo nutné instalovat žádné dodatečné balíčky a knihovny. Na ukázce níže je možné si prohlédnout zmíněnou navigační metodu.

```
void navigateToScreen(String screenName) {
    switch (screenName) {
      case 'Scenario2Screen':
        Navigator.push(
          context,
          MaterialPageRoute(builder: (context) => Scenario2Screen()),
        );
        break;
      case 'Scenario3Screen':
        Navigator.push(
          context,
          MaterialPageRoute(builder: (context) => Scenario3Screen()),
        );
        break;
    }
  }
```

**Ukázka kódu 14: Metoda, která provádí změnu aktivní obrazovky – Flutter. [Zdroj: Autor práce]**

Autor dále vytvořil obsah obrazovky `scenario2Screen.dart`. Tímto byly otestovány možnosti tvorby uživatelského rozhraní pomocí frameworku Flutter. Autor vytvořil uživatelské rozhraní podle návrhu, který byl vytvořen v jedné z předešlých kapitol.

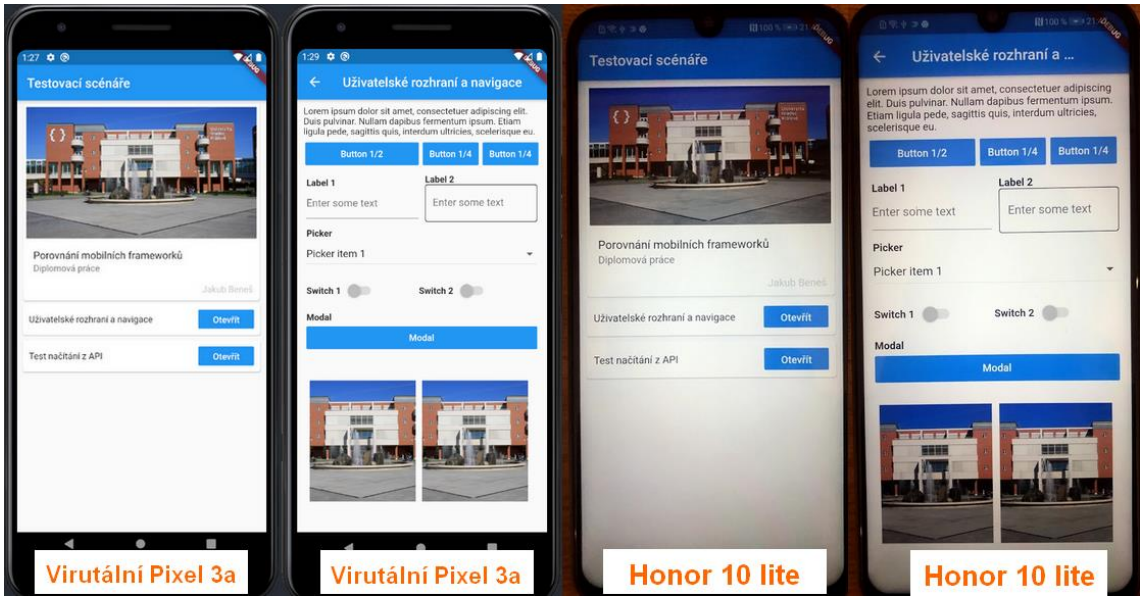
Možnosti tvorby uživatelského rozhraní byly otestovány i při vytváření úvodní obrazovky. Přestože neměl autor práce žádné zkušenosti s frameworkem Flutter ani s programovacím jazykem Dart, byla práce s frameworkem svižná a intuitivní. Framework autorovi poskytl veškeré prvky uživatelského rozhraní, které byly potřebné.

```
Container(
    padding: EdgeInsets.only(
        top: 5.0, right: 5.0, left: 5.0),
    child: const Image(
        image: AssetImage('assets/header.jpg'))),
const ListTile(
  title: Text('Porovnání mobilních frameworků'),
  subtitle: Text('Diplomová práce'),
),
Container(
  alignment: Alignment.centerRight,
  padding: EdgeInsets.only(right: 10, bottom: 10),
  child: Text('Jakub Beneš',
      style: TextStyle(
```

```
            color: const Color.fromRGBO(
                204, 204, 204, 1.0))),
)
```

**Ukázka kódu 15: Část zdrojového kódu uživatelského rozhraní úvodní obrazovky – Flutter. [Zdroj: Autor práce]**

Na obrázcích níže je možné pozorovat výsledné uživatelské rozhraní úvodní obrazovky a obrazovky druhého scénáře. Rozhraní bylo otestování na virtuálním zařízení Pixel 3a a reálném zařízení Honor 10 lite.




**Obrázek 21: Uživatelské rozhraní úvodní obrazovky a obrazovky druhého scénáře – Flutter. [Zdroj: Autor práce]**

### 5.3.3 Testovací scénář 3 - načítání a zobrazení seznamu objektů z REST API

Pro načítání dat ze síťového zdroje, Flutter poskytuje balíček nazvaný http. Balíček není součástí frameworku a je tedy nutné provést jeho instalaci. Instalace je velmi jednoduchá a byla provedena přidáním závislosti do konfiguračního souboru projektu. Následně byl balíček naimportován v hlavičce souboru obrazovky 3. testovacího scénáře.

```
dependencies:
  http: ^0.12.0+4
```

**Ukázka kódu 16: Přidání závislosti balíčku http – Flutter. [Zdroj: Autor práce]**

```
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;
```

**Ukázka kódu 17: Import balíčku http v hlavičce souboru – Flutter. [Zdroj: Autor práce]**

Dále autor práce vytvořil metodu `fetchScenarioItems()`, která pomocí metody `http.get()` vrací instanci třídy `Future` obsahující odpověď ze serveru. Třída `Future` je součástí jazyka Dart a je určena pro práci s asynchronními operacemi. Jelikož třída `Future` obsahuje odpověď ve formě nestrukturovaných dat, může být práce s touto třídou později nepohodlná. Z tohoto důvod autor vytvořil třídu `ScenarioItem`. Odpověď ze serveru je konvertována na instance objektů této třídy pomocí metody `ScenarioItem.fromJson()`. V ukázkách kódů níže je zobrazena implementace třídy `ScenarioItem` a metody `fetchScenarioItems()`.

```
class ScenarioItem {
  final int id;
  final String name;
  final String description;
  final String image;

  ScenarioItem({this.id, this.name, this.description, this.image});

  factory ScenarioItem.fromJson(Map<String, dynamic> json) {
    return ScenarioItem(
      id: json['id'],
      name: json['name'],
      description: json['description'],
      image: json['image'],
    );
  }
}
```

**Ukázka kódu 18: Třída určená pro práci s objekty získanými z API – Flutter. [Zdroj: Autor práce]**

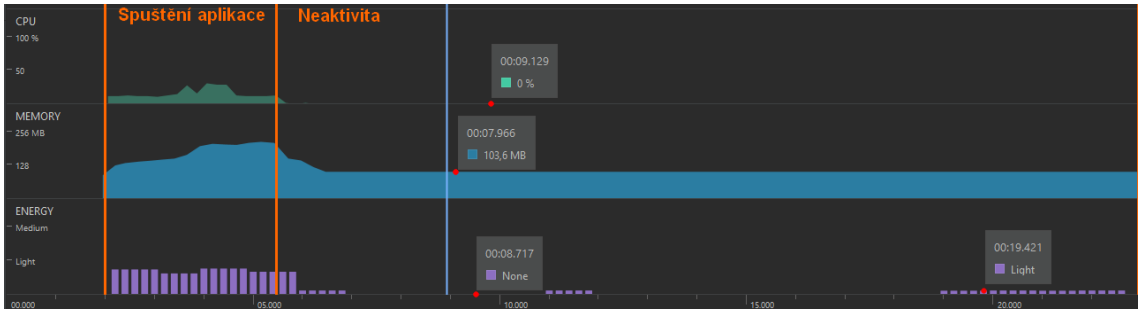
```
Future<List<ScenarioItem>> fetchScenarioItems() async {
  final response = await http.get('http://192.168.214.49:5000/');

  if (response.statusCode == 200) {
    // Pokud je ze serveru vrácena odpověď s kódem 200
    // Parsujeme json
   final parsed = jsonDecode(response.body).cast<Map<String, dynamic>>();
   return parsed
```

```
        .map<ScenarioItem>((json) => ScenarioItem.fromJson(json))
        .toList();
  } else {
    // Pokud server vrátit jinou odpověď, zobrazíme chybu
    throw Exception('Failed to load');
  }
}
```

**Ukázka kódu 19: Načtení dat ze síťového zdroje – Flutter. [Zdroj: Autor práce]**

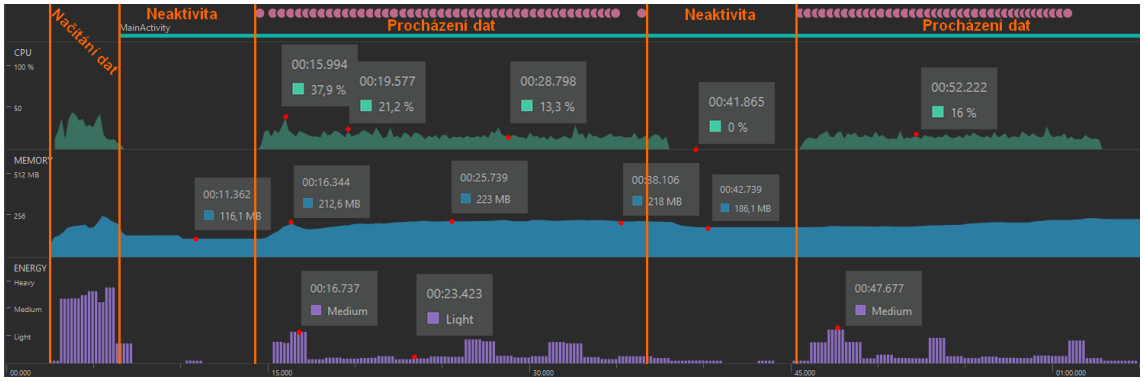
Následně autor vytvořil uživatelské rozhraní, které slouží k zobrazení seznamu načtených objektů. Při implementaci rozhraní se autor držel původního návrhu a použil widgety, které poskytuje framework Flutter. Nebylo nutné použít žádné externí knihovny. Použity byly např. widgety `ListView`, `Card`, `Container`, `Text`, `FlatButton` či `Image`.

V dalším kroku bylo autorem provedeno měření, jehož cílem bylo otestovat vytížení zdrojů mobilního zařízení při načítání a procházení dat. Stejně jako při testování frameworku React Native, i zde bylo testování opakováno třikrát. Při každém opakování byla autorem otevřena testovací obrazovka. Po načtení dat autor začal data procházet a pomocí nástroje profiler, byla zaznamenávána data. Výsledné naměřené údaje je možné pozorovat na obrázcích níže.

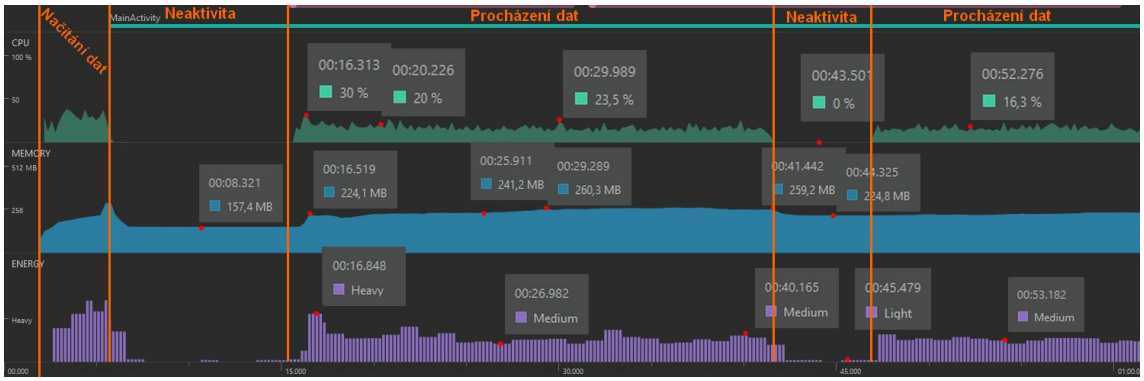



**Obrázek 22: Využití zdrojů při neaktivitě – Flutter. [Zdroj: Autor práce]**

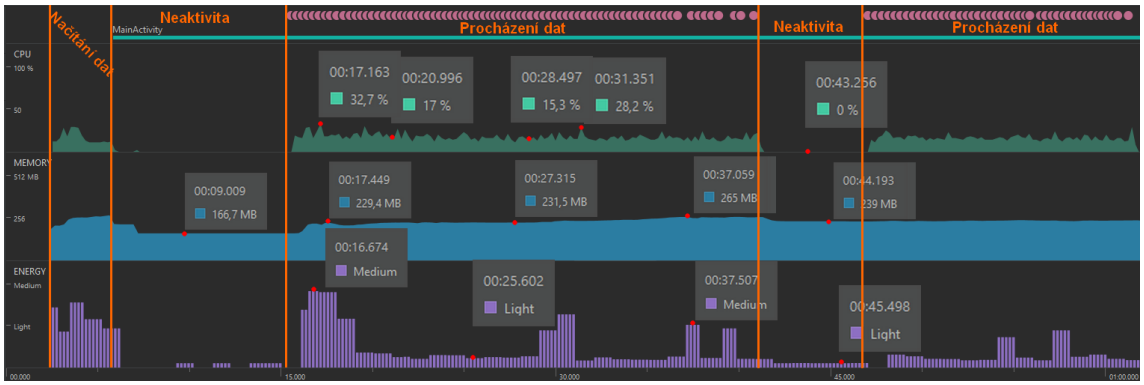
Při neaktivitě byly profilerem naměřeny konstantní hodnoty, podle kterých lze předpokládat, že je využití procesoru velmi nízké. Aplikace konstantně využívala přibližně 104 MB paměti RAM. Spotřeba energie je označena jako nízká, v určitých místech dokonce jako žádná.




**Obrázek 23: Využití zdrojů při práci s velkým množstvím dat – 1. test Flutter. [Zdroj: Autor práce]**




**Obrázek 24: Využití zdrojů při práci s velkým množstvím dat – 2. test Flutter. [Zdroj: Autor práce]**




**Obrázek 25: Využití zdrojů při práci s velkým množstvím dat – 3. test Flutter. [Zdroj: Autor práce]**

Při všech třech pokusech byly výsledné naměřené hodnoty velmi podobné. Při načítání dat se hodnota využití procesoru pohybovala okolo 35 % a bylo využíváno až přibližně 230 MB paměti RAM. Využití energie bylo ve dvou případech označeno jako střední. Při prvním testování se blížilo označení vysoké. Po načtení testovacích dat klesla hodnota využití procesoru velmi blízko hodnotě 0 %. Bylo

využíváno průměrně kolem 145 MB RAM a využití energie bylo označeno jako nízké. Během procházení dat byly hodnoty opět velmi podobné. Bylo využíváno přibližně 15 % až 38 % procesoru a 220 MB až 260 MB paměti RAM. Až na malou výjimku po načtení testovacích dat v jednom z pokusů, bylo využití energie označováno jako nízké až střední.

| | CPU (%) | RAM (MB) | Využití energie |
|---|---|---|---|
| **Neaktivita** | 0 | 104 | Žádné – nízké |
| **Načítání dat** | 35 | 230 | Nízké - vysoké |
| **Neaktivita po načtení dat** | 0 | 145 | Nízké |
| **Procházení dat** | 15 - 38 | 220 - 260 | Nízké - střední |

**Tabulka 5: Přehled naměřených přibližných hodnot – Flutter. [Zdroj: Autor práce]**

## 5.4 Xamarin

Jak již bylo popsáno v první části této diplomové práce, k vývoji aplikací pomocí frameworku Xamarin je primárně používán jazyk C#. Vývojáři mají možnost využít i jazyky F# a Visual Basic. K tvorbě uživatelského rozhraní je možné přistupovat dvojím způsobem. Jedním z těchto způsobů je využití frameworku Xamarin.Forms, který dává vývojářům možnost vytvořený kód sdílet napříč platformami. Podle článku What is Xamarin.Forms [66], tento nástroj využívá značkovací jazyk XAML, který je postavený nad jazykem XML. Výsledný kód je vykreslen do podoby nativních prvků pro danou platformu.

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="StackLayoutTutorial.MainPage">
    <StackLayout Margin="20,35,20,25">
        <Label Text="The Padding property can be set to specify the dista
nce between the StackLayout and its children." />
        <Label Text="The Spacing property can be set to specify the dista
nce between views in the StackLayout." />
    </StackLayout>
</ContentPage>
```

**Ukázka kódu 20: Vytvoření uživatelského rozhraní pomocí jazyka XAML - Xamarin. [67]**

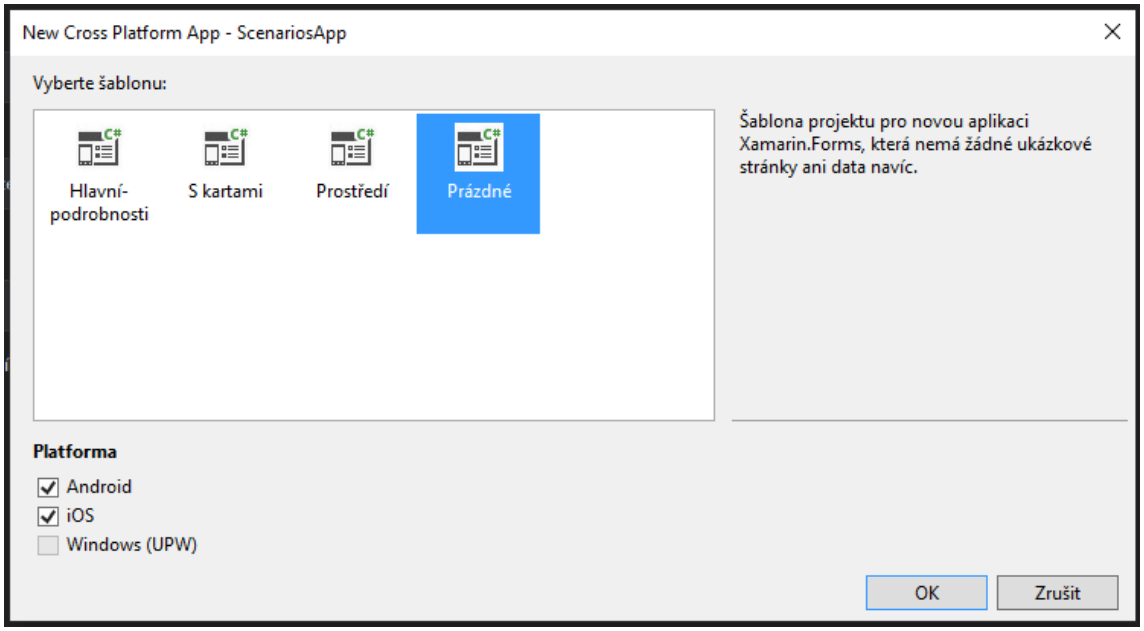
V ukázce zdrojového kódu je definováno uživatelské rozhraní pro jednu obrazovku aplikace. Obrazovka obsahuje dva prvky `Label`, které slouží k zobrazení textu. Text prvků je definován pomocí atributu `Text`. Zmíněné prvky jsou potomky prvku `StackLayout`. Atribut `Margin` nastavuje pozici prvku vůči jeho rodiči. V tomto případě je rodičem `ContentPage`, který definuje obrazovku jako takovou.

Dalším způsobem, jak tvořit uživatelské rozhraní aplikace, je využít standardní nástroje dané platformy. V takovémto případě není vytvořený kód sdílený a je nutné vytvořit uživatelské rozhraní, pro každou platformu zvlášť. V případě platformy Android jsou vytvářeny aktivity[13] a je používán značkovací jazyk AXML. Pro tvorbu uživatelského rozhraní pro platformu iOS je využíván tzv. storyboard[14].

### 5.4.1 Testovací scénář 1 - instalace, vytvoření projektu a ladění aplikace

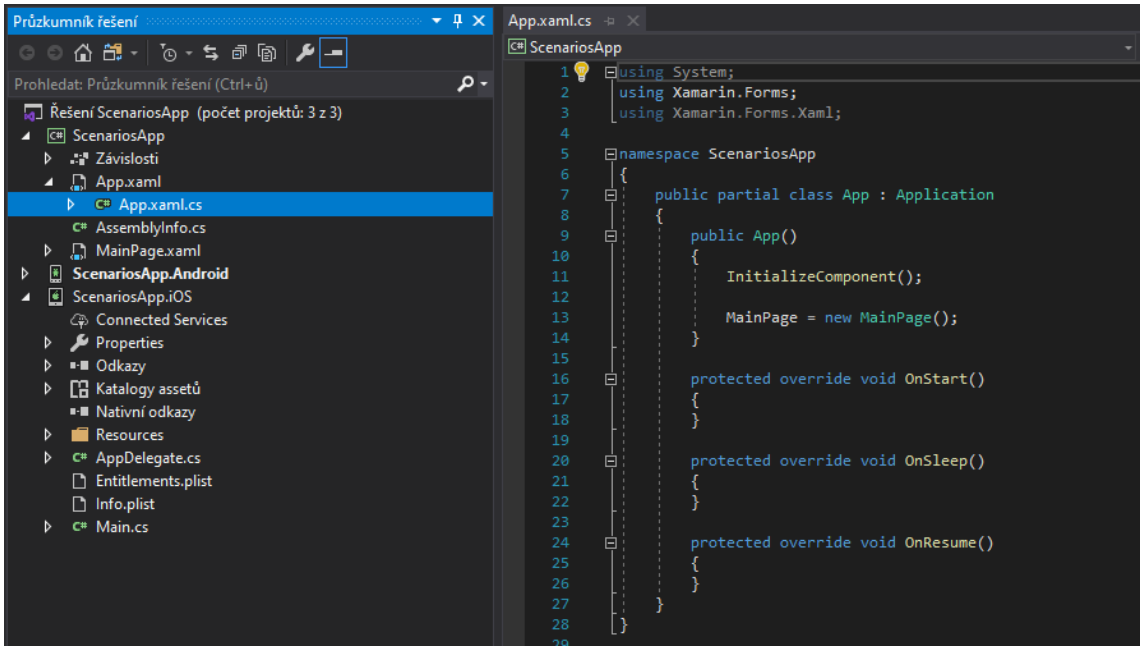
Při instalaci a vytváření projektu aplikace, autor práce postupoval podle oficiální dokumentace frameworku. Instalace byla provedena přes instalační balíček vývojářského nástroje Visual Studio. Přes uživatelské rozhraní stejného nástroje, autor vytvořil projekt aplikace. Autorovi bylo nabídnuto hned několik typů projektů. Pro navrhnuté testovací scénáře byl zvolen typ projektu nazvaný `Mobilní aplikace (Xamarin.Forms)`. V dalším kroku byl autor dotázán na typ šablony, nabízena byla např. šablona s postranním menu či šablona s kartami. Autor pro tuto práci zvolil šablonu prázdnou.

[13] Třída, která slouží jako obrazovka aplikace. Typicky se aplikace skládá z více aktivit, mezi kterými se uživatel pohybuje a interaguje s nimi.
[14] Vizuální reprezentace uživatelského rozhraní aplikací pro platformu iOS.




**Obrázek 26: Možnost zvolení šablony projektu ve frameworku Xamarin. [Zdroj: Autor Práce]**

Struktura vytvořeného projektu je zobrazena na obrázku níže. Důležité je zmínit, že projekt je rozdělen na 3 části, konkrétně `ScenarioApp`, `ScenarioApp.Android` a `ScenarioApp.iOS`. Sekce `ScenarioApp` obsahuje sdílené zdrojové kódy hlavních funkcí aplikace a uživatelského rozhraní. Důležitým souborem je `App.xaml.cs`, který obsahuje hlavní třídu `App`, která slouží jako výchozí bod aplikace. V projektu je připravena jedna obrazovka pojmenovaná `MainPage`. Další dvě části projektu, tedy `ScenarioApp.Android` a `ScenarioApp.iOS`, jsou určené pro specifický nesdílený kód daných platforem.




**Obrázek 27: Struktura projektu Xamarin a ukázka třídy App. [Zdroj: Autor práce]**

Autor provedl první spuštění aplikace pomocí programu Visual Studio. Spuštění bylo rychlé a proběhlo bez jakýchkoliv problémů. Autor otestoval použití ladících nástrojů, které poskytuje přímo nástroj Visual Studio. Ladící nástroje jsou velmi komplexní a poskytují vývojářům široké možnosti.

Při změně ve zdrojovém kódu uživatelského rozhraní, tedy v souborech typu XAML, se změny okamžitě projevily ve spuštěné aplikaci bez nutnosti restartu či opětovné instalace. Při zásahu do tříd např. `MainPage` je nutné provést restart aplikace.

### 5.4.2 Testovací scénář 2 - tvorba uživatelského rozhraní a navigace

Podobně jako při implementaci 2. scénáře v ostatních frameworcích, i v tomto případě začal autor s tvorbou navigačního systému. Byly vytvořeny třídy `Scenario2` a `Scenario3`, které slouží jako obrazovky, mezi kterými se může uživatel pohybovat. Třída `MainPage`, která slouží jako úvodní obrazovka, byla součástí šablony projektu. O samotnou navigaci mezi obrazovkami se stará třída `NavigationPage`, která je součástí frameworku Xamarin. Autorem byla na hlavní obrazovce vytvořena tlačítka. Akce kliknutí na konkrétní tlačítko je obsloužena následujícími metodami.

```
async private void btnScenario2_Clicked(object sender, EventArgs e)
{
    await Navigation.PushAsync(new Scenario2());
}

async private void btnScenario3_Clicked(object sender, EventArgs e)
{
    await Navigation.PushAsync(new Scenario3());
}
```

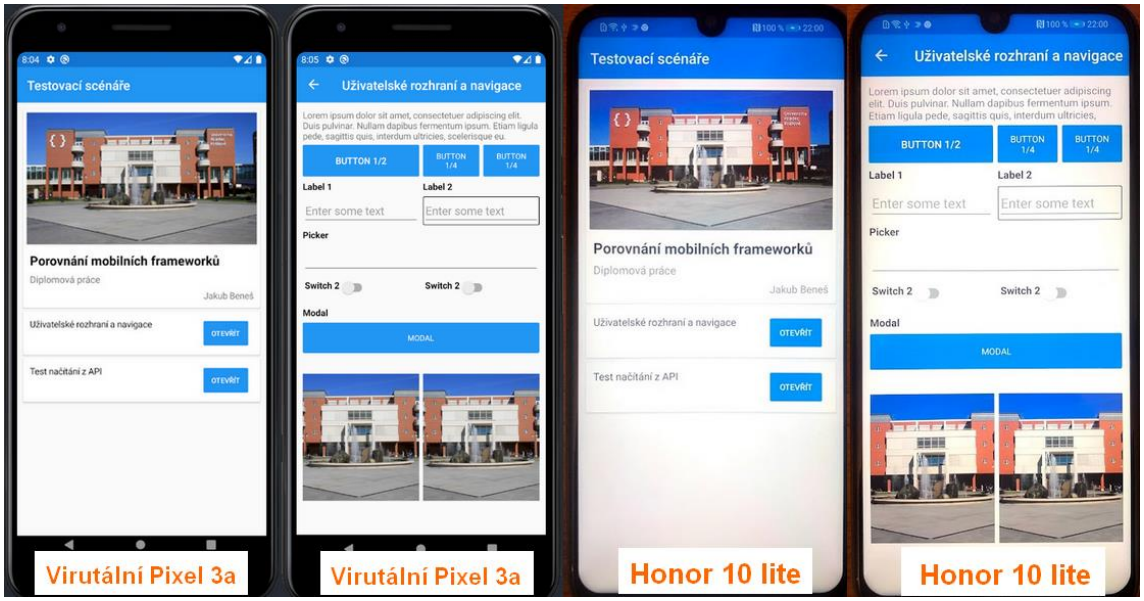
**Ukázka kódu 21: Obsluha kliknutí a provedení změny aktivní obrazovky pomocí NavigationPage – Xamarin. [Zdroj: Autor práce]**

Autor se dále věnoval tvorbě uživatelského rozhraní podle původního návrhu. Během tvorby využil framework Xamarin.Forms. Níže je k nahlédnutí ukázka zdrojového kódu v jazyce XAML. Jedná se o úvodní obrazovku aplikace. K jejímu vytvoření byly použity komponenty jako `StackLayout`, `Frame`, `Image` či `Label`. Autor si vystačil s komponentami, které poskytuje samotný framework. Nebylo nutné instalovat dodatečné knihovny či používat připravená uživatelská rozhraní třetích stran.

```
<Frame Padding="5">
    <StackLayout>
        <Image Source="header.jpg" />
            <StackLayout Padding="5">
                <Label
                FontSize="20"
                TextColor="Black"
                FontAttributes="Bold"
                Text="Porovnání mobilních frameworků"/>
                <Label
                FontSize="15"
                Text="Diplomová práce"/>
                <Label
                HorizontalOptions="FillAndExpand"
                HorizontalTextAlignment="End"
                Text="Jakub Beneš"/>
            </StackLayout>
        </StackLayout>
</Frame>
```

**Ukázka kódu 22: Část kódu uživatelského rozhraní úvodní obrazovky v jazyce XAML – Xamarin. [Zdroj: Autor práce]**

Na obrázcích níže je možné pozorovat výsledné uživatelské rozhraní úvodní obrazovky a obrazovky druhého scénáře. Rozhraní bylo otestování na virtuálním zařízení Pixel 3a a reálném zařízení Honor 10 lite.




**Obrázek 28: Uživatelské rozhraní úvodní obrazovky a obrazovky druhého scénáře – Xamarin. [Zdroj: Autor práce]**

### 5.4.3 Testovací scénář 3 - načítání a zobrazení seznamu objektů z REST API

Aby byl splněn cíl 3. testovacího scénáře, byly v prvním kroku vytvořeny třídy, jejichž úkolem je načítat data ze síťového zdroje. Konkrétně se jedná o třídy `ScenarioItemanager` a `RestService`. Třída `RestService` implementuje metodu `RefreshDataAsync()`, která pomocí třídy `HttpClient` provádí samotné načtení síťových dat. Třída `HttpClient` je součástí frameworku. V ukázce níže se nachází zmíněná metoda `RefreshDataAsync()`.

```
public async Task<List<ScenarioItem>> RefreshDataAsync()
{
    Items = new List<ScenarioItem>();
    var uri = new Uri(string.Format("http://192.168.214.49:5000", string.Empty));
    try {
        var response = await _client.GetAsync(uri);
        if (response.IsSuccessStatusCode) {
            var content = await response.Content.ReadAsStringAsync();
            Items = JsonConvert.DeserializeObject<List<ScenarioItem>>(content);
        }
    }
    catch (Exception ex){
        Debug.WriteLine(@"Došlo k chybě: ", ex.Message);
    }
    return Items;
}
```

**Ukázka kódu 23: Načtení dat ze síťového zdroje – Xamarin. [Zdroj: Autor práce]**

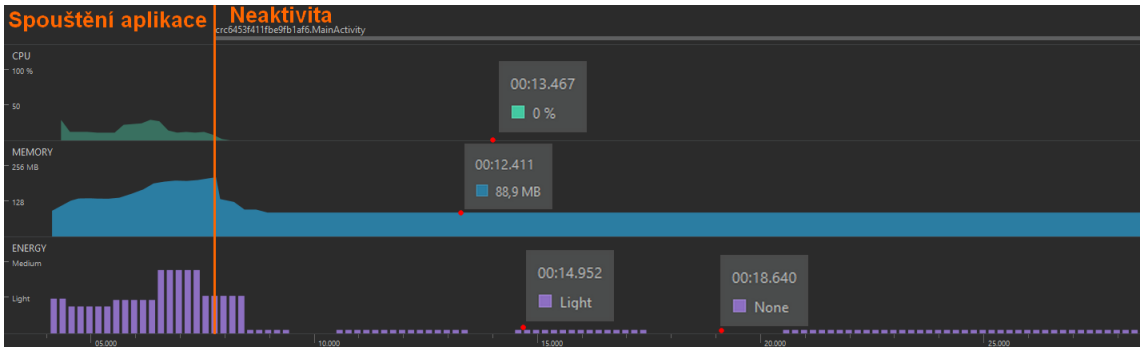
V ukázce zdrojového kódu se nachází i zatím nezmíněná třída `ScenarioItem`. Tato třída byla rovněž vytvořena autorem a slouží pro práci s daty načtenými z testovacího REST API. Třída je k nahlédnutí v ukázce níže.

```
public class ScenarioItem
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
    public string Image { get; set; }
}
```

**Ukázka kódu 24: Třída ScenarioItem – Xamarin. [Zdroj: Autor práce]**

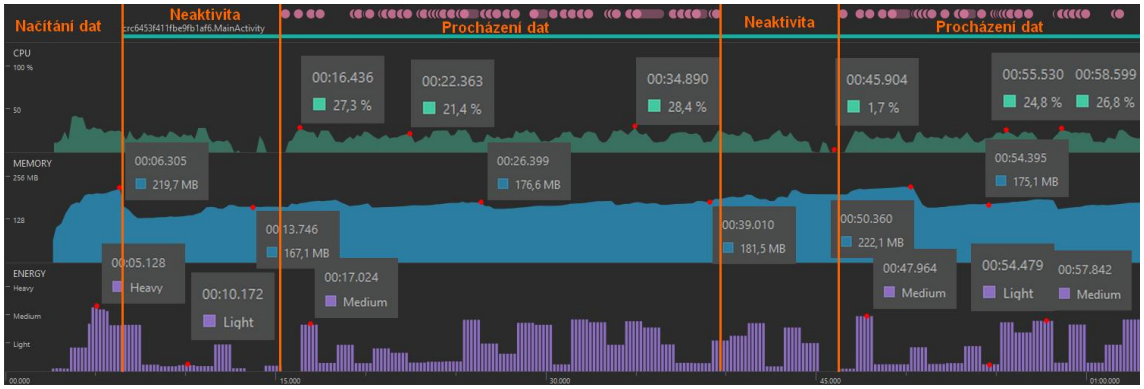
Dále bylo vytvořeno uživatelské rozhraní, které slouží k zobrazení načtených dat. K zobrazení seznamu hodnot poskytuje framework Xamarin komponentu nazvanou `ListView`. Podle oficiální dokumentace frameworku je vhodné pro velké množství různě strukturovaných dat použít komponentu `CollectionView`. Díky této komponentě je možné dosáhnout větší flexibility a většího výkonu oproti alternativní komponentě `ListView`.

Následně bylo autorem provedeno testování využití zdrojů. Jako v předchozích případech, i zde byl k měření hodnot využit profiler integrovaný ve vývojovém prostředí Android Studio. Autor se zaměřil na využití zdrojů při neaktivitě a při procházení načtených dat. Opět byl test opakován třikrát. Na obrázcích níže se nachází naměřené hodnoty.

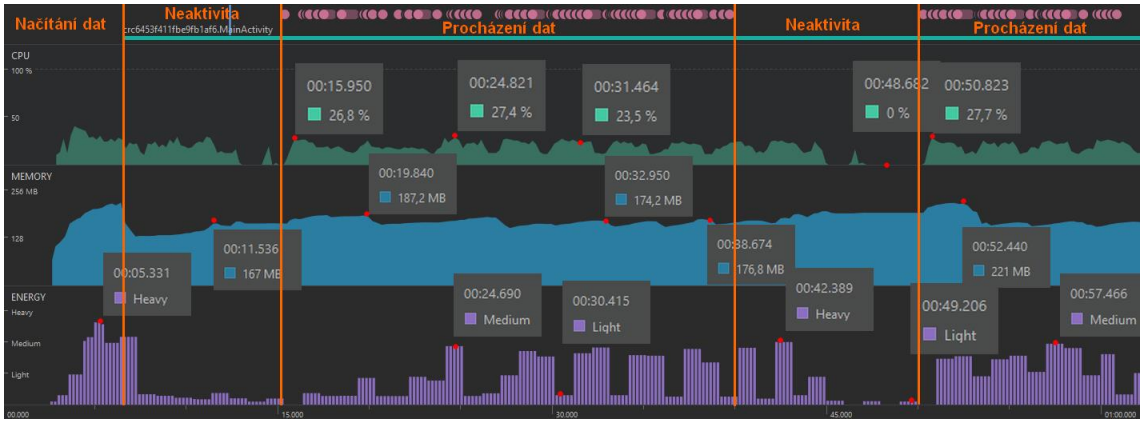



**Obrázek 29: Využití zdrojů při neaktivitě – Xamarin. [Zdroj: Autor práce]**

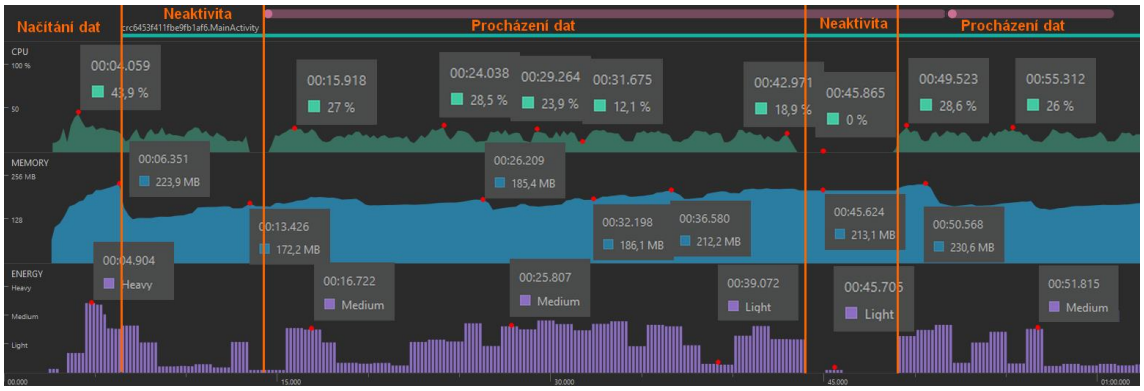
Při neaktivitě nedocházelo k žádným výkyvům. Po spuštění aplikace a následné neaktivitě byla naměřená hodnota využití procesoru 0 %. Využití paměti RAM bylo rovněž konstantní a naměřeno bylo přibližně 89 MB. Využití energetických zdrojů bylo označeno jako žádné až nízké.




**Obrázek 30: Využití zdrojů při práci s velkým množstvím dat – 1. test Xamarin. [Zdroj: Autor práce]**




**Obrázek 31: Využití zdrojů při práci s velkým množstvím dat – 2. test Xamarin. [Zdroj: Autor práce]**




**Obrázek 32: Využití zdrojů při práci s velkým množstvím dat – 3. test Xamarin. [Zdroj: Autor práce]**

Všechny tři opakované testy přinesly velmi podobné výsledky. Při načítání dat se využití procesoru pohybovalo okolo 26 % až 44 %. Aplikace využívala až 225 MB paměti RAM a využití energetického zdroje bylo označeno jako vysoké. Po načtení dat klesla hodnota využití procesoru až na hodnotu 0 %. Pokles však nebyl

okamžitý. Hodnota využití paměti RAM se pohybovala okolo 170 MB a využití energie bylo označeno jako nízké. Při následném procházení dat se využití procesoru pohybovalo mezi 12 % až 27 %. Bylo využito 170 MB až 230 MB paměti RAM a využití energetických zdroj bylo označeno jako nízké až střední.

| | CPU (%) | RAM (MB) | Využití energie |
|---|---|---|---|
| **Neaktivita** | 0 | 59 | Žádné – nízké |
| **Načítání dat** | 26 až 44 | 225 | Vysoké |
| **Neaktivita po načtení dat** | 0 | 170 | Nízké |
| **Procházení dat** | 12 až 27 | 170 až 230 | Nízké – střední |

**Tabulka 6: Přehled naměřených přibližných hodnot – Xamarin. [Zdroj: Autor práce]**

# 6 Shrnutí výsledků

V praktické části této práce byly navrhnuty testovací scénáře, které si kladly za cíl otestovat vybrané mobilní multiplatformní frameworky podle několika kritérií. První scénář byl zaměřen na složitost instalace a vytvoření první mobilní aplikace ve vybraném frameworku. Zároveň byly prozkoumány ladící a testovací nástroje. Ve druhém scénáři se autor práce věnoval vývoji navrhnutého uživatelského rozhraní a prozkoumal možnosti, které frameworky během tvorby rozhraní poskytují. Třetí scénář si kladl za cíl otestovat výkon a využití zdrojů mobilního zařízení. Testováno bylo využití procesoru a pamětí RAM. Sledováno bylo i využití energetického zdroje.

## *6.1 Testovací scénář 1 – instalace, vytvoření projektu a ladění aplikace*

Instalace všech vybraných frameworků byla velmi jednoduchá a intuitivní. Autor postupoval podle oficiálních dokumentací daných frameworků, které ve všech případech poskytují veškeré potřebné informace. Mnohdy jsou informace velmi podrobné a vysvětlují postupy velmi detailně.

Rozdíly mezi frameworky je však možné nalézt ve velikosti samotných frameworků. V případě frameworku React Native se velikost pohybuje v řádech několika málo desítek MB. Malá velikost je způsobena tím, že framework obsahuje skutečně jen základní funkce a komponenty. Instalační balíček frameworku Flutter již dosahuje velikosti větší, konkrétně se jedná o stovky MB. Framework Xamarin posouvá hranici ještě výše. Oficiální dokumentace frameworku doporučuje využívat vývojové prostředí Visual Studio, velikost zmíněného softwaru dosahuje necelých 700 MB. Následně je nutné nainstalovat samotný framework Xamarin společně s frameworkem .NET. Zmíněná instalace dosahuje velikosti několika GB.

Podobně jako instalace frameworku, tak i vytvoření projektu aplikace bylo v každém zvoleném frameworku velmi jednoduchým procesem. Autor opět postupoval podle dokumentace, kde našel všechny potřebné informace. Frameworky React Native a Xamarin poskytují předdefinované šablony, které jsou vývojářům plně k dispozici. Framework Flutter tuto možnost nenabízel. Následné

spuštění aplikace proběhlo ve všech případech bez problémů. Všechny frameworky podporují funkci zobrazení změn v reálném čase.

Autor dále vyzkoušel dostupné ladící nástroje. Frameworky poskytují bohaté možnosti testování a ladění. Při ladění aplikace napsané ve frameworku React Native bylo nutné doinstalovat rozšíření webového prohlížeče. Rozšíření autorovi poskytlo informace o komponentách, jejich vlastnostech a stavech. Framework Flutter umožňuje ladění přímo v rámci vývojového prostředí Android Studio. Nechybí možnost ladění aplikací přímo v prohlížeči pomocí nástroje Dart DevTools. Ladící nástroje frameworku Xamarin jsou velmi komplexní a jsou dostupné skrze vývojové prostředí Visual Studio.

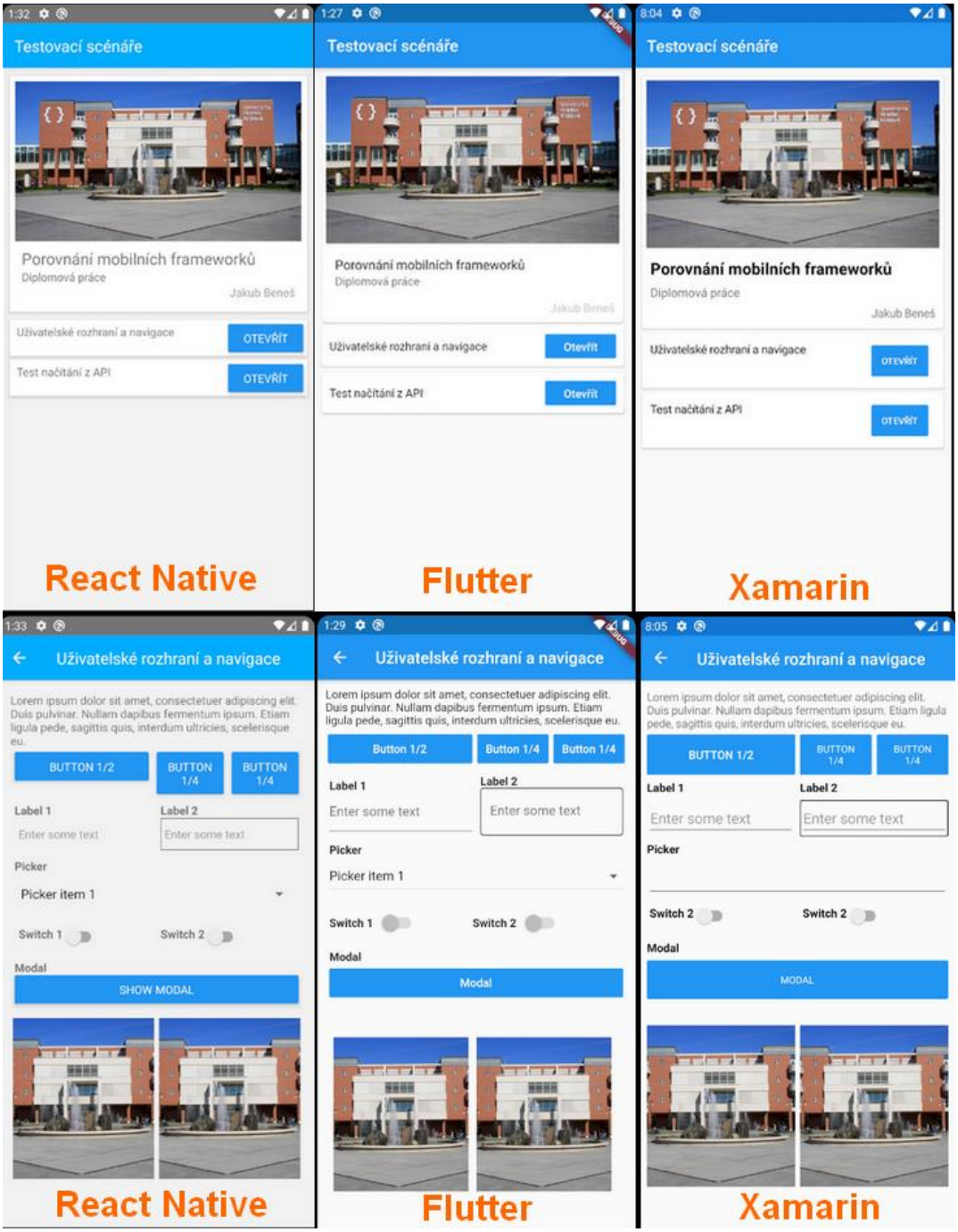
| Framework | Velikost (MB) | Předdefinované šablony | Funkce hot-reload | Ladění a testování |
|---|---|---|---|---|
| React Native | 29,6 | Ano | Ano | Nástroje třetích stran |
| Flutter | 645 | Ne | Ano | Android Studio |
| Xamarin (Visual Studio, .NET, Xamarin) | 7454,72 | Ano | Ano | Visual Studio |

**Tabulka 7: Porovnání vybraných frameworků – 1. testovací scénář. [Zdroj: Autor práce]**

## 6.2 Testovací scénář 2 - tvorba uživatelského rozhraní a navigace

V prvním kroku druhého testovacího scénáře byl implementován navigační systém aplikace a byly vytvořeny obrazovky, mezi kterými navigace probíhá. Framework React Native navigační systém neobsahuje a bylo nutné použít knihovnu třetích stran. Konkrétně byla použita knihovna `react-navigation`. Samotná instalace knihovny a následná implementace byla velmi jednoduchá a autor práce nenarazil na žádné komplikace. Framework Flutter naopak navigační systém již obsahuje a nebylo nutné instalovat knihovny třetích stran. Implementace navigačního systému proběhla bez komplikací. Xamarin rovněž obsahuje navigační systém a nebylo tedy nutné instalovat dodatečné balíčky. Systém je velmi jednoduchý na pochopení i implementaci.

Následně bylo autorem práce vytvořeno uživatelské rozhraní aplikace. Autor se zaměřil na úvodní obrazovku a obrazovku druhého testovacího scénáře. Výsledné uživatelské rozhraní je možné pozorovat na obrázku níže. Autor práce dodržel původní návrh obrazovek a výsledné uživatelské rozhraní je ve všech třech vybraných frameworcích velmi podobné.




**Obrázek 33: Porovnání vytvořeného uživatelského rozhraní ve vybraných frameworcích. [Zdroj: Autor práce]**

Během tvorby rozhraní nebylo nutné použít žádné knihovny třetích stran, autor si tedy vystačil s komponentami samotných frameworků. Tvorba uživatelského rozhraní byla ve všech frameworcích pohodlná a autor neměl problémy s dosažením definovaných cílů. Je nutné zmínit, že autor práce má základní zkušenosti s knihovnou React, na které je framework React Native postaven. Přesto byla doba potřebná k vytvoření navrhnutého uživatelského rozhraní téměř totožná.

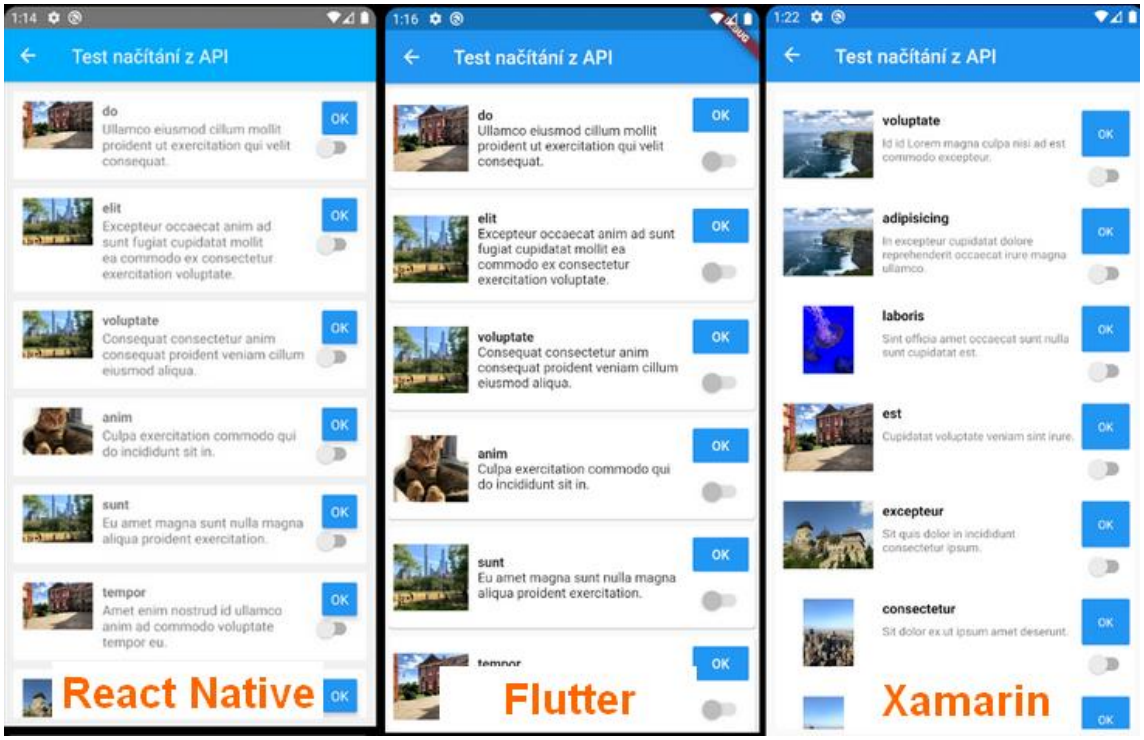
| **Framework** | **Navigační systém** | **UI – Nutnost použít knihovny třetích stran** | **Čas potřebný k dosažení navrhnutého UI** |
|---|---|---|---|
| React Native | Nutné použít knihovny třetích stran | Ne | Přibližně 40min |
| Flutter | Obsahuje framework | Ne | Přibližně 1h |
| Xamarin | Obsahuje framework | Ne | Přibližně 50min |

**Tabulka 8: Porovnání vybraných frameworků – 2. testovací scénář. [Zdroj: Autor práce]**

## *6.3 Testovací scénář 3 - načítání a zobrazení seznamu objektů z REST API*

Třetí testovací scénář byl zaměřen na výkon framework, konkrétně na schopnost zpracovávat velké množství dat. Během testování byla měřena plynulost aplikace, rychlost načítání dat a nároky aplikace na zdroje mobilního zařízení.

Autor práce navrhl a vytvořil testovací REST API, které vytvořeným aplikacím poskytlo testovací data prostřednictvím síťové komunikace. Následně úspěšně vytvořil testovací obrazovku podle původního návrhu. Během následného testování byly pomocí profileru změřeny následující hodnoty: využití CPU, využití paměti RAM, náročnost na využití energetického zdroje. Profiler zajistil, že naměřené hodnoty pochází pouze z vytvořené aplikace a nejsou ovlivněny aplikacemi dalšími, či samotným systémem.




**Obrázek 34: Porovnání vytvořeného uživatelského rozhraní načtených dat ve vybraných frameworcích.**

Prvním testem byly měřeny hodnoty při neaktivitě. Autor spustil aplikaci a dále již nepokračoval v interakci. Po dobu přibližně 20 až 25 vteřin byly naměřeny hodnoty, které je možné si prohlédnout v tabulce níže. Z naměřených hodnot lze konstatovat, že náročnost vytvořených aplikací na zdroje mobilního zařízení při neaktivitě je téměř totožná. Rozdíl lze pozorovat ve využití paměti RAM.

| Framework | CPU (%) | RAM (MB) | Využití energie |
|---|---|---|---|
| React Native | 1 - 3 | 63 | Nízké |
| Flutter | 0 | 104 | Žádné - nízké |
| Xamarin | 0 | 59 | Žádné - nízké |

**Tabulka 9: Porovnání naměřených hodnot při neaktivitě. [Zdroj: Autor práce]**

Dále autor provedl testování načítání dat z externího síťového zdroje a následné procházení těchto dat. Po spuštění aplikace byla autorem otevřena obrazovka třetího testovacího scénáře, čímž začalo načítání testovacích dat. Načtení dat bylo ve všech aplikacích provedeno v čase přibližně 6 vteřin. Využití procesoru bylo ve všech případech velmi podobné. Rozdíly byly viditelné ve využití paměti

RAM, kde byl framework React Native šetrnější než frameworky ostatní. Všechny aplikace dosahovaly v určitých opakováních vysokého vytížení energetických zdrojů. Framework Xamarin dosahoval vysokého využití energie ve všech třech pokusech.

| Framework | CPU (%) | RAM (MB) | Využití energie |
|---|---|---|---|
| React Native | 35 | 145 | Střední - vysoké |
| Flutter | 35 | 230 | Nízké - vysoké |
| Xamarin | 26 - 44 | 225 | Vysoké |

**Tabulka 10: Porovnání naměřených hodnot při načítání dat. [Zdroj: Autor práce]**

Následná neaktivita po načtení testovacích dat přinesla opět velmi podobné výsledky. Frameworky byly šetrné na využití CPU, pamětí RAM i energetického zdroje.

| Framework | CPU (%) | RAM (MB) | Využití energie |
|---|---|---|---|
| React Native | 1 | 145 | Nízké |
| Flutter | 0 | 145 | Nízké |
| Xamarin | 0 | 170 | Nízké |

**Tabulka 11: Porovnání naměřených hodnot při neaktivitě po načtení dat. [Zdroj: Autor práce]**

Během následného procházení dat byly naměřeny hodnoty, které lze pozorovat v tabulce níže. Aplikace vytvořená pomocí frameworku React Native vytěžovala procesor v určitých chvílích až ze 40 %. Většinu času se však hodnoty pohybovaly v rozmezí 22 % až 35 %. Framework Flutter rovněž v určitých chvílích dosáhl podobné hodnoty, konkrétně 38 % využití procesoru. Většinu času byla však hodnota nižší. Nejméně procesor vytěžovala aplikace vytvořená ve frameworku Xamarin. Nejvýše se dostala na hodnotu 27 %, ale stejně jako v předchozích případech, i zde bylo většinu času vytížení nižší. Vytížení paměti RAM bylo ve všech aplikacích velmi podobné. Nejnáročnější aplikací na využití energie byla aplikace vytvořená pomocí frameworku React Native. Aplikace neklesla pod střední vytížení a dosahovala i vytížení vysokého. Frameworky Flutter a Xamarin si vedly o poznání

lépe. Vytížení energetického zdroje těchto frameworků bylo při procházení dat označeno jako nízké až střední.

| Framework | CPU (%) | RAM (MB) | Využití energie |
|---|---|---|---|
| React Native | 22 - 40 | 190 - 250 | Střední – vysoké |
| Flutter | 15 - 38 | 220 - 260 | Nízké – střední |
| Xamarin | 12 - 27 | 170 - 230 | Nízké - střední |

**Tabulka 12: Porovnání naměřených hodnot při procházení dat. [Zdroj: Autor práce]**

# 7 Závěry a doporučení

Cílem této práce bylo prozkoumat a představit čtenáři možnosti vývoje mobilních aplikací. Autor se dále zaměřil na multiplatformní vývoj. Zvolil několik frameworků, které podrobně popsal. Frameworky byly zvoleny podle jejich oblíbenosti, ale text práce se věnoval i méně známým frameworkům.

Jelikož se jedná o stále se rozvíjející oblast, lze v budoucnu doporučit průzkum frameworků opakovat. Zcela jistě se objeví frameworky nové a některé z aktuálně popsaných mohou zaniknout. Dále je nutné zmínit, že se práce nevěnovala všem existujícím frameworkům. Záměrně byly vynechány hybridní frameworky, jejichž porovnání by mohlo být rovněž velmi zajímavé. Práce se zaměřila na multiplatformní frameworky, které umožňují nativní vývoj. Avšak ani z této oblasti nebyly představeny všechny existující technologie.

Práce se v praktické části zabývala návrhem a implementací testovacích scénářů, jejichž cílem bylo porovnat vybrané multiplatformních frameworky podle různých kritérií. I zde je vhodné doporučit opakování, jelikož se vybrané frameworky stále vyvíjí a porovnání by mohlo v budoucnu přinést jiné výsledky. Rovněž se nabízí k porovnání zvolit jiné frameworky a navrhnout nové testovací scénáře.

# 8 Seznam použité literatury


[1] „Brief History of Mobile Apps - JetRuby". https://expertise.jetruby.com/brief-history-of-mobile-apps-286fbbf766a9 (viděno bře. 09, 2020).

[2] „Operating System List". https://www.operating-system.org/betriebssystem/_english/os-liste.htm (viděno led. 05, 2020).

[3] „Mobile Operating System Market Share Worldwide | StatCounter Global Stats". https://gs.statcounter.com/os-market-share/mobile (viděno led. 20, 2020).

[4] „Mobile Operating System Market Share United States Of America | StatCounter Global Stats". https://gs.statcounter.com/os-market-share/mobile/united-states-of-america/ (viděno led. 20, 2020).

[5] „Mobile Operating System Market Share Europe | StatCounter Global Stats". https://gs.statcounter.com/os-market-share/mobile/europe (viděno led. 20, 2020).

[6] „Mobile Operating System Market Share China | StatCounter Global Stats". https://gs.statcounter.com/os-market-share/mobile/china/ (viděno led. 20, 2020).

[7] „The history of Android OS: its name, origin and more". https://www.androidauthority.com/history-android-os-name-789433/ (viděno led. 06, 2020).

[8] „5 Reasons Why Android Is So Much More Popular Than iPhone". https://www.makeuseof.com/tag/android-more-popular-iphone/ (viděno led. 06, 2020).

[9] „• Mobile share of website visits worldwide 2018 | Statista". https://www.statista.com/statistics/241462/global-mobile-phone-website-traffic-share/ (viděno led. 20, 2020).

[10] „What are Progressive Web Apps?" https://web.dev/what-are-pwas/ (viděno bře. 09, 2020).

[11] „Ionic Article: What is Hybrid App Development?" https://ionicframework.com/resources/articles/what-is-hybrid-app-development (viděno bře. 09, 2020).

[12] „WKWebView - WebKit | Apple Developer Documentation". https://developer.apple.com/documentation/webkit/wkwebview (viděno bře. 09, 2020).

[13] „WebView | Android Developers". https://developer.android.com/reference/android/webkit/WebView (viděno bře. 09, 2020).

[14] „Cross-Platform vs Hybrid | EBS Integrator". https://ebs-integrator.com/blog/cross-platform-vs-hybrid-two-different-stories/ (viděno bře. 09, 2020).

[15] „Cross-platform mobile development 2020: trends and frameworks". https://www.merixstudio.com/blog/cross-platform-mobile-development/ (viděno led. 11, 2020).

[16] „The Mobile App Comparison Chart: Hybrid vs. Native vs. Mobile Web (2019 Update) - mrc's Cup of Joe Blog". https://www.mrc-productivity.com/blog/2019/10/the-mobile-app-comparison-chart-hybrid-vs-native-vs-mobile-web/ (viděno bře. 09, 2020).

[17] „Native apps vs. progressive web apps: Pros and cons | Adjust". https://www.adjust.com/blog/native-app-vs-progressive-web-app/ (viděno bře. 09, 2020).
[18] „Progressive Web App vs Native App vs Cross Platform vs Hybrid App - SPEC INDIA". https://www.spec-india.com/blog/progressive-web-app-vs-native-app-vs-cross-platform-vs-hybrid-app (viděno bře. 09, 2020).
[19] „React Native, Flutter, NativeScript, Xamarin - Prozkoumat - Trendy Google". https://trends.google.com/trends/explore?cat=31&date=2017-12-01%202019-12-12&q=React%20Native,Flutter,NativeScript,Xamarin (viděno led. 20, 2020).
[20] „React Native · A framework for building native apps using React". https://facebook.github.io/react-native/ (viděno led. 13, 2020).
[21] „Flutter vs React Native — Comparing the Features of Each Framework". https://levelup.gitconnected.com/flutter-vs-react-native-comparing-the-features-of-each-framework-f61bfd146a90 (viděno bře. 10, 2020).
[22] „Supporting React Native at Pinterest - Pinterest Engineering Blog - Medium". https://medium.com/pinterest-engineering/supporting-react-native-at-pinterest-f8c2233f90e6 (viděno led. 15, 2020).
[23] „Why Discord is Sticking with React Native - Discord Blog". https://blog.discordapp.com/why-discord-is-sticking-with-react-native-ccc34be0d427 (viděno led. 15, 2020).
[24] „Support third-party 64-bit libraries on Android · Issue #2814 · facebook/react-native · GitHub". https://github.com/facebook/react-native/issues/2814 (viděno led. 18, 2020).
[25] „Releasing React Native 0.59 · React Native". http://facebook.github.io/react-native/blog/2019/03/12/releasing-react-native-059 (viděno led. 18, 2020).
[26] „Showcase - Flutter". https://flutter.dev/showcase (viděno úno. 24, 2020).
[27] „What is Flutter and Why You Should Learn It in 2020". https://www.freecodecamp.org/news/what-is-flutter-and-why-you-should-learn-it-in-2020/ (viděno úno. 24, 2020).
[28] „Blog | Flutter, Google's Disruptive Innovation". https://digiryte.com/blog/posts/flutter-google-s-disruptive-innovation (viděno bře. 10, 2020).
[29] „Reflectly — From React Native to Flutter - Reflectly Engineering - Medium". https://medium.com/reflectly-engineering/reflectly-from-react-native-to-flutter-2e3dffced2ea (viděno úno. 29, 2020).
[30] „Xamarin Customer Showcase | .NET", *Microsoft*. https://dotnet.microsoft.com/apps/xamarin/customers (viděno bře. 11, 2020).
[31] „What is Xamarin? - Xamarin | Microsoft Docs". https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin (viděno úno. 18, 2020).
[32] „We worked with Xamarin for almost three years, this is what we learned - DEV Community". https://dev.to/ljosmyndun/we-worked-with-xamarin-for-almost-three-years-this-is-what-we-learned-34ph (viděno bře. 11, 2020).
[33] „Native mobile apps with Angular, Vue.js, TypeScript, JavaScript - NativeScript". https://www.nativescript.org/ (viděno bře. 05, 2020).

[34] „Technical Overview - NativeScript Docs". https://docs.nativescript.org/angular/core-concepts/technical-overview (viděno bře. 10, 2020).
[35] T. Q. Company, „The Qt Company". https://www.qt.io/company (viděno bře. 10, 2020).
[36] „Why You Should Choose Qt For Cross-Platform App Development", *Hiring Headquarters*, srp. 01, 2017. https://www.upwork.com/hiring/for-clients/qt-cross-platform-app-development/ (viděno bře. 10, 2020).
[37] „Fuse Open | Create better native apps for iOS and Android with a new breed of development tools." https://fuseopen.com/ (viděno bře. 03, 2020).
[38] „What is Weex? | WEEX". https://weex.apache.org/guide/introduction.html#overview (viděno bře. 03, 2020).
[39] „Weex framework - fresh blood in the cross-platform development ring". https://clockwise.software/blog/weex-framework-for-cross-platform-app-development/ (viděno bře. 10, 2020).
[40] „• Apple: most popular app store categories 2019 | Statista". https://www.statista.com/statistics/270291/popular-categories-in-the-app-store/ (viděno bře. 10, 2020).
[41] „Unity Real-Time Development Platform | 3D, 2D VR & AR Visualizations". https://unity.com/ (viděno bře. 04, 2020).
[42] „Online and in-person Unity courses & training in 2D, 3D, AR, & VR development!" https://unity.com/learn (viděno bře. 04, 2020).
[43] „Unreal Engine | The most powerful real-time 3D creation platform". https://www.unrealengine.com/en-US/ (viděno bře. 04, 2020).
[44] „Unreal Engine | Frequently Asked Questions". https://www.unrealengine.com/en-US/faq (viděno bře. 04, 2020).
[45] „Corona: Free Cross-Platform 2D Game Engine". https://coronalabs.com/ (viděno bře. 04, 2020).
[46] S. INDIA, „React Native vs. Ionic vs. Flutter: Comparison of Top Cross-Platform App Development Tools", *Medium*, čvc. 05, 2019. https://codeburst.io/react-native-vs-ionic-vs-flutter-comparison-of-top-cross-platform-app-development-tools-71c8011309ac (viděno bře. 13, 2020).
[47] „Xamarin vs React Native vs Ionic vs NativeScript: Cross-platform Mobile Frameworks Comparison", *AltexSoft*. https://www.altexsoft.com/blog/engineering/xamarin-vs-react-native-vs-ionic-vs-nativescript-cross-platform-mobile-frameworks-comparison/ (viděno bře. 13, 2020).
[48] „Flutter Versus Other Mobile Development Frameworks: A UI And Performance Experiment. Part 2 - Codemagic blog custom", *Codemagic blog*, pro. 29, 2019. https://blog.codemagic.io/flutter-vs-android-ios-xamarin-reactnative/ (viděno bře. 13, 2020).
[49] „Miro: the collaborative whiteboard platform for distributed teams", *https://miro.com/*. https://miro.com/ (viděno bře. 15, 2020).
[50] „npm | build amazing things". https://www.npmjs.com/ (viděno bře. 17, 2020).
[51] „About | Node.js". https://nodejs.org/en/about/ (viděno bře. 17, 2020).

[52] „An Introduction to JavaScript". https://javascript.info/intro (viděno led. 18, 2020).
[53] „index | TIOBE - The Software Quality Company". https://www.tiobe.com/tiobe-index/ (viděno úno. 17, 2020).
[54] „The Fall and Rise of Dart, Google's ‚JavaScript Killer´". https://insights.dice.com/2019/03/27/fall-rise-dart-google-javascript-killer/ (viděno bře. 01, 2020).
[55] „Why Flutter uses Dart?|Relationship between Flutter and Dart". https://kodytechnolab.com/why-flutter-uses-dart (viděno dub. 24, 2020).
[56] „Dart programming language | Dart". https://dart.dev/ (viděno bře. 01, 2020).
[57] „About Mono | Mono". https://www.mono-project.com/docs/about-mono/ (viděno dub. 01, 2020).
[58] „The history of C# - C# Guide | Microsoft Docs". https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-version-history (viděno úno. 17, 2020).
[59] „Node.js". https://nodejs.org/en/ (viděno bře. 17, 2020).
[60] „Learn the Basics · React Native". https://facebook.github.io/react-native/docs/tutorial (viděno led. 18, 2020).
[61] „Debugging · React Native". https://reactnative.dev/ (viděno bře. 16, 2020).
[62] „Using List Views · React Native". https://reactnative.dev/docs/using-a-listview (viděno bře. 17, 2020).
[63] „React.Component – React". https://reactjs.org/docs/react-component.html#componentdidmount (viděno bře. 19, 2020).
[64] „Introduction to widgets - Flutter". https://flutter.dev/docs/development/ui/widgets-intro (viděno úno. 29, 2020).
[65] „Windows install". https://flutter.dev/docs/get-started/install/windows (viděno bře. 24, 2020).
[66] profexorgeek, „What is Xamarin.Forms? - Xamarin". https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin-forms (viděno bře. 11, 2020).
[67] davidbritch, „Xamarin.Forms StackLayout Tutorial - Xamarin". https://docs.microsoft.com/en-us/xamarin/get-started/tutorials/stacklayout/ (viděno bře. 11, 2020).

*Zadání práce*

Univerzita Hradec Králové
Fakulta informatiky a managementu

# Zadání diplomové práce

| | |
|---|---|
| **Autor:** | **Bc. Jakub Beneš** |
| Studium: | I1700315 |
| Studijní program: | N1802 Aplikovaná informatika |
| Studijní obor: | Aplikovaná informatika |
| **Název diplomové práce:** | **Porovnání mobilních frameworků** |
| Název diplomové práce AJ: | Comparison of mobile frameworks |

**Cíl, metody, literatura, předpoklady:**

doplnit v zari Seznamte se s frameworky pro tvorbu mobilních aplikací. Porovnejte přístupy ve vývoji ve vybraných (alespoň 2) frameworcích. Porovnejte možnosti frameworků z pohledu jejich využitelnosti napříč operačními systémy. Navrhněte vzorovou aplikaci demonstrující práci s mobilními aplikacemi. Vzorovou aplikaci implementujte ve vybraných frameworcích. Porovnejte práci s vybranými frameworky a jejich možnosti z pohladu vývoje v rámci vzorové aplikace.

doplnit v zari

| | |
|---|---|
| Garantující pracoviště: | Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu |
| Vedoucí práce: | doc. Ing. Filip Malý, Ph.D. |
| Datum zadání závěrečné práce: | 14.1.2018 |