

# Vývoj aplikací pro Universal Windows Platform

Martin Bojnanský

---

Bakalářská práce  
2016



Univerzita Tomáše Bati ve Zlíně  
Fakulta aplikované informatiky

---

# **ZADÁNÍ BAKALÁŘSKÉ PRÁCE**

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Martin Bojnanský**

Osobní číslo: **A13143**

Studijní program: **B3902 Inženýrská informatika**

Studijní obor: **Informační a řídicí technologie**

Forma studia: **prezenční**

Téma práce: **Vývoj aplikací pro Universal Windows Platform**

Téma anglicky: **Application Development for Universal Windows Platform**

## **Zásady pro vypracování:**

- 1. Vypracujte literární rešerši na téma vývoj aplikací pro Universal Windows Platform.**
- 2. Popište jazyky XAML a C#.**
- 3. Analyzujte a demonstруйте základní úlohy a techniky z oblasti vývoje aplikací pro danou platformu.**
- 4. Analyzujte a demonstруйте techniky přizpůsobení aplikací různým aspektům zařízení Windows 10.**
- 5. Demonstруйте výsledky.**

Rozsah bakalářské práce: -  
Rozsah příloh: -  
Forma zpracování bakalářské práce: tištěná/elektronická

Seznam odborné literatury:

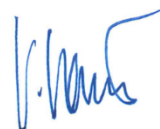
1. MOEMEKA, Edward a Elizabeth MOEMEKA. Real World Windows 10 Development. 2. ed. New York: Apress Media LLC, 2015. ISBN 978-1-4842-1450-3.
2. SHARP, John. Microsoft Visual C# Step by Step, 8th Edition. 8. ed. Redmond, Washington: Microsoft Press, 2015, pages cm. ISBN 9781509301041.
3. PETZOLD, Charles. Mistrovství ve Windows Presentation Foundation: [aplikace = kód + markup]. Vyd. 1. Brno: Computer Press, 2008, 928 s. Mistrovství. ISBN 978-80-251-2141-2.
4. Microsoft Developer Network [online]. [cit. 2016-01-24]. Dostupné z: <https://msdn.microsoft.com/>
5. Microsoft Virtual Academy [online]. [cit. 2016-01-24]. Dostupné z: <https://mva.microsoft.com/>
6. Channel 9 [online]. [cit. 2016-01-24]. Dostupné z: <https://channel9.msdn.com>

Vedoucí bakalářské práce: Ing. Erik Král, Ph.D.  
Ústav počítačových a komunikačních systémů  
Datum zadání bakalářské práce: 19. února 2016  
Termín odevzdání bakalářské práce: 27. května 2016

Ve Zlíně dne 19. února 2016



doc. Mgr. Milan Adámek, Ph.D.  
děkan



prof. Ing. Vladimír Vašek, CSc.  
ředitel ústavu

### Prohlašuji, že

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

### Prohlašuji,

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

Ve Zlíně, dne 23.5.2016

.....  
podpis diplomanta

## **ABSTRAKT**

Bakalárska práca sa zaoberá vývojom aplikácii pre Universal Windows Platform. V práci sú diskutované možnosti vývoja aplikácie pre viacero zariadení ako sú mobilné telefóny, tablety, stolné počítače, zariadenie IoT a iné. Dôraz je kladený na techniky a technológie prispôsobenia používateľského rozhrania aplikácie rôznym veľkostiam obrazoviek a spôsobom interakcií. Súčasťou práce je univerzálna aplikácia pre vzdialenú obsluhu internetového rádia v domácnosti.

Kľúčové slová: Universal Windows Platform, .NET, C#, XAML, Windows 10, vývoj aplikácii, adaptívne používateľské rozhranie

## **ABSTRACT**

The Bachelor's thesis is focused on Universal Windows Platform Application Development. The thesis discusses the application development options for multiple devices, such as smartphones, tablets, desktop computers, IoT devices, and others. It also emphasizes techniques and technologies for the user-interface customization according to different screen resolutions and forms of inputs. The second part of the thesis is a sample universal application for the remote control of a household's internet radio.

Keywords: Universal Windows Platform, .NET, C#, XAML, Windows 10, application development, adaptive user interface

Touto formou by som rád poďakoval vedúcemu práce Ing. et Ing. Erikovi Královi, Ph.D. za odborné vedenie, cenné rady a konzultácie bakalárskej práce.

Taktiež ďakujem svojej rodine za podporu a možnosť štúdia na vysokej škole.

## OBSAH

<b>ÚVOD.....</b>	<b>9</b>
<b>I. TEORETICKÁ ČASŤ .....</b>	<b>10</b>
<b>1 OPERAČNÝ SYSTÉM WINDOWS 10 .....</b>	<b>11</b>
1.1 ZMENY V POUŽÍVATELSKOM ROZHRAŇÍ .....	11
1.2 ROZŠÍRENIE FUNKCIONALÍT SYSTÉMU .....	14
<b>2 UNIVERSAL WINDOWS PLATFORM.....</b>	<b>16</b>
2.1 HISTÓRIA UWP .....	16
2.2 RODINY ZARIADENÍ .....	17
2.3 .NET.....	18
2.3.1 .NET CORE .....	18
2.3.2 .NET NATIVE .....	19
2.4 TECHNOLÓGIE VÝVOJA .....	20
<b>3 PROGRAMOVACÍ JAZYK C# .....</b>	<b>22</b>
3.1 NOVINKY VO VERZII JAZYKA C# 6.0.....	22
<b>4 JAZYK XAML .....</b>	<b>25</b>
4.1 NOVINKY JAZYKA XAML.....	25
<b>5 TVORBA UWP APLIKÁCIÍ V JAZYKOCH C# A XAML .....</b>	<b>28</b>
5.1 ZÁKLADNÉ TYPY PROJEKTOV .....	28
5.2 ŠTRUKTÚRA PROJEKTU BLANK APP .....	29
5.3 ŽIVOTNÝ CYKLUS APLIKÁCIE.....	30
5.3.1 OBSLUHA ŽIVOTNÉHO CYKLU.....	31
5.4 ADAPTÍVNE POUŽÍVATEĽSKÉ ROZHRAŇIE .....	31
5.4.1 EFEKTÍVNE PIXELY A ŠKÁLOVANIE VEĽKOSTÍ .....	32
5.4.2 ADAPTÍVNY DIZAJN .....	32
5.4.3 DIZAJN NA MIERU .....	35
5.5 ADAPTÍVNY KÓD.....	36
<b>II. PRAKTICKÁ ČASŤ.....</b>	<b>38</b>
<b>6 INTERNETOVÉ RADIO .....</b>	<b>39</b>
6.1 CIELE .....	39
6.1.1 FUNKCIONÁLNE POŽIADAVKY .....	39
6.1.2 NE-FUNKCIONÁLNE POŽIADAVKY .....	39
6.1.3 ARCHITEKTONICKÉ POŽIADAVKY .....	40
<b>7 ARCHITEKTÚRA APLIKÁCIE .....</b>	<b>41</b>
<b>8 SIEŤOVÁ KOMUNIKÁCIA MEDZI APLIKÁCIAMI.....</b>	<b>42</b>
8.1 STREAMSOCKET SERVER .....	43
8.2 STREAMSOCKET KLIENT .....	45
8.3 SERIALIZÁCIA DÁT .....	47
<b>9 WINDOWS 10 IOT CORE APLIKÁCIA.....</b>	<b>49</b>
9.1 ŠTRUKTÚRA PROJEKTU.....	49
9.2 PRÁCA S ÚLOŽISKOM - SPRÁVA RÁDIOVÝCH STANÍC.....	50

9.3	OBSLUHA PREHRÁVANIA RÁDIOVÝCH STANÍC.....	51
<b>10</b>	<b>KLIENTSKÁ APLIKÁCIA INTERNETOVÉ RÁDIO .....</b>	<b>52</b>
10.1	ARCHITEKTÚRA PROJEKTU .....	52
10.1.1	MODEL-VIEW-VIEWMODEL (MVVM) .....	52
10.1.2	APLIKAČNÝ DIAGRAM .....	53
10.1.3	VIAZANIE DÁT A UDALOSTÍ .....	54
10.2	NAVIGAČNÁ SLUŽBA.....	55
10.3	TVORBA POUŽÍVATEĽSKÉHO ROZHRANIA .....	56
10.3.1	ZOBRAZENIE S NAVIGAČNOU SEKCIOU .....	56
10.3.2	ZMENA ZOSKUPENIA ZOBRAZENÍ.....	58
10.3.3	PRISPÔSOBENIE ZOBRAZENIA PRI ZMENE FORMY INTERAKCIE .....	59
	<b>ZÁVER .....</b>	<b>61</b>
	<b>ZOZNAM POUŽITEJ LITERATÚRY .....</b>	<b>63</b>
	<b>ZOZNAM POUŽITÝCH SYMBOLOV A SKRATIEK.....</b>	<b>66</b>
	<b>ZOZNAM OBRÁZKOV .....</b>	<b>67</b>
	<b>ZOZNAM ZDROJOVÝCH KÓDOV .....</b>	<b>68</b>
	<b>ZOZNAM TABULIEK .....</b>	<b>70</b>
	<b>ZOZNAM PRÍLOH.....</b>	<b>71</b>



## ÚVOD

Technológie modernej doby čoraz viac ovplyvňujú naše životy a stávajú sa ich neodmysliteľnou súčasťou. Osobné počítače, mobilné telefóny, tablety alebo iné zariadenia 21. storočia sa s pribúdajúcim časom tešia stále väčšej popularite. Táto skutočnosť je podmienená ich praktickým prínosom do osobného a pracovného života. Prítomnosť a vývoj týchto technológií nám umožňuje spájať sa s blízkymi, pracovať efektívnejšie a spoľahlivejšie, byť informovanejšími ako kedykoľvek predtým alebo nás jednoducho zabávať.

Význam a využitie týchto zariadení sa neustále približuje, a preto sa tvorcovia ich operačných systémov čoraz viac zaoberajú otázkou jednotnosti z pohľadu používateľov systému a vývojárov aplikácií. Ich snaha ponúka viacero odlišných prístupov k tomuto problému. Jedna z najväčších spoločností tejto oblasti, Microsoft, sa rozhodla zjednotiť ich operačné systémy pre rôzne platformy vytvorením spoločného jadra systému. Ten je v prípade rôznych zariadení rozširovaný o špecifické funkcie jednej z kategórií, do ktorej spadá.

Cieľom bakalárskej práce je analyzovať možnosti vývoja univerzálnych aplikácií pre túto platformu. Ďalej bude práca ponúkať pohľad na využitie špecifických funkcií kategórií zariadení a oboznamovať čitateľa o možnostiach prispôsobenia aplikácii týmto kategóriám, rôznym veľkostiam obrazoviek a typom spôsobu interakcie s nimi. Súčasťou práce bude vytvorenie ukážkovej aplikácie internetového rádia pre Universal Windows Platform. Univerzálna aplikácia bude slúžiť k vzdialenej obsluhu prehrávania rádiových staníc na zariadení Raspberry Pi.

## **I. TEORETICKÁ ČASŤ**

## 1 OPERAČNÝ SYSTÉM WINDOWS 10

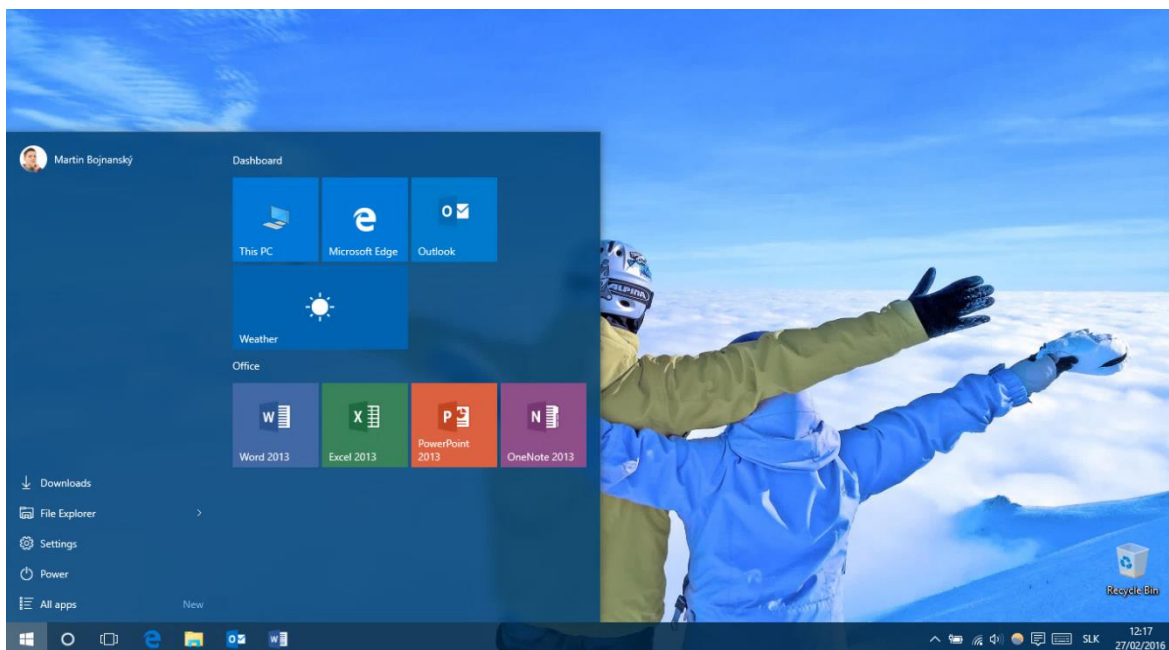
Dňa 29.7.2015 spoločnosť Microsoft uviedla na trh nový operačný systém s označením Windows 10 [1]. Podobne ako jeho predchodcovia je operačný systém založený na jadre Windows NT [2]. Jeho príchod prináša veľké množstvo zmien, technologických a dizajnových novín pre používateľov a vývojárov Windows aplikácií. Windows 10 predstavuje aplikačnú platformu Universal Windows Platform, nové bezpečnostné prvky a vylepšenia, osobnú asistentku Cortana a mnoho iných novín. [3]

### 1.1 Zmeny v používateľskom rozhraní

V rámci OS Windows 8 boli realizované jedny z najväčších zmien v používateľskom rozhraní už od vzniku Windows 95 [3]. Napriek modernému prístupu k návrhu rozhrania sa spoločnosť Microsoft dočkala kritiky a nepochopeniu zo strany používateľov [4]. Tlačidlo Štart, ktoré sa stalo ikonické pre systém Windows už od verzie 95, bolo odstránené z panelu úloh a ponuka Štart bola rozšírená na celú obrazovku so živými dlaždicami. Avšak zmenou klasickej ponuky sa pre bežných používateľov stalo zložitejšie pristupovať k vypínaniu, nastaveniam alebo pracovnej ploche počítača. Vo verzii OS Windows 8 boli taktiež po prvýkrát predstavené moderné aplikácie, ktorých zobrazenie už nebolo možné v prispôsobiteľných oknách, ale jedine v celoobrazovkovom alebo rozdelenom režime aplikácii. Pri tvorbe nového rozhrania sa spoločnosť rozhodla využiť konštruktívnu kritiku používateľov a k návrhu rozhrania pristúpiť odlišným spôsobom. [5]

#### Navigačná ponuka Štart

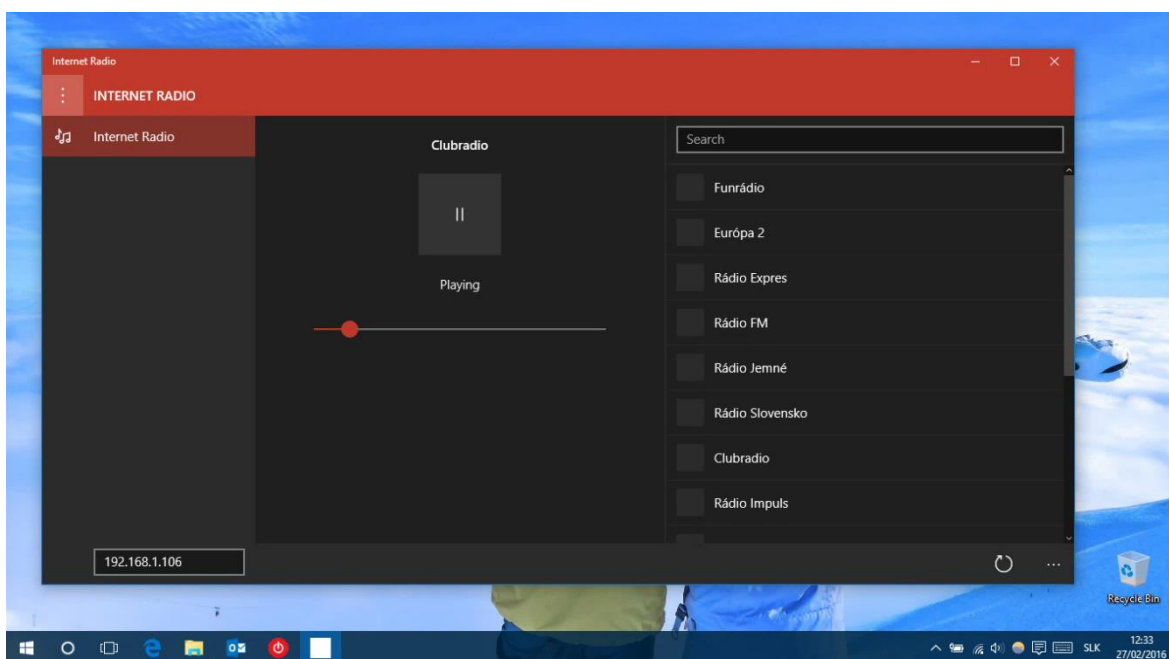
Windows 10 prichádza späť s ponukou Štart a jej tlačidlom v ľavom dolnom rohu obrazovky. Táto ponuka vznikla kombináciou klasickej ponuky z verzie Windows 7 a modernej ponuky z verzie Windows 8. Ponuku je možné rozšíriť o živé dlaždice alebo využiť zobrazenie v Tablet režime, ktoré je podobné celoobrazovkovej ponuke Štart predstavenej vo verzii Windows 8.



*Obrázok 1 Ponuka Štart*

### **Zobrazenie moderných aplikácií v oknách**

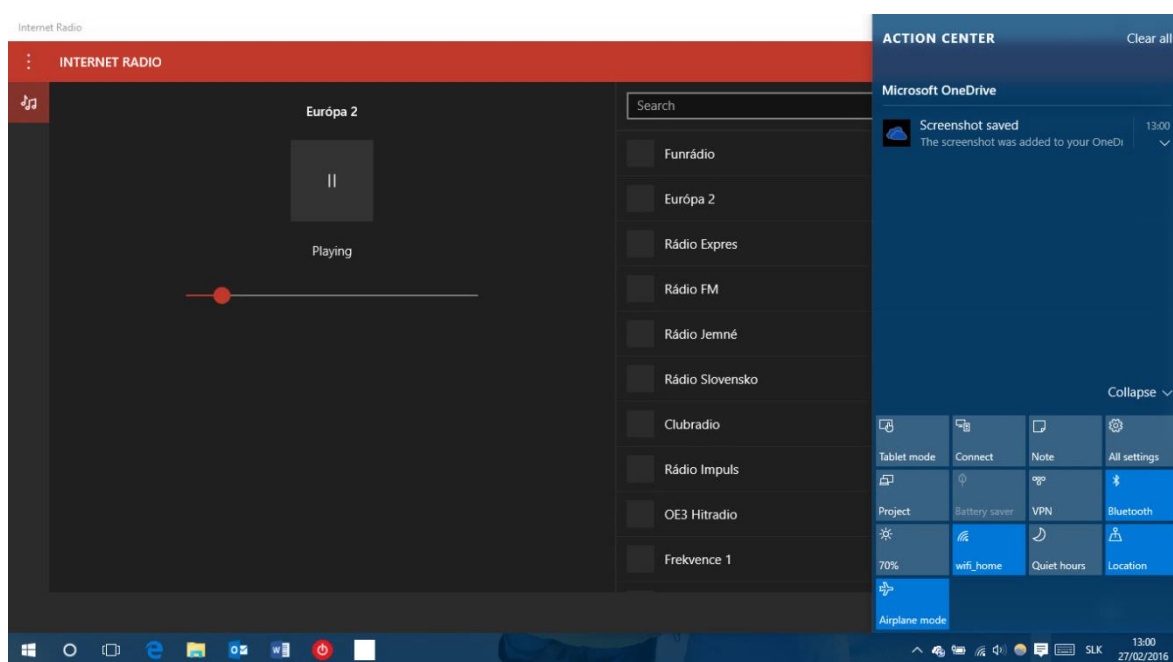
Ďalšou výraznou zmenou je zobrazenie moderných aplikácií v prispôsobiteľných oknách. V klasickom režime je veľkosť okna aplikácie plne adaptívna, na čo je dôležité myslieť už pri samotnom návrhu používateľského rozhrania. V režime Tablet môže byť aplikácia zobrazená na celú obrazovku alebo môže zdieľať jej časť s druhou aplikáciou, podobným spôsobom ako v predchádzajúcej verzii Windows 8.1.



*Obrázok 2 Zobrazenie modernej aplikácie v adaptívnom okne*

## Centrum akcií

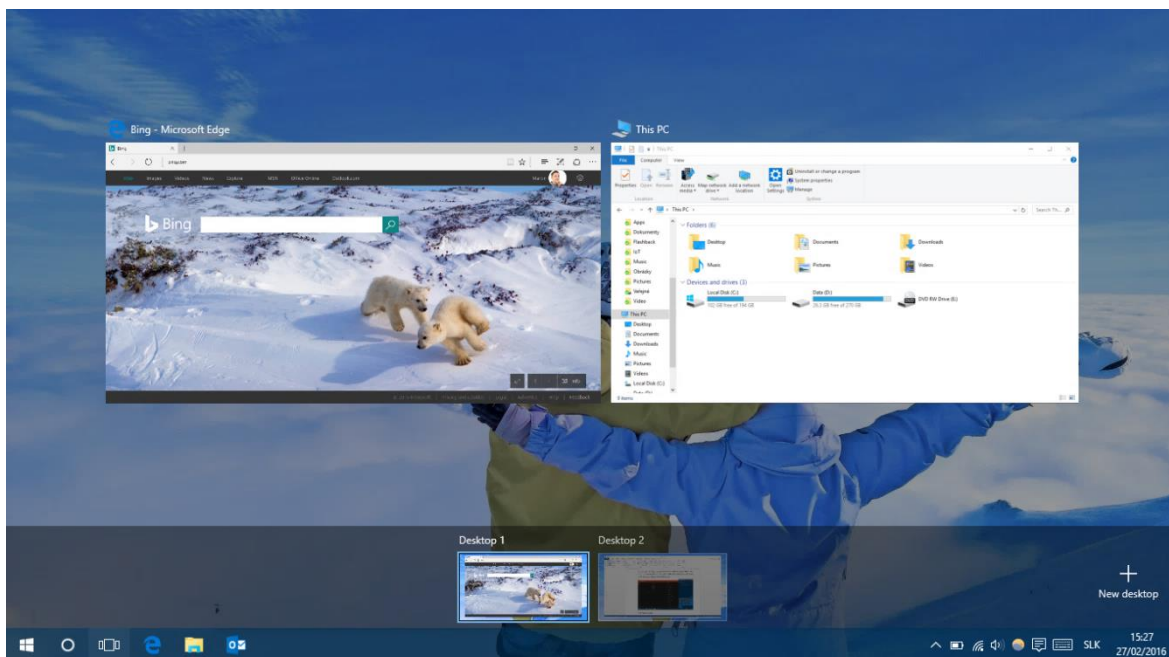
Do rozhrania pribudlo centrum akcií, v ktorom nájdeme oznámenia z aplikácií a systému, skratky pre rýchle nastavenia alebo bežné úlohy. Štvorica z týchto skratiek je prispôsobiteľná používateľom v nastaveniach systému a ich celková ponuka závisí od zariadenia a jeho možností [3]. Oznámenia sú zoskupované podľa aplikácií, ktoré ich vytvorili a po kliknutí na oznámenie alebo ich skupinu sa vyvolá príslušná aplikácia a jej odpovedajúca úloha. Centrum akcií zobrazíme ikonou v pravom dolnom rohu panelu úloh alebo klávesovou skratkou *Windows + A*.



Obrázok 3 Centrum akcií

## Virtuálne pracovné plochy

Medzi ďalšie novinky patrí možnosť vytvorenia viacerých pracovných plôch, medzi ktorými je možné prepínať, a tak si úlohy (aplikácie) oddeliť podľa vlastného uváženia [3]. Pre jednoduchší prístup pribudlo na panely úloh nové tlačidlo, pretože k tomuto zobrazeniu sa v predošlých verziách pristupovalo jedine pomocou klávesovej skraty *Windows + Tab*. V jeho spodnej časti sú umiestnené náhľady aktuálne vytvorených pracovných plôch a vyššie je situovaný prehľad aktuálnych úloh aktívnej pracovnej plochy. Pre vytvorenie plochy je k dispozícii tlačidlo v pravej dolnej časti zobrazenia alebo klávesová skratka *CTRL + Windows + D*. Pre rýchlejší a jednoduchší pohyb medzi plochami existuje alternatíva v podobe klávesovej skratky *CTRL + Windows + Vľavo/Vpravo*.

*Obrázok 4 Zobrazenie úloh*

## 1.2 Rozšírenie funkcionalít systému

### Univerzálne aplikácie

Pretože v dnešnom svete sme obklopaní stále väčším množstvom rôznych zariadení, ktorých použitie je široké, spoločnosť Microsoft sa pri návrhu a vývoji OS Windows 10 rozhodla vytvoriť čo najjednoduchší systém pre všetky bežné zariadenia. V rámci tejto ideológie Windows 10 predstavuje univerzálne aplikácie, ktoré je možné spustiť na zariadeniach typu osobný počítač, tablet, mobilný telefón, herná konzola XBOX, zariadenia kategórie Internet vecí, SurfaceHub, HoloLens a iných. Aplikácie disponujú možnosťami prispôbovať sa rôznym veľkostiam obrazoviek, vstupným zariadeniam a iným špecifickým faktorom jednotlivých zariadení ich kategórií. [1], [5]

### Účet Microsoft

Od verzie Windows 8 je možné pre prihlásenie do systému namiesto lokálneho účtu využiť účet Microsoft. Jeho výhodou je zdieľanie nastavení ako sú napr. téma, nastavenia aplikácií, údaje webového prehliadania alebo hesiel medzi všetkými používateľovými zariadeniami Windows. Pokiaľ je systém preinštalovaný alebo aktivovaný na inom zariadení, všetky tieto nastavenia je možné jednoduchým spôsobom preniesť z údajov uchovaných v osobnom internetovom úložisku OneDrive. [3]

## **Cortana**

Súčasťou Windows 10 je aj osobná asistentka Cortana, ktorá sa po prvýkrát objavila v operačnom systéme Windows Phone 8.1. Cortana inteligentne kombinuje vyhľadávanie v úložisku počítača a na internete, a zároveň uľahčuje množstvo bežných úloh na základe hlasových alebo textových inštrukcií. Osobná asistentka ponúka vytváranie pripomienok podmienených na čas, polohu alebo kontakte, upozorňuje, kedy je vhodné vyraziť na stretnutie alebo či zakúpený letecký spoj aktuálne mešká. Spomínané funkcie sú len malým množstvom z toho, čo asistentka reálne poskytuje. Zoznam rozpoznávaných inštrukcií je kedykoľvek dostupný prostredníctvom povelu: „What can I say?“. Vývojári univerzálnych aplikácií ocenia najmä možnosť implementácie vlastných inštrukcií a prepojenia asistentky s vlastnou aplikáciou. [3]

## **Windows Hello**

Po klasických možnostiach prihlásenia a overenia používateľa systému Windows, ako je heslo a kód PIN, pribudla služba Windows Hello využívajúca biometrické údaje. Súčasťou tejto služby sú možnosti prihlásenia pomocou rozpoznávania tváre, dúhovky alebo odtlačku prsta. Táto funkcia však vyžaduje špecifický hardware ako je infračervená kamera, a preto sa nemusí objaviť na všetkých zariadeniach. [3]

## **Aktualizácie**

Veľká zmena nastala aj v oblasti aktualizácií systému. Tie boli mnohokrát odkladané do vydania nového servisného balíčku alebo vyššej verzie systému, čo spôsobovalo rozsiahle a nie veľmi časté aktualizácie [2]. Kvôli tomu sa tvorcovia rozhodli, že aktualizácie budú prebiehať prostredníctvom služby Windows Update, a všetky nové funkcie budú do systému pridávané a spravované častejšie a v menších aktualizáciách ako tomu bolo doteraz. [3]

## 2 UNIVERSAL WINDOWS PLATFORM

Myšlienka univerzálnych aplikácií, ktoré je možné spustiť na zariadeniach Windows s rôznymi rozlíšeniami a vstupnými interakciami, je realizovaná vytvorením spoločnej aplikačnej platformy nazývanej Universal Windows Platform (UWP). Táto platforma je súčasťou zjednoteného jadra OS Windows 10 a je dostupná na všetkých jeho zariadeniach. UWP je označovaná viacerými termínmi ako aplikačný model, služba a platforma najmä preto, že umožňuje kompilovať a spúšťať aplikácie na zariadeniach Windows 10. Návrh tejto platformy pomáha cieľiť jeden aplikačný balík viacerým kategóriám zariadení a to prostredníctvom jediného spoločného obchodu s aplikáciami (Windows Store). [6]



Obrázok 5 Cielenie univerzálnych aplikácií [6]

### 2.1 História UWP

Pretože jadrá operačných systémov Windows, Windows Phone 7.5 a XBOX 360 boli odlišné, prvým veľmi dôležitým krokom na ceste vzniku konvergentnej<sup>1</sup> platformy bolo použitie jednotného jadra systému. V novšej verzii OS Windows Phone 8 bolo jadro Windows CE nahradené za Windows NT, a rovnako tak sa stalo aj v prípade OS pre XBOX One. O rok neskôr, s príchodom Windows 8.1, bol predstavený nový aplikačný model nazývaný Windows Runtime (Windows RT) a predstavoval spoločnú vrstvu sady rozhraní programovania aplikácií (API<sup>2</sup>) pre operačné systémy Windows 8.1 a Windows Phone 8.1. Táto skutočnosť umožnila zdieľať veľké množstvo kódu medzi aplikáciami cieľenými na

<sup>1</sup> Termín označujúci zbiehavosť alebo približujúci sa charakter.

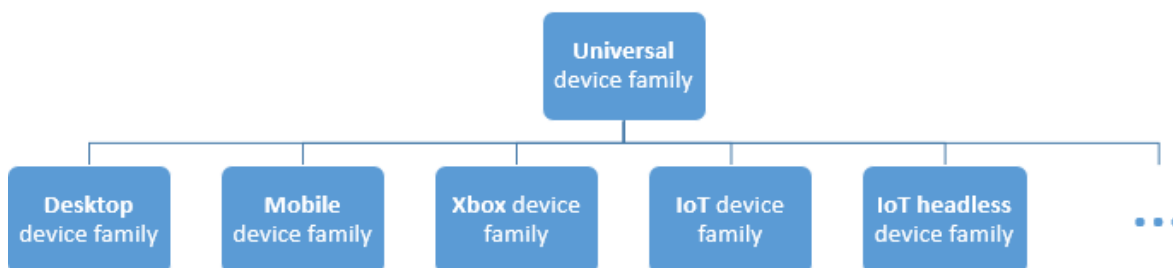
<sup>2</sup> Súhrn metód a funkcií, umožňujúcich interakciu aplikácie a operačného systému.



dva odlišné operačné systémy pomocou tretieho zdieľaného projektu a kompilačných direktív<sup>3</sup>. [2]

## 2.2 Rodiny zariadení

UWP vznikla rozšírením aplikačného modelu Windows RT tak, aby aplikácie už neboli obmedzené len na určitú vrstvu sady rozhraní API spoločnej pre všetky zariadenia, ale mohli pristupovať aj k jeho špecifickým rozhraniam a využiť všetky jeho schopnosti. Zariadenia sú kategorizované do rodín zariadení podľa ich vlastností, využitia a hardwarových možností. Pre tieto rodiny zariadení existujú rozšírenia v podobe rozhraní API a obsahujú charakteristické funkcionality. To znamená, že aplikácia využívajúca len jadro UWP bude poskytovať rovnakú funkcionality na každom jednom zariadení, avšak ak využijeme špecifickej vlastnosti ako napr. poslanie SMS správy, tak táto funkcia bude prístupná len na zariadeniach patriacich do rodiny mobilných zariadení. Z tohto dôvodu aplikácie už nie sú cielené pre operačný systém, ale pre ľubovoľný počet rodín zariadení. [6]



Obrázok 6 Hierarchia rodín zariadení [6]

Rodina zariadení predstavuje sadu rozhraní API spoločnú pre zariadenia patriace do tejto kategórie a tvorí základ OS daného zariadenia. Pretože väčšina rozhraní API je spoločná pre všetky rodiny zariadení, každá z nich dedí spoločnú sadu od Univerzálnej rodiny zariadení a rozširuje ju o špecifické rozhrania. Výhodou týchto rodín je možnosť vytvoriť aplikáciu na základe garantovanej sady rozhraní API a pomocou podmienok v kóde kontrolovať a využívať prítomnosť určitého rozhrania špecifického pre niektoré zo zariadení. [6]

<sup>3</sup> Kódy, ktoré slúžia ako podmienky prekladaču.

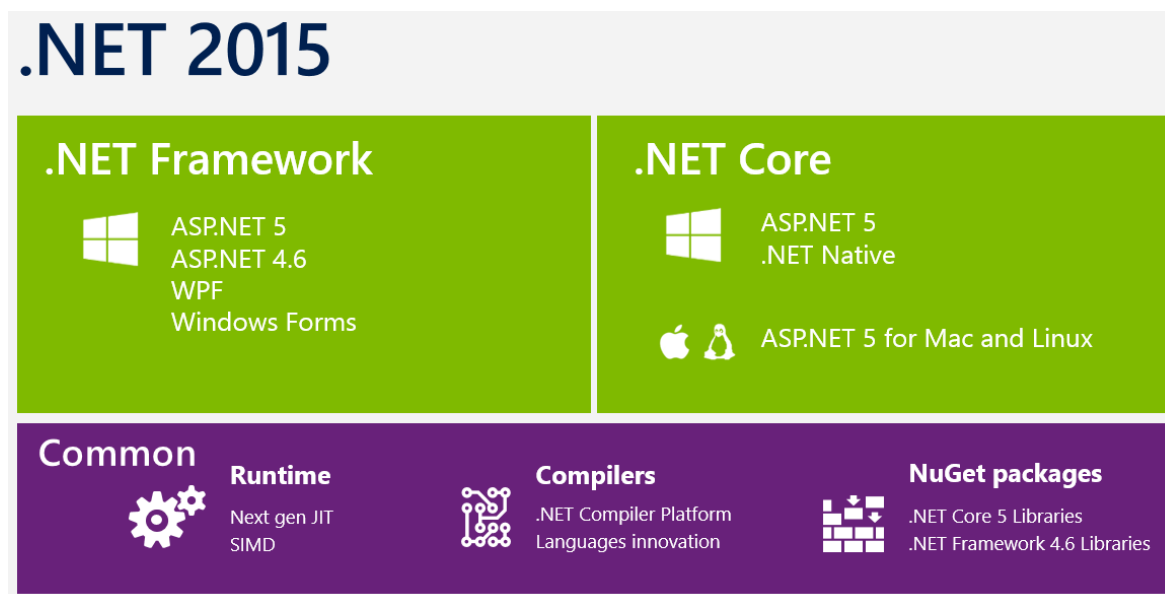
Storage	Networking	NFC & Bluetooth	Appointments / Calendar	Maps & Location
Data.XML	XAML	Inking	DirectX 12	Holographic
App to App Services	Voice recognition & Cortana	Authentication Broker	Background Transfer	Sensors
Tiles & Notifications	Background Tasks	Data Roaming	Media Casting	and more..

*Obrázok 7 Vybraný obsah univerzálnej sady rozhraní API*

## 2.3 .NET

### 2.3.1 .NET Core

Aplikačný rámec (framework) .NET Core 1.0 vznikol ako menšia modulárna verzia klasického rámca .NET Framework 4.6. Jeho architektúra je flexibilná tak, aby našiel uplatnenie na rôznych miestach. Predstavuje podmnožinu kompletného rámca .NET Framework a je prenositeľný medzi platformami Windows, Linux a Mac OS. Súčasťou tejto podmnožiny nie sú najmä silne závislé sady rozhraní na platforme Windows, medzi ktoré patrí napr. WPF, Windows Forms, ADO.NET a podobne. Na rozdiel od klasického aplikačného rámca je licencovaný ako otvorený softvér (open-source) a je dostupný na serveri GitHub pre prispievateľov z komunity. Veľmi podstatnou zmenou je aj fakt, že aplikácia nie je viac závislá na verzii aplikačného rámca nainštalovaného v OS, ale potrebná a využívaná časť sa stáva jeho súčasťou v podobe NuGet balíčkov. K dispozícii je ale viacero možností vrátane zdieľaného komponentu pre viaceré aplikácie. [7], [8]



Obrázok 8 Vetvenie aplikačného rámca .NET v roku 2015 [9]

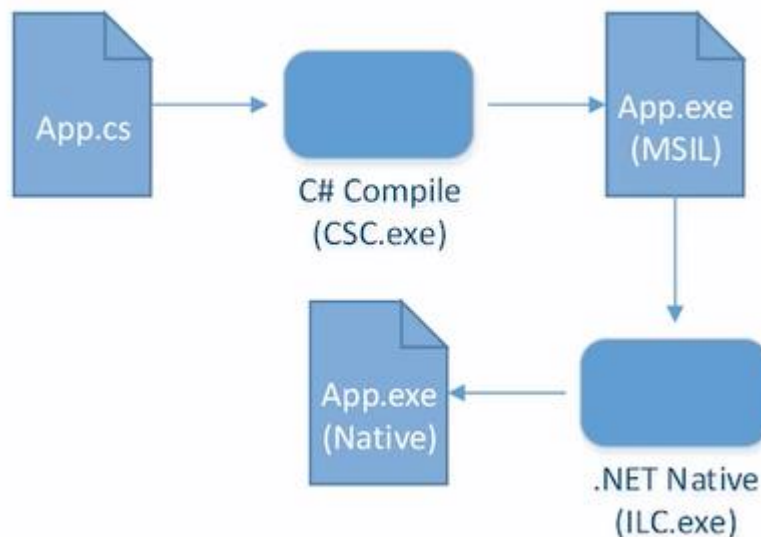
### 2.3.2 .NET Native

.NET Native predstavuje špeciálnu optimalizovanú implementáciu (variantu) .NET Core [7]. Táto implementácia bola vyvinutá za účelom zvýšenia výkonu aplikácií a vďaka nej je možné dosiahnuť výkonu natívneho kódu C++ aj pri implementácii kódu v jazyku C#. Namiesto JIT (Just-in-time) kompilácie sa jedná o kompiláciu statickú, kedy je aplikačný kód implementovaný pomocou jazyku C# priamo prekladaný na strojový kód. Pre tento preklad kompilátor využíva Visual C++ optimalizátor. Napriek absencii JIT kompilácie zostáva písanie aplikačného kódu platformovo nezávislé. Jedinou zmenou je nutnosť tvorby aplikačných balíčkov pre každú architektúru zvlášť. [10], [11]

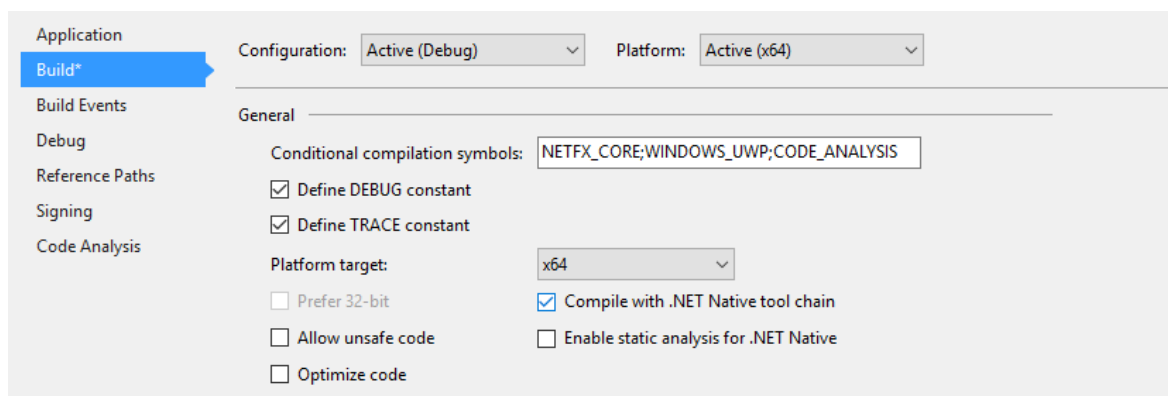
Statické kompilovanie sa prevádza ako druhý krok hneď po klasickej kompilácii programu do jazyku MSIL<sup>4</sup>. Spustiteľný súbor v jazyku MSIL je ďalej kompilovaný cez C++ optimalizátor do natívneho kódu. Počas svojho behu aplikácia už viac nevyužíva klasický aplikačný rámec, ale nový a menší nazývaný CLR, ktorý je súčasťou balíčku aplikácie. Rovnako ako pri .NET Core, aplikácia nie je závislá na verzii aplikačného rámca nainštalovaného v systéme. Aplikácie kompilované pomocou .NET Native vykazujú o mnoho rýchlejší studený aj teplý štart, ako aj menšie nároky na pamäť procesu, pretože nie je nutné načítať celý aplikačný rámec .NET Framework do jeho pamäte. Visual Studio

<sup>4</sup> MSIL – Microsoft Intermediate Language

automaticky prevádza obidva kroky kompilácie pokiaľ je v nastaveniach projektu aktivovaná možnosť kompilovania pomocou reťazového nástroja .NET Native. [10]



Obrázok 9 Proces statickej kompilácie pomocou .NET Native [10]

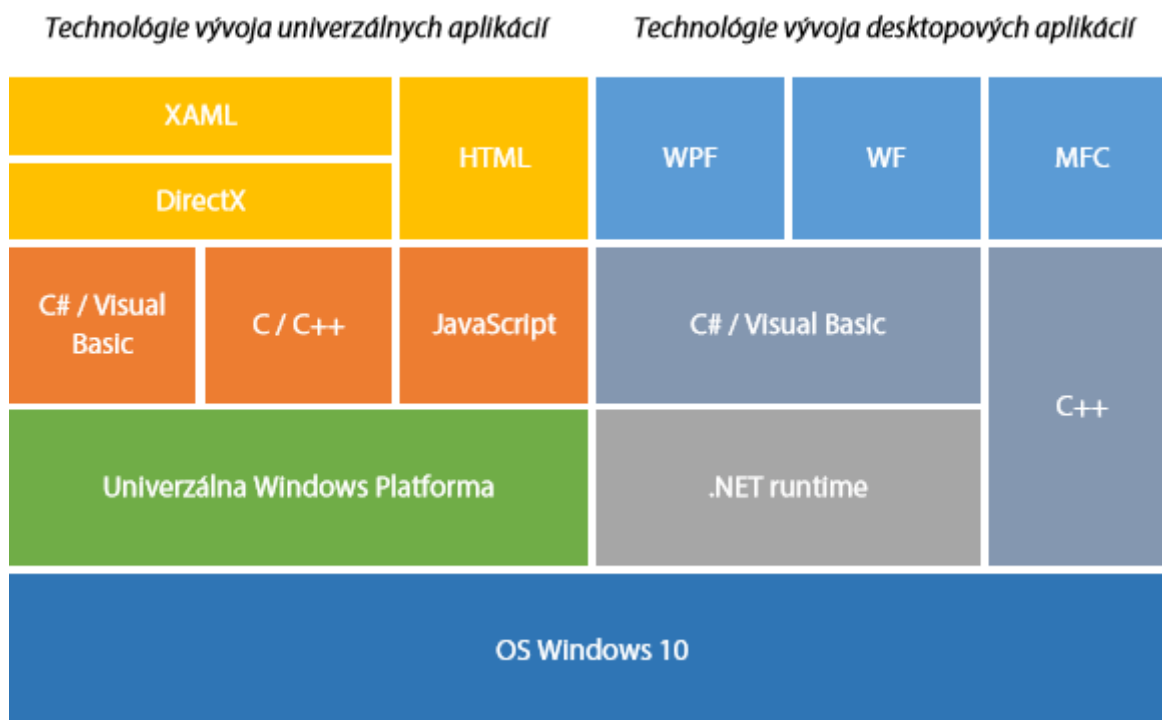


Obrázok 10 Nastavenie kompilácie pomocou reťazového nástroja .NET Native

## 2.4 Technológie vývoja

Pre vývoj univerzálnych aplikácií existujú viaceré alternatívy jazykov aplikačnej logiky a návrhu používateľského rozhrania. Jednou z možností je značkový jazyk XAML alebo DirectX pre návrh používateľského rozhrania a v kombinácii s jazykom C#, Visual Basic alebo C++ implementovať aplikačnú logiku. Druhou možnosťou je tvorba rozhrania za pomoci štandardov HTML5 / CSS3 a aplikačnej logiky v jazyku JavaScript. Všetky technológie vývoja poskytujú plnohodnotný prístup k rozhraniam API platformy, a tak

neobmedzujú vývojárov k čisto jednému natívnemu jazyku. Navyše aplikácia môže používať knižnice a komponenty napísané v inom jazyku podporovanom platformou UWP. Okrem vývoja univerzálnych aplikácií pre Windows naďalej existuje možnosť vývoja desktop aplikácií, tie avšak neposkytujú multiplatformové benefity ako aplikácie univerzálne. Spoločnosť Microsoft pripravuje aj alternatívy premostenia iOS, webových a klasických desktop aplikácií na aplikácie UWP. [2]



Obrázok 11 Technológie vývoja aplikácií pre Windows 10

### 3 PROGRAMOVACÍ JAZYK C#

Microsoft Visual C# je moderný objektovo a komponentovo orientovaný programovací jazyk primárne určený vývojárom aplikácií pre rámec .NET Framework. V jazyku C# môžeme vytvárať klientské a serverové aplikácie, webové služby, databázové systémy a mnoho ďalších [12]. Jedná sa o jednoduchý a výkonný jazyk, ktorý zdedil množstvo vlastností z jazykov C++ a Visual Basic. Syntax je odvodená z jazykov C a C++, pričom disponuje množstvom podobností s jazykom Java. Na rozdiel od väčšiny programovacích jazykov C# nemá svoje vlastné prostredie behu, ale je závislý na knižniciach rámca .NET Framework. C# podporuje koncepty, konštrukcie a prvky bežných moderných objektovo orientovaných jazykov, medzi ktoré patria napr. dedičnosť, mnohotvárnosť, tvorba rozhrania či triedy, štruktúry a mnoho iných. [13], [14]

Program písaný v jazyku C# sa skladá najmä z typov (tried), dát a funkcií. Každý typ môže uchovávať v sebe dáta vo forme premenných a funkčné členy vo forme metód alebo úloh. Tieto metódy reprezentujú postupnosť príkazov za určitým cieľom. Žiadna z metód a premenných sa nesmie v programe vyskytovať samostatne, ale vždy len ako súčasť triedy. Jedine ku statickým metódam je možný prístup prostredníctvom typu, inak je nutnosťou vytvorenie jeho inštancie (objektu). Jazyk C# poskytuje aj inteligentné zachytávanie chýb (výnimiek) v kóde pomocou špeciálnych blokov, a tak aj spôsob ako sa vyhnúť pádu aplikácie v prípade jej nečakaného priebehu. [14]

Jazyk C# vyvinuli páni Ander Hejlsberg, ktorý už v minulosti stál za tvorbou Turbo Pascalu a návrhom Delphi, Scott Wiltamuth a Peter Golde. Prvá verzia C# 1.0 bola predstavená v roku 2001, pričom už v máji roku 2000 spoločnosť Microsoft sprístupnila verejnosti prvú predbežnú sadu SDK, spoločne s kompilátorom pre rámec .NET Framework. Vo verzii 2.0 sa objavili novinky ako generické typy, iterátory a anonymné metódy. V C# 3.0 pribudli rozširujúce metódy, lambda výrazy a veľmi mohutný prvok LINQ (Language-Integrated Query), pre uľahčenie práce s kolekciami dát. Ďalšie verzie C# 4.0 a 5.0 priniesli prvky pre paralelné spracovanie úloh a natívnu podporu asynchrónnych úloh použitím označenia *async* a operátoru *await* [15]. [13], [14]

#### 3.1 Novinky vo verzii jazyka C# 6.0

C# 6.0 prináša zopár nových syntaktických vylepšení pre prehľadnejší a čistejší zdrojový kód programu. V novej verzii napr. nie je nutné používať privátne členy pre deklaráciu vlastností určených len na čítanie. Pri takej vlastnosti je možné využiť automatickú

inicializáciu priamo za jej deklaráciou. Pre časté overovanie objektov, či nie sú nulové, sa tvorcovia rozhodli vytvoriť nový jednoduchý operátor `?`, ktorý posluží namiesto vytvárania rozsiahlej podmienky v kóde. Ďalšou z často opakovaných rutín je volanie statických funkcií. Pri použití direktívy *static using* nie je viac potrebné pri volaní funkcií udávať názov príslušného typu. Novinkou je aj zjednodušenie formátovania reťazcov textu pomocou String Interpolation. String Interpolation nahradzuje použitie `String.Format` a dovoľuje na vlastnosti odkazovať priamo v reťazci. [16]

```
/// Deklarácia vlastnosti len pre čítanie
public string Name { get; }

/// Automatická inicializácia vlastnosti len pre čítanie
public string Address { get; } = "Unknown";

/// Overenie nenulového objektu
public char GetInitial()
{
    return (char)(Name?.ElementAt(0));
}

/// Volanie statických funkcií bez definície názvu triedy pri využití statickej
direktívy using static
/// using static System.String;
public string FormatName()
{
    return Format("Name: {0}", Name);
}

/// String interpolation pre zjednodušenie formátovania reťazcov
public string FormattedName
{
    get { return $"Name: {Name}"; }
}
```

#### *Zdrojový kód 1 Novinky v jazyku C# 6.0 – časť 1*

Medzi ďalšie novinky v C# patrí aj nový spôsob zápisu jednoriadkových metód a vlastností určených len pre čítanie za pomoci syntaxi lambda výrazu. Pri programovaní je často potrebné predávať názov typu ako hodnotu reťazca. Preto sa pri ich premenovávaní mnohokrát stáva, že programátor zabudne v kóde premenovať všetky názvy typu reprezentované touto hodnotou. Spomínaný problém rieši nový operátor *nameof*, ktorý preberá typ a vracia jeho názov ako reťazec. Nápomocné môže byť aj zachytávanie výnimiek len za splnenia určitých podmienok a rovnako tak aj využitie volania asynchrónnych úloh v blokoch *catch* a *finally*. V C# 6.0 pribudla aj alternatíva inicializácie prvkov kolekcie priamo v jej deklarácii. [16]

```
// Lambda zápis jednorádkových metod a vlastností
public override string ToString() => $"Name: {Name}, Address: {Address}";
public string FormattedNameSimplified => $"Name: {Name}";

// Nameof operátor vracia názov typu ako reťazec
public static string NameOfType(Type type)
{
    return nameof(type);
}

// Podmienečné výnimky
public void MethodWithCondition()
{
    try { }
    catch (ArgumentNullException ex) when (ex.ParamName == nameof(Person)) { }
}

// Použitie await operátoru v bloch catch a finally
public async void MethodWithAsyncException()
{
    try { }
    catch { await Task.Run(new Action(MethodWithCondition)); }
    finally { await Task.Run(new Action(MethodWithCondition)); }
}

// Inicializácia prvkov kolekcie priamo v deklarácii
public void CreateDictionary()
{
    var dictionary = new Dictionary<string, string>()
    {
        ["Name"] = Name,
        ["Address"] = Address
    };
}
```

*Zdrojový kód 2 Novinky v jazyku C# 6.0 – časť 2*



## 4 JAZYK XAML

Jazyk XAML (Extensible Application Markup Language) je deklaratívny značkovací jazyk vyvinutý spoločnosťou Microsoft pre zjednodušenie tvorby grafického prostredia .NET aplikácií. XAML, je založený na jazyku XML a striktne dodržiava jeho syntaktické a sémantické pravidlá. Pomocou začiatkových a koncových párových a nepárových značiek popisuje zobrazenie ovládacích prvkov a ich stromovú hierarchiu. Každý XAML súbor obsahuje práve jeden koreňový element, pričom každý ďalší element môže obsahovať jeden alebo viac prvkov. Počet prvkov závisí od toho, či je obsah objektov definovaný ako kolekcia alebo ako jeden samostatný objekt. [17], [18]

Na rozdiel od väčšiny značkovacích jazykov, XAML využíva definovanie menných priestorov pre vymedzenie špecifických objektov, ktoré môže grafické rozhranie aplikácie obsahovať. Vďaka tomu je programátor schopný udržiavať kód prehľadnejší a vyhnúť sa kolíziám spojeným s rovnakými názvami objektov. Každá značka reprezentuje práve jednu inštanciu objektu a môže obsahovať atribúty a hodnoty spojené s vlastnosťami objektov. Jazyk bol vyvinutý tak, aby umožnil dizajnérom a programátorom pracovať na aplikáciách oddelene a prípadne k tomu využívať odlišné nástroje. V prípade XAML, je možné rozhranie definovať v čistom kóde alebo využiť nástroj Blend for Visual Studio. Pre zdieľanie hodnôt vlastností prvkov sú použité štýly, ktoré plnia podobný účel ako kaskádové štýly v HTML [19]. Štýly je možné aplikovať lokálne alebo globálne a v príslušných prvkoch naň odkazovať prostredníctvom mien statických zdrojov. Prvky rozhrania sú oddelené od aplikačnej logiky a prepojené s jej kódom na pozadí v druhom súbore. Jednou zo slabostí XAML je veľká náchylnosť na chyby v kóde. V prípade chyby nebude aplikácia schopná vykresliť rozhranie, čo je rozdielom napríklad oproti jazyku HTML. Avšak nástroje pre vývoj a kompilátor inteligentne upozorňujú na nezrovnalosti v kóde a snažia sa týmto situáciám zabrániť. [17], [18]

### 4.1 Novinky jazyka XAML

Jazyk XAML bol s príchodom UWP rozšírený o funkcie zamerané na zvýšenie výkonnosti vykresľovania a možnosti adaptívneho rozhrania. Jedným z prípadov, kedy by bolo náročné a zdĺhavé načítať všetky prvky rozhrania, je situácia zobrazenia veľkého množstva prvkov v zozname. Ovládacie prvky ako ListView a GridView načítavajú len viditeľné objekty a ich blízke okolie. Pri zmene pozície v zozname sa postupne inicializujú ďalšie položky. V prípade rýchleho posunu v zozname sa ale môže stať, že položky nie sú načítané včas

a nezobrazujú žiadny obsah. Ako čiastočné riešenie tohto problému je vykresľovanie novo inicializovaných položiek vo viacerých fázach podľa priority ich zobrazenia. Táto funkcia je založená na rozšírení `x:Bind` a je dostupná len pre spomínané ovládacie prvky `ListView` a `GridView`. Priorita sa definuje ako hodnota atribútu `x:Phase` a preberá celočíselnú hodnotu. Štandardná hodnota je nula, pričom vyššia hodnota značí neskoršiu fázu vykresľovania. Nie je nutné aby postupnosť týchto hodnôt bola založená na kroku 1, ale veľké množstvo fáz nie je relevantné. [20], [21]

```
<ListView>
  <ListView.ItemTemplate>
    <DataTemplate x:DataType="local:ImageModel">
      <StackPanel>
        <Image x:Phase="1" Source="{x:Bind Source}"/>
        <TextBlock Text="{x:Bind Description}"/>
      </StackPanel>
    </DataTemplate>
  </ListView.ItemTemplate>
</ListView>
```

#### *Zdrojový kód 3 Vykresľovanie prvkov rozhrania vo viacerých fázach*

Programátori a dizajnéri často vytvárajú prvky, ktoré sú v rámci stránok neviditeľné alebo sú plne priehľadné. Takéto prvky sú aj napriek neviditeľnosti pri inicializácii rozhrania vykresľované a spomaľujú štart aplikácie. Preto ďalším novým rozšírením je odloženie vytvorenia elementu a všetkých jeho potomkov pomocou atribútu `DeferLoadStrategy="Lazy"`. Odloženie znižuje čas potrebný pre štart aplikácie, ale mierne zvyšuje jej využitie pamäte. Výhodou je aj spojené odloženie inicializácie viazania dát, pokiaľ nie sú potrebné. Takéto odloženie je možné aplikovať len na prvky používateľského rozhrania alebo typy odvodené od základnej triedy `FlyoutBase`. Podmienkou je aj definovanie atribútu `x:Name`. Element, ako aj jeho obsah bude vykreslený až v momente volaní funkcií `FindName`, `GetTemplateChild` alebo aplikovania zmeny hodnoty vo vizuálnom stave. Treba si ale uvedomiť, že element do zavolania skutočne neexistuje, a preto je potrebné sa uistiť o jeho inicializácii pred akoukoľvek manipuláciou s ním. [20], [22]

```
<Image x:Name="DeferredImage" x:DeferLoadStrategy="Lazy"/>
```

#### *Zdrojový kód 4 Deklarovanie odloženia vykreslenia prvku*

```
FindName("DeferredImage");
```

#### *Zdrojový kód 5 Príklad volania vykreslenia odloženého prvku*

Rozšírenie `x:Bind` ponúka alternatívny mechanizmus pre viazanie dát medzi grafickým rozhraním a aplikačnou logikou. Na rozdiel od klasického spôsobu viazania dát (`Binding`), rozšírenie vykonáva účelový kód vygenerovaný už v čase kompilácie. To umožňuje

rýchlejšiu inicializáciu väzieb a zníženie nárokov na využitie pamäte. Takto vytvorené objekty sú schopné sledovať zmeny hodnôt a na základe toho vykonať potrebné aktualizácie v rozhraní alebo dátovej štruktúre. Rovnako ako pôvodný mechanizmus, pracuje v troch módoch: jednorazovom, jednosmernom a obojsmernom. Rozdielom je ale štandardné použitie jednorazového módu. Rozšírenie viac nevyužíva dátového kontextu, ale vlastností samotnej stránky. Preto väzba s modelmi dát musí prebehnúť za pomoci verejnej premennej definovanej v kóde na pozadí stránky alebo môže priamo čerpať z obsahu jej elementov. Výhodou je aj schopnosť viazania ovládačov udalostí alebo možnosť ladenia a overenia väzieb už v čase kompilácie. Napriek mnohým výhodám rozšírenie `x:Bind` neposkytuje také množstvo funkcií ako pôvodný mechanizmus. Rozšírenie nemôže byť použité v štýloch, s výnimkou dátových šablón, ani bez špecifikácie viazaného typu. [23], [24]

```
<TextBox Text="{x:Bind Text, Mode=TwoWay}"/>
```

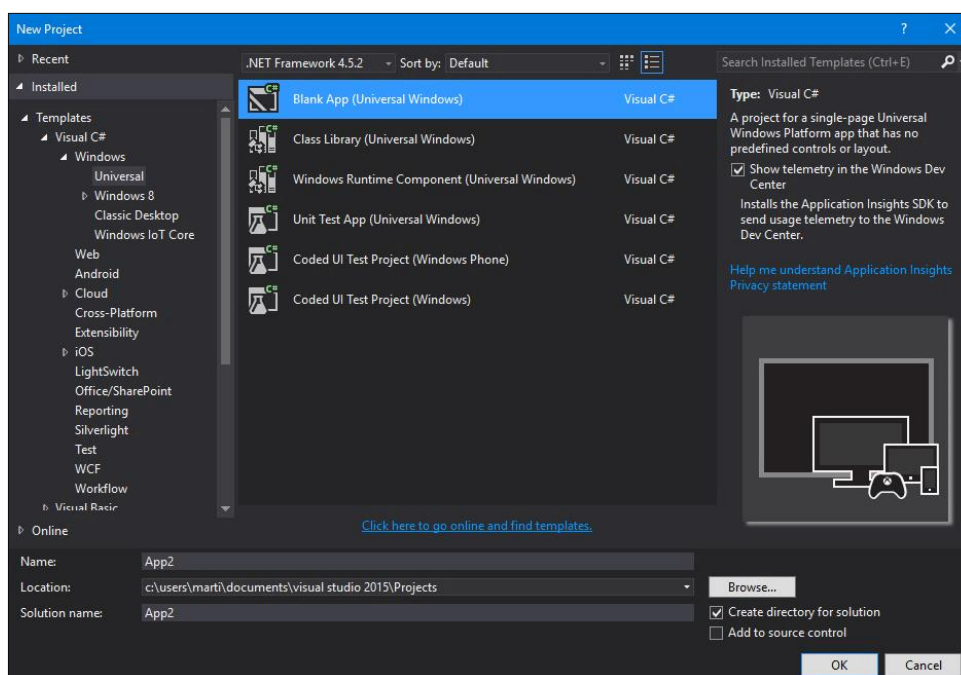
#### *Zdrojový kód 6 Príklad použitia rozšírenia `x:Bind`*

Medzi ďalšie novinky v XAML patria aj adaptívne spúšťače, rozšírenia `Setter` vo vizuálnych stavoch a relatívny panel, ktoré budú podrobnejšie diskutované neskôr.

## 5 TVORBA UWP APLIKÁCIÍ V JAZYKOC H C# A XAML

### 5.1 Základné typy projektov

Rovnako ako aj iné projekty, UWP potrebuje viacero špecifických súborov a nastavení na to, aby bolo možné projekt skompilovať a spustiť. Na tieto účely vývojárske nástroje poskytujú v prostredí Visual Studio preddefinované šablóny. Ako znázorňuje nasledujúci obrázok (Obr. 12), oproti predchádzajúcej verzii Windows 8.1 je ponuka preddefinovaných šablón výrazne menšia.



Obrázok 12 Typy projektov Universal Windows

- **Blank App** – Tento typ projektu vytvorí čisto novú jednostránkovú aplikáciu so všetkými potrebnými náležitosťami pre jej prvé spustenie. Táto aplikácia neobsahuje žiadne ovládacie prvky ani rozloženie.
- **Class Library** – Projekt, ktorého výsledkom je DLL<sup>5</sup> súbor použiteľný v aplikáciách písaných v jazykoch .NET.
- **Windows Runtime Component** – Projekt, ktorého výsledkom je DLL súbor s metadátami, použiteľný v aplikáciách písaných v akomkoľvek z dostupných jazykov. Napriek tomu, veľkým obmedzením je, že komponent nepodporuje verejné

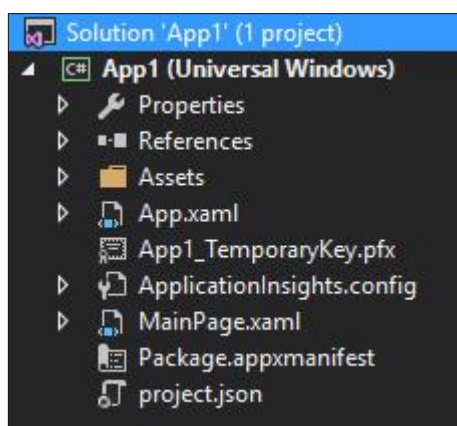
<sup>5</sup> Zdieľaná knižnica viacerých programov obsahujúca zdrojový kód a dáta.

abstraktné typy a môže exportovať len typy Windows RT. Jedinou výnimkou sú typy, ktoré dedia od tried menného priestoru Windows.UI.Xaml. [25]

- **Unit Test App** – Testovacia jednotka aplikácie.

## 5.2 Štruktúra projektu Blank App

Ako je vidieť na (Obr. 13), novo vytvorený projekt univerzálnej aplikácie obsahuje viacero záložiek a súborov. Každý z nich má svoj špecifický účel a funkciu, ktorá je priblížená nižšie.



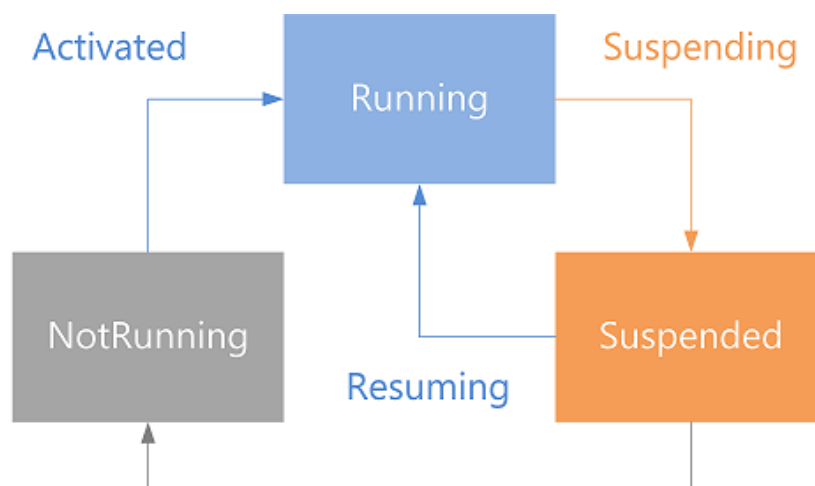
Obrázok 13 Štruktúra projektu

*Blank App*

- **Package.appxmanifest** – Služi k popísaniu detailov aplikácie ako sú napr.: názov, ikony, povolenia, číslo verzie a podobne.
- **App.xaml** – XAML kód hlavnej stránky aplikácie. Zdroje definované na tejto úrovni aplikácie sú globálne prístupné.
  - **App.xaml.cs** – Kód na pozadí hlavnej stránky. Tento súbor je stavebným prvkom aplikácie, ktorý vytvára hlavné okno a reaguje na udalosti životného cyklu.
- **MainPage.xaml** – Stránka, na ktorú je okno aplikácie odkázané po jej spustení.
- **Assets** – Priechinok určený pre prílohy aplikácie. V tomto priečinku sú umiestnené ikony a dlaždice aplikácie.
- **References** – Spravuje odkazy na externé knižnice, balíčky NuGet a iné komponenty aplikácie.
- **project.json** – Konfiguračný súbor balíčkov NuGet.

### 5.3 Životný cyklus aplikácie

Podľa toho, či je aplikácia spustená, pozastavená alebo vôbec nie je aktívna, rozlišujeme jej rôzne stavy. Tie sú popísané životným cyklom aplikácie (Obr. 14). Cyklus je navrhnutý tak, aby umožnil zariadeniam šetriť čo najväčšie množstvo batérie a operačnej pamäte. Životný cyklus desktop a mobilných aplikácií je vzhľadom na ich účel a veľkosť mierne odlišný. Zatiaľ čo na desktop zariadeniach sú aktívne (Running) všetky spustené aplikácie, ktoré neboli inak používateľom minimalizované, na mobilnom zariadení je aktívna vždy len jediná viditeľná aplikácia v popredí. Minimalizovaním aplikácia prechádza do stavu pozastavenia (Suspended). Takto pozastavená aplikácia môže byť znovu oživená opätovnou navigáciou, eventuálne deaktivovaná operačným systémom alebo používateľom. V prípade, že systém rozhodne uvoľniť pamäť využívanú touto aplikáciou, dochádza k jej následnej deaktivácii. Takéto uzavretie nie je možné v kóde detegovať a preto je nutné zvážiť ukladanie potrebných dát už počas udalosti pozastavovania (Suspending). Oživovanie údajov neaktívnych aplikácií je vykonávané prostredníctvom špeciálnych úloh na pozadí (Background Tasks), ktoré môžu bežať bez ohľadu na stav aplikácie. Úlohy na pozadí sú počas behu aplikácie zaregistrované na jednu alebo viaceré udalosti spomedzi systémových, časových alebo geolokačných spúšťačov. V prípade časových intervalov je nutné počítať s obmedzením určujúcim minimálnu dĺžku intervalu rovnú 15 minútam. [26]



Obrázok 14 Životný cyklus aplikácie [27]

### 5.3.1 Obsluha životného cyklu

V rámci obsluhy životného cyklu sú k dispozícii 3 dostupné udalosti prostredníctvom globálnej triedy aplikácie App: spustenie (OnLaunched), pozastavovanie (Suspending) a obnovovanie aplikácie (Resuming). Udalosť spustenia je používaná pre vytvorenie a obnovenie hlavného okna aplikácie a zároveň poskytuje možnosť detekcie formy spustenia alebo jej predošlého stavu. Takéto rozlišovanie už pri spustení je vhodné najmä v prípadoch kedy aplikácia implementuje rôzne hlasové povely, sekundárne dlaždice, upozornenia a podobne. [26]

```
if(e.PreviousExecutionState == ApplicationExecutionState.ClosedByUser)
{
    ...
}
```

*Zdrojový kód 7 Detekcia predošlého stavu aplikácie*

```
if(e.Kind == ActivationKind.VoiceCommand)
{
    ...
}
```

*Zdrojový kód 8 Detekcia formy aktivácie aplikácie*

Udalosť pozastavenia je spustená v prípade minimalizácie aplikácie a slúži pre uloženie potrebných dát a stavu. Naopak udalosť obnovovania aplikácie slúži pre oživenie dát, časovačov a iných potrebných funkcií počas behu aplikácie. Na uloženie dát má aplikácia maximálne 5 sekúnd, pričom najskôr musí požiadať o odklad a následne dokončenie oznámiť. Aplikácia môže požiadať aj o rozšírený čas, ktorý v prípade dostatku pamäte operačný systém povolí alebo zamietne. [26]

```
private async void Current_Suspending(object sender,
Windows.ApplicationModel.SuspendingEventArgs e)
{
    var deferral = e.SuspendingOperation.GetDeferral();
    await SaveDataAsync();
    deferral.Complete();
}
```

*Zdrojový kód 9 Ukladanie dát počas pozastavovania aplikácie*

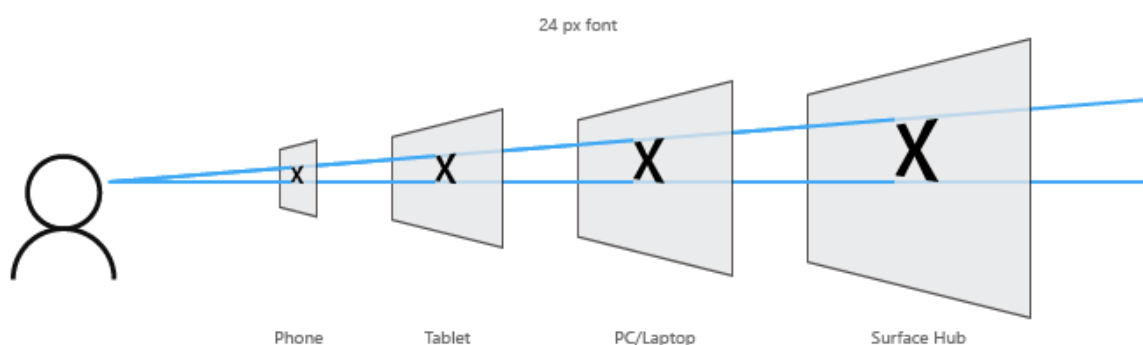
## 5.4 Adaptívne používateľské rozhranie

Pri cielení aplikácie viacerým zariadeniam je mnohokrát vhodné alebo potrebné rozloženie stránky a ovládacích prvkov určitým spôsobom prispôbiť rôznym veľkostiam obrazoviek a vstupným faktorom. Pre tieto účely UWP poskytuje radu funkcií na zjednodušenie tvorby adaptívneho používateľského rozhrania. Platforma ponúka širokú škálu bežne používaných

ovládacích prvkov, ktoré sú automaticky prispôsobené a lokalizované pre daný typ zariadenia, vstupného faktoru a preferencií používateľa. Tieto ovládacie prvky obsahujú štýly pre podporu dvoch základných tém (tmavá alebo svetlá) a témy vysokého kontrastu. Súčasťou štýlov sú aj animácie spojené s manipuláciou prvkov a vizuálnych stavov indikujúcich ich stav zobrazenia. Oproti dizajnovým manuálom predchádzajúcich verzií, nabáda spoločnosť Microsoft k väčšej kreativite, prispôbeniu prostredia vlastnej značky a použitiu vlastných typov písma. Okrem predom pripravených ovládacích prvkov, poskytuje platforma viacero techník ako vytvárať adaptívne rozhranie pre rôzne veľkosti obrazoviek, vstupných faktorov a rodiny zariadení. Pri návrhu dizajnu aplikácie môžeme ľubovoľne kombinovať dva základne spôsoby prístupu k tomuto návrhu. [28], [29]

#### 5.4.1 Efektívne pixely a škálovanie veľkostí

Pre lepšiu čitateľnosť zobrazenia je v rámci platformy zabudovaný špeciálny algoritmus pre automatické škálovanie veľkostí objektov. Tento algoritmus upravuje ich veľkosti tak, aby objekty na rôzne veľkých zariadeniach vyzerali rovnako veľké. Zohľadňuje bežnú vzdialenosť používateľa od zariadenia a jeho hustotu fyzických pixelov. Pri návrhu rozhrania sa používajú tzv. efektívne pixely, ktoré sa skladajú z viacerých fyzických pixelov a ich počet sa líši od konkrétneho zariadenia. Použitie efektívnych pixelov oproti programátorov od zohľadňovania rozlíšenia a DPI. Pri využití takéhoto škálovania veľkostí je odporúčané používať veľkosti, ktoré sú násobkami čísla 4. [28], [29]



Obrázok 15 Škálovanie veľkostí na základe vzdialenosti zariadenia [28]

#### 5.4.2 Adaptívny dizajn

V prípade adaptívneho dizajnu, stránky alebo ovládacie prvky reagujú na zmeny veľkosti okna aplikácie, jej orientácie a eventuálne iným programátorom definovaným udalosťami.



Pre zmeny vlastností rozhrania a prvkov sa využívajú vizuálne stavy (VisualStates), ktoré je po novom možné aktivovať aj pomocou adaptívnych spúšťačov priamo v XAML kóde. Základným zástupcom takéhoto spúšťača je AdaptiveTrigger, ktorý aktivuje vizuálny stav podľa minimálnej šírky alebo minimálnej výšky rozlíšenia okna aplikácie. Ďalšie vlastné spúšťače je možné implementovať za pomoci triedy StateTriggerBase. Okrem nastavovania vlastností objektov vo vizuálnych stavoch definovaním animácií závislých na čase pribudla možnosť využitia rozšírenia Setter ako skokovej zmeny hodnoty. [29]

```
<VisualStateManager.VisualStateGroups>
  <VisualStateGroup>
    <VisualState>
      <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="1024"/>
      </VisualState.StateTriggers>
      <VisualState.Setters>
        <Setter Target="ElementName.Visibility" Value="Visible"/>
      </VisualState.Setters>
    </VisualState>
  </VisualStateGroup>
</VisualStateManager.VisualStateGroups>
```

#### *Zdrojový kód 10 Adaptívny dizajn pomocou vizuálnych stavov*

UWP ponúka 6 základných adaptívnych techník ako sa vysporiadať so zmenami veľkosti a usporiadania v rámci rozloženia stránky.

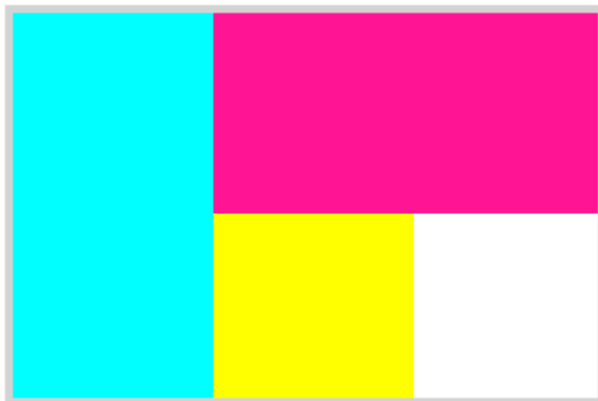
- **Zmena pozície**

Zrejme najvhodnejším prvkom pre zmenu pozícií je nový prvok nazývaný RelativePanel. Relatívny panel slúži k návrhu rozmiestnenia jeho potomkov na základe podmienených vlastností zarovnania alebo umiestnenia voči iným prvkom. Jednotlivým prvkom je priradený parameter x:Name a následne sú zadefinované žiadané vlastnosti zarovnania a umiestnenia. Takto vytvorené vzťahy je možné upravovať vo vizuálnych stavoch a vo výsledku usporadovať rozloženie prvkov na základe zmien veľkosti a podobne. Pri návrhu si ale treba dávať pozor na cyklické alebo konfliktné vlastnosti. [30]

```
<RelativePanel
  Width="300" Height="200"
  BorderBrush="LightGray" BorderThickness="4">
  <Rectangle
    x:Name="CyanRectangle" Fill="Cyan" Width="100" Height="200"/>
  <Rectangle
    x:Name="PinkRectangle" Fill="DeepPink" MinHeight="100"
    RelativePanel.RightOf="CyanRectangle"
    RelativePanel.AlignRightWithPanel="True"/>
  <Rectangle
    x:Name="YellowRectangle" Fill="Yellow" Width="100" Height="100"
    RelativePanel.RightOf="CyanRectangle"
```

```
RelativePanel.Below="PinkRectangle"/>
</RelativePanel>
```

*Zdrojový kód 11 Relatívny panel*



*Obrázok 16 Relatívny panel*

- **Zmena veľkosti**

Skokové zmeny veľkosti môžeme uskutočniť v rámci vizuálnych stavov alebo kódu na pozadí. K automatickým zmenám slúži prvok mriežky (Grid), v ktorom sa určuje relatívna veľkosť definíciou pomeru jednotlivých stĺpcov a riadkov ku veľkosti okna aplikácie.

```
<Grid>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="Auto"/>
    <ColumnDefinition Width="2*" />
    <ColumnDefinition Width="3*" />
  </Grid.ColumnDefinitions>
</Grid>
```

*Zdrojový kód 12 Zmena relatívnej veľkosti*

- **Zmena toku**

Príkladom vhodnej situácie pre zmenu toku je čítanie textu na malej a veľkej obrazovke. Zatiaľ čo na malej obrazovke je text zobrazený v jednom stĺpci, na veľkej obrazovke je vhodné pre lepšiu orientáciu a čitateľnosť text rozdeliť do viacerých stĺpcov.

- **Odhalenie**

Odhalenie je užitočné v prípadoch zobrazenia prvku za prítomnosti špecifickej funkcie rodiny zariadenia.

- **Nahradenie**

V rámci rozhrania môžeme v závislosti na veľkosti obrazovky zmeniť zobrazenie navigačného menu, šablóny položky a podobne.

- **Zmena architektúry**

Zmenu architektúry je vhodné využiť napríklad pri zobrazovaní položiek a ich detailov. Zatiaľ čo na veľkej obrazovke vidíme zoznam položiek aj detail, na malej obrazovke využijeme pre zobrazenie detailu presmerovanie na novú stránku.

Všetky objekty zobrazenia XAML sú schopné reagovať na množstvo udalostí spojených s manipuláciou objektu ako kliknutie, dvojklik, podržanie a podobne. Platforma automaticky zabezpečuje, že forma vyvolania udalostí smie byť ľubovoľná, a preto nie je nutné implementovať jednu a tú istú udalosť pre rôzne vstupy viackrát. [28]

### 5.4.3 Dizajn na mieru

Dizajn stránky alebo ovládacieho prvku je na mieru vytvorený jednej alebo viacerým rodinám zariadení. Tento prístup je vhodné využiť najmä v prípadoch, kedy je žiadané používateľovi poskytnúť odlišný spôsob zobrazovania obsahu na rôznych zariadeniach alebo v situáciách kedy by bolo využitie adaptívnych techník príliš rozsiahle a neefektívne. Dizajn na mieru je možné tvoriť aj pre určité škály veľkostí rozlíšenia, ale len za pomoci detekcie zmeny veľkosti okna v kóde na pozadí a následným presmerovaním na požadovanú stránku. Pre definíciu na základe rodiny zariadení sú k dispozícii dva rôzne spôsoby:

- **Presmerovanie z kódu na pozadí**

Jedným zo spôsobov je získanie označenia rodiny zariadenia pomocou triedy `ResourceContext` a následné presmerovanie na novú stránku v kóde na pozadí.

```
var rootFrame = Window.Current.Content as Frame;
var deviceFamily =
ResourceContext.GetForCurrentView().QualifierValues["DeviceFamily"];

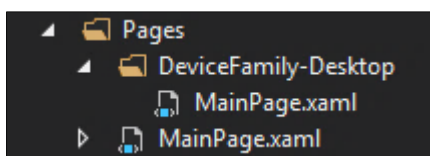
switch (deviceFamily)
{
    case "Mobile":
        rootFrame.Navigate(typeof(MobilePage));
        break;
}
```

*Zdrojový kód 13 Detekcia rodiny zariadenia*

- **XAML View**

Druhou možnosťou je vytvorenie nového zobrazenia v podobe XAML View súboru pomenovaného rovnako ako pôvodná stránka. V tomto súbore definujeme unikátne rozhranie pre určitú rodinu zariadení a vložíme ho do adresára s názvom zloženého z reťazca „DeviceFamily-“ a názvu cieľovej rodiny zariadení. Takto vytvorené nové

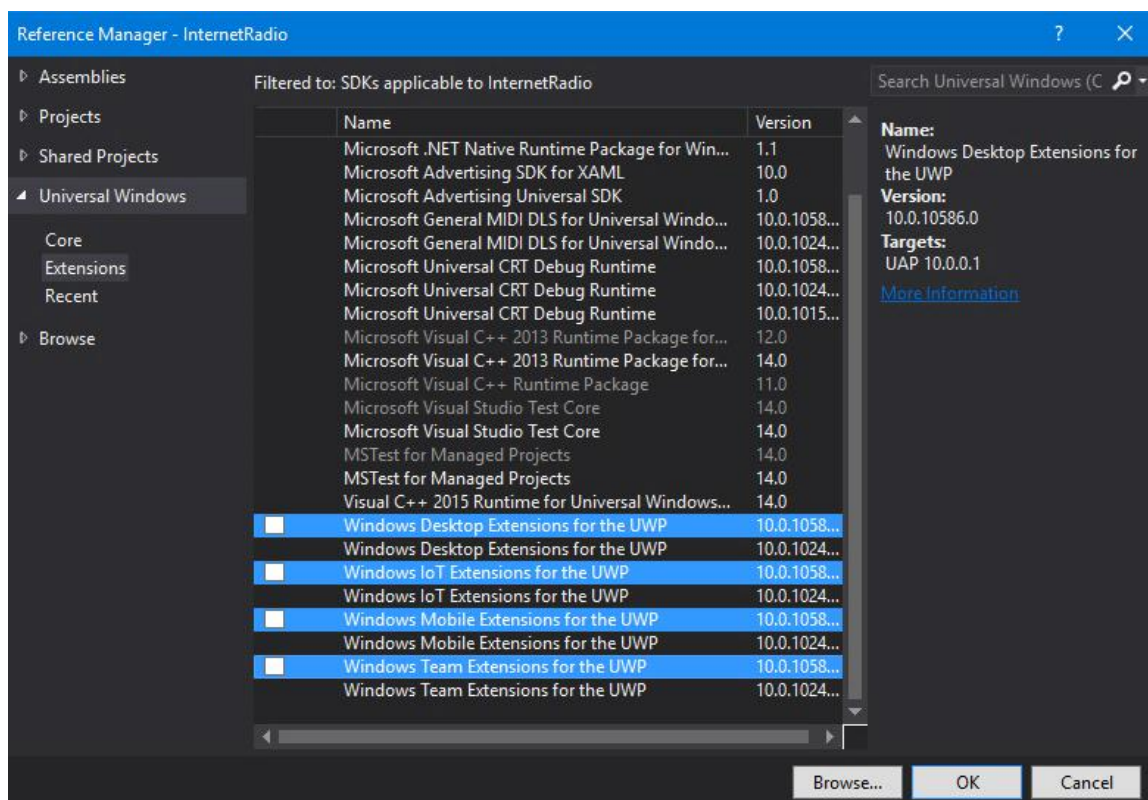
zobrazenie je automaticky aktivované pri jeho volaní a využíva pôvodný súbor s kódom na pozadí. [29]



Obrázok 17 Zobrazenie na mieru  
pre unikátnu rodinu zariadenia

## 5.5 Adaptívny kód

Adaptívny kód nahradzuje logiku kompilačných direktív z predchádzajúcej verzie Windows 8.1. Špecifické knižnice boli kontrolované v čase kompilácie, čoho hlavnou nevýhodou bola nutnosť tvorby viacerých aplikačných balíčkov. V prípade UWP je prítomnosť špecifických knižníc kontrolovaná v čase behu programu, pomocou klasických podmienok v kóde aplikácie. Pre prístup k špecifickým knižniciam rodín zariadení sú k dispozícii rozšírenia, ktoré sa do aplikácie pridávajú rovnakým spôsobom ako referencie na externé komponenty alebo knižnice. Verzia SDK 10.0.10586.0 obsahuje rozšírenia pre desktop zariadenia, mobilné zariadenia, zariadenia IoT a SurfaceHub. [31]



Obrázok 18 Rozšírenia rodín zariadení

Po pridaní rozšírenia je možné pracovať so špecifickými knižnicami rodiny zariadenia. Kompletný obsah rozšírení je dostupný v oficiálnej dokumentácii na stránke: <https://msdn.microsoft.com/en-us/library/windows/apps/dn706137.aspx>. Pred použitím knižnice je nutné overiť jej prítomnosť, k čomu slúžia viaceré funkcie triedy `Windows.Foundation.Metadata.ApiInformation`. Nasledujúce dva príklady demonštrujú kontrolu prítomnosti špecifického rozhrania API a kontrolu prítomnosti špecifickej verzie rozhrania API. [31]

```
if
(Windows.Foundation.Metadata.ApiInformation.IsTypePresent("Windows.Media.Capture.CameraCaptureUIContract"))
{
    Windows.Media.Capture.CameraOptionsUI.Show(mediaCaptureMgr);
}
```

*Zdrojový kód 14 Kontrola prítomnosti špecifického rozhrania API*

```
if
(Windows.Foundation.Metadata.ApiInformation.IsApiContractPresent("Windows.Media.Capture.CameraCaptureUIContract", 1, 0))
{
    Windows.Media.Capture.CameraOptionsUI.Show(mediaCaptureMgr);
}
```

*Zdrojový kód 15 Kontrola prítomnosti špecifickej verzie rozhrania API*

V prípade, že by v kóde ku kontrole nedošlo a rozhranie nie je prístupné, aplikácia aktivuje výnimku. Zabráneniu takýchto situácií pomáha NuGet balíček `PlatformSpecific.Analyzer`, ktorý rozšíri funkciu `IntelliSense` o kontrolu a chybové hlásenia. Rozšírenie disponuje aj možnosťou doplnenia podmienky.

## **II. PRAKTICKÁ ČASŤ**

## 6 INTERNETOVÉ RADIO

V praktickej časti tejto práce je podrobne demonštrovaný proces tvorby univerzálnej klient / server aplikácie internetového rádia pre inteligentnú domácnosť. Aplikácia je implementovaná pomocou programovacích jazykov C# a XAML, pričom využíva moderné technológie pre tvorbu Universal Windows Platform aplikácií a adaptívneho používateľského rozhrania. Ovládanie a správa staníc internetového rádia je možná prostredníctvom komunikácie s rôznymi typmi zariadení po sieti Wi-Fi a LAN.

### 6.1 Ciele

Na začiatku procesu tvorby aplikácie je dôležitá špecifikácia cieľov popisujúcich jej funkcionality, správanie a architektúru samotného projektu. Pre jednoduchšie pochopenie sú ciele aplikácie rozdelené do troch základných skupín požiadaviek.

#### 6.1.1 Funkcionálne požiadavky

Funkcionálne požiadavky aplikácie reprezentujú žiadanú funkcionality, použiteľnosť, správanie a návrh používateľského rozhrania. Medzi tieto požiadavky patrí:

- Vytvoriť serverovú aplikáciu cielenú pre zariadenie Windows 10 IoT Core na prehrávanie hudby a spracovávanie požiadaviek klienta
- Vytvoriť klientskú aplikáciu pre ovládanie a správu staníc zariadenia
- Implementovať sieťovú komunikáciu medzi zariadeniami
- Vytvoriť adaptívne používateľské rozhranie pre rôzne veľkosti a využitie zariadení

#### 6.1.2 Ne-funkcionálne požiadavky

Do kategórie ne-funkcionálnych požiadaviek zaradíme ciele popisujúce spôsob efektívnej implementácie. Nefunkčné požiadavky teda neurčujú to, čo by aplikácia mala byť schopná vykonávať, ale akým spôsobom ju implementovať tak, aby poskytovala používateľovi čo najväčší komfort. Ne-funkcionálne požiadavky riešia najmä otázky bezpečnosti, škálovania a výkonu aplikácie. Medzi tieto požiadavky patrí:

- Využiť oficiálne pokyny a metódy tvorby UWP aplikácií
- Vysporiadať sa s obmedzením komunikácie sieťového pripojenia
- Vykonávať požiadavky asynchrónne
- Informovať používateľa o aktuálnom stave a priebehu

- Vysporiadať sa s nežiadanými stavmi aplikácie

### 6.1.3 Architektonické požiadavky

Architektonické ciele sú zamerané na proces vývoja a popisujú spôsob štruktúry a organizácie programového kódu aplikácie. Sú zamerané na efektívnosť, prehľadnosť, prenositeľnosť, údržbu a rozširiteľnosť kódu v prípade potreby. Medzi tieto požiadavky patrí:

- Separácia spoločných modelov a funkcií
- Tvorba a separácia pomocných tried a ovládacích prvkov
- Využitie architektonického vzoru MVVM



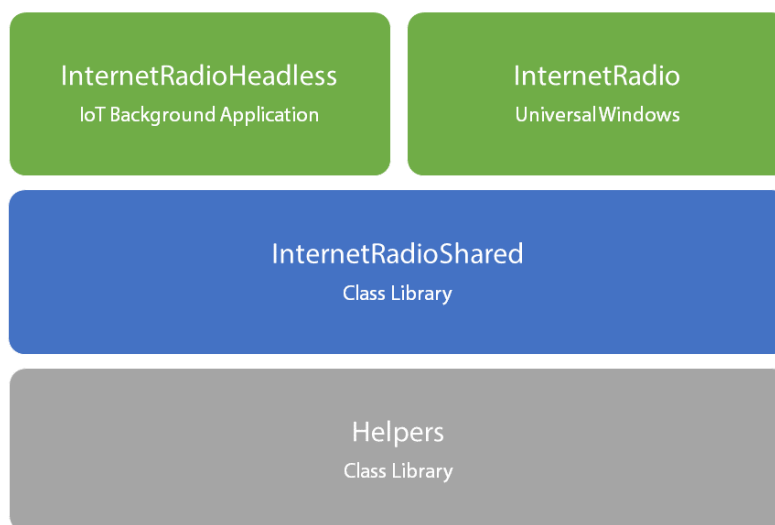
## 7 ARCHITEKTÚRA APLIKÁCIE

Už na samotnom začiatku návrhu architektúry je aplikácia rozdelená medzi dva základné projekty. Prvým projektom je aplikácia InternetRadioHeadless, pre zariadenie Windows 10 IoT Core a zabezpečuje prehrávanie rádiových staníc a spracovávanie klientských požiadaviek. Druhým základným projektom je univerzálna klientská aplikácia InternetRadio pre platformu UWP, ktorá poskytuje používateľské rozhranie pre vzdialenú obsluhu rádia.

Tretím projektom je triedna knižnica InternetRadioShared určená pre zdieľané modely (triedy) oboch aplikácií. Pre predstavu, oba projekty pracujú so spoločným modelom reprezentujúcim rádiovú stanicu alebo modelom vyjadrujúcim jeho aktuálny stav. Preto je efektívne takéto modely umiestniť do zdieľanej triednej knižnice a v oboch projektoch naň odkázať prostredníctvom referencií. Takáto implementácia nielenže zabráni duplikácií modelov, ale rovnako tak zjednoduší ich budúce rozšírenie a údržbu.

Štvrtým projektom je zdieľaná triedna knižnica Helpers obsahujúca pomocné triedy, modely a vlastné ovládacie prvky používateľského rozhrania. Zmyslom tohto komponentu je vytvárať a zhromažďovať spoločné prvky ako je navigačná logika alebo navigačná ponuka a neskôr ich využiť v akejkoľvek budúcej aplikácii. Časom si programátor vytvorí rozsiahlu zbierku, ktorá mu môže značne ušetriť čas strávený implementáciou už v minulosti vytvoreného prvku.

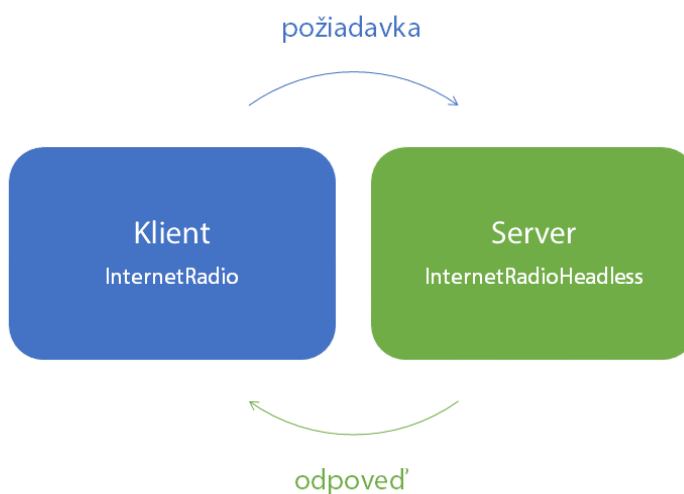
Výsledná vyššie popísaná štruktúra aplikácie je znázornená na obrázku (Obr. 19). Architektúra a štruktúra čiastkových aplikácií bude popísaná v nasledujúcich kapitolách.



Obrázok 19 Štruktúra aplikácie

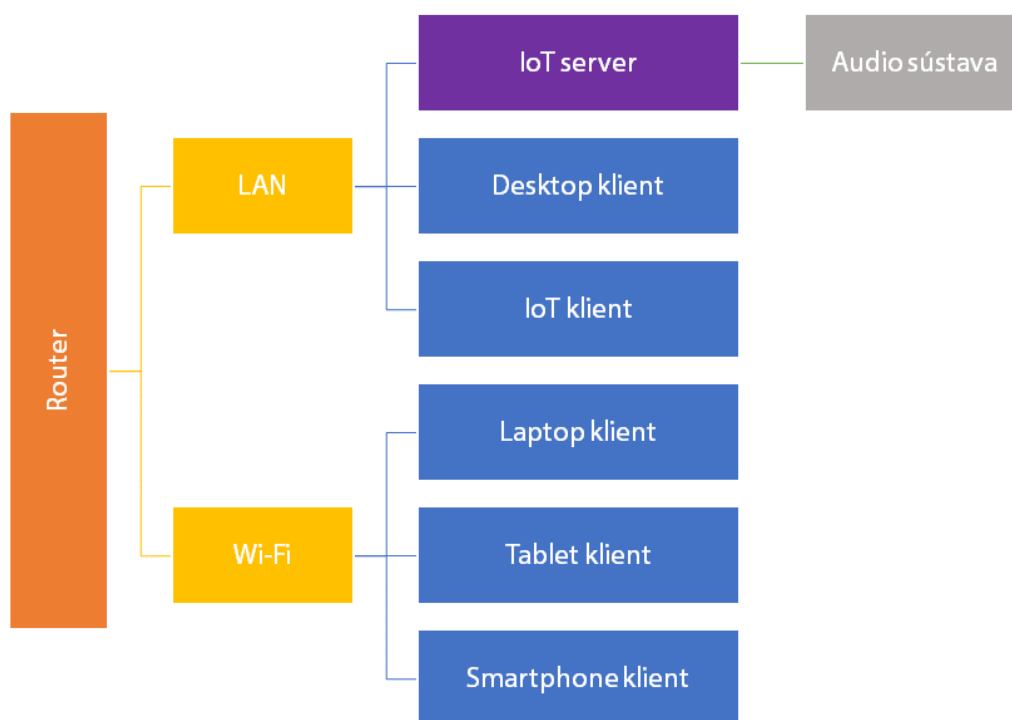
## 8 SIEŤOVÁ KOMUNIKÁCIA MEDZI APLIKÁCIAMI

Veľmi dôležitou súčasťou tohto projektu je vytvorenie sieťovej komunikácie medzi klientským a serverovým zariadením pre prehrávanie rádiových staníc. Klient odosiela požiadavky na získanie alebo zmenu stavu rádia a rovnako tak požiadavky na získanie alebo správu rádiových staníc uložených v pamäti serverového zariadenia. Ako je možné vidieť na obrázku (Obr. 23), klient odosiela požiadavku serverovému zariadeniu a čaká na jeho odpoveď. Táto odpoveď obsahuje informáciu o úspešnosti a telo odpovede.



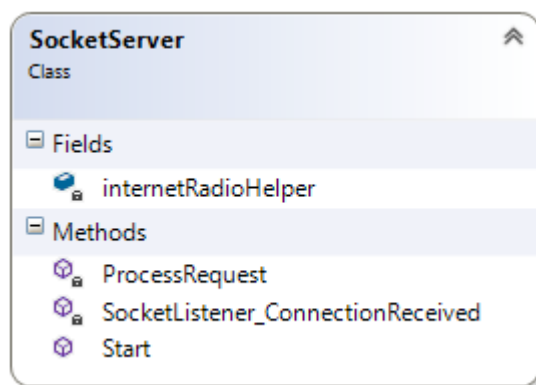
Obrázok 20 Komunikácia klient / server

Na implementáciu vzdialenej komunikácie týchto dvoch uzlov je využitý mechanizmus `StreamSocket`. Prenášanie údajov je zabezpečené prostredníctvom spojenia typu `TCP/IP`. Výhodou `StreamSocket` mechanizmu je poskytovanie trvalého spojenia, čo v konečnom dôsledku prináša spoľahlivý a konzistentný prenos dát. Na obrázku (Obr. 24) je znázornené zapojenie viacerých klientských a serverového zariadenia do spoločnej domácej siete. V rámci tejto siete sú klientské zariadenia schopné vymieňať si údaje so serverom.



Obrázok 21 Zapojenie klient / server zariadení v domácej sieti

## 8.1 StreamSocket server



Obrázok 22 SocketServer model

K vytvoreniu komunikačného serveru je použitá inštancia triedy `Windows.Networking.Sockets.StreamSocketListener`, ktorá dokáže spustiť službu na žiadanom porte zariadenia a informovať o nadviazaní nového spojenia. Pre lepšiu organizáciu kódu sú obe tieto funkcionality spoločne so spracovávaním prijatej požiadavky zakomponované do novej triedy nazvanej `SocketServer`. Tá v rámci funkcie `Start` inicializuje nový server na žiadanom porte a registruje ovládač udalosti pre nadviazanie nového spojenia.

```
public async void Start(int port)
{
    try
    {
        StreamSocketListener socketListener = new StreamSocketListener();

        // Registrácia udalosti vzniku nového spojenia
        socketListener.ConnectionReceived += SocketListener_ConnectionReceived;

        // Spustenie služby na žiadanom porte
        await socketListener.BindServiceNameAsync(port.ToString());
    }
    catch { }
}
```

#### *Zdrojový kód 16 Inicializácia serverovej služby*

Po nadviazaní spojenia s klientom je udalosťou spustená funkcia `SocketListener_ConnectionReceived`, ktorá poskytuje vstupný a výstupný tok dát medzi dvoma zariadeniami. Pre udržanie trvalého spojenia je nutné pracovať v slučke. Po každom prečítaní je požiadavka spracovaná a odpoveď zapísaná na výstupný tok dát komunikácie.

```
private async void SocketListener_ConnectionReceived(StreamSocketListener sender,
StreamSocketListenerConnectionReceivedEventArgs args)
{
    try
    {
        using (DataReader reader = new DataReader(args.Socket.InputStream))
        {
            using (DataWriter writer = new DataWriter(args.Socket.OutputStream))
            {
                while (true)
                {
                    uint sizeFieldCount = await reader.LoadAsync(sizeof(uint));
                    if (sizeFieldCount != sizeof(uint))
                    {
                        return;
                    }
                    // Prečíta textový reťazec
                    uint stringLength = reader.ReadUInt32();
                    uint actualStringLength = await reader.LoadAsync(stringLength);
                    if (stringLength != actualStringLength)
                    {
                        return;
                    }

                    string request = reader.ReadString(actualStringLength);

                    // Spracovanie požiadavky
                    SocketResponse response = ProcessRequest(request);
                    string responseString =
                        JsonHelper.ToJson<SocketResponse>(response);

                    // Zápis veľkosti a obsahu reťazca do lokálneho čítača
                    writer.WriteUInt32(writer.MeasureString(responseString));
                    writer.WriteString(responseString);
                    // Zápis lokálneho čítača do siete
                    await writer.StoreAsync();
                }
            }
        }
    }
}
```

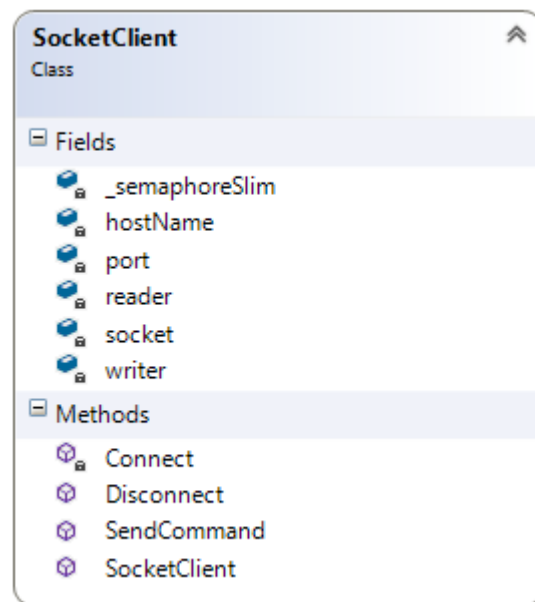
```

    }
}
catch(Exception ex)
{
    // Kontrola fatálnej chyby
    if (Windows.Networking.Sockets.SocketError.GetStatus(ex.HResult) ==
        SocketErrorStatus.Unknown)
    {
        throw;
    }
}
}

```

*Zdrojový kód 17 Ovládač udalosti nadviazania spojenia*

## 8.2 StreamSocket klient



*Obrázok 23 Model SocketClient*

Pre vytvorenie komunikačného klienta je definovaná trieda `SocketClient`, ktorá pomocou inštancie triedy `Windows.Networking.Sockets.StreamSocket` komunikuje so serverom. Objekt tejto triedy je inicializovaný v tele funkcie `Connect`, kde sa zároveň pripája na žiadaný server. Ten je reprezentovaný IP adresou alebo názvom zariadenia a doplnený o číslo portu.

```

private async Task Connect(int responseTimeout)
{
    CancellationTokenSource cts = new CancellationTokenSource();

    try
    {
        cts.CancelAfter(responseTimeout);

        socket = new StreamSocket();
        socket.Control.KeepAlive = false;
    }
}

```

```

        socket.Control.NoDelay = true;

        HostName serverHost = new HostName(hostName);
        String serverPort = port.ToString();
        await socket.ConnectAsync(serverHost, serverPort).AsTask(cts.Token);

        reader = new DataReader(socket.InputStream);
        writer = new DataWriter(socket.OutputStream);
    }
    catch
    {
        Disconnect();
    }
}

```

### *Zdrojový kód 18 Nadviazanie pripojenie k serveru*

Pokiaľ by sa do vyrovnávacej pamäte zapisovalo viacero požiadaviek naraz, mohlo by dôjsť k narušeniu konzistentnosti odosielaných dát a server by s najväčšou pravdepodobnosťou vyhodnotil takúto komunikáciu ako predčasne uzavretú. Z tohto dôvodu je vo funkcii odosielania požiadavky zakomponovaný semafor zabráňujúci vykonávaniu viacerých požiadaviek paralelne. Po overení spojenia je požiadavka serializovaná do formátu JSON a odoslaná na server. Ten vracia klientovi odpoveď, ktorá je následne deserializovaná a vrátená volaniu úlohy SendCommand.

```

/// Semafor pre zabránenie paralelného prístupu viacerých požiadaviek k čítaču
private SemaphoreSlim _semaphoreSlim = new SemaphoreSlim(1);

public async Task<SocketResponse> SendCommand(SocketCommand command, int
connectionTimeout = 2000)
{
    // Čaká na dokončenie predchádzajúcej požiadavky
    await _semaphoreSlim.WaitAsync();

    try
    {
        // Nadviazanie spojenia
        if (socket == null)
            await Connect(connectionTimeout);

        // Serializácia objektu požiadavky do dátového formátu json
        string request = JsonHelper.ToJson<SocketCommand>(command);
        // Zápis veľkosti a obsahu reťazca do lokálneho čítača
        writer.WriteUInt32(writer.MeasureString(request));
        writer.WriteString(request);
        // Zápis lokálneho čítača do siete
        await writer.StoreAsync();

        uint sizeFieldCount = await reader.LoadAsync(sizeof(uint));
        if (sizeFieldCount != sizeof(uint))
        {
            return new SocketResponse(SocketResponse.StatusCode.EXCEPTION, "The
underlying socket was closed before we were able to read the whole data.");
        }
        // Prečítanie odpovede
        uint stringLength = reader.ReadUInt32();
        uint actualStringLength = await reader.LoadAsync(stringLength);
    }
}

```

```

        if (stringLength != actualStringLength)
        {
            return new SocketResponse(SocketResponse.StatusCode.EXCEPTION, "The
underlying socket was closed before we were able to read the whole data.");
        }
        string responseString = reader.ReadString(actualStringLength);
        return JsonHelper.FromJson<SocketResponse>(responseString);
    }
    catch (Exception ex)
    {
        Disconnect();
        throw ex;
    }
    finally
    {
        _semaphoreSlim.Release();
    }
}

```

*Zdrojový kód 19 Úloha odoslania požiadavky na server*

### 8.3 Serializácia dát

Vzhľadom na to, že v rámci implementovanej komunikácie sú dáta vymieňané ako textový reťazec, je nutné ich pred posielaním určitým spôsobom serializovať. Na tieto účely je vytvorená pomocná trieda `JsonHelper` obsahujúca statické funkcie prevádzajúce akýkoľvek serializovateľný objekt na textovú reprezentáciu formátu JSON a späť. Táto trieda má využitie nie len pri posielaní dát, ale rovnako tak aj pri ich ukladaní do pamäte. Z tohto dôvodu je trieda `JsonHelper` zaradená do triednej knižnice `Helpers`.

```

public static string ToJson<T>(T obj)
{
    DataContractJsonSerializer ser = new DataContractJsonSerializer(typeof(T));

    using (MemoryStream ms = new MemoryStream())
    {
        ser.WriteObject(ms, obj);
        var jsonArray = ms.ToArray();
        return Encoding.UTF8.GetString(jsonArray, 0, jsonArray.Length);
    }
}

```

*Zdrojový kód 20 Serializácia dát*

```

public static T FromJson<T>(string json)
{
    DataContractJsonSerializer ser = new DataContractJsonSerializer(typeof(T));

    using (MemoryStream stream = new MemoryStream(Encoding.UTF8.GetBytes(json)))
    {
        return (T)ser.ReadObject(stream);
    }
}

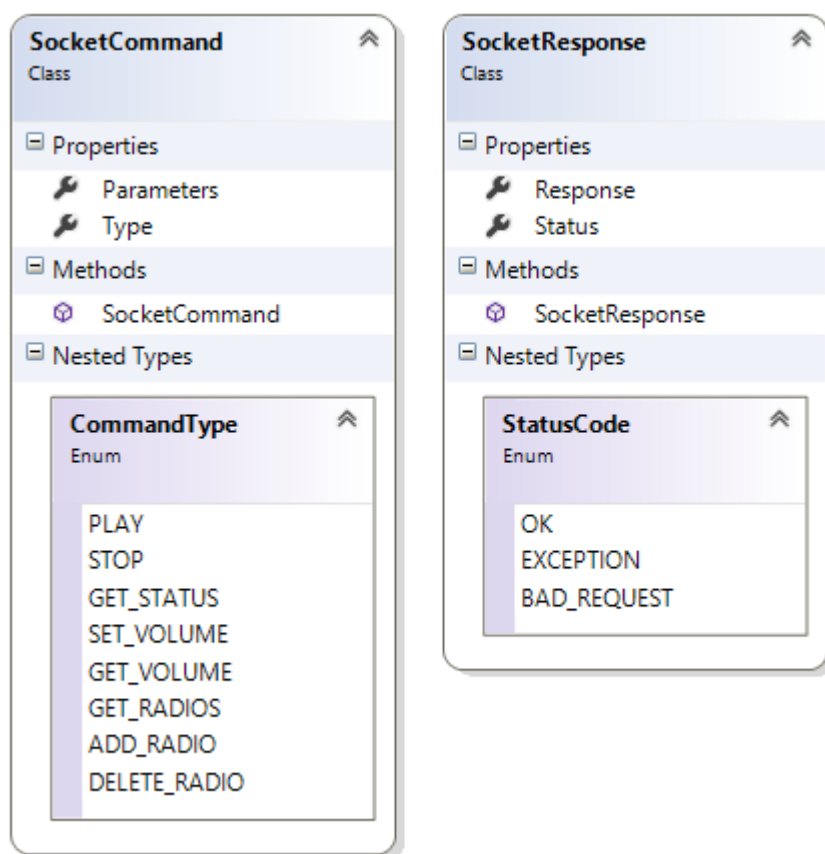
```

*Zdrojový kód 21 Deserializácia dát*

Pri posielaní požiadaviek je potrebné špecifikovať ich typ a parameter nesúci informáciu o žiadanej hodnote. Rovnako tak pri odpovedi serveru je potrebné odoslať informáciu o tom či požiadavka bola úspešná a vrátiť výslednú hodnotu (odpoveď serveru). Z týchto dôvodov boli vytvorené zdieľané serializovateľné modely `SocketCommand` a `SocketResponse`. Na to, aby objekt určitej triedy bolo možné serializovať, musí byť trieda označená atribútom `[DataContract]` a jej hodnoty atribútom `[DataMember]`.

```
[DataContract]
public class SocketCommand
{
    [DataMember]
    public CommandType Type { get; set; }
    [DataMember]
    public object[] Parameters { get; set; }
}
```

*Zdrojový kód 22 Označenie atribútmi serializácie*



*Obrázok 24 Modely SocketRequest a SocketResponse*



## 9 WINDOWS 10 IOT CORE APLIKÁCIA

V serverovej aplikácii InternetRadioHeadless je zakomponovaná obsluha prehrávania audio streamov a hlasitosti, manipulácia s úložiskom rádiových staníc a obsluha prichádzajúcich požiadaviek klientov. Aplikácia je typu Windows Universal IoT Background Application a je nainštalovaná do zariadenia Raspberry Pi 2 s operačným systémom Windows 10 IoT Core. Tento systém plne podporuje spúšťanie UWP aplikácii a ponúka viacero rozšírení určených pre odvetvie Internet vecí. Jeho hlavným rozdielom je možnosť prítomnosti jedinej aplikácie v popredí. Z tohto dôvodu je projekt cielený ako služba na pozadí, čím zariadenie neobmedzuje na jediný účel. Výhodou využitia tejto služby je možnosť inštalácie klientskej alebo inej aplikácie do popredia. Aplikácia InternetRadioHeadless implementuje nasledujúce typy požiadaviek:

Typ požiadavky	Parameter požiadavky	Odpoveď	Popis požiadavky
PLAY	Model rádiovej stanice	len kód stavu	Spustí prehrávanie rádiovej stanice.
STOP	bez parametra	len kód stavu	Zastaví prehrávanie.
GET_VOLUME	bez parametra	Číselná hodnota hlasitosti v rozmedzí 0-1	Vráti hodnotu aktuálnej hlasitosti.
SET_VOLUME	Číselná hodnota hlasitosti v rozmedzí 0-1	len kód stavu	Nastaví hlasitosť na požadovanú hodnotu.
GET_STATUS	bez parametra	Model stavu rádia	Vráti aktuálny stav prehrávania.
GET_RADIOS	bez parametra	Kolekcia modelov rádiových staníc	Vráti zoznam rádiových staníc v pamäti zariadenia.
ADD_RADIO	Model rádiovej stanice	len kód stavu	Pridá rádiovú stanicu do pamäte zariadenia.
DELETE_RADIO	ID rádiovej stanice	len kód stavu	Odstráni rádiovú stanicu z pamäte zariadenia.

Tabuľka 1 Typy implementovaných požiadaviek

### 9.1 Štruktúra projektu

Štruktúra aplikácie rovnako ako bežné úlohy na pozadí (Background Tasks) obsahuje hlavnú triedu `StartupTask`, ktorá dedí od rozhrania `Windows.ApplicationModels.Background.IBackgroundTask`. Hlavným rozdielom oproti klasickej úlohe na pozadí je absencia registrácie na systémové alebo časové udalosti. Výsledná aplikácia môže byť spúšťaná manuálne alebo automaticky po štarte systému. Pre podporu automatického spúšťania je v aplikačnom manifeste deklarované rozšírenie úlohy

typu Startup. Vstupným bodom serverovej aplikácie je metóda Run, v rámci ktorej je inicializovaná komunikačná inštancia triedy SocketServer.

```
<BackgroundTasks>
  <iot:Task Type="startup" />
</BackgroundTasks>
```

*Zdrojový kód 23 Rozšírenie pre automatický štart pri spustení systému*

## 9.2 Práca s úložiskom - správa rádiových staníc

Rádiové stanice sú ukladané do lokálnej pamäte aplikácie prostredníctvom pomocnej statickej triedy LocalSettingsHelper. Statická trieda je súčasťou triednej knižnice Helpers, pretože ju je možné použiť v akomkoľvek projekte UWP aplikácii. Pamäť kontajnerového typu sa nachádza v aplikačnom sandboxe, čo znamená že žiadna iná aplikácia nemôže pristupovať k uloženým dátam. Kolekcia rádiových staníc je pri ukladaní serializovaná do dátového formátu JSON. Záznamy sú v kontajneroch uchovávané, aktualizované a prístupné pod unikátnymi textovými kľúčmi definovanými programátorom.

```
public static void SetValue(string key, object value, string container = null)
{
    ApplicationDataContainer applicationDataContainer;

    if (container != null)
    {
        if (localSettings.Containers.ContainsKey(container))
        {
            applicationDataContainer = localSettings.Containers[container];
        }
        else
        {
            applicationDataContainer = localSettings.CreateContainer(container,
ApplicationDataCreateDisposition.Always);
        }
    }
    else
    {
        applicationDataContainer = localSettings;
    }
    if (applicationDataContainer.Values.ContainsKey(key))
    {
        applicationDataContainer.Values[key] = value;
    }
    else
    {
        applicationDataContainer.Values.Add(key, value);
    }
}
```

*Zdrojový kód 24 Pomocná metóda pre uloženie dát do lokálnej pamäte aplikácie*

```
public static object GetValue(string key, string container = null)
{
    ApplicationDataContainer applicationDataContainer;
```

```
if (container != null)
{
    if (localSettings.Containers.ContainsKey(container))
    {
        applicationDataContainer = localSettings.Containers[container];
    }
    else
    {
        throw new KeyNotFoundException();
    }
}
else
{
    applicationDataContainer = localSettings;
}

if (applicationDataContainer.Values.ContainsKey(key))
{
    return applicationDataContainer.Values[key];
}
else
{
    throw new KeyNotFoundException();
}
}
```

*Zdrojový kód 25 Pomocná metóda pre čítanie dát z lokálnej pamäte aplikácie*

### 9.3 Obsluha prehrávania rádiových staníc

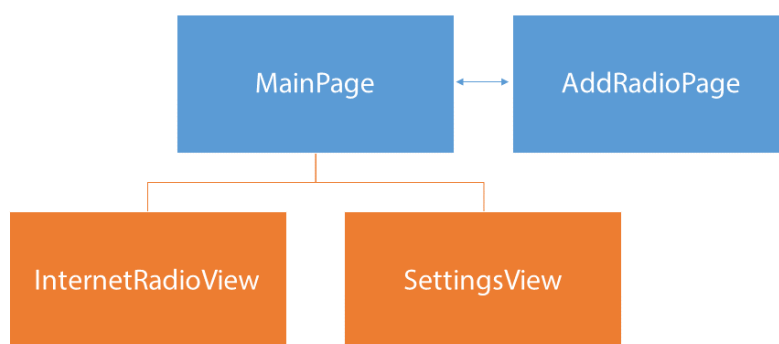
Obsluha prehrávania rádiových staníc je implementovaná prostredníctvom triedy `Windows.Media.Playback.BackgroundMediaPlayer`. Tá poskytuje prístup k mediálnemu prehrávaču spusteného na pozadí. Jednotlivé požiadavky pre jeho obsluhu sú deklarované tak, aby v prípade úspechu či neúspechu vrátili klientovi relevantnú odpoveď. Nasledujúca ukážka zdrojového kódu demonštruje obsluhu požiadavky pre spustenie rádiovej stanice.

```
public static SocketResponse PlayRadio(Radio radio)
{
    try
    {
        BackgroundMediaPlayer.Current.SetUriSource(radio.Uri);
        return new SocketResponse(SocketResponse.StatusCode.OK);
    }
    catch(Exception ex)
    {
        return new SocketResponse(SocketResponse.StatusCode.EXCEPTION,
ex.ToString());
    }
}
```

*Zdrojový kód 26 Spustenie rádiovej stanice*

## 10 KLIENTSKÁ APLIKÁCIA INTERNETOVÉ RÁDIO

Klientská aplikácia InternetRadio slúži k vzdialenej obsluhu rádia a demonštruje tvorbu univerzálnej aplikácie cielenej pre všetky zariadenia s operačným systémom Windows 10. Výsledná aplikácia je vo forme jediného aplikačného balíku formátu .appx. Tento balík môže byť nainštalovaný priamo do zariadenia pomocou nástroja Visual Studio alebo publikovaný do obchodu Windows Store, a tak sprístupnený pre široké publikum. Aplikácia je optimalizovaná a testovaná na zariadeniach typu desktop, tablet, mobil a IoT. Vzhľadom na rozdiely zariadení najmä v rozlíšení obrazoviek a spôsobe interakcie so zariadením, táto kapitola poukazuje na možnosti implementácie adaptívneho používateľského rozhrania. Okrem toho je diskutovaná tvorba základnej architektúry, spracovania požiadaviek a implementácia odporúčaných navigačných techník.



Obrázok 25 Navigačný diagram

Aplikácia obsahuje dve základné stránky prvej úrovne. Prvá z nich, stránka MainPage je spustená ihneď po štarte aplikácie a zahŕňa ďalšie zobrazenia, medzi ktorými je prechod umožnený prostredníctvom navigačnej sekcie. Zatiaľ čo zobrazenia stránky MainPage poskytujú viaceré funkcie pre ovládanie, správu a nastavenia komunikácie, stránka AddRadioPage slúži výhradne na účely vytvorenia novej rádiovej stanice.

### 10.1 Architektúra projektu

#### 10.1.1 Model-View-ViewModel (MVVM)

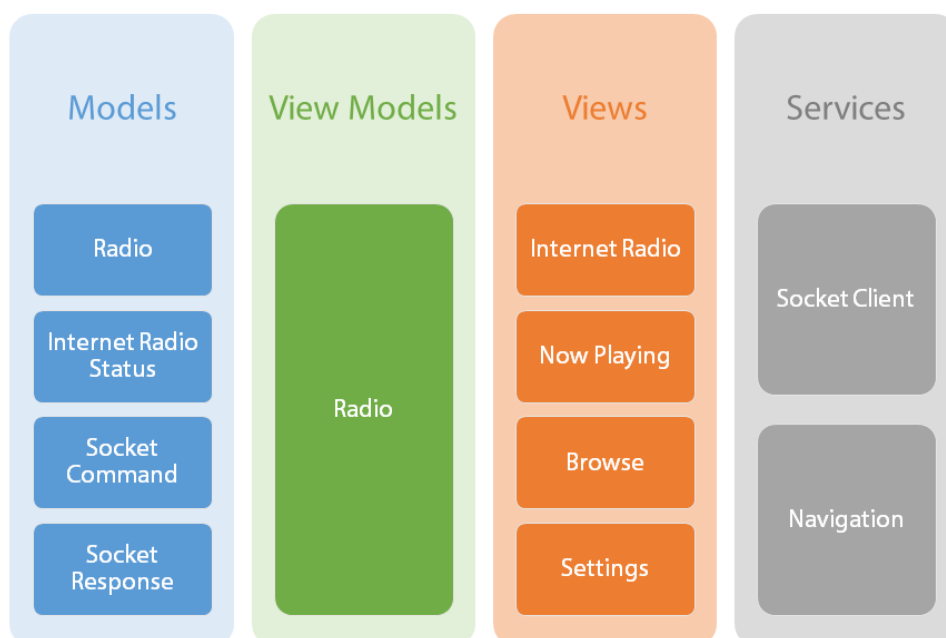
Architektúra aplikácie využíva bežne používaný a odporúčaný návrhový vzor MVVM. Jeho hlavnou výhodou je oddelenie aplikačného kódu od závislosti na zobrazení. To umožňuje oveľa lepšiu prenositeľnosť kódu nie len medzi rôznymi aplikáciami, ale dokonca aj

rozdielnymi platformami. Návrhový vzor rozdeľuje aplikačný kód do troch špecifických kategórii:

- **Models** – Za modely sú považované triedy, ktorých inštancie sú využívané pre tvorbu aplikačnej logiky.
- **ViewModels** – ViewModel využíva inšancií tried modelov a poskytuje dáta, príkazy a ovládače udalostí pre ich spracovanie.
- **Views** – Jedná sa o používateľské zobrazenie (XAML súbory), do ktorého sú naviazané dáta, príkazy a ovládače udalostí z príslušnej inštancie triedy ViewModel.

### 10.1.2 Aplikačný diagram

Aplikačný diagram zobrazuje rozdelenie tried do príslušných vrstiev architektúry aplikácie. Okrem troch základných vrstiev, ktorých význam bol diskutovaný vyššie, obsahuje štvrtú kategóriu nazývanú Services. Tá slúži pre aplikačné služby ako je navigačná služba alebo komunikačný klient pre vzdialenú obsluhu rádia. Vzhľadom na to, že triedy modelov a služieb nie sú využívané len klientskou aplikáciou, nie sú priamou súčasťou jej zdrojových súborov. Triedy sú do projektu pripojené pomocou referencií z triednych knižníc, čo zabráňuje duplikácii a viacnásobným zmenám v zdrojových kódov v prípade ich údržby.



Obrázok 26 Aplikačný diagram

### 10.1.3 Viazanie dát a udalostí

Na účely viazania dát, príkazov a ovládačov udalostí medzi triedou ViewModel a používateľským zobrazením je použité rozšírenie x:Bind, ktorého výhody a funkcionality boli diskutované v teoretickej časti. Viazanie dát pracuje v troch módoch, ktoré upresňujú či je hodnota v zobrazení aktualizovaná v prípade jej zmien a či ju je možné meniť priamo zo zobrazenia. Medzi tieto módy patrí:

- **OneTime (jednorazový)** – hodnota je aktualizovaná len jeden-krát, pri inicializácii zobrazenia.
- **OneWay (jednosmerný)** – hodnota je aktualizovaná v prípade zmeny hodnoty vlastnosti v triede ViewModel.
- **TwoWay (obojsmerný)** – rozširuje jednosmerný mód o schopnosť aktualizovať hodnotu vlastnosti v triede ViewModel priamo z používateľského zobrazenia.

```
<TextBlock
    Text="{x:Bind RadioViewModel.InternetRadioStatus.State, Mode=OneWay}"
/>
```

*Zdrojový kód 27 Naviazanie hodnoty dát a deklarácia módu viazania*

```
<AppBarButton Click="{x:Bind RadioViewModel.RefreshInternetRadioData}"/>
```

*Zdrojový kód 28 Naviazanie ovládača udalosti*

Na to, aby ViewModel informoval používateľské zobrazenie a bol informovaný o zmene hodnoty vlastnosti, je nutné implementovať do jeho triedy rozhranie INotifyPropertyChanged. Rovnako tak je potrebné pri nastavovaní hodnoty vlastnosti vyvolať udalosť tohto rozhrania. Na tieto účely je vytvorená abstraktná trieda ViewModelBase.

```
[DataContract]
public abstract class ViewModelBase : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    public virtual void RaisePropertyChanged(string propertyName)
    {
        this.PropertyChanged?.Invoke(this, new
        PropertyChangedEventArgs(propertyName));
    }
}
```

*Zdrojový kód 29 Abstraktná trieda ViewModelBase*

```
private InternetRadioStatus _internetRadioStatus = new InternetRadioStatus();
public InternetRadioStatus InternetRadioStatus
{
    get
```

```

    {
        return _internetRadioStatus;
    }
    set
    {
        _internetRadioStatus = value;
        RaisePropertyChanged(nameof(InternetRadioStatus));
        NowPlayingView.Current.UpdatePlayPauseState();
    }
}

```

*Zdrojový kód 30 Deklarácia vlastnosti*

## 10.2 Navigačná služba

Na rozdiel od predchádzajúcej verzie, UWP priamo neposkytuje triedu pre navigačnú službu. Z tohto dôvodu je vytvorená pomocná služba `NavigationService`, ktorá uľahčuje prechod medzi jednotlivými stránkami aplikácie. Služba je implementovaná ako statická trieda a obsahuje funkcie pre registráciu ovládača udalosti tlačidla späť, prechod na žiadanú stránku, prechod vpred a vzad a aktualizáciu viditeľnosti tlačidla späť pre zariadenia typu desktop. Výhodou takejto statickej služby je možnosť navigácie kdekoľvek z aplikácie zadáním parametra funkcie služby. Okrem tohto parametra funkcia preberá aj nepovinný ľubovoľný objekt ako parameter nesúci informáciu nasledujúcej stránky. Takúto navigáciu demonštruje nasledujúca ukážka.

```
NavigationService.Navigate(typeof(NewRadioPage));
```

*Zdrojový kód 31 Použitie navigačnej služby*

```

public static Frame RootFrame
{
    get { return Window.Current.Content as Frame; }
}

public static void Navigate(Type type, object args = null)
{
    if(RootFrame.Content != null)
    {
        RootFrame.Navigate(type, args);
    }
}

```

*Zdrojový kód 32 Implementácia navigačnej funkcie*

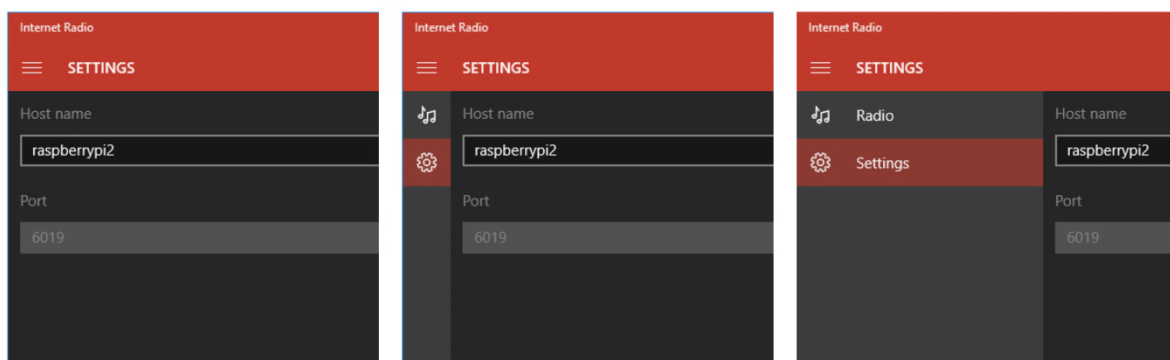
Keďže sa v tejto aplikácii jedná o jednoduchú navigáciu medzi stránkami, služba neumožňuje zmenu ovládača udalosti pre špecifickú stránku. V takomto prípade by ju nebolo možné použiť globálne, ale bolo by potrebné ju registrovať pre každú stránku podliehajúcu navigácii zvlášť.

## 10.3 Tvorba používateľského rozhrania

Tvorba jednotného používateľského rozhrania pre zariadenia s rôznymi rozlíšeniami obrazoviek nemusí byť vždy jednoduchou a vhodnou záležitosťou. Z tohto dôvodu je výhodne dodržiavať základné dizajnové manuály vytvorené výrobcom platformy. Okrem toho, že tieto manuály ponúkajú v praxi overené techniky, rovnako tak upozorňujú na situácie, kedy je ich použitie nevhodné. Klientská aplikácia kombinuje a prispôsobuje viaceré odporúčané navigačné a adaptívne techniky tak, aby zaistila jednoduchosť ovládania a efektívnosť využitia dostupnej plochy aplikácie pre širokú škálu typov a veľkostí zariadení.

### 10.3.1 Zobrazenie s navigačnou sekciou

Zobrazenie s navigačnou sekciou je technika určená pre posun medzi rovnocennými (peer-to-peer) stránkami obsahu. S prihliadnutím na to, že veľkosť navigačnej časti sa v závislosti na veľkosti obrazovky aplikácie mení, prípadne skrýva, je jej použitie vhodnejšie pre stránky, medzi ktorými nie je očakávaný častý prechod. Táto technika je v aplikácii využitá pre posun medzi obsluhou prehrávania rádiových staníc a zobrazením pre nastavenia vzdialenej komunikácie s rádiom. Aplikácia demonštruje implementáciu tejto techniky pomocou ovládacieho prvku SplitView a vytvára nový prispôsobiteľný prvok pre ďalšie použitie v tej istej alebo inej aplikácii.



Obrázok 27 Zobrazenie s adaptívnou navigačnou sekciou

Ako už bolo spomenuté, pre efektívne využitie dostupného miesta obrazovky je veľkosť navigačnej ponuky, prípadne jeho viditeľnosť aktualizovaná. Pri mobilných zariadeniach alebo úzkom okne aplikácie je navigačná sekcia skrytá a jej zobrazenie je plne závislé na stave navigačného tlačidla v ľavom hornom rohu aplikácie. So zväčšujúcou sa šírkou okna sa postupne mení mód zobrazenia tejto sekcie na kompaktný s prekrytím, kedy je v ľavej časti pevne umiestnený navigačný pás s príslušnými ikonami zobrazení. V oboch týchto



módach je po stlačení navigačného tlačidla okno aktuálnej stránky čiastočne prekryté sekciou navigačných tlačidiel. V poslednom prípade, kedy je šírka okna aplikácie väčšia alebo rovná šírke 1024 pixelov, je zobrazenie navigačnej sekcie plne odkryté a namiesto prekrytia časti zobrazenia s aktuálnou stránkou sa stáva jej rovnocennou súčasťou. Použitím ovládacieho prvku SplitView je docielenie takejto funkcionality takmer triviálne. V rámci vizuálnych stavov aktivovaných pomocou spúšťačov minimálnej šírky okna aplikácie je mód zobrazenia ovládacieho prvku zmenený. Nasledujúci kód demonštruje definovanie vizuálnych stavov zabezpečujúcich zmeny zobrazenia.

```
<VisualStateManager.VisualStateGroups>
  <VisualStateGroup>
    <!--tablet-->
    <VisualState>
      <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="720"/>
      </VisualState.StateTriggers>
      <VisualState.Setters>
        <Setter Target="HamburgerMenuSplitView.DisplayMode"
Value="CompactOverlay"/>
        <Setter Target="HamburgerMenuToggleButton.IsChecked" Value="False"/>
      </VisualState.Setters>
    </VisualState>
    <!--desktop-->
    <VisualState>
      <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="1024"/>
      </VisualState.StateTriggers>
      <VisualState.Setters>
        <Setter Target="HamburgerMenuSplitView.DisplayMode"
Value="CompactInline"/>
        <Setter Target="HamburgerMenuToggleButton.IsChecked" Value="True"/>
      </VisualState.Setters>
    </VisualState>
  </VisualStateGroup>
</VisualStateManager.VisualStateGroups>
```

### *Zdrojový kód 33 Zmeny stavu zobrazenia navigačnej sekcie*

Novo vytvorený ovládací prvok HamburgerMenu preberá informácie o navigačných sekciách a ich zobrazeniach prostredníctvom poľa špeciálnej triedy NavLink. Tá uchováva vlastnosti názvu, ikony, a inštanciu ovládacieho prvku, ktorý má byť zobrazený. Následná zmena výberu jedného z navigačných odkazov inicializuje ovládač udalosti pre zmenu aktuálnej sekcie. Ten jednoduchým spôsobom nahradí obsah prvku sekcie za nový.

```
private void HamburgerMenuNavigationListView_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    NavLink navLink = HamburgerMenuNavigationListView.SelectedItem as
NavLink;
    HamburgerMenuSplitView.Content = navLink.Control;
```

```

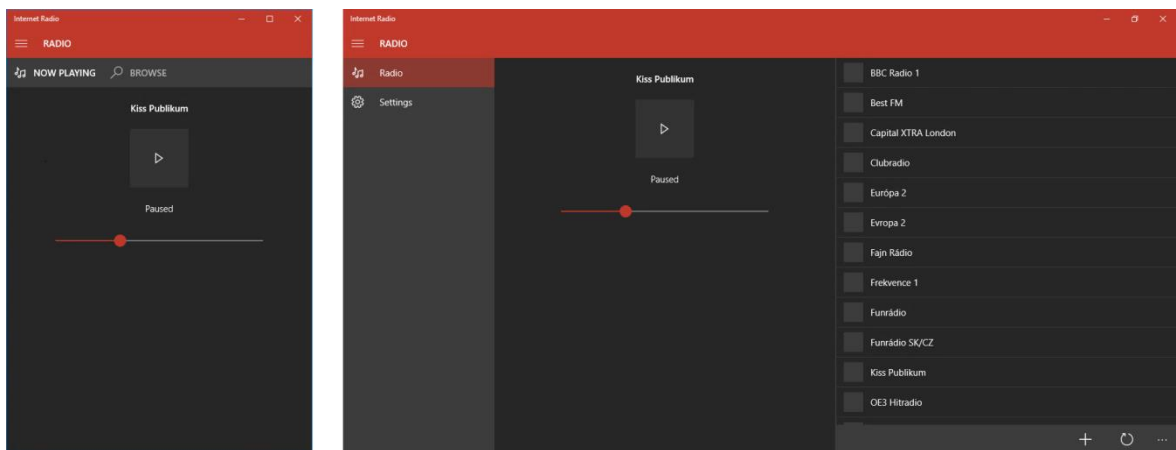
    if (HamburgerMenuSplitView.DisplayMode == SplitViewDisplayMode.Overlay ||
        HamburgerMenuSplitView.DisplayMode == SplitViewDisplayMode.CompactOverlay)
        HamburgerMenuToggleButton.IsChecked = false;
}

```

*Zdrojový kód 34 Ovládač udalosti zmeny výberu aktuálnej sekcie*

### 10.3.2 Zmena zoskupenia zobrazení

Stránka pre vzdialenú obsluhu internetového rádia (RadioView) združuje dve jednotlivé zobrazenia. Prvé z nich (NowPlayingView) obsahuje ovládacie prvky pre zmenu stavu prehrávania a hlasitosti. Druhé zobrazenie (BrowseView) poskytuje výber zo zoznamu dostupných rádiových staníc. Zobrazenia sú umiestnené vedľa seba, pričom každé z nich využíva presne polovicu dostupnej šírky. Avšak takýto spôsob rozdelenia obrazovky na dve časti nie je vhodný pre zariadenia s nižším rozlíšením a preto sú v tomto prípade umiestnené do ďalšieho navigačného prvku Pivot. Pivot slúži pre zobrazenie jednotlivých stránok v záložkách, medzi ktorými je navigácia realizovaná gestami pohybu do strán alebo stlačením odkazov v jeho hornej časti. Podobne ako v predchádzajúcej ukážke zoskupenia podliehajú zmene viditeľnosti pri zväčšení alebo zmenšení okna aplikácie. Okrem zmeny viditeľnosti, zoskupenia využívajú odloženia vytvorenia `DeferLoadStrategy="Lazy"`. To pri štarte aplikácie zabezpečuje vytvorenie len jedného zoskupujúceho prvku, ktorý je určený pre danú šírku okna.



*Obrázok 28 Porovnanie zoskupenia zobrazení*

```

<!--vizuálne stavy-->
<VisualStateManager.VisualStateGroups>
    <VisualStateGroup x:Name="InternetRadioScreensVisualStateGroup">
        <!--mobilný vizuálny stav-->
        <VisualState>
            <VisualState.StateTriggers>
                <AdaptiveTrigger MinWindowWidth="0"/>
            </VisualState.StateTriggers>
            <VisualState.Setters>

```

```

        <Setter Target="MobileScreen.Visibility" Value="Visible"/>
    </VisualState.Setters>
</VisualState>
<!--desktopový vizuálny stav-->
<VisualState>
    <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="1024"/>
    </VisualState.StateTriggers>
    <VisualState.Setters>
        <Setter Target="DesktopScreen.Visibility" Value="Visible"/>
    </VisualState.Setters>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>

<!--mobilná obrazovka-->
<Grid
    x:Name="MobileScreen"
    x:DeferLoadStrategy="Lazy"
    Visibility="Collapsed">
    <Pivot>
    ...
    </Pivot>
</Grid>

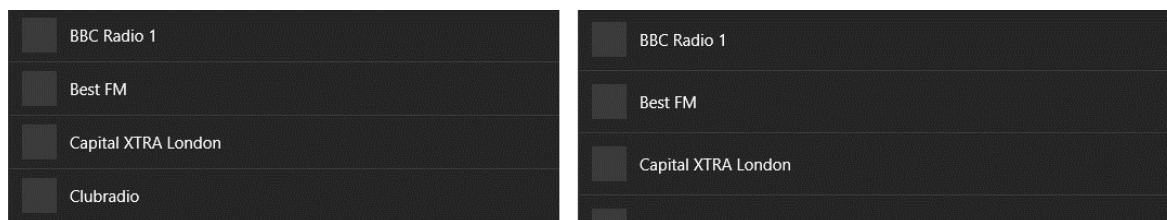
<!--desktop screen-->
<Grid
    x:Name="DesktopScreen"
    x:DeferLoadStrategy="Lazy"
    Visibility="Collapsed">
    ...
</Grid>

```

*Zdrojový kód 35 Zmena zoskupenia zobrazení*

### 10.3.3 Prispôsobenie zobrazenia pri zmene formy interakcie

Okrem vyššie uvedených zmien v zobrazení aplikácia prispôsobuje veľkosť položiek pre jednotlivé formy interakcie. Pokiaľ používateľ aplikáciu ovláda pomocou myši, jeho citlivosť je vyššia a umožňuje mu jednoducho manipulovať aj s menšími položkami. Pri dotyku prstom sú položky mierne zväčšené, tak aby ich ovládanie bolo jednoduchšie.



*Obrázok 29 Prispôsobenie zobrazenia pri zmene formy interakcie*

Nasledujúci zdrojový kód demonštruje implementáciu adaptívneho spúšťača na základe zmeny formy módu interakcie používateľa. Pri zmene veľkosti okna spúšťač porovnáva

žiadaný mód s aktuálnym. Na základe zhody je spúšťač aktivovaný alebo deaktivovaný prostredníctvom funkcie SetActive.

```
public class UserInteractionModeTrigger : StateTriggerBase
{
    private UserInteractionMode _userInteractionMode;
    public UserInteractionMode UserInteractionMode
    {
        get { return _userInteractionMode; }
        set { _userInteractionMode = value; WindowSizeChanged(); }
    }

    public UserInteractionModeTrigger()
    {
        Window.Current.SizeChanged += new WindowSizeChangedEventArgs((object
sender, WindowSizeChangedEventArgs e) =>
        {
            WindowSizeChanged();
        }));
    }

    private void WindowSizeChanged()
    {
        SetActive(UIViewSettings.GetForCurrentView().UserInteractionMode.Equals(UserInteract
ionMode));
    }
}
```

#### *Zdrojový kód 36 Implementácia spúšťača formy interakcia*

Následné použitie spúšťača v XAML kóde:

```
<VisualState.StateTriggers>
    <triggers:UserInteractionModeTrigger UserInteractionMode="Touch"/>
</VisualState.StateTriggers>
```

#### *Zdrojový kód 37 Použitie spúšťača formy interakcie*

## ZÁVER

Cieľom bakalárskej práce bola analýza a demonštrácia vývoja univerzálnych aplikácií pre platformu Universal Windows Platform. Práca sa zaoberala najmä možnosťami prispôsobenia používateľského rozhrania rôznym veľkostiam rozlíšení a spôsobom interakcie. V prvej časti práce je uvedený stručný prehľad novínok operačného systému Windows 10, v rámci ktorého bola platforma UWP predstavená.

Ďalej práca objasňuje základnú myšlienku tvorby jednej aplikácie pre viacero typov zariadení. V tejto oblasti je spomenutý proces vývoja platformy a jej vznik ako rozšírenie aplikačného modelu Windows Runtime. Práca poskytuje pohľad na rôzne technológie vývoja aplikácií pre operačný systém Windows 10. Okrem vývoja univerzálnych aplikácií naďalej existuje možnosť vývoja aplikácií pre .NET Framework. Tie avšak napriek väčšej popularite a schopnostiam neposkytujú spomínané multiplatformové benefity a výhody spojené s jednotným aplikačným obchodom. UWP navyše disponuje možnosťami implementácie aplikácií vo viacerých programovacích jazykoch a kombinácie ich komponentov. Za účelom zvýšenia výkonu aplikácií bola vyvinutá optimalizovaná implementácia aplikačného rámca .NET Core. Implementácia využíva statickej kompilácie a umožňuje zrýchlenie štartu aplikácií a zníženie nárokov na pamäť procesu.

Programovací jazyk C# získal vo verzii 6.0 zopár syntaktických vylepšení zameraných na tvorbu čistejšieho a prehľadnejšieho zdrojového kódu. Pre podporu nových adaptívnych techník a zvýšenia výkonu vykresľovania používateľského rozhrania bolo v jazyku XAML pridaných viacero rozšírení. Na zabezpečenie lepšej konzistentnosti zobrazenia na veľkom množstve rozlíšení obrazoviek a zjednodušenie procesu prispôsobenia používateľského rozhrania, disponuje UWP špeciálnym algoritmom pre škálovanie veľkostí. Ten je založený na bežnej fyzickej vzdialenosti pozorovateľa od obrazovky a jej hodnoty DPI. Adaptívne techniky ponúkajú viacero alternatív prispôsobenia používateľského rozhrania veľkostiam rozlíšení obrazoviek alebo rodinám zariadení. Výhodou sú možnosti prispôsobenia nielen v jazyku XAML, ale aj prostredníctvom kódu na pozadí. Medzi ďalšie výhody patrí možnosť implementácie vlastných spúšťačov inicializujúcich zmeny používateľského rozhrania, ktoré môžu byť aktivované na základe ľubovoľných podmienok programátora a schopností platformy. Výhodou je, že platforma automaticky zabezpečuje vyvolanie správnej udalosti pre rôzne formy interakcií. Avšak pre prispôsobenie používateľského rozhrania rôznym formám je programátor odkázaný na implementáciu vlastných spúšťačov a podmienok založených na dostupných rozhraniach.

V praktickej časti je na autorom vytvorenej prípadovej štúdii demonštrovaný proces tvorby univerzálnej aplikácie pre zariadenia Windows 10 a jej prepojenie s čiastkovou aplikáciou na pozadí zariadenia kategórie Internet vecí - Raspberry Pi. Prostredníctvom ukážok zdrojového kódu sú objasňované implementácie tvorby štruktúry, navigácie, adaptívneho používateľského rozhrania, práce s úložiskom nastavení a sieťovej komunikácie s druhou čiastkovou aplikáciou pre prehrávanie rádiových staníc. Implementácia využíva odporúčaný návrhový vzor MVVM a demonštruje viazanie dát a ovládačov udalostí pomocou nového rozšírenia `x:Bind`. Výsledná aplikácia bola otestovaná na zariadeniach typu mobilný telefón, tablet, osobný počítač a Raspberry Pi 2. Po otestovaní je zrejmé, že UWP je flexibilný nástroj pre tvorbu multiplatformových aplikácií.

## ZOZNAM POUŽITEJ LITERATÚRY

- [1] KUMAR, Senthil, Lohith GOUDAGERE NAGARAJ, Pathik RAWAL a Pryank ROHILLA. *Windows 10 Development Recipes: A Problem-Solution Approach in HTML and JavaScript*. New York: Apress Media, 2016. ISBN 978-1-4842-0720-8.
- [2] CHAUHAN, Shen a Andy WIGLEY. The Universal Windows Platform. *Channel 9* [online]. Microsoft, 2015 [cit. 2016-02-20]. Dostupné z: <https://channel9.msdn.com/events/Windows/Developers-Guide-to-Windows-10-RTM/The-Universal-Windows-Platform>
- [3] BOTT, Ed. *Introducing Windows 10 for IT Professionals, Preview Edition*. Microsoft Press, 2015. ISBN 978-0-7356-9696-9.
- [4] MOEMEKA, Edward a Elizabeth MOEMEKA. *Real World Windows 10 Development*. 2. ed. New York: Apress Media, 2015. ISBN 978-1-4842-1450-3.
- [5] EKUAN, Martin. What's a Universal Windows Platform (UWP) app? *Microsoft Developer Network* [online]. Microsoft, 2016 [cit. 2016-03-04]. Dostupné z: <https://msdn.microsoft.com/en-us/windows/uwp/get-started/whats-a-uwp>
- [6] WHITNEY, Tyler. Guide to Universal Windows Platform (UWP) apps. *Microsoft Developer Network* [online]. Microsoft, 2016 [cit. 2016-03-04]. Dostupné z: <https://msdn.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide>
- [7] LANDER, Richard. Introducing .NET Core: A Cross-Platform Runtime. *Channel 9* [online]. Microsoft, 2015 [cit. 2016-03-12]. Dostupné z: <https://channel9.msdn.com/events/Visual-Studio/Connect-event-2015/104>
- [8] LANDRY, Nick a David GIARD. Episode 420: Nick Landry on .NET Framework and .NET Core. *Channel 9* [online]. Microsoft, 2016 [cit. 2016-03-12]. Dostupné z: <https://channel9.msdn.com/Blogs/Technology-and-Friends/tf420>
- [9] LANDWERTH, Immo. Introducing .NET Core. In: *.NET Blog* [online]. Microsoft Corporation, 2014 [cit. 2016-03-13]. Dostupné z: <https://blogs.msdn.microsoft.com/dotnet/2014/12/04/introducing-net-core/>
- [10] FARKAS, Shawn. .NET Native. *Channel 9* [online]. Microsoft, 2014 [cit. 2016-03-14]. Dostupné z: <https://channel9.msdn.com/events/Visual-Studio/Connect-event-2014/112>

- [11] Inside .NET Native. *Channel 9* [online]. Microsoft, 2014 [cit. 2016-03-14]. Dostupné z: <https://channel9.msdn.com/Shows/Going+Deep/Inside-NET-Native>
- [12] Introduction to the C# Language and the .NET Framework. *Microsoft Developer Network* [online]. Microsoft [cit. 2016-03-24]. Dostupné z: <https://msdn.microsoft.com/en-us/library/z1zx9t92.aspx>
- [13] SHARP, John. *Microsoft Visual C# 2008: krok za krokem*. Brno: Computer Press, 2008. Krok za krokem. ISBN 9788025120279.
- [14] DRAYTON, Peter, Ben ALBAHARI a Ted NEWARD. *C# v kostce: pohotová referenční příručka*. Praha: Grada, 2003. ISBN 8024704439.
- [15] SHARP, John. *Microsoft Visual C# Step by Step, 8th Edition*. 8. ed. Redmond, Washington: Microsoft Press, 2015, pages cm. ISBN 9781509301041.
- [16] Co je nového v C# 6.0. In: *Czech MSDN Blog* [online]. Microsoft, 2015 [cit. 2016-03-25]. Dostupné z: <https://blogs.msdn.microsoft.com/vyvojari/2015/05/11/co-je-novho-v-c-6-0/>
- [17] XAML Overview (WPF). *Microsoft Developer Network* [online]. Microsoft [cit. 2016-04-01]. Dostupné z: [https://msdn.microsoft.com/cs-cz/library/ms752059\(v=vs.110\).aspx](https://msdn.microsoft.com/cs-cz/library/ms752059(v=vs.110).aspx)
- [18] LACKO, Luboslav. *Vývoj aplikací pro Windows 8.1 a Windows Phone*. Brno: Computer Press, 2014. ISBN 9788025138229.
- [19] PETZOLD, Charles. *Mistrovství ve Windows Presentation Foundation: [aplikace = kód + markup]*. Brno: Computer Press, 2008. ISBN 9788025121412.
- [20] WIGLEY, Andy a Shen CHAUHAN. Improving XAML Performance. *Microsoft Virtual Academy* [online]. Microsoft, 2015 [cit. 2016-04-01]. Dostupné z: [https://mva.microsoft.com/en-US/training-courses/a-developer-s-guide-to-windows-10-12618?l=pZhtA3pRB\\_9105095281](https://mva.microsoft.com/en-US/training-courses/a-developer-s-guide-to-windows-10-12618?l=pZhtA3pRB_9105095281)
- [21] WALKER, Jim. X:Phase attribute. *Microsoft Developer Network* [online]. Microsoft, 2016 [cit. 2016-04-04]. Dostupné z: <https://msdn.microsoft.com/en-us/windows/uwp/xaml-platform/x-phase-attribute>
- [22] WALKER, Jim. X:DeferLoadStrategy attribute. *Microsoft Developer Network* [online]. Microsoft, 2016 [cit. 2016-04-04]. Dostupné z: <https://msdn.microsoft.com/en-us/windows/uwp/xaml-platform/x-deferloadstrategy-attribute>



- [23] WALKER, Jim. {x:Bind} markup extension. *Microsoft Developer Network* [online]. Microsoft, 2016 [cit. 2016-04-05]. Dostupné z: <https://msdn.microsoft.com/en-us/windows/uwp/xaml-platform/x-bind-markup-extension>
- [24] WIGLEY, Andy a Shen CHAUHAN. Improvements to XAML Data Binding. *Microsoft Virtual Academy* [online]. Microsoft, 2015 [cit. 2016-04-05]. Dostupné z: [https://mva.microsoft.com/en-US/training-courses/a-developer-s-guide-to-windows-10-12618?l=Mt1woVqRB\\_8405095281](https://mva.microsoft.com/en-US/training-courses/a-developer-s-guide-to-windows-10-12618?l=Mt1woVqRB_8405095281)
- [25] YOSIFOVICH, Pavel. Windows 8 Store Apps: Class Library vs. Windows Runtime Component. In: *Pavel's Blog* [online]. Microsoft, 2012 [cit. 2016-04-07]. Dostupné z: <http://blogs.microsoft.co.il/pavely/2012/10/30/windows-8-store-apps-class-library-vs-windows-runtime-component/>
- [26] WIGLEY, Andy a Shen CHAUHAN. Application Lifecycle. *Microsoft Virtual Academy* [online]. Microsoft, 2015 [cit. 2016-04-08]. Dostupné z: [https://mva.microsoft.com/en-US/training-courses/a-developer-s-guide-to-windows-10-12618?l=nXtix0pRB\\_5205095281](https://mva.microsoft.com/en-US/training-courses/a-developer-s-guide-to-windows-10-12618?l=nXtix0pRB_5205095281)
- [27] WHITNEY, Tyler. App lifecycle. *Microsoft Developer Network* [online]. Microsoft, 2016 [cit. 2016-04-08]. Dostupné z: <https://msdn.microsoft.com/en-us/windows/uwp/launch-resume/app-lifecycle>
- [28] JACOBS, Mike. Introduction to UWP app design. *Microsoft Developer Network* [online]. Microsoft, 2016 [cit. 2016-04-15]. Dostupné z: <https://msdn.microsoft.com/windows/uwp/layout/design-and-ui-intro>
- [29] WIGLEY, Andy a Shen CHAUHAN. Adaptive UI. *Microsoft Virtual Academy* [online]. Microsoft, 2015 [cit. 2016-04-15]. Dostupné z: [https://mva.microsoft.com/en-US/training-courses/a-developer-s-guide-to-windows-10-12618?l=o3HUevpRB\\_8105095281](https://mva.microsoft.com/en-US/training-courses/a-developer-s-guide-to-windows-10-12618?l=o3HUevpRB_8105095281)
- [30] RelativePanel class. *Microsoft Developer Network* [online]. [cit. 2016-04-16]. Dostupné z: <https://msdn.microsoft.com/en-us/library/windows/apps/windows.ui.xaml.controls.relativepanel.aspx>
- [31] WIGLEY, Andy a Shen CHAUHAN. Adaptive Code. *Microsoft Virtual Academy* [online]. Microsoft, 2015 [cit. 2016-04-16]. Dostupné z: [https://mva.microsoft.com/en-US/training-courses/a-developer-s-guide-to-windows-10-12618?l=4y6SBxpRB\\_5405095281](https://mva.microsoft.com/en-US/training-courses/a-developer-s-guide-to-windows-10-12618?l=4y6SBxpRB_5405095281)

**ZOZNAM POUŽITÝCH SYMBOLOV A SKRATIEK**

.NET	Názov platformy pre vývoj softvéru
API	Application Program Interface
C#	Objektovo orientovaný programovací
CLR	Common Language Runtime
DLL	Dynamic-link library
DPI	Dots per inch
iOS	Mobilný operačný systém spoločnosti Apple
IoT	Internet of things
IP	Internet Protocol
JIT	Just in time
JSON	Dátový formát
LAN	Local area network
LINQ	Language Integrated Query
MSIL	Microsoft Intermediate Language
MVVM	Model View ViewModel
OS	Operačný systém
RT	Runtime
SDK	Software development kit
TCP	Transmission Control Protocol
UWP	Universal Windows Platform
WPF	Windows Presentation Foundation
XAML	Extensible Application Markup Language
XML	Extensible Markup Language

## ZOZNAM OBRÁZKOV

<i>Obrázok 1 Ponuka Štart.....</i>	<i>12</i>
<i>Obrázok 2 Zobrazenie modernej aplikácie v adaptívnom okne.....</i>	<i>12</i>
<i>Obrázok 3 Centrum akcií.....</i>	<i>13</i>
<i>Obrázok 4 Zobrazenie úloh.....</i>	<i>14</i>
<i>Obrázok 5 Cielenie univerzálnych aplikácií [6].....</i>	<i>16</i>
<i>Obrázok 6 Hierarchia rodín zariadení [6].....</i>	<i>17</i>
<i>Obrázok 7 Vybraný obsah univerzálnej sady rozhraní API.....</i>	<i>18</i>
<i>Obrázok 8 Vetvenie aplikačného rámca .NET v roku 2015 [9].....</i>	<i>19</i>
<i>Obrázok 9 Proces statickej kompilácie pomocou .NET Native [10].....</i>	<i>20</i>
<i>Obrázok 10 Nastavenie kompilácie pomocou reťazového nástroja .NET Native.....</i>	<i>20</i>
<i>Obrázok 11 Technológie vývoja aplikácií pre Windows 10.....</i>	<i>21</i>
<i>Obrázok 12 Typy projektov Universal Windows.....</i>	<i>28</i>
<i>Obrázok 13 Štruktúra projektu Blank App.....</i>	<i>29</i>
<i>Obrázok 14 Životný cyklus aplikácie [27].....</i>	<i>30</i>
<i>Obrázok 15 Škálovanie veľkostí na základe vzdialenosti zariadenia [28].....</i>	<i>32</i>
<i>Obrázok 16 Relatívny panel.....</i>	<i>34</i>
<i>Obrázok 17 Zobrazenie na mieru pre unikátnu rodinu zariadenia.....</i>	<i>36</i>
<i>Obrázok 18 Rozšírenia rodín zariadení.....</i>	<i>36</i>
<i>Obrázok 19 Štruktúra aplikácie.....</i>	<i>41</i>
<i>Obrázok 20 Komunikácia klient / server.....</i>	<i>42</i>
<i>Obrázok 21 Zapojenie klient / server zariadení v domácej sieti.....</i>	<i>43</i>
<i>Obrázok 22 SocketServer model.....</i>	<i>43</i>
<i>Obrázok 23 Model SocketClient.....</i>	<i>45</i>
<i>Obrázok 24 Modely SocketRequest a SocketResponse.....</i>	<i>48</i>
<i>Obrázok 25 Navigačný diagram.....</i>	<i>52</i>
<i>Obrázok 26 Aplikačný diagram.....</i>	<i>53</i>
<i>Obrázok 27 Zobrazenie s adaptívnou navigačnou sekciou.....</i>	<i>56</i>
<i>Obrázok 28 Porovnanie zoskupenia zobrazení.....</i>	<i>58</i>
<i>Obrázok 29 Prispôsobenie zobrazenia pri zmene formy interakcie.....</i>	<i>59</i>

**ZOZNAM ZDROJOVÝCH KÓDOV**

<i>Zdrojový kód 1 Novinky v jazyku C# 6.0 – časť 1</i>	23
<i>Zdrojový kód 2 Novinky v jazyku C# 6.0 – časť 2</i>	24
<i>Zdrojový kód 3 Vykresľovanie prvkov rozhrania vo viacerých fázach</i>	26
<i>Zdrojový kód 4 Deklarovanie odloženia vykreslenia prvku</i>	26
<i>Zdrojový kód 5 Príklad volania vykreslenia odloženého prvku</i>	26
<i>Zdrojový kód 6 Príklad použitia rozšírenia x:Bind</i>	27
<i>Zdrojový kód 7 Detekcia predošlého stavu aplikácie</i>	31
<i>Zdrojový kód 8 Detekcia formy aktivácie aplikácie</i>	31
<i>Zdrojový kód 9 Ukladanie dát počas pozastavovania aplikácie</i>	31
<i>Zdrojový kód 10 Adaptívny dizajn pomocou vizuálnych stavov</i>	33
<i>Zdrojový kód 11 Relatívny panel</i>	34
<i>Zdrojový kód 12 Zmena relatívnej veľkosti</i>	34
<i>Zdrojový kód 13 Detekcia rodiny zariadenia</i>	35
<i>Zdrojový kód 14 Kontrola prítomnosti špecifického rozhrania API</i>	37
<i>Zdrojový kód 15 Kontrola prítomnosti špecifickej verzie rozhrania API</i>	37
<i>Zdrojový kód 16 Inicializácia serverovej služby</i>	44
<i>Zdrojový kód 17 Ovládač udalosti nadviazania spojenia</i>	45
<i>Zdrojový kód 18 Nadviazanie pripojenie k serveru</i>	46
<i>Zdrojový kód 19 Úloha odoslania požiadavky na server</i>	47
<i>Zdrojový kód 20 Serializácia dát</i>	47
<i>Zdrojový kód 21 Deserializácia dát</i>	47
<i>Zdrojový kód 22 Označenie atribútmi serializácie</i>	48
<i>Zdrojový kód 23 Rozšírenie pre automatický štart pri spustení systému</i>	50
<i>Zdrojový kód 24 Pomocná metóda pre uloženie dát do lokálnej pamäte aplikácie</i>	50
<i>Zdrojový kód 25 Pomocná metóda pre čítanie dát z lokálnej pamäte aplikácie</i>	51
<i>Zdrojový kód 26 Spustenie rádiovkej stanice</i>	51
<i>Zdrojový kód 27 Naviazanie hodnoty dát a deklarácia módu viazania</i>	54
<i>Zdrojový kód 28 Naviazanie ovládača udalosti</i>	54
<i>Zdrojový kód 29 Abstraktná trieda ViewModelBase</i>	54
<i>Zdrojový kód 30 Deklarácia vlastnosti</i>	55
<i>Zdrojový kód 31 Použitie navigačnej služby</i>	55
<i>Zdrojový kód 32 Implementácia navigačnej funkcie</i>	55

<i>Zdrojový kód 33 Zmeny stavu zobrazenia navigačnej sekcie .....</i>	<i>57</i>
<i>Zdrojový kód 34 Ovládač udalosti zmeny výberu aktuálnej sekcie .....</i>	<i>58</i>
<i>Zdrojový kód 35 Zmena zoskupenia zobrazení .....</i>	<i>59</i>
<i>Zdrojový kód 36 Implementácia spúšťača formy interakcia.....</i>	<i>60</i>
<i>Zdrojový kód 37 Použitie spúšťača formy interakcie .....</i>	<i>60</i>

**ZOZNAM TABULIEK**

<i>Tabuľka 1 Typy implementovaných požiadaviek .....</i>	<i>49</i>
--	-----------

## ZOZNAM PRÍLOH

P I      CD - ROM