České zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Vývoj univerzálních aplikací pro Windows 10

Šimon Kubita

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Šimon Kubita

Informatika

Název práce

Vývoj univerzálních aplikací pro Windows 10

Název anglicky

Development of Windows 10 Universal Apps

Cíle práce

Práce je zaměřena na problematiku tvorby aplikací pro Windows 10 využívajících platformu Universal Apps. Cílem práce je představit tuto platformu a možnosti vývoje univerzálních aplikací a na ukázkovém příkladu demonstrovat vývoj aplikace na této platformě.

Metodika

Metodika řešené bakalářské práce je založena na studiu odborných informačních zdrojů. Na základě analýzy zjištěných poznatků budou shrnuty vlastnosti platformy Universal Apps a popsány specifika vývoje aplikací pro tuto platformu. Dále bude navržena a implementována ukázková aplikace, která bude demonstrovat možnosti této platformy. Postup tvorby bude popsány a dále zhodnocen.

Doporučený rozsah práce

35-40 stran

Klíčová slova

C#, .NET, Windows 10, Universal Apps, platforma, univerzální aplikace

---

Doporučené zdroje informací

MAREŠ, A. 1001 tipů a triků pro C# 2010: Sbírka nejužitečnějších řešení programátorských úloh. Praha: Computer Press, 2011. ISBN 978-80-251-3250-0

NASH, T. C# 2010: Rychlý průvodce novinkami a nejlepšími postupy. Praha: Computer Press, 2010. ISBN 978-80-251-3034-6

SHARP, J. Microsoft Visual C# 2010: Krok za krokem. Praha: Computer Press, 2013. ISBN 978-80-251-3147-3

Předběžný termín obhajoby

2015/16 LS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 20. 2. 2016

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 20. 2. 2016

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 27. 02. 2016

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci „Vývoj univerzálních aplikací pro Windows 10“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené bakalářské práce prohlašuji, že jsem při jejím vytvoření neporušil žádná autorská práva třetích osob.

V Praze dne _____

Poděkování

Touto cestou bych chtěl velice poděkovat Ing. Jiřímu Brožkovi, Ph.D. za pomoc, jeho rady a svědomité vedení bakalářské práce.

Vývoj univerzálních aplikací pro Windows 10

Development of Windows 10 Universal Apps

Souhrn

Teoretická část bakalářské práce je zaměřena obecně na vývoj aplikací pro operační systém Windows 10. Nejdříve představí samotný operační systém Windows 10, poté detailněji popisuje vývojové prostředí využité v praktické části a celkovou problematiku univerzálních aplikací. Dále pak porovnává chování a zobrazení aplikace na jednotlivých typech zařízení s možnostmi vývoje a optimalizace.

Praktická část se zabývá samotným návrhem a vývojem aplikace na platformě Universal Apps. Na začátku představuje použité technologie a jejich specifikaci. Následně obecně nastíní základní struktury a zakládání nového projektu. Na vývoji ukázkové aplikaci demonstruje problematiku vývoje na této platformě. A nakonec ukazuje základní možnosti nasazení hotového projektu.

Summary

The theoretical part is focused on developing application for the Windows 10 operating system. First, introduces Windows 10 in general, then describes development environment used in the practical part in detail and show overall issue of universal applications. After that compares the behavior and application design of various types of devices with potential for development and optimization.

The practical part deals with the design and development of applications on Universal Apps platform. At the beginning the used technology and their specification are introduced. Then the basic structure and foundation of the new project is shown. The issue of development on this platform are then demonstrated on development of s sample application. Finally, it shows the basic deployment options of the finished project.

Klíčová slova: C#, .NET, Windows 10, Universal Apps, platforma, univerzální aplikace

Keywords: C#, .NET, Windows 10, Universal Apps, platform, universal application

Obsah

1. Úvod

Firma Microsoft v červenci roku 2015 představila novou verzi nejpoužívanějšího operačního systému na světě. Tento operační systém má název Windows 10 a je údajně nejosobitějším a nejdokonalejším operačním systémem, této firmy, vůbec. Systém nabízí známé prostředí z předchozích verzí. Ve spoustě věcí včetně nabídky Start, se podobá oblíbené verzi systému Windows 7. Z verze Windows 8, zase přebírá systém dlaždic a mnoho dalšího.

S novým operačním systémem přišla i nová platforma pro vývoj aplikací. Díky této platformě lze vyvíjet univerzální aplikace, které umožňují programátorovi napsat pouze jeden balíček aplikace, a ten půjde spustit na všech zařízeních. O správném zobrazení a přizpůsobení grafického uživatelského rozhraní se postará samotný operační systém a programátorovi ušetří mnoho práce s odlaďováním aplikace na různá zařízení. Platforma Universal Apps umožňuje použití rozšiřujících sad SDK pro jednotlivá zařízení, které umožňuje použití specifických funkcí pro daný typ zařízení.

Poprvé je společné jádro na všech platformách. Pro uživatelské rozhraní jsou pouze dva frameworky. Jeden Framework uživatelského rozhraní ve formátu XAML a druhý ve formátu HTML.

Tato bakalářská práce ukazuje možnosti těchto aplikací a jejich velkou rozmanitost napříč všemi zařízeními, na kterých může být tato aplikace spuštěna. Následně rozebere trochu detailněji samotné vytváření univerzálních aplikací. Představí možnosti použití různých technologií a vybere technologie, které následně využije na vývoj ukázkové aplikace. Tento vývoj bude dokumentovat a prezentovat na jednoduchých ukázkách kódu. Po úspěšném vytvoření ukázkového projektu ukáže možnosti použití.

2. Cíl práce a metodika

2.1. Cíl práce

Cílem bakalářské práce je ukázat možnosti vývoje aplikací pro operační systém Windows 10 na platformě Universal App, vlastnosti této platformy na jednotlivých zařízeních a způsob vytváření a odladění aplikace na jednotlivých zařízeních.

V praktické části bakalářské práce je cílem vytvořit ukázkovou aplikaci ve vývojovém prostředí Visual Studio 2015, na které se budou demonstrovat jednotlivé vlastnosti platformy. Vývoj aplikace bude dokumentován a měl by osvětlit základní problematiku vývoje jednoduché aplikace na Windows 10. Součástí praktické části budou ukázky kódu ze zdrojového kódu aplikace. S pomocí těchto ukázek kódu a komentářů bude možné vytvořit vlastní aplikaci. Základní postup a struktura projektu bude vysvětlena.

2.2. Metodika

Za použití získaných informací z použitých odborných zdrojů a praxe v programování, budou popsány principy vývoje aplikací pro operační systém Windows 10 na platformě Universal App.

Nabyté znalosti z teoretické části budou následně využity pro vytvoření ukázkové aplikace, která bude demonstrovat vlastnosti platformy. Technologie použité v praktické části budou detailně vysvětleny v úvodu praktické části bakalářské práce a následně použity při samotném vývoji. Krátkými ukázkami kódu bude prezentován základ zdrojového kódu aplikace. Tyto ukázky budou detailně popsány a vysvětleny, aby bylo možné s nabytými znalostmi bez problémů vytvořit jednoduchou aplikaci na tuto platformu.

3. Teoretická část

3.1 Windows 10

3.1.1 Úvod

Firma Microsoft v průběhu roku 2015 vydala nový operační systém s názvem Windows 10. Vydání ovšem nepřišlo najednou, ale probíhalo postupně, prostřednictvím programu Windows Insider. V tomto programu Microsoft zpřístupňoval účastníkům, nedokončené a neodladěné verze operačního systému. Své zákazníky tímto programem vyzíval, aby přispěli k vývoji a budoucnosti Windows 10. Zpětná vazba účastníků byla jedním z nejdůležitějších faktorů při úspěšném dokončení funkčního operačního systému. [1]

Konečná verze pro veřejnost vyšla 29. července a Microsoft ji uváděl jako přelomovou verzi systému, který je to nejlepší co doposud vytvořili. Přelomový hlavně díky tomu, že sjednocuje všechna zařízení. Počítače, notebooky, tablety, chytré telefony, Xbox a další zařízení budou sjednoceny pod jedním systémem, který bude pro každý typ zařízení trochu modifikovaný, ale jádro bude vždy stejné.

Měsíc po vydání oficiální verze, je systém nainstalován na 75 milionech zařízení. Microsoft předpokládá, že do roku 2018 toto číslo vzroste na jednu miliardu.

Tak rychlé rozšíření bylo způsobeno hlavně jednoduchým upgradem z předchozích verzí Windows 7 a Windows 8. Microsoft tyto upgrady nabídl po dobu trvání jednoho roku zdarma pro držitele licencí na přechozí verze. Upgrade neměl žádný vliv na stávající soubory, ani programy, což byl jeden z hlavních důvodů, proč se lidé nebránili přechodu na nový systém.

3.1.2 Specifikace

Pro upgrade je potřeba, aby Váš hardware splňoval určité požadavky. Ať už na počítači, chytrém telefonu, nebo na jakémkoli jiném zařízení. Musíte používat nejnovější verzi systému, jako je Windows 7, nebo Windows 8.1. Procesor musí být taktován minimálně na 1 GHz, operační paměť musí dosahovat minimálně 1 GB a celý systém zabere na pevném disku 16 až 20 GB, podle verze. [2]

Po úspěšném upgradu budou uživatelé verze Home dostávat aktualizace přes Windows Update automaticky, bez nutnosti potvrzení. U verze Pro a Enterprise bude možnost aktualizace odložit, ale jen na omezenou dobu.

Windows 7[2]	
Edice před upgradem	Edice po upgradu
Windows 7 Starter	Windows 10 Home
Windows 7 Home Basic	
Windows 7 Home Premium	
Windows 7 Professional	Windows 10 Pro
Windows 7 Ultimate	

Windows 8[3]	
Edice před upgradem	Edice po upgradu
Windows Phone 8.1[5]	Windows 10 Mobile
Windows 8.1[4]	Windows 10 Home
Windows 8.1 Pro	Windows 10 Pro
Windows 8.1 Pro pro studenty	

Obrázek 1 - Edice Windows 10 po upgradu ZDROJ: www.microsoft.com

Systém přestal podporovat několik zastaralých funkcí. Mezi ně patří například funkce správy mobilních zařízení, miniaplikace na ploše z Windows 7, předinstalované hry z Windows 7 a některé další funkcí. [2]

Cortana, jakožto inteligentní virtuální asistentka, je v současné době podporována v sedmi zemích. Tyto země jsou Spojené státy, Spojené království, Čína, Francie, Itálie, Německo a Španělsko. Spolehlivost rozpoznávání hlasu se bude lišit v závislosti na mikrofonu zařízení.

Funkce Windows Hello, se kterou se můžete přihlásit do systému pouhým pohledem, vyžaduje speciální infračervenou kameru pracující za všech světelných podmínek, nebo čtečku otisků prst, která podporuje architekturu Windows Biometric Framework.

Continuum umožňuje uživateli přizpůsobit rozhraní v závislosti na situaci. Tato funkce je k dispozici, ve všech desktopových edicích Windows, ručním zapnutím a vypnutím tabletu v Centru akcí. Při splnění určitých požadavků je možné nastavit automatické přepínání do režimu tabletu. Dostupnost této technologie pro telefony je při uvedení na trh omezena na vybrané prémiové telefony. Příslušenství kompatibilní s technologií Continuum se prodává samostatně. [2]

3.2 Visual Studio 2015

3.2.1 Úvod

Microsoft uvedl nové Visual Studio 2015, také s novou verzí frameworku .NET Framework 4.6. Hlavní novinkou je nástroj pro tvorbu univerzálních aplikací pro Windows 10. Další výraznou novinkou je podpora vývoje aplikací pro platformy Android a iOS.

Plná verze Visual Studia 2015 obsahuje podporu pro mnoho jazyků, jako například C#, Visual Basic, Javu, JavaScript a mnoho dalších. K dispozici jsou knihovny Android SDK a

Xamarin zahrnující podporu pro iOS zařízení. Pro vývoj na cizích platformách je možné použít nástroj Visual Studio Tools for Apache cordova. [3]

Vývojové prostředí se nabízí ve třech verzích, a to Community, Professional a Enterprise. Community verze je zdarma, pro nekomerční využití. Verze se liší funkcemi, v nejvyšší je zahrnuta plná podpora týmové práce nebo nástroje pro podrobné testování.

3.2.2 Novinky ve verzi 2015

Instalace Visual Studia 2015 byla rozdělena na jednotlivé části zaměřené na užší zaměření vývoje. Stačí si nainstalovat části, které opravdu použijete. Základní instalace zahrnuje nástroje pro vývoj webových a desktopových aplikací v C# a Visual Basic. Pokud chcete vyvíjet v ostatních odvětvích, zvolíte si vlastní možnosti instalace a vyberete knihovny třetích stran k instalaci.

V nové verzi jsou možnosti efektivního přihlašování uživatelů online, i v případě, že máte více účtů. Po přihlášení budete automaticky přihlášeni ke všem instancím Visual Studia ve svém počítači. Přihlášení automaticky spustí sdílení osobního nastavení. [4]

Visual Studio 2015 podporuje multiplatformní vývoj pro mobilní zařízení. Můžete napsat aplikace a hry určené pro iOS, Android a Windows, které budou sdílet stejný kód.

Pro multiplatformní vývoj můžete využít například Xamarin. Xamarin je mobilní rozhraní vytvářející nativní aplikace za pomocí jazyku C#.

Velmi důležitou novinkou je platforma univerzálních aplikací. S touto platformou přichází jedno společné Windows jádro. To dovoluje spustit jednu aplikaci na jakémkoli zařízení s Windows 10, od počítačů až po mobilní zařízení. [4]

3.3 Platforma univerzálních aplikací

Obrázek 2- Universal Windows Platform – ZDROJ: [5]

3.3.1 Úvod

Universal Windows Platform (UWP), neboli platforma univerzálních aplikací, byla poprvé představena ve Windows 8, jako Windows Runtime. Windows Runtime je architektura aplikací obsažena v operačním systému Windows 8, která nativně podporuje architektury x86 i ARM a běží v uzavřeném prostředí. To umožňuje větší stabilitu a bezpečnost. [5]

Jádrem univerzálních aplikací je myšlenka, že uživatelé chtějí, aby jejich prostředí bylo mobilní napříč všemi jejich zařízeními. Měli by používat různá zařízení, podle aktuálního nejvhodnějšího použití pro daný úkol, avšak se stejným uživatelským prostředím.

Windows 10 a univerzální aplikace usnadňují především vývoj aplikací s pouze jedním API základem. Výsledkem vývoje aplikace je také pouze jeden balíček aplikace, který lze nainstalovat na všechny zařízení s Windows 10, čímž může být PC, tablet, telefon a další. Platforma usnadňuje práci s celou řadou velikostí obrazovek, a také různé modely interakce, ať už je to dotyk, myš a klávesnice, nebo herní ovladač. [5]

3.3.2 Specifikace

Windows 8 představil Windows Runtime, který byl evolucí modelu aplikací pro Windows. Tím byla zamýšlena společná aplikační architektura. Když vyšel mobilní operační systém Windows Phone 8.1, byl Windows Runtime sladěn mezi tímto systémem a systémem Windows. To umožnilo vývojářům vytvářet Universal Windows 8 aplikace, které byly zaměřeny na Windows i Windows Phone pomocí sdílené codebase. [5]

Windows 10 představuje Universal Windows Platform, která dále vyvíjí model Windows Runtime a přináší do Windows 10 sjednocená jádra. Jako součást jádra, UWP nyní poskytuje společnou platformu aplikace dostupnou na každém zařízení s Windows 10. S tímto stylem vývoje univerzální aplikace mohou používat API Windows Runtime, která jsou společná pro všechny zařízení, ale také API specifická pro jednotlivé druhy zařízení, jako Win32 a .NET API. UWP poskytuje garantovanou jádrovou vrstvu API napříč všemi zařízeními. To znamená, že lze vytvořit pouze jeden aplikační balíček, který lze nainstalovat na širokou škálu zařízení. A tento jediný balíček nahrát na Windows Store, který poskytuje jednotný distribuční kanál aplikací použitelný na všech typech zařízení. [6]

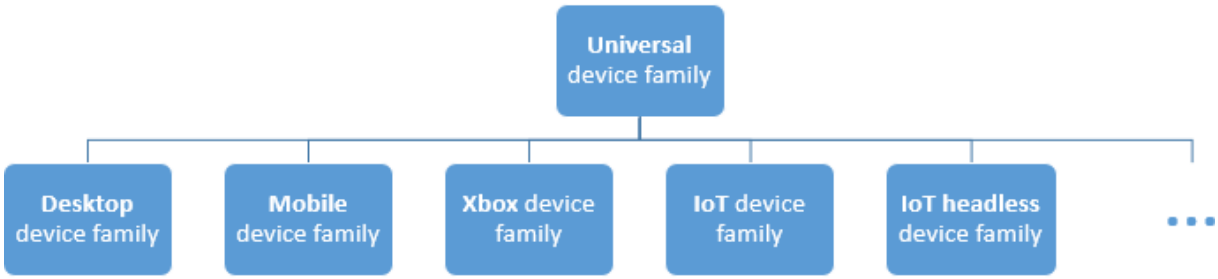
Protože univerzální aplikace běží na nejrůznějších zařízeních s různými obrazovkami a vstupními způsoby, je nutné je navrhnout tak, aby byly schopné využít možnosti každého zařízení. Každý druh zařízení má své rozšiřující API, které lze používat podmíněně v závislosti na aktuálním zařízení, na kterém aplikace zrovna běží. Tímto způsobem lze plně využít potenciál všech zařízení s jedním programovým základem.

3.3.3 Rodiny zařízení

U předchozí verze operačního systému se vývoj aplikací cílil na typ operačního systému. Bylo možné rozlišovat Windows a Windows Phone. U univerzálních aplikací se vývoj cílí na jednotlivé rodiny zařízení. Každá rodina zařízení má svoje rozšiřující API, charakterizující systém a chování, které lze očekávat v různých zařízeních v rámci určité rodiny.

Rodina zařízení je sada rozhraní API shromážděných dohromady, který je základem operačního systému. Například stolní počítače mají operační systém založený na rodině desktop. Telefony a tablety mají mobilní operační systém založený na rodině mobile. Rodina není přímo základem každého operačního systému, ale je zaručeno, že bude v každém systému v závislosti na zařízení. [6]

Jednou z výhod rodin je možnost určitého chodu aplikace na libovolném, nebo všech zařízeních od telefonů a stolních počítačů, až po Xbox konzole. Díky rodinám lze vytvořit adaptivní kód, který umí dynamicky detekovat funkce zařízení a používat je.



Obrázek 3- Rodiny zařízení – ZDROJ: [6]

Jako výchozí nastavení cílové rodiny je ve Visual Studiu vždy nastavena rodina Universal device. Tudíž aplikace s výchozím nastavením půjde nainstalovat na všech zařízeních.

3.3.4 Uživatelské prostředí

Univerzální aplikace může běžet na mnoha odlišných zařízeních, která mají různé formy vstupu, rozlišení obrazovky, hustotu displeje a další jedinečné vlastnosti. Windows 10 poskytuje nové univerzální ovládací prvky, rozvržení panelů a nástrojů, které pomáhají optimálně přizpůsobit uživatelské prostředí aplikace. Jeden z důležitých faktorů je rozlišení obrazovky, které se velice liší například mezi monitorem stolního počítače a obrazovkou na mobilním zařízení.

Některé aspekty uživatelského rozhraní se automaticky přizpůsobí podle zařízení, na kterém je aplikace spuštěna. Ovládací prvky, jako jsou tlačítka a jezdci se automaticky přizpůsobí podle zařízení a jeho vstupním režimu. Avšak některé prvky prostředí musí být přizpůsobeny přesně podle jejich využití. Například aplikace s fotografiemi by měla přizpůsobit své prostředí, při spuštění na mobilním zařízení tak, aby bylo zajištěno, že lze prvky ovládat jednou rukou. Naopak při spuštění na stolním počítači, by aplikace měla využít dostatečného prostoru obrazovky. [6]

Windows svými funkcemi pomáhá správně přizpůsobit aplikaci typu zařízení. Univerzální ovládací prvky, jako jsou panely, pomohou optimalizovat prostředí podle rozlišení

displeje. Manipulaci s obvyklými vstupy umožňuje přijímat vstup přes dotek, pero, myš nebo regulátor na Xboxu. [6]

U některých aplikací se musí přizpůsobit celkové rozvržení prostředí na základě rozlišení obrazovky zařízení, na kterém aplikace běží. Například aplikace sloužící ke komunikaci, která lze využívat na stolním počítači i na mobilu, musí mít toto adaptivní prostředí. Když aplikace běží na velké obrazovce, může obsahovat odesílaný obraz v obraze volajícího a ovládací prvky dobře ovladatelné myší. Při spuštění na telefonu s menší obrazovkou, se musí aplikace přeuspořádat, aby šla jednoduše ovládat a byl dobře vidět obraz volajícího. [6]

Obrázek 4- Adaptace UI – ZDROJ: [6]

3.3.5 Programování

Při vytváření univerzální aplikace existuje možnost výběru mezi několik programovacími jazyky a způsoby vytvoření uživatelského prostředí. Na výběr jsou programovací jazyky, jako například Visual C++, C#, Visual Basic a JavaScript. Pro Visual C++, C# a Visual Basic můžete použít XAML pro návrh uživatelského prostředí. Tvorba s Visual C++ nabízí možnost výběru mezi návrhem v XAML nebo vykreslování přes rozhraní DirectX. S použitým JavaScriptem je možnost tvorby prezentační vrstvy s pomocí HTML. Velká část kódu a rozhraní budou univerzální a budou spuštěny stejným způsobem na všech zařízeních. Ale pro přizpůsobení aplikace jednotlivým zařízením a pro uživatelské prostředí šité na míru se musí použít adaptivní kód a adaptivní rozhraní. K tomu lze použít více způsobů.

Jedním ze způsobů je používání API, které je implementováno v cílové rodině zařízení. Kdykoli se bude volat API, je potřeba vědět, jestli je API implementováno v rodině zařízení,

na které je aplikace cílena. Pro upřesnění informací ohledně API se může použít referenční příručka API dokumentace. Pokud je aplikace směrována na všechny rodiny zařízení, neboli pro univerzální rodinu, můžete použít její API a je zaručeno, že bude na všech zařízeních. [6]

Visual Studio programátorovi pomáhá s rozpoznáváním správného API. IntelliSense implementované ve Visual Studio nerozpozná API, pokud není implementováno v cílové rodině zařízení, na kterou je aplikace vytvářena.

K vytvoření adaptivního kódu jsou potřeba dva kroky. Prvním krokem je, aby API, které je potřeba mít přístupné k projektu, bylo připojeno pomocí reference na SDK, které reprezentuje danou rodinu zařízení. [6]

Druhým krokem je využít třídu Windows.Foundation.Metadata.ApiInformation na ověření, zda je API, které chceme použít, v rodině zařízení, na kterém aplikace běží. V malém měřítku lze požít takto:

```
bool isHardwareButtonsAPIPresent =
Windows.Foundation.Metadata.ApiInformation.IsTypePresent("Windows.Phone
.UI.Input.HardwareButtons");

if (isHardwareButtonsAPIPresent)
{
   Windows.Phone.UI.Input.HardwareButtons.CameraPressed +=
     HardwareButtons_CameraPressed;
}
```

Ukázka kódu 1- Použití třídy ApiInformation ZDROJ: vlastní zpracování

V tomto případě jsme si jisti, že můžeme pracovat s událostí CameraPressed třídy HardwareButtons. Třída ApiInformation poskytuje mnoho metod na získání informací o jednotlivých API. [6]

3.3.6 Balíček aplikace

Po dokončení programové fáze vývoje aplikace, je výstupem balíček aplikace. Visual Studio balíček samo podle průvodce vygeneruje. Tento balíček je pouze jeden pro všechny rodiny zařízení. Obsahuje však verze x86, x64 a ARM. Tyto verze jsou nainstalovány podle typu procesoru na jednotlivých zařízeních. Balíčky lze nainstalovat ručně na zařízení, nebo nahrát na Windows Store. Při výběru možnosti nahrání balíčku na Windows Store je potřeba

vývojářský Microsoft účet. Balíček se může nahrát pro jakýkoli operační systém, pro který je aplikace cílena. Když se balíček instaluje z Windows Store, obchod se automaticky podívá na všechny dostupné verze balíčku aplikace a automaticky poskytne každému zákazníkovi správnou verzi, podle rodiny zařízení zákazníka. [7]

Před samotným vygenerováním balíčku je doporučeno prověřit aplikaci s Windows App Certification Kit. Tento nástroj je součástí Windows SDK, takže není nutné starat se o jeho pořízení, ale lze ho využívat ihned.

Při vytváření balíčku ve Visual Studiu, je nutné zkontrolovat přihlášení pomocí Microsoft účtu, přes který, se bude aplikace nahrávat na Windows Store. Některé části manifestu aplikace mají specifické detaily vázané na přihlášený účet. Visual Studio poté vygeneruje balíček souborů typu .appx, který lze nainstalovat na Windows 10. Balíček lze vygenerovat také ručně, ale Visual Studio k tomu má jednoduchý a spolehlivý nástroj.

Volitelná součást balíčku je konfigurační soubor StoreManifest.xml. Jeho cílem je umožnit funkce, jako například deklaraci aplikace jako Windows Store Device App, která je privilegovanější, než normální aplikace. Nebo deklarovat nutné doplňky k správnému chodu aplikace, které balíček neobsahuje. StoreManifest.xml je předkládán s balíčkem aplikace a musí být v kořenové složce hlavního projektu. [7]

Každý balíček musí mít číslo verze uložené v manifestu aplikace, nebo v atributu verze balíčku. Windows Store vynucuje určitá pravidla týkající se číslování verzí, která pracují v různých operačních systémech odlišně. Číslo verze jakéhokoli balíčku aplikace pro Windows 10 musí být vždy vyšší, než jakékoli číslo verze aplikace pro Windows 8.1 a nižší, které jsou vydávány za stejnou aplikaci. [8]

Při instalaci aplikace z Windows Store bude vždy vybrána verze s nejvyšším číslem, která je použitelná na aktuální zařízení. To poskytuje větší flexibilitu a kontrolu, které balíčky budou poskytovány na konkrétní typy zařízení. Důležité je, že se balíčky mohou nahrávat na Windows Store nezávisle, tudíž není problém navyšovat verze pro jednotlivé typy zařízení odlišně. Dokonce je možné nahrávat více Windows 10 balíčků se stejným číslem verze. Nicméně balíčky, které sdílejí číslo verze, nemohou mít také stejnou architekturu, protože Windows Store nemůže mít více identických balíčků. [8]

Při odhalení chyby v aplikaci po nahrání nové verze, je za předpokladu, že jsou uloženy archivní verze aplikace, možnost vrátit starší verzi, a zamezit tím škodám u uživatelů, dokud se chyba neopraví a znovu nahraje nová verze. Uživatelům se pak automaticky aktualizuje aplikace na opravenou aktuálně nejvyšší verzi. [8]

4. Praktická část

4.1 Úvod

V praktické části si nejprve představíme technologie, které budeme následně používat k demonstraci platformy Universal Windows Platform. Mezi tyto technologie patří programovací jazyk C#, ve kterém budeme aplikaci vytvářet, a jazyk XAML, ve kterém budeme tvořit uživatelské prostředí aplikace. Aplikaci za pomocí těchto technologií budeme vytvářet ve vývojovém prostředí Visual Studio 2015 od firmy Microsoft. Tyto technologie jsem vybral, protože jsou nejpoužívanějším a univerzálním prostředkem k vytvoření univerzální aplikace pro Windows 10.

4.2 Použité technologie

4.2.1 C#

Tento jazyk byl navržen společností Microsoft a vydán spolu s vývojovým prostředím Visual Studio a rozhraní .NET Framework od stejné firmy. Je to programovací jazyk navržený pro vytváření různorodých aplikací, které běží v rozhraní .NET Framework. C# budeme používat díky jeho jednoduchosti, výkonosti a typové bezpečnosti. Tento jazyk je plně objektově orientovaný a umožňuje rychlý vývoj aplikací a zároveň zachování elegance jazyků stylu C. [9]

Jazyk velice připomíná Javu a v jistých ohledech nese některé myšlenky Visual Basicu. Vývojáři Microsoftu, při představení jazyka, říkali, že přebrali to nejlepší z jazyka C++, Visual Basicu a Javy. [9]

Důvody Microsoftu pro vytvoření nového jazyka byly nejméně dva. První politický, druhý technický. Vedl se spor Microsoft versus Sun, jehož důsledkem došlo více méně k zániku jazyka Java MS provinence. Visual J++, jak se verze z Redmondu jmenovala, měla na architekturu jazyka C# nemalý vlit díky svým kvalitám v mnohých oblastech. Přechod od Javy k C# byl pro Microsoft způsob jak urovnat vleklý spor a mít jazyk obdobných kvalit. [9] [12]

Základní charakteristiky jazyka je mnoho. C# je objektově orientovaný, obsahuje nativní podporu komponentového programování, používá jednoduchou dědičnost s možností násobné implementace rozhraní. Vedle členských dat a metod přidává vlastnosti a události, správa paměti je realizována prostřednictvím Carbage Collection. Mimo jiné podporuje zpracování chyb formou výjimek, zajišťuje typovou bezpečnost a podporuje řízení verzí. Podporuje atributové programování a zajišťuje integraci se stávajícím kódem na binární i zdrojové úrovni. [12]

Značnou část vlastností C# přebírá z funkcionality .NET Framework a zprostředkovává ji programátorům.

4.2.2 XAML

XAML je deklarativní značkovací jazyk. Vychází XML, což je značkovací jazyk navržený tak, aby si do něj každý mohl přidat vlastní značky a používat ho úplně k čemukoli. Zkratka XAML označuje eXtensible Aplication Markup Language. [10]

U modelu .NET Framework programování zjednodušuje vytváření uživatelského rozhraní pro aplikace založené na této platformě. Je to pro mě nejsympatičtější způsob pro vytvoření uživatelského rozhraní aplikace pro Windows 10. Z tohoto důvodu jsem ho vybral i pro tvorbu ukázkové aplikace. Jazykem XAML lze vytvořit viditelné prvky uživatelského rozhraní a oddělit definici rozhraní s programovou logikou kódu na pozadí, připojenou k rozhraní pomocí definic dílčích tříd. XAML přímo reprezentuje instance objektů ve specifických typových sadách definovaných v sestavách. To je odlišné od většiny jiných značkovacích jazyků, které jsou obvykle interpretovány, aniž by takové přímé vazby na programové zázemí aplikace existovaly. XAML umožňuje separovaný běh logické strany aplikace a rozhraní, za použití odlišných nástrojů. [10] [11]

Když je XAML reprezentován jako text, soubory jsou základně s příponou .xaml. Soubory mohou být zakódovány jakýmkoli kódováním XML, ale typické je kódování UTF-8.

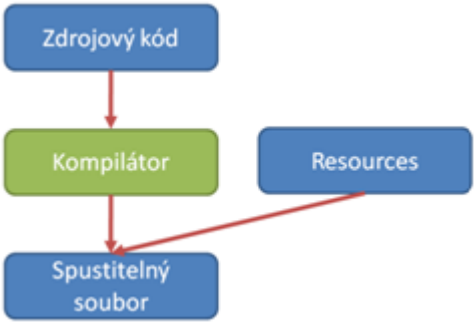
4.2.3 .Net Framework

.NET Framework je rozsáhlá softwarová platforma, která je určena pro vývoj mnoha různých druhů aplikací. Za pomoci .NET Frameworku můžeme vyvíjet nejen klasické aplikace pro Windows, ale mimo jiné i webové aplikace a služby, aplikace pro mobilní zařízení a mnoho dalších.

Celý Framework obsahuje kromě velké sady knihoven a funkcí i samotné běhové prostředí, které zajišťuje běh a kompilaci aplikací. Obrovská škála funkcí nám zajišťuje to, že se nemusíme starat o psaní věcí, které se používají často a dá se v nich udělat mnoho chyb. Mezi takovéto případy patří například třídění pole, stažení webové stánky, vytvoření okna, práce s XML soubory a další. Díky běhovému prostředí jsou aplikace rozumně rychlé a bezpečné. Vývoj v .NET je rychlý, pohodlný a méně náchylný na chyby programátora. [9]

Základním rysem k pochopení systému tohoto frameworku je způsob, jakým dochází v .NET ke kompilaci aplikací.

Při klasické kompilaci aplikace například v C++, Delphi, nebo dalších jazycích, výstupem kompilátoru je přímo strojový kód. Ten je samozřejmě závislí na platformě, na které bude spouštěn. Na operačním systému a na konkrétním procesoru. Jedná se tedy o tzv. neřízený kód. Je obvykle velmi rychlý, protože procesor pracuje přímo se strojovým kódem, ale je zde také mnoho nevýhod. Největší nevýhodou je náchylnost na chyby v samotné aplikaci a bezpečnostní rizika. [13]



Obrázek 5 - Klasická kompilace – ZDROJ: [13]

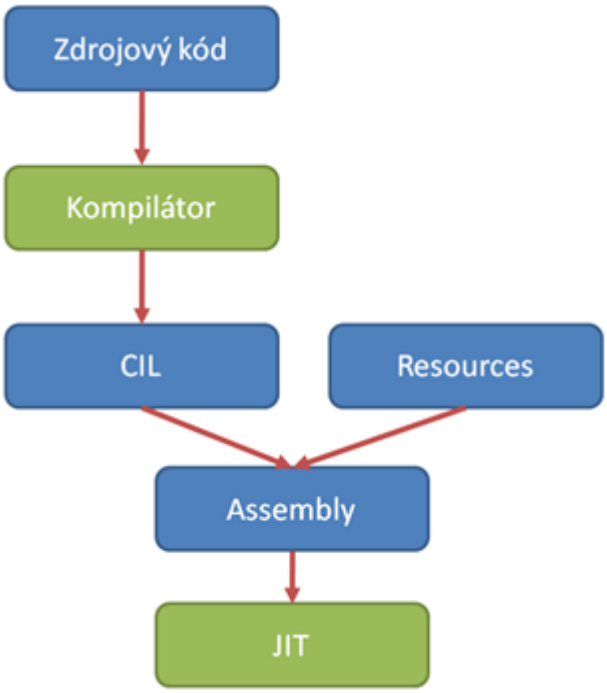
V .NET Frameworku to ovšem takhle není. Kompilátor v .NET vezme zdrojové kódy a jeho výsledkem není strojový kód, ale Common Intermediate Language, neboli CIL. Jedná se o kód podobný kódu strojovému, ale jeho výhodou je to, že je nezávislý na platformě a dá se spustit všude tam, kde je běhové prostředí .NET Frameworku. [13]

Kód přeložený do jazyka CIL se spolu s přidanými datovými soubory, jako jsou například obrázky, zabalí do assembly, což je soubor s příponou exe nebo dll. Na první pohled výsledné soubory vypadají stejně jako výstup kompilátoru v C++. Obrovský rozdíl je až při jejich spuštění. Při spuštění .NETové assembly se provede částečná kompilace, toho co je pro běh aplikace zatím potřeba, což se přeloží do strojového kódu, optimalizovaného pro konkrétní platformu, a poté se aplikace spustí. Následně se překládají pouze části kódu, které jsou aktuálně potřeba a jejich výsledek se zachovává do dalšího spuštění té dané části.

Pokud je na kompilaci dost času, JIT, neboli „just in time“ překladač, může provádět velmi složité a užitečné optimalizace, takže výkon aplikace může být dokonce lepší než u aplikace v C++. [13]

Tomuto kódu se říká řízený. Už od základu se počítá s bezpečností a korektností přístupu do paměti, jelikož se nedá přistupovat do paměti kamkoli a není možné přepsat někde kus paměti, se kterým aktuálně nepracujeme, což by mohlo způsobit rozbití jiného mechanismu.

Dále je zde zaručena striktní typová kontrola, díky které nelze jednoduše přetypovat něco na úplně jiný typ. Je zde implementována obsluha výjimek a mnoho dalšího.



Obrázek 6- .NET kompilace – ZDROJ: [13]

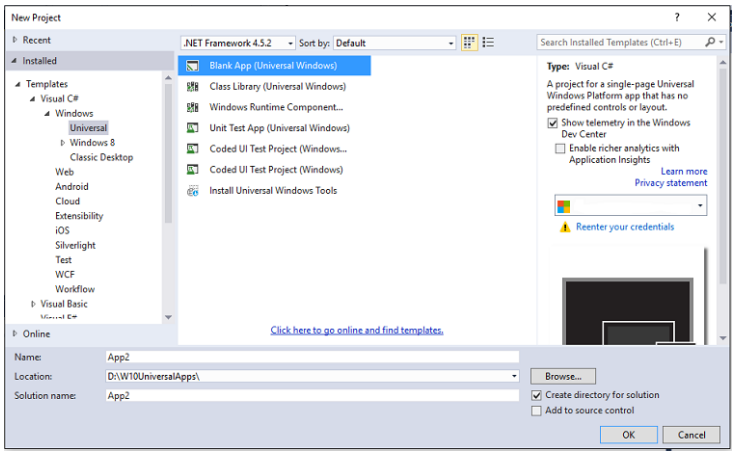
4.3 Vývoj

4.3.2 Základy projektu

4.3.2.1 Založení nového projektu

Když jsme si vysvětlili technologie, které budeme využívat, můžeme začít vyvíjet ukázkovou aplikaci. Tato aplikace bude velice jednoduchá a bude sloužit pouze jako demonstrace základních principů vytváření univerzálních aplikaci pro operační systém Windows 10. Pro pochopení se předpokládá, alespoň minimální znalosti vývoje aplikací pro Windows a základy jazyků typu C a XML, jelikož programování bude primárně realizováno pomocí jazyka C# a XAML, které jsme si už jednoduše představili v předchozích kapitolách. Základní struktury kódu, jako konstruktory a podobné, nebudou prezentovány v ukázkách kódu, a proto je nutné, alespoň základy programování znát. S vývojem začneme se spuštěním vývojového prostředí Visual Studio 2015 a vytvořením nového projektu aplikace na plaformě Universal Apps. Pro ukázkový projekt se použilo vývojové prostředí volně dostupné pro studentské účely. Po kliknutí na tlačítko pro vytvoření nového projektu, ve stromové struktuře

otevřeme postupně složky Tamplates, poté Visual C#, jelikož budeme k vývoji používat tento jazyk. V této složce máme na výběr několik platforem, na které můžeme pomocí tohoto jazyka vyvíjet aplikace. My zvolíme platformu Windows a v ní typ aplikace Universal. Tímto se nám zobrazil náhled aplikací, které můžeme začít vytvářet.

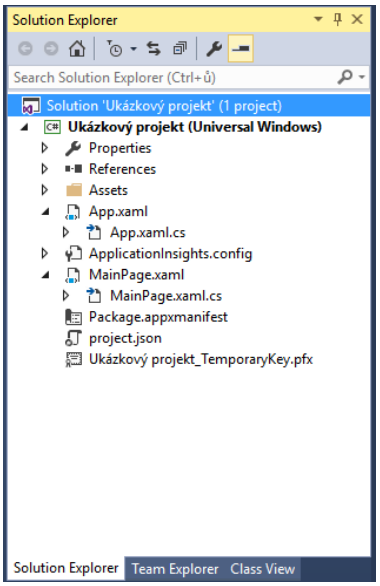


Obrázek 7- Visual Studionový projekt - ZDROJ: vlastní zpracování

Nastavíme si název projektu a můžeme vytvořit projekt typu Blank App (Universal Windows) a stiskneme tlačítko OK.

4.3.2.2 Souborová struktura projektu

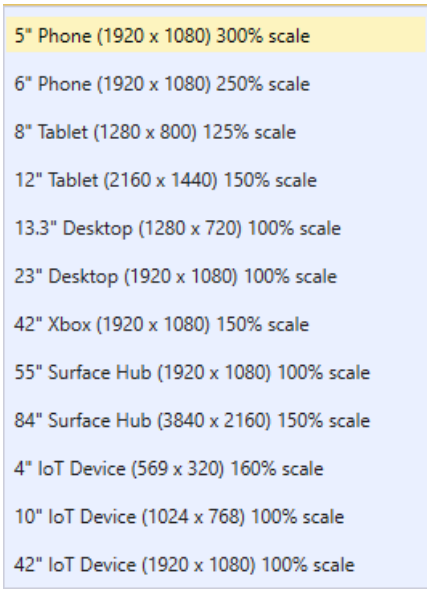
Po vytvoření projektu se nám vygenerovala souborová struktura projektu. Tuto strukturu si ukážeme a trochu popíšeme, abychom věděli co psát do jakého souboru. Jeden z největších problémů při přechodu mezi platformami je vždy tato souborová struktura, která se velmi liší podle konkrétního způsobu vývoje. Pochopení není vždy úplně intuitivní, a proto pro základní orientaci v projektu je velmi důležité znát jednotlivé soubory a jejich funkci. Tato znalost je vždy velice jednoduchý, ale má za následek velké ulehčení samotného vývoje. Při založení základního prázdného projektu není těchto souboru velké množství, takže nebude složité tuto strukturu pochopit. Postupně si probereme jeden soubor po druhém. O těch souborech, které budeme využívat, si řekneme o něco více, než o ostatních, které nám k ničemu v této bakalářské práci nebudou a využívat je nebudeme. Vzhledem k perfektní dokumentaci firmy Microsoft, na jejich webových stánkách, nebude problém si dohledat informace o ostatních souborech a jejich funkci, kdyby nebyly dostatečné informace prezentovány v tomto dokumentu.



Obrázek 8- Souborová struktura projektu - ZDROJ: vlastní zpracování

Nejdůležitějším souborem v projektu pro nás bude `MainPage.xaml` a s ním spojený `MainPage.xaml.cs`. V těchto dvou souborech se skrývá uživatelské prostředí a programová část hlavní stránky.

`MainPage.xaml` je soubor se strukturou jazyku XAML a je v něm zapsán vzhled hlavní stránky, který můžeme pomocí designeru vidět nejenom v textové formě, ale i vizuálně. Náhled vizuální části lze nastavit podle zařízení, které sami vybereme v nabídce zobrazených zařízení.



Obrázek 9- Možnosti zobrazení – ZDROJ: vlastní zpracování

V souboru `MainPage.xaml.cs` je zdrojový kód hlavní stránky. Každá jednotlivá stránka aplikace musí v konstruktoru zavolat metodu `InitializeComponent()`. Tato

metoda je potřeba na zpracování XAML kódu, aby programová část aplikace mohla pracovat s komponentami vizuální části. Dále v tomto souboru můžeme vytvářet kód stránky a používat její metody. Jedna z důležitých metod třídy `Page` je například `OnNavigatedTo()`, kterou budeme používat v ukázkovém příkladu. Tato metoda má vstupní parametr třídy `NavigationEventArgs`, do kterého můžeme vložit jakýkoli námi definovaný vstupní parametr, který budeme v nově otevřené stránce potřebovat. Metoda je volána vždy po přechodu navigace na tuto stránku. Navigaci můžeme zavolat v jakékoli události, ale my ji budeme volat minimálně při kliknutí na tlačítko, což je asi nejpoužívanější možnost a proto si ji ukážeme. K navigaci nám bude sloužit metoda `Navigate()` třídy `Frame`. V ukázkovém kódu si otevřeme novou stránku po stisknutí tlačítka s nějakým vstupním parametrem.

```
private void btnTlacitko_Click(object sender, RoutedEventArgs e)
{
            VstupniParametr param = new VstupniParametr();
            this.Frame.Navigate(typeof(NovaStranka), param);

}
```

Ukázka kódu 2- Navigace - ZDROJ: vlastní zpracování

Po zavolání metody `Navigate()`, se otevře stránka třídy `NovaStranka` odděděna od třídy `Page`. Tato naše nová stránka v metodě `OnNavigatedTo()` dostane vstupní parametr třídy `NavigationEventArgs` a po přetypování s ním může pracovat jako s objektem třídy, na který byl přetypován, což v našem případě bude třída `VstupniParametr`. V ukázkovém kódu si znázorníme tohle použití na příkladu.

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
      string Nazev = ((VstupniParametr)e.Parameter).Nazev;
}
```

Ukázka kódu 3- OnNavigatedTo - ZDROJ: vlastní zpracování

Takto jsme si stručně ukázali možný obsah souboru `MainPage.xaml.cs` a můžeme pokračovat ve vysvětlování souborové struktury projektu. Nejdůležitější soubory pro naši práci už jsme si ukázali, takže následující soubory a složky vysvětlíme velmi rychle a stručně.

V souborech `App.xaml` a `App.xaml.cs` najdeme obecné informace o aplikaci. Uvnitř `App.xaml` je kód jazyka XAML a například v něm můžeme nastavit téma aplikace. Základně je nastaveno na hodnotu „Light“, což znamená světlé pozadí a tmavý text. Naopak v souboru `App.xaml.cs` je v našem případě C# kód, ve kterém je samotné jádro aplikace. Spouštění, navigace a různé metody, které můžeme použít. Například lze použít metodu `OnNavigationFailed()` a v ní spustit kód, který chceme, aby se provedl, pokud selže navigace mezi stránkami v aplikaci.

Složka `Assets` pro nás bude důležitá, pokud budeme chtít v aplikaci mít uložené například nějaké obrázky nebo ikony, které budeme jakkoli používat. Implementovaný kód v tomto souboru je už velmi hezky okomentován, takže není problém se v jeho základech vyznat.

Do `References` přidáváme používané knihovny, které nejsou základně implementovány v projektu. Většinou se jedná o knihovny třetích stran, které můžeme do projektu jednoduše implementovat a stáhnout pomocí nástroje NugGet, který je implementován ve Visual Studiu. Funguje jako manažer přídavných balíčků, které se mohou implementovat do projektu. S jeho pomocí je dodatečné stažení přidaných balíčků velice banální operací, která nezabere mnoho času. V našem případě to bude například knihovna SQLite, kterou budeme používat na ukládání některých informací.

V `Properties` se ukrývají dva soubory, které nijak používat nebudeme a proto ani nemusíme znát nějaké podrobnosti o tom co v nich je. Stačí nám říci, že v souboru `AssemblyInfo.cs` jsou nějaké informace o `Assembly`, které nebudeme editovat. Mezi tyto informace patří například atributy jako jméno, verze, popis a nějaké další. V souboru `Default.rd.xml` je místo pro doplnění direktiv pro .NET, které ale také používat nebudeme, a proto to pro nás není důležité.

Soubor `Package.appxmanifest` je zajímavější, než předchozí soubory, protože v něm můžeme pohodlně nastavovat atributy projektu. Například základní jazyk pro používání aplikace, jméno aplikace, které se bude ukazovat a některé další. Tento soubor také nebudeme používat, takže není nutné ho procházet nějak podrobněji.

Poslední soubor, o kterém si něco řekneme, je `Project.json`. V tomto souboru jsou informace o aplikaci ve formátu json. Tyto informace nebudeme editovat, a proto pro nás nejsou nijak zajímavé.

Tímto jsou probrány všechny základní soubory a je možné začít vyvíjet samotnou ukázkovou aplikaci.

4.3.3 Vývoj ukázkové aplikace

V následující kapitole bude popsán průběh vytváření ukázkové aplikace k bakalářské práci. Aplikace bude sloužit pouze k demonstraci vývoje a problematiky s ním spojené. Za tímto účelem bude aplikace velice jednoduchá a bude používat ukázkové řešení základních problematik vývoje aplikace na Windows 10.

Aplikace bude mít pouze dvě stránky, na kterých bude prezentovat data, která budou uloženy v SQLite databázi. V databázi budou data o uložených odkazech. Struktura aplikace bude připomínat jednoduchou aplikaci na správu a editaci poznámek. Jedna stránka aplikace bude obsahovat seznam vytvořených odkazů a stručné informace o nich. Na této stránce bude možné otevřít jednotlivé poznámky, přechod na stránku pro vtvoření nového odkazu a mazání jednotlivých existujících odkazů z databáze. Při přechodu na druhou stránku, která bude sloužit jako detail jednotlivých odkazů, se pomocí objektu třídy odkazu, přenesou informace o konkrétním odkazu, který budeme otevírat. V případě, že budeme vytvářet nový odkaz, vstupní objekt bude prázdný a tím stránka pozná, že má vytvářet nový odkaz, který potom uloží do databáze. Toto bude veškerá funkce aplikace, která ovšem okryje základní problematiku vývoje a objasní základní principy.

Na začátku budeme postupovat stejně, jako na začátku předchozí kapitoly. Vytvoříme si nový projekt typu Blank App ve složce Universal. Tímto krokem se vytvořila nová aplikace s již známou souborovou strukturou.

Hlavní stránka `MainPage.xaml` bude sloužit k zobrazení listu uložených odkazů. Dále na této stránce budou tlačítka na přidání nové poznámky a vymazání už existující. Takže si otevřeme tento soubor a počkáme, až se načte designer. Designer je grafické zobrazení aplikace. Toto grafické zobrazení můžeme použít podle různých rozměrů, jak už jsme si říkali dříve. Pro tuto aplikaci zvolíme zobrazení na „5¨ Phone (1920 x 1080) 300% scale“, jelikož pro tuto aplikaci to bude ideální zobrazení. Orientaci designeru necháme nastavenou na `Portrait`. To znamená, že designer bude zobrazovat aplikaci na výšku.Teď existují dva způsoby k vytvoření uživatelského prostředí popsaného v předchozím odstavci.

První způsob je vložit grafické komponenty ručně na stránku. Tyto komponenty najdeme v panelu Toolbox, který je základně zobrazen v levé části Visual Studia. V tomto panelu je mnoho komponent, které si nebudeme blíže vysvětlovat. Na této stránce budeme používat komponenty `ListView` a `AppBarButton`. Pokud použijeme tento způsob vkládání komponent, vložíme nejdříve dvě tlačítka. Jedno do levého horního a druhé do pravého horního rohu. Poté najdeme v panelu Toolbox komponentu ListView a pod tlačítka ji vložíme. Následně

se stejně nevyhneme editaci XAML kódu, jelikož použijeme předávání hodnot do komponenty. Takže výsledný kód XAML, který budeme používat v druhém způsobu tvorby grafického rozhraní, bude shodný.

Druhým způsobem je vytvořit rozhraní přímo pomocí editace XAML kódu. Tento způsob je rychlý pokud máte znalosti XAML, a proto je hojněji využíván mezi programátory. V následující ukázce kódu si ukážeme strukturu komponent a dále popíšeme jednotlivé řádky kódu.

```
<ListView x:Name="listOdkazy" RenderTransformOrigin="0.5,0.5" Margin="0,42,0,158">
   <ListView.ItemTemplate>
      <DataTemplate>
         <Grid>
           <StackPanel>
              <StackPanel Orientation="Horizontal" >
                 <TextBlock  Text="{Binding Nazev}" FontWeight="Bold"></TextBlock>
                 <TextBlock Text="{Binding Datum}"></TextBlock>
              </StackPanel>
              <TextBlock Text="{Binding Url}"></TextBlock>
           </StackPanel>
         </Grid>
      </DataTemplate>
   </ListView.ItemTemplate>
</ListView>

<AppBarButton x:Name="btnPridat" Icon="Add"  HorizontalAlignment="Left"
VerticalAlignment="Top" Width="45" Height="44" Click="btnPridat_Click"/>

<AppBarButton x:Name="btnVymazat" Icon="Delete" HorizontalAlignment="Right"
VerticalAlignment="Top" Width="45" Height="44" Click="btnVymazat_Click"/>
```

Ukázka kódu 4- MainPage.xaml komponenty - Zdroj: vlastní zpracování

V této ukázce kódu vytváříme tři komponenty. Dvě tlačítka typu `AppBarButton`, která mají nějakou iconu a po kliknutí na ně se zavolá metoda v parametru `Click`.

První tlačítko s názvem `btnPridat` bude sloužit pro přidání nového odkazu a tlačítko `btnVymazat` vymazat pro jeho vymazání. V atributech alignment nastavujeme zarovnání komponent. Tlačítko btnPridat chceme mít v levém horním rohu, a proto je vertikální zarovnání nastaveno na „Top“ a horizontální na „Left“.

Druhé tlačítko chceme mít také nahoře, ale vpravo, takže atribut horizontálního zarovnání má hodnotu „Right“. To byla jednodušší část ukázky kódu. V následující části si vysvětlíme nějaké atributy komponenty `ListView`, které jsou použity.

`ListView` je pojmenováno `listOdkazy`, což nám zaručí snadnou identifikaci v kódu. První atribut `ItemTemplate` zapouzdřuje to, co bude `ListView` zobrazovat v jednotlivých

řádcích. V našem případě to bode Grid, ve kterém bude klasický `StackPanel`. Tento `StackPanel` v tomto použití slouží pouze pro uchování dalších komponent po řádcích. Následně je do něj vložen další pomocný `StackPanel`, který ovšem má nastaven atribut Orientation na hodnotu „Horizontal“, tudíž se komponenty v něm obsažené budou zobrazovat po sloupcích. Tyto komponenty zobrazené po sloupcích budou dva `TextBlock`y, které budou zobrazovat název a datum vytvoření odkazu. Pod celým tímto `StackPanel`em orientovaným horizontálně bude další `TextBlock` s informací o url odkazu. Nejdůležitější částí této vytvořené komponenty je použití `Data Binding`. Tímto nástrojem nastavujeme dynamicky text komponentám typu `TextBlock` podle toho co vstoupí do `ListView` jako `ItemsSource`. Například do prvního `TextBlocku` se dynamicky, při plnění `ListView`, nastaví hodnota z proměnné `Nazev`. Do druhého proměnná `Datum` a do třetího proměnná `Url`. Tato možnost velice ulehčuje dynamické plnění komponent, protože se nemusí složitě vymýšlet cyklus na plnění jednotlivých prvků. Jednoduše se vloží objekt do `ItemsSource` a komponenta si sama vezme hodnotu proměnných, jak je určené v XAML kódu. Toto vložení objektu bude realizováno v souboru `MainPage.xaml.cs`. Rozhraní této stránky je tímto vyřešeno.

Nyní je nutné vytvořit nějakou třídu, ve které budeme držet informace o odkazu. Vytvoříme tedy třídu `Odkaz` v souboru `Odkaz.cs`. V této třídě budeme mít čtyři atributy. Budou jimi název, datum, url a popis. Název, url a popis budou typu `string` a datum bude typu `DateTime`. Uděláme prázdný konstruktor a konstruktor se vstupními parametry. Nakonec uděláme metodu na naklonování objektu třídy `Odkaz`, kterou v budoucnu použijeme. Tato metoda bude velice jednoduchá a ukážeme si ji v následující ukázce kódu. [14][15]

```
class Odkaz
{
    public string Nazev { get; set; }
    public string Url { get; set; }
    public DateTime Datum { get; set; }
    public string Popis { get; set; }

    public Odkaz(string Nazev, string Url, DateTime Datum, string Popis)…

    public Odkaz()…

    public Odkaz Clone()
    {
        Odkaz odkaz = new Odkaz(this.Nazev, this.Url, this.Datum, this.Popis);
        return odkaz;
    }
}
```

Ukázka kódu 5 - Třída Odkaz - Zdroj: vlastní zpracování

Tato ukázka kód obsahuje strukturu této třídy. Těla konstruktorů jsem pro jejich nedůležitost a jednoduchost nechal skrytý. Jejich obsah by měl být jasný.

Nyní už máme třídu na zapouzdření informací o odkazu, ale pořád nemáme jak ho ukládat. Způsob ukládání jsem zvolil pomocí SQLite. SQLite je relační databázový systém, který ovšem nefunguje na principu klient-server, ale běží pouze jako samostatný proces. Je to malá knihovna napsaná v jazyce C, která se dynamicky přilinkuje k aplikaci. Tento způsob je velice jednoduchý na použití.

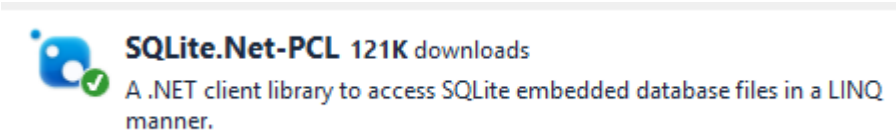
Oproti normálním serverovým databázím má ovšem nevýhodu a tou je, že není možnost přistupovat ke stejným datům z různých zařízení. Tato ukázková aplikace by mohla využívat přístupu k datům z různých zařízení uživatele, jelikož uživatel může potřebovat například ukládat odkazy na mobilním zařízení, ale práci s nimi provozovat na stolním počítači. Tuto nedokonalost nebudeme řešit, jelikož toto je jen ukázková aplikace, která má pouze demonstrovat vývoj univerzálních aplikací a pro ukázku tento způsob bude vhodný. Tento způsob je o mnoho jednodušší, než serverové řešení z mnoha důvodů.

Jedním z nich je například fakt, že nebudeme řešit chod nějaké databáze a pro vývoj bude potřeba polovina prostředků.

Nejdůležitějším faktem v tomto ohledu ovšem je komplikace, kterou vývoj pro Windows 10, oproti vývoji na Windows 7 a nižší, přináší. Tato komplikace je, že platforma univerzálních aplikací nedovoluje přímý přístup do externí databáze. Problematika přístupu do externí databáze by musela být řešena, buď webovou službou, která by zprostředkovávala data z databáze. Poslal by se požadavek na webovou službu a webová služba by vstoupila do databáze a provedla příkaz. Pokud by se získávala některá data z databáze, tak by webová služba aplikaci vracela data ve formátu JSON, se kterými by se dále mohlo pracovat. Tento způsob by byl zbytečně složitý a pro tuto aplikaci zbytečný. Nicméně forma přístupu do databáze přes webovou službu zaručuje určitou bezpečnost, kterou by přímý přístup, například z mobilního zařízení, mít nemohl.

Když jsme si vysvětlili důvod použití SQLite, řekneme si, jak a co použít, aby vše správně fungovalo a data se nám ukládala, editovala a mazala správně. Projekt zatím nezná třídu `SQLite`, kterou budeme používat, a proto musíme nejdříve tuto třídu implementovat do projektu, pro její bezproblémové použití. V záložce Tools si najdeme NuGet Package Manager a otevřeme „Manage NuGet packages for solution“. Tímto se zobrazí nástroj pro správu a stahování přídavných balíčků, které budou použity v projektu. V tomto nástroji je veliká škála různých knihoven, které je možné použít do budoucích projektů. Nás zajímá balíček pro SQLite s názvem „SQLite.Net-PCL“, ve kterém jsou třídy, které budeme potřebovat k vytvoření a

správné funkci projektu. Pomocí vyhledávače lze najít daný balíček, který je poté nutno stáhnout a nainstalovat.



Obrázek 10- NuGet SQLite balíček - ZDROJ: vlastní zpracování

Pokud se balíček správně stáhnul a nainstaloval, můžeme pokračovat a začít používat jeho třídu. Pro kontrolu je potřeba zkontrolovat, jestli se balíček přidal do složky References v souborové struktuře projektu. Pokud zde není, musíme ho přidat ručně. Pomocí „Add reference“ otevřeme Reference Manager k projektu. V tomto nástroji najdeme složku Extensions, ve které musí být daný balíček, který pouze označíme a vložíme do projektu.

Nyní lze používat knihovny SQLite. Pro jednoduchost vstupu do databáze, využijeme nově vytvořenou třídu `Prihlaseni` v novém souboru `Prihlaseni.cs`. Třída `Prihlaseni` bude mít pouze jednu metodu `Prihlasit()`, která bude vracet objekt třídy `SQLite.Net.SQLiteConnection`. Tento objekt bude přihlášené connection, se kterým budeme pracovat. Tato metoda, bude demonstrována, v následující ukázce kódu. [14][15]

```
public static SQLite.Net.SQLiteConnection Prihlasit()
{
  var path = Path.Combine
  (
  Windows.Storage.ApplicationData.Current.LocalFolder.Path,  "db.sqlite"
  );

  SQLite.Net.SQLiteConnection conn = new

  SQLite.Net.SQLiteConnection
  (
  new SQLite.Net.Platform.WinRT.SQLitePlatformWinRT(), path
  );

  return conn;
}
```

Ukázka kódu 6 - Metoda Prihlasit() - ZDROJ: vlastní zpracování

V metodě `Prihlasit()` se naplní proměnná `path` spojením aktuálního adresáře aplikace a textového řetězce „db.sqlite“. Tuto proměnnou použije při vytvoření nového

`SQLiteConnection`, které metoda vrací. Do konstruktoru tohoto connection vstupuje proměnná platformy, na které se má vytvořit nebo připojit databáze a cesta, kde se má soubor s databází vytvořit, jelikož každá databáze na bázi SQLite je reprezentována jedním souborem. Tímto je třída `Prihlaseni` hotová. Použili jsme samostnatnou třídu, abychom zajistili optimální přehlednost kódu při čtení. Nyní můžeme jít programovat další funkcionalitu v souboru `MainPage.xaml.cs`.

Třída `MainPage` zatím obsahuje pouze jednu metodu, kterou obsahuje každá třída stránky univerzální aplikace. Touto metodou je konstruktor, ve kterém proběhne inicializace komponent ze souborů s kódem XAML. Při vytvoření této třídy tedy dojde k přeložení kódu XAML a vytvoření komponent, aby se dali používat v programové části. Do této třídy budeme postupně přidávat metody pro obsluhu této stránky.

Jako první použijeme metodu třídy `Page`, která se volá vždy, při načítaní stránky. V tomto případě se metoda bude jmenovat `MainPage_Loaded()` a bude mít nějaké vstupní parametry, které ovšem nebudeme používat. V naší aplikaci bude důležité, aby se vždy při obnovení této stránky načetl obsah z databáze a obnovilo `ListView` s uloženými odkazy. Toto načtení z databáze a obnovení `ListView` bude probíhat při každém obnovení hlavní stránky. V následující ukázce si představíme kód metody `MainPage_Loaded()` a později si ho detailněji popíšeme.

```
private void MainPage_Loaded(object sender, RoutedEventArgs e)
{
  SQLite.Net.SQLiteConnection conn = Prihlaseni.Prihlasit();
  List<Odkaz> ListOutput = new List<Odkaz>();
  ListOutput = (from p in conn.Table<Odkaz>() select p).ToList();
  listOdkazy.ItemsSource = ListOutput;
}
```

Ukázka kódu 7- Načítání položek z databáze do ListView - ZDROJ: vlastní zpracování

V ukázce kódu je vidět, že nejprve pomocí metody `Prihlasit()` vytvoříme objekt třídy `SQLiteConnection`. Toto connection už je přihlášeno do správné databáze, kterou představuje soubor s databází. Tento soubor je uložen na adrese, kterou jsme použili v proměnné `path` v metodě `Prihlasit()`. Soubor by měl být v tomto případě ve složce, kde je aplikace na používaném zařízení nainstalována. Tuto skutečnost lze ověřit prohledáním správného adresáře. Pokud už máme přihlášené connection můžeme jít na další krok, ve kterém budeme potřebovat nějakým způsobem naplnit vizuální komponenty aplikace nějakými daty

z databáze, na kterou bylo connection připojeno. Jako první krok musíme zajistit, abychom data z databáze mohli uchovat v nějaké proměnné, ze které budeme následně komponenty plnit. Vytvoříme list objektů třídy `Odkaz`, kterou jsme už vytvořili, a slouží k uchování informací o odkazu. Tento list je nutno naplnit informacemi z databáze, a na to použijeme třetí řádek kódu. Na tomto řádku použijeme zápis podobný dotazu v jazyce SQL. Tento zápis vrátí data uložené v SQLite databázi v tabulce se strukturou stejnou jako má třída `Odkaz`. Knihovna pro práci SQLite, je přizpůsobena k jednoduchému použití v kódu, proto není nutné znát dopodrobna strukturu a syntaxi jazyka SQL, který je normálně používat v externích databázích, jako hlavní dotazovací jazyk. V tomto případě se metody knihoven SQLite postarají, aby tento zápis opravdu vrátil data, které chceme používat. Tyto data, ale bohužel ještě není možné vložit do vytvořeného listu. Byla by možnost kód zkrátit a nepoužívat proměnnou `ObjectList`, do které nejdříve naplníme data z databáze, ale kód by byl výrazně méně přehledný a pro účely demonstrace nepotřebujeme, aby aplikace pracovala nějakou závratnou rychlostí. Nejdříve musíme data nějak strukturovat, abychom je mohli použít jako list. K tomuto účelu slouží metoda `ToList()`, která je použita v předchozí ukázce kódu. V této fázi je vytvořený list naplněn objekty z databáze a můžeme s ním pracovat. Poslední řádek z ukázky připojí naplněný list do `ItemsSource` v `ListView` a tím zaručí správné zobrazení v aplikaci. `ListView` si list vložený do `ItemsSource` samo zpracuje a vloží jednotlivé objekty do řádků `ListView`. Tyto objekty se, podle zápisu kódu v XAML části, zpracují podle použitého data binding. V tomhle případě dojde k přiřazení hodnot proměnných v objektech z databáze do jednotlivých komponent, které se vykreslují v řádcích `ListView`.

Nyní vytvoříme druhou stránku, na které bude detail odkazu. Bude se zde zobrazovat název, url, datum a popis. Tyto informace musí být vidět na jedné obrazovce i na zařízeních s malou obrazovkou. Proto je důležité přizpůsobit toto zobrazení především na tyto zařízení. Pro zobrazení na větších zařízení zafunguje podpora platfrormy Universal Apps a sama rozhraní přizpůsobí, aby vypadalo srávně. Vytvoříme rozhraní, kde budou dvě tlačítka a čtyři `TextBlocky` s názvy jednotlivých atributů a čtyři `TextBoxy`, do kterých budeme plnit jednotlivé informace o odkazu. Návrh by mohl vypadat, jako v následující ukázce kódu. Pokud by tato aplikace nebyla pouze ukázková, byla by nutná optimalizace na různá zařízení. Tato optimalizace by sloužila především pro správné přizpůsobení velikostí komponent podle velikosti obrazovky zařízení, na kterém by byla aplikace spuštěna. Například pevné velikosti některých komponent, jako je to v následujícím kódu, by nefungovali úplně spolehlivě na všech

velikostech obrazovek. Všechny atributy pozic a rozměrů komponent, by měli být přizpůsobivé, bez pevně daných čísel. K tomuto účelu ovšem poslouží uspokojivě.

```
<AppBarButton x:Name="btnZpet" HorizontalAlignment="Left" Icon="Back"
                VerticalAlignment="Top" Width="45" Height="44"
                Click="btnZpet_Click"/>

<AppBarButton x:Name="btnUlozit" HorizontalAlignment="Right" Icon="Save"
                VerticalAlignment="Top" Width="45" Height="44"
                Click="btnUlozit_Click"/>

<TextBlock x:Name="txtNazev" HorizontalAlignment="Left" Margin="10,83,0,0"
TextWrapping="Wrap" Text="Název:" VerticalAlignment="Top"/>
<TextBox x:Name="boxNazev" HorizontalAlignment="Left" Margin="59,75,0,0"
TextWrapping="Wrap" Text="TextBox" VerticalAlignment="Top" Width="291"/>
<TextBlock x:Name="txtUrl" HorizontalAlignment="Left" Margin="10,120,0,0"
TextWrapping="Wrap" Text="URL:" VerticalAlignment="Top"/>
<TextBox x:Name="boxUrl" HorizontalAlignment="Left" Margin="59,115,0,0"
TextWrapping="Wrap" Text="TextBox" VerticalAlignment="Top" Width="291"/>
<TextBlock x:Name="txtDatum" HorizontalAlignment="Left" Margin="10,162,0,0"
TextWrapping="Wrap" Text="Datum editace:" VerticalAlignment="Top"/>
<TextBox x:Name="boxDatum" HorizontalAlignment="Left" Margin="115,155,0,0"
TextWrapping="Wrap" Text="TextBox" VerticalAlignment="Top" Width="235"
IsReadOnly="True"/>
<TextBlock x:Name="txtPopis" HorizontalAlignment="Left" Margin="6,233,0,0"
TextWrapping="Wrap" Text="Popis:" VerticalAlignment="Top"/>
<TextBox x:Name="textBox" HorizontalAlignment="Left" Margin="50,233,0,0"
TextWrapping="Wrap" Text="boxPopis" VerticalAlignment="Top"
RenderTransformOrigin="0.523,-0.466" Height="150" Width="300"/>
```

Ukázka kódu 8 - XAML kód stránky Detail - ZDROJ: vlastní zpracování

Tímto je základ druhé stránky hotový. Teď je nutné vrátit se k `MainPage` a přidat události s navigací na stránku s detailem. Při kliknutí na položku `ListView` se otevře detail odkazu, na který bylo kliknuto. Na toto kliknutí použijeme událost komponenty `ListView` s názvem `Tapped()`. V těle této události naklonujeme objekt odkazu, na který bylo kliknuto a vložíme ho jako parametr do metody `Navigate()` třídy `Frame`, která nám zaručí přesměrování na stránku `Detail`. Tento jednoduchý kus kódu si ukážeme v následující ukázce kódu.

```
private void listOdkazy_Tapped(object sender, TappedRoutedEventArgs e)
{
    if (listOdkazy.SelectedItem != null)
    {
        Odkaz pozn = ((Odkaz)listOdkazy.SelectedItem).Clone();
        this.Frame.Navigate(typeof(Detail), pozn);
    }
}
```

Ukázka kódu 9 - událost Tapped() - ZDROJ: vlastní zpracování

Další navigační prvek, který potřebujeme je v události `Click()` tlačítka `btnPridat`. Při stisknutí tohoto tlačítka očekáváme navigaci na prázdnou stránku detail, ve které lze editovat nový odkaz. K tomuto kroku nepotřebujeme žádný vstupní parametr, takže navigace bude jednoduché zavolání metody `Navigate()` do které vložíme typ třídy `Detail`.

Nyní máme přechod na stránku s detailem zajištěn, takže bude nutné udělat zpracování vstupního parametru při načtení stránky s detailem. Budou existovat dvě situace, ke kterým může dojít. Jedna z nich je, že se stránka otevře bez vstupního objektu odkazu. To znamená vytváření nového odkazu. V tomto případě není nutné nic načítat, pouze vyčistit všechny komponenty. Na to použijeme metodu třídy `Detail`, kterou vytvoříme s názvem `Cistit()`. Tato bude zapsána stejným způsobem, jako by byla zapsána při vytváření jiného projektu, například na Windows Form. Metoda bude demonstrována v následující ukázce kódu.

```
public void Cistit()
{
  boxDatum.Text = "";  boxNazev.Text = "";  boxPopis.Text = "";  boxUrl.Text = "";
}
```

Ukázka kódu 10 - stejný zápis, jako u Win Form app - ZDROJ: vlastní zpracování

Tuto metodu je nutné zavolat vždy při vytvoření stránky. Tím, že se zavolá metoda Cistit() v konstruktoru třídy Detail, zaručíme prázdné komponenty pro následnou práci. Je nutné tuto metodu zavolat v konstruktoru, protože následně budeme vyplňovat údaje z přijatého objektu odkazu, do jednotlivých komponent. Kdyby se metoda volala v nějaké z metod, které následují po vytvoření samotné instance třídy stránky, riskovalo by se smazání vyplněných údajů.

Na naplnění jednotlivých komponent údaji z přijatého objektu odkazu budeme realizovat ve zvláštní metodě třídy `Page`. Touto metodou bude v našem případě `DetailLoaded()`. V této metodě zpracujeme proměnnou `lOdkaz` třídy `Odkaz`. Tato

proměnná je deklarována na začátku třídy `Detail`. Abychom ovšem tuto proměnnou mohli používat, musíme ji nejdříve naplnit správnými daty. K tomu nám poslouží vstupní objekty odkazu, který zpracujeme a vložíme do proměnné `lOdkaz`, abychom s ním mohli jednoduše pracovat. Použijeme metodu `OnNavigatedTo()`, kterou jsme si už v této bakalářské práci představovali. K této události dojde při přepnutí navigace na tuto stránku. Do této metody vstoupí objekt odkazu jako vstupní parametr, který je definován jinou třídou. Stačí parametr přetypovat na třídu, kterou potřebujeme použít a můžeme s ním pracovat, jako s odkazem, jak potřebujeme.

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    if (e.Parameter != null)
    {
        lOdkaz = ((Odkaz)e.Parameter);
    }
    else
    {
        lOdkaz = new Odkaz();
    }
}
```

Ukázka kódu 11 - zpracování vstupního parametru odkaz - ZDROJ: vlastní zpracování

Nyní je lokální proměnná `lOdkaz`, která je třídy odkazu, naplněna správnými informacemi. Zbývá samotné plnění komponent v případě, že nám při navigaci vstoupí nějaký odkaz jako vstupní parametr do stránky. K tomuto úkolu bude sloužit další metoda třídy `Page`, kterou si ukážeme. Její název je v tomto případě `Detail_Loaded()` a v jejím těle pouze naplníme komponenty jednotlivými atributy v objektu typu odkazu, který vstoupil jako vstupní parametr na tuto stránku.

```
private void Detail_Loaded(object sender, RoutedEventArgs e)
{
    if (!lOdkaz.JePrazdny())
    {
    boxDatum.Text = lOdkaz.Datum.ToString();
    boxNazev.Text = lOdkaz.Nazev;
    boxPopis.Text = lOdkaz.Popis;
    boxUrl.Text = lOdkaz.Url;
    }
}
```

Ukázka kódu 12 - Naplnění komponent daty - ZDROJ: vlastní zpracování

V ukázce kódu můžeme vidět metodu třídy odkazu `JePrazdny()`, která vrací hodnotu `true` nebo `false`. Tato metoda jednoduše zjišťuje, jestli v lokální proměnné odkazu nejsou základní hodnoty, které se vkládají v prázdném konstruktoru třídy odkazu. Tyto hodnoty jsou u všech řetězcových proměnných nastaveny na hodnotu „base“.

Nyní už stránka umí zobrazit informace o odkazu, takže jediné co nám zbývá, je ukládání nového, nebo zeditovaného odkazu. Na to nám poslouží metody z knihoven SQLite, které použijeme na práci s databází. Je více způsobů, jakými můžeme ukládat informace, a hlavně jaké informace ukládat, například do proměnné datum. Já jsem se rozhodl použít při každém ukládání aktuální datum a čas.

Ukládat budeme pouze při zmáčknutí tlačítka uložit. Když odkaz neuložíme, pouze se vrátíme na hlavní stránku a změny nikam ukládat nebudeme. To znamená, že vložíme kód pro ukládání do obsluhy události `Click()` tlačítka `btnUlozit`. Na začátku musíme vložit data z komponent do proměnné odkazu, abychom měli co ukládat. Následně se aplikace připojí do databáze, vytvoří tabulku, pokud neexistuje a vloží objekt odkazu. Po uložení se vrátíme na hlavní stránku, kde se nový odkaz zobrazí. V následující ukázce kódu bude demonstrována metoda pro uložení odkazu.

```
private void btnUlozit_Click(object sender, RoutedEventArgs e)
{
    NactiDoOdkazu();
    SQLite.Net.SQLiteConnection conn = Prihlaseni.Prihlasit();
    conn.CreateTable<Odkaz>();
    conn.Insert(lOdkaz);
    this.Frame.Navigate(typeof(MainPage));
}
```

Ukázka kódu 13 - uložení odkazu - ZDROJ: vlastní zpracování

Nyní je hotova plně funkční struktura na zobrazování a vytváření jednotlivých odkazů. Poslední krok, který zbývá je mazání jednotlivých odkazů z databáze. K tomuto účelu bude sloužit stisknutí tlačítka vymazat na hlavním formuláři. Po zmáčknutí tohoto tlačítka se aplikace připojí do databáze a vymaže data podle vybraného objektu z `ListView`. V této metodě je nutné ošetřit možnost, kdy nebude vybrán žádný objekt, což by skončilo výjimkou. V následující ukázce kódu bude vše zobrazeno.

```
private void btnVymazat_Click(object sender, RoutedEventArgs e)
{
    if (listOdkazy.SelectedItem != null)
    {
        SQLite.Net.SQLiteConnection conn = Prihlaseni.Prihlasit();
        conn.Delete((Odkaz)listOdkazy.SelectedItem);
    }
}
```

Ukázka kódu 14 - Mazání označeného odkazu - ZDROJ: vlastní zpracování

Tento kód je správně napsaný, ale bohužel, když se program spustí a dojde k stisknutí tlačítka, aplikace zhavaruje. Chyba je v konstrukci třídy, která byla definována, ještě před implementováním SQLite do projektu, a proto jsme nemohli použít zásadní věc k definici této třídy. Touto věcí je přidání primárního klíče, aby databáze mohla správně identifikovat odkaz, který chceme používat. Jelikož tabulku vytváříme přímo z třídy odkazu, budeme definovat primární klíč tam. K proměnným přidáme atribut ID, který se bude sám zvyšovat v databázi a bude definovat určitý objekt odkazu. V následující ukázce bude finální struktura třídy odkazu.

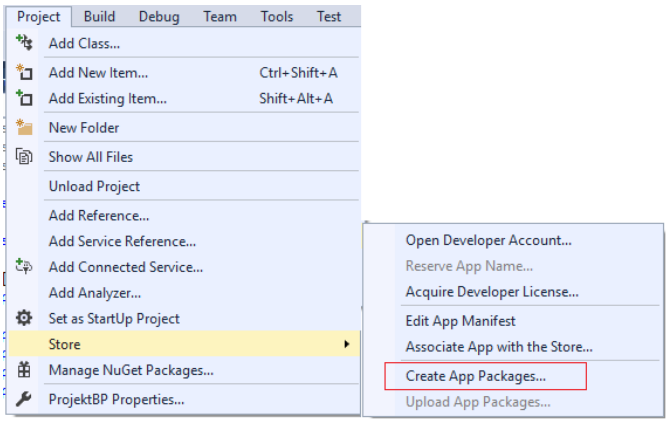
```
[PrimaryKey, AutoIncrement]
public int Id { get; set; }

public string Nazev { get; set; }
public string Url { get; set; }
public DateTime Datum { get; set; }
public string Popis { get; set; }
```

Ukázka kódu 15 - Finální struktura třídy Objekt - ZDROJ: vlastní zpracování

S touto strukturou už jsou knihovny SQLite schopny jednoznačně identifikovat jednotlivé odkazy, takže nyní vše bude fungovat, jak má. Aplikace je tímto hotova a poslední část této kapitoly bude demonstrovat vytvoření balíčku, který lze nainstalovat na jednotlivá zařízení.

Balíček aplikace umí Visual Studio jednoduše vygenerovat. K tomu slouží nástroj, který najdeme v záložce Project, podskupina Store. Zde nalezneme položku Create App Packages.



Obrázek 11 - Vytváření balíčku projektu - ZDROJ: vlastní zpracování

Kliknutím na tuto položku otevřeme nástroj na vytvoření balíčku aplikace. Na první stránce je k zaškrtnutí, jestli má být balíček vytvořen pro nahrání na Windows Store, nebo ne. V případě nahrávání balíčku na Windows Store je potřeba mít developerský Microsoft účet, který je zpoplatněn. V tomto případě vybereme možnost bez nahrávání balíčku na Store a klikneme na tlačítko OK. Na následující obrazovce jsou možnosti generování balíčku, v této ukázkové aplikaci není nutno nic měnit, takže nastavení zůstane základní. Následujeme kliknutím na tlačítko Create. V tuto chvíli už Visual Studio generuje balíček do vybrané složky v předchozím nastavení.

Tuto složku složku otevřeme a vybereme platformu, na které chceme balíček instalovat. Najdeme soubor typu WindowsPoweShellScript, otevřeme jeho nabídku a stiskneme „Run with PowerShell“. Otevře se nástroj Power Shell, kde je nutné povolit instalaci bezpečnostních certifikátu. Po doběhnutí celého skriptu je aplikace úspěšně nainstalována.

4.4 Zhodnocení

V praktické části bakalářské práce byly nejdříve představeny jednotlivé technologie, které byly použity. Tyto technologie byly následně postupně použity při vývoji ukázkové aplikace. Tato aplikace byla navržena velmi jednoduše, jelikož sloužila pouze jako ukázka základního vývoje aplikace na Windows 10. Vývoj byl postupně a strukturovaně popisován. V ukázkách kódu byly demonstrovány jednotlivé metody a třídy, které byly použity při samotném vývoji. Tato aplikace byla úspěšně spuštěna. Následně byly vygenerovány balíčky pro instalaci a popsán postup instalace. Tímto byly obsaženy všechny části vývoje nutné pro vytvoření vlastní jednoduché aplikace na platformě Universal Apps.

5. Závěr

V úvodu teoretické části bakalářské práce byl představen obecně operační systém Windows 10. Nejprve byly popsány jeho počátky a následně bližší specifikace samotného systému. Po vysvětlení základních principů Windows 10 bylo představeno vývojové prostředí Visual Studio 2015, které bylo použito v praktické části k vytvoření ukázkové aplikace. V rámci kapitoly o tomto vývojovém prostředí byly představeny především novinky ve verzi 2015.

Další obsáhlá kapitola se věnovala samotné platformě Universal Apps a vysvětlila její specifikaci a vlastnosti. Dále byly vysvětleny základní možnosti při vývoji aplikací na tuto platformu, čímž byl splněn první cíl bakalářské práce. Rodinám zařízení se věnovala celá kapitola 3.3.3 Rodiny zařízení, ve které byly popsány detailně skupiny, do kterých patří jednotlivá zařízení.

V praktické části bakalářské práce bylo cílem vytvořit ukázkovou aplikaci, která by demonstrovala základní principy vývoje aplikací na Windows 10. Tento cíl se podařilo splnit v kapitole 4.1.3 Vývoj ukázkové aplikace. V této kapitole byl popsán vývoj aplikace, který byl prezentován ukázkami kódu. Po přečtení této kapitoly, by měli být jasné všechny kroky nutné k vytvoření jednoduché aplikace pro Windows 10.

Se znalostmi nabytými z teoretické části bakalářské práce vznikla aplikace a její vývoj byl strukturovaně popsán, čímž se dá považovat cíl práce za plně splněný. Aplikace slouží pouze jako demonstrativní ukázka základních principů, a proto není úplně vhodná pro praktické použití. Po doplnění některých funkcí, jako ukládání dat do externí databáze, otevírání odkazů v prohlížeči a možností připojit obrázek k jednotlivým odkazům, by tato aplikace mohla sloužit i pro praktické využití.

6. Použité zdroje informací

1. Historie Windows. *Windows a Windows 10 – Microsoft* [online]. 2015 [cit. 2016-01-31]. Dostupné z: http://windows.microsoft.com/cs-cz/windows/history#T1=era11
2. Specifikace Windows 10. *Oficiální domovská stránka Microsoft* [online]. 2015 [cit. 2016-01-31]. Dostupné z: https://www.microsoft.com/cs-cz/windows/windows-10-specifications?OCID=win10_null_vanity_win10specs
3. POLESNÝ, David. *Vyšlo Visual Studio 2015, podporuje Android a iOS* [online]. 2015 [cit. 2016-01-31]. Dostupné z: http://www.zive.cz/bleskovky/vyslo-visual-studio-2015-podporuje-android-a-ios/sc-4-a-179067/default.aspx
4. Novinky ve Visual Studio 2015. *Výuka pro vývojáře na webu Microsoft Developer Network | MSDN* [online]. 2015 [cit. 2016-01-31]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/bb386063.aspx
5. Universal Windows Platform. *Výuka pro vývojáře na webu Microsoft Developer Network | MSDN* [online]. [cit. 2016-02-20]. Dostupné z: https://msdn.microsoft.com/library/windows/apps/dn726767.aspx
6. Universal Windows Platform. *Výuka pro vývojáře na webu Microsoft Developer Network | MSDN* [online]. [cit. 2016-02-20]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/apps/dn894631.aspx
7. UWP balíčky aplikací pro Windows 10. *Výuka pro vývojáře na webu Microsoft Developer Network | MSDN* [online]. [cit. 2016-02-21]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/apps/mt148542.aspx
8. UWP balíčky aplikací pro Windows 10. *Výuka pro vývojáře na webu Microsoft Developer Network | MSDN* [online]. [cit. 2016-02-21]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/apps/mt188602.aspx
9. KAČMÁŘ, Dalibor. *Programujeme .NET aplikace ve Visual Studiu .NET*. Vyd. 1. Praha: Computer Press, 2001. Všechny cesty k informacím. ISBN 80-722-6569-5.
10. ČÁPKA, David. Jazyk XAML v C# .NET WPF. *IT network* [online]. **2013**, 3 [cit. 2016-02-25]. Dostupné z: http://www.itnetwork.cz/csharp/wpf/c-sharp-tutorial-wpf-jazyk-xaml/
11. Přehled XAML (WPF). *Výuka pro vývojáře na webu Microsoft Developer Network | MSDN* [online]. [cit. 2016-02-25]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/ms752059(v=vs.110).aspx

12. SHARP, John. *Microsoft Visual C# 2010: krok za krokem*. Vyd. 1. Brno: Computer Press, 2010. Krok za krokem (Computer Press). ISBN 978-80-251-3147-3.
13. HERCEG, Tomáš. Úvod do .net frameworku. *DotNetPortal.cz* [online]. 2009 [cit. 2016-02-25]. Dostupné z: http://www.dotnetportal.cz/clanek/125/Uvod-do-NET-Frameworku
14. MAREŠ, Amadeo. *1001 tipů a triků pro C# 2010*. Brno: Computer Press, 2011. ISBN 978-80-251-3250-0.
15. NASH, Trey. *C# 2010: rychlý průvodce novinkami a nejlepšími postupy*. Vyd. 1. Brno: Computer Press, 2010. ISBN 978-80-251-3034-6.

7. Přílohy na CD

Složka s projektem, vytvořeném ve vývojovém prostředí Visual Studio 2015. V této složce jsou také balíčky aplikace pro instalaci.