BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

Framework pro tvorbu informačních systémů nad Universal Windows Platform (ISUF)

Jan Rajnoha*

Abstrakt
Informační systém je v dnešní době jeden z nejpoužívanějších typů aplikace a je potřeba zjednodušovat jejich návrhy a realizaci pomocí frameworků, kterých existuje celá řada, ale zatím neexistuje framework, který by byl schopný pracovat nad Windows 10 a technologií Universal Windows Platform.
V tomto článku se zaměřím právě na návrh frameworku pro Windows 10 s požadavky na jednoduchost tvorby jednotlivých modulů, aktualizaci a automatickou tvorbu generovaných formulářů. Součástí řešení je i ukázková aplikace, která prezentuje možnosti frameworku jednoduchou formou včetně zdrojových kódů.
Windows 10 je v aktuální době nejrozšířenější operační systém a nabízí velmi širokou škálu zařízení a je tedy výhodné směřovat právě na něj.

Klíčová slova: Framework — Informační systém — Windows 10 — UWP — Aplikační design

Přiložené materiály: GitHub repozitář řešení — Ukázková aplikace

*xrajno09@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Úvod

Frameworky a informační systémy jsou nedílnou součástí dnešních rutin, obzvláště pak ve firemní nebo studijní sféře. Existuje pouze malé procento aplikací a programů, které jsou vytvořeny ”na zelené louce” bez jakékoliv podpory nějakého frameworku a, v případě stavby aplikace nad nějakým operačním systémem, je to už skoro nemožné.

Pokud se podíváme na oblast informačních systémů, tak většina programů je v dnešní době právě z této kategorie. Na fakultě informačních technologií v Brně se na denní bázi používá WIS, který slouží jak pro studenty, tak pro zaměstnance. VUT v Brně používá Apollo, vývojáři například Azure DevOps na verzování kódu nebo ekonomové např. program Pohoda

Dalším velkým oborem v informačních technologiích, na který je potřeba dbát, jsou operační systémy, na který informační systém cílíme. Posledních pár let se rozšiřuje operační systém Windows 10 od společnosti Microsoft. Pro vývojáře je to velmi pohodlný systém obzvláště v oblasti distribuce software pro Windows a jejich aktualizací díky Microsoft Store, který zároveň nabízí i široké možnosti testování mezi uživateli a distribuci doplňujících balíčků.

Windows 10 také nabízí vlastní prostředí nazvané Windows Universal Platform (UWP), jenž díky frameworku .NET Native [1], na kterém když vytvoříme aplikaci, je spustitelné na všech zařízeních běžících na tomto operačním systému. Další výhodou využití Windows 10 je jeho rozšíření, kdy v aktuální době tento operační systém používá více jak 1 miliarda zařízení

[2].

V současné době neexistuje framework, který by se zaměřoval právě na kombinaci těchto aspektů, a z toho důvodu jsem se rozhodl vytvořit vlastní, jenž kombinuje vlastnosti těchto oblastí a nabízí velmi jednoduchý způsob tvorby aplikací, jejich distribuce a aktualizace a přenositelnosti mezi zařízeními právě s Windows 10. Při návrhu bylo dbáno na co nejjednodušší možnost vytvoření funkční aplikace, ať už se jedná o tvorbu modulů, formulářů, správu nastavení a obecně dalšího chování aplikace, kterou je poté možné dále upravovat k vlastním potřebám. Framework nese název "ISUF", v delším znění pak "Framework pro informační systémy nad Windows 10".

2. Požadavky na framework nad UWP

Když se řekne framework, spousta lidí se představí nějakou knihovnu s množstvím obecných funkcionalit zaměřené na danou kategorii. Stejné je to i v našem případě frameworku pro informační systém, kde se očekává řada nástrojů pro jednoduchý návrh a možnou přizpůsobitelnost stávajícího řešení, ať už jde o rozšíření nebo upravení celé funkcionality.

V prvé řadě je potřeba nabídnout vývojáři možnost pracovat odděleně a nezávisle na ostatních částech, aby výsledný produkt nenabobtnal funkcemi, které v závěru ani nevyužijeme. Dále je potřeba oddělit celé řešení na jednotlivé části i z důvodu jejich zaměření. Ne vždy totiž potřebujeme například část pro tvorbu grafického rozhraní nebo pro přístup do úložiště.

Stejně tak je potřeba se zamyslet nad vytvářením jednotlivých modulů, práce s nimi, propojování mezi sebou a jejich zapojení do systému a následném spravování na úrovni reprezentace v celém frameworku. Ne vždy chceme modul zobrazovat pomocí UI nebo všechny jeho data ukládat (používání vypočítaných hodnot). Některé moduly mohou být pouze informativní a jako celek se neukládají, protože se používají pro reprezentaci stavu aplikace.

Kromě rozdělení na samostatné části je nutné dbát i na samotné vývojáře a ulehčení jejich práce vhodnou volbou návrhu jednotlivých modelů, práce s atributy a úpravou vlivu modelu na chod aplikace, jako je např. schování položky ve formuláři nebo její ignorování při ukládání do databáze. Samozřejmostí je také možnost úpravy chování pomocí globálních konstant a nastavení aplikace, jejich ukládání a rozšiřování.

Zaměřit se také musíme i na oblast, na jakou cílíme. Jestli výsledný produkt bude spíše obecný nebo bude řešit jednu velmi specifickou úlohu. Jestli budeme v návrhu počítat s úpravami ku potřebám vývojáře nebo to bude čistě pevně dané řešení, které se upraví pouze o základní oblasti, jako jsou např. modely. Zde se vychází především z již existujících řešení, jako jsou např. informační systémy GTrade a ILTEGRO pro stejnojmenné firmy vyvinuté lidmi z IT Spektrum nebo řešení Microsoft Dynamics [3].

Dalším aspektem, jenž je důležitý pro výsledný produkt, je rychlost řešení, tedy optimalizace. Zde je potřeba správně navrhnout přístup do úložiště, načítání uživatelského rozhraní a celkovou náročnost řešení (množství provedených operací mezi požadavkem uživatele a odpovědí aplikace). Do této kategorie spadá i volba technologie, na které bude daný framework realizován. Jak již zaměření napovídá, vybral jsem technologii Universal Windows Platform pro Windows 10.

Jak jsem již ale zmínil, Windows 10 a prostředí UWP nabízí pro vývojáře velmi příjemné prostředí pro distribuci software a velmi povedenou optimalizaci práce aplikace se systémovými zdroji, ale v případě vývoje může být značně omezující, protože se snaží, aby aplikace vytvořené právě pomocí UWP technologie byly co nejbezpečnější, což v důsledku znamená možné nedostatky a omezení. Jedním z nich je například přístup k souborovému systému. Aplikace je ve výchozím stavu schopná pracovat ve svém adresáři nebo v Downloads složce, ale již nedokáže získat přístup třeba k základním uživatelským složkám, jako jsou Pictures, Music nebo třeba Videos a dalším složkám souborového systému. Další omezení přináší samotná technologie jako taková, protože je limitována pouze na operační systém Windows 10 a instalace balíčků je doporučována přes Store (nicméně lze instalovat aplikace i mimo něj), kde, pokud chceme publikovat, si musíme koupit licenci.

Když to shrneme a vytvoříme nad těmito body SWOT analýzu, můžeme vidět, že problémy při návrhu budou dělat značná omezení použité technologie. Na druhou stranu, unikátnost řešení a možnosti distribuce vyvažují celý další postup natolik, že výsledný produkt (v případě této práce pouze koncept) bude jedinečný a ukáže možnosti, které tato technologie má.

SWOT analýza:

Strengths - silné stránky:

- Distribuce aktualizací
- Jednoduchost řešení
- Možnosti přizpůsobení chování

Weaknesses - slabé stránky:

- Omezený přístupu k souborovému systému
- Omezené možnosti při vývoji
- Pouze Windows 10

Distribuce přes Microsoft Store vyžaduje licenci

Opportunities - příležitosti:

Jedinečný koncept v rámci Windows 10

Threats - hrozby:

Aktuálně nejasný vývoj UWP platformy

3. Dostupná řešení

Pokud se podíváme, jaká jsou dostupná řešení, tak v oblasti UWP zatím podobný produkt neexistuje a tím se tento koncept stává ještě více ojedinělým. V případě jiných desktop frameworků, založených na technologiích WinForms a WPF, již můžeme vybírat. Mezi známější patří například:

DevExpress eXpressApp Framework

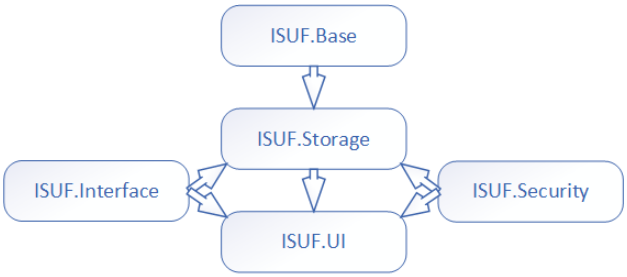
Společnost Developer Express Inc. patří v dnešní době k jedním z největších dodavatelů různých vývojových komponent v oblasti vzhledu aplikací, ale i služeb. Jejich frameworky jsou cílené na spousty platforem a řešení. Jednou z těchto oblastí je i tvorba informačních systémů, kterou pokrývá právě eXpressApp Framework [4]. Tento nástroj je placený, ale nabízí i 30denní zkušební dobu. Distribuce je pomocí jednoho velkého balíku řešení (přes 600 prvků grafického rozhraní), který je nutné zakoupit jako celek a je tedy finančně náročnější, nicméně součástí je velmi podrobná dokumentace s postupným návodem, jak pracovat s daným frameworkem. Řešení podporuje tvorbu uživatelského rozhraní a nabízí podporu pro většinu specifik, jenž byly zmíněny v sekci "Požadavky na framework nad UWP" (Sekce 2).

MM .NET Framework

Framework společnosti Oak Leaf Enterprise, Inc. [5] nabízí řešení jak pro WinForms, tak pro WPF. Cena za jednoho vývojáře na rok není příliš vysoká, ale při porovnání s řešením od DevExpress je limitující, obzvláště pak v případě práce s databází a automaticky generovaných formulářů, které nepodporuje vůbec. Dokumentace k řešení je podrobná a na vyžádání je možné vyzkoušet zkušební verzi.

Vlastní řešení od A do Z

Další možností je také tvorba vlastního řešení z již existujících komponent při využití více či méně pomocných knihoven. Mezi tvůrci komponent je nutné zmínit společnost Syncfusion, Inc. [6] nebo Progress Software Corporation a jejich balíček Telerik [7]. Problémem vlastního řešení je ztráta všech specializovaných funkcí, jako je např. automaticky generované tabulky v databázi nebo generované formuláře. Na druhou stranu, dané řešení může být rychlejší, protože není zatížené obecnými funkcemi, které musí framework obsahovat.



Obrázek 1. Vazby mezi částmi ISUF frameworku

4. Návrh frameworku ISUF

Nejprve, než se podíváme na návrh řešení, vysvětlíme si základní pojmy, které se zde vyskytují.

- **Modul** - Jednotka aplikace. Obsahuje způsob, jak s ním aplikace pracuje a datovou třídu (entita).
- **Komponenta** - Část designu. Uživatelský prvek, z nichž se potom budou skládat jednotlivé view (zobrazení).
- **View, zobrazení** - Části okna, jenž využívají view modely.

Ještě jednou vyjmenuji požadavky na framework, které jsme si rozebrali v kapitole 2. a vyberu i oblasti, které jsem se rozhodl realizovat:

- Rozčleněné řešení do více samostatných částí
- Jednoduchá tvorba modulů s následnou tvorbou databáze
- Možnost propojovat moduly mezi sebou a ostatní úpravy
- Přednastavené moduly
- Automatické generování formulářů
- Obecný přístup, které si vývojář přepíše v případě nutnosti pro vlastní potřeby
- Optimalizace
- Využití možností Windows 10 a UWP technologie a možná přenositelnost na jiné platformy

Nyní se podíváme na nejdůležitější požadavky, které nejvíce ovlivňují následný vývoj celého řešení a které si zároveň i podrobněji rozebereme.

Rozdělení do samostatných částí

Při návrhu jsem se snažil dát vývojáři možnost pracovat pouze s nějakými částmi řešení. Pokud chce, může si na něm postavit vlastní, které již vyšší části

potřebovat nebude. Stejně tak může využít celý framework a upravovat jen dílčí chování a reakce na události. Další výhodou je také jednodušší aktualizování, kdy se nemusí výsledná aplikace distribuovat s velkým balíkem, ale pouze jen s upravenou částí.

Jednoduchá tvorba modulů

Každý modul je tvořen velmi jednoduchým způsobem, jako samostatná třída, která musí implementovat základní rozhraní. Vše ostatní se poté řeší pomocí atributů. Každý datový typ sebou nese jistá omezení atributů, ale v základu má vývojář volnou ruku a podporovány jsou všechny základní datové typy. Jednotlivé moduly jsou registrovány a při vytváření spojení s databází se kontroluje, zda jsou tabulky aktuální a není nutnost je aktualizovat. Tím má vývojář zjednodušenou práci, protože nemusí znát žádné parametry použité databáze, jen mu stačí znát přístup, nejlépe připojovací řetězec (connection string) a framework obstará všechny ostatní nutné kroky k nastavení spojení.

Propojování modulů

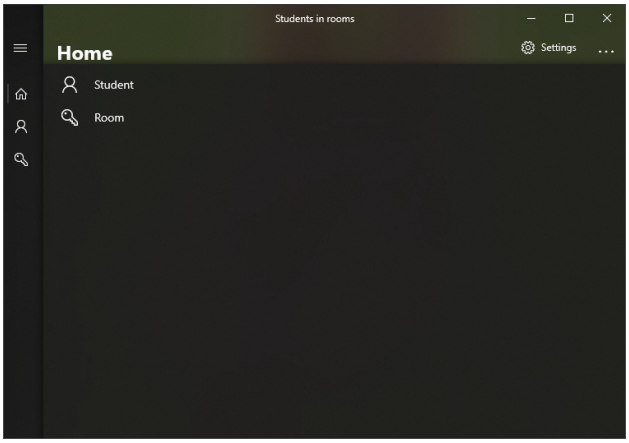
Jak jsem již zmiňoval, veškeré úpravy se tvoří pomocí atributů. Relace mezi tabulkami se pak dělají pomocí číselných datových typů a seznamu číselných datových typů, které určují vazbu 1:N. Pomocí atributu specifikujeme modul, ke kterému se daná vazba váže a uložená hodnota pak určuje ID jednotlivých propojených záznamů. Vazba N:M v aktuální době není podporována.

Automatické generování formulářů

Další specialitou frameworku jsou automaticky generované formuláře, které právě pomocí modulu a atributů jednotlivých vlastností modulu vytvoří daný design. Jsou 2 typy formulářů, jeden pro práci se záznamem (přidání a editace), další pak pro zobrazení dat v režimu pouze pro čtení, kdy se zobrazí i další vypočítané hodnoty, které se do tabulky neukládají. Navíc všechny data ke vzhledu formuláře se po vygenerování ukládají, aby při dalším otevření bylo jejich načítání rychlejší.

Windows 10, UWP a možná přenositelnost

Stěžejní požadavek na řešení, propojení frameworku s Windows 10 a postavení na technologii UWP tak, aby využíval co nejvíce jejich možností, jako jsou aktualizace právě přes Microsoft Store. Framework je již od prvního momentu stavěn výhradně pro platformu UWP, ale vzhledem k vývoji na .NET Core je jeho možnost vytvoření alternativy pro ostatní systémy



Obrázek 2. Hlavní stránka ukázkové aplikace

velmi jednoduchá.

Po podrobné analýze těchto požadavků byl navržen systém registrace modulů a přednastaveného UI, kdy vývojáři stačí vytvořit modul a registrovat jej při spuštění aplikace. V případě registrace je zapotřebí mít unikátní datovou třídu použitou v našem modulu a unikátní název daného modulu. Pokud chceme využít vyšších možností modulu, např. zobrazení v aplikaci, je nutné implementovat další požadované vlastnosti. Framework proto nabízí kromě základního modulu také 2 další typy, UI modul a datový modul, který je možné použít i pro ukládání do databáze a vytváření relací mezi jednotlivými tabulkami.

Při každé registraci se pak provede analýza modulu pomocí reflexe a anotací jednotlivých vlastností, zda splňuje požadované podmínky, aby nedošlo k práci s chybnými datovými typy anebo nastavení protichůdných atributů. Framework interně pracuje v oblasti designu pomocí návrhového vzoru MVVM (Model-View-ViewModel), a proto pro tvorbu UI částí je nutné vytvořit všechny view (design, zobrazení) modulu a jeho view modely, ale jen pro potřeby dědění, protože framework už většinu dat doplní sám a jsou pak místem pro vývojářovi další úpravy.

5. Použití frameworku ISUF v aplikaci

Implementace frameworku je velmi jednoduchá. V aplikaci stačí vytvořit základní zobrazení (hlavní stránka a zaobalení celé aplikace), jejich příslušné view modely, moduly, UI částí modulu a jeho view modely, ale pro všechny tyto části jsou připraveny abstraktní třídy, které stačí v daném souboru zdědit. V této kapitole ukážu, jak by se vytvořil jednoduchý systém s přednáškovými místnostmi a studenty v nich.

Prvně si naimportujeme z NuGet balíčky ISUF.UI, který stáhne všechny potřebné balíčky nižších vrstev, dále Template10 ve verzi 1.1.12 a Newtonsoft.Json ve

verzi 10.0.3.

Pro ukázku studenta nám postačí vytvořit modul se jménem studenta a místností, ve které se nachází. Pro místnost obdobně název a seznam studentů v ní.

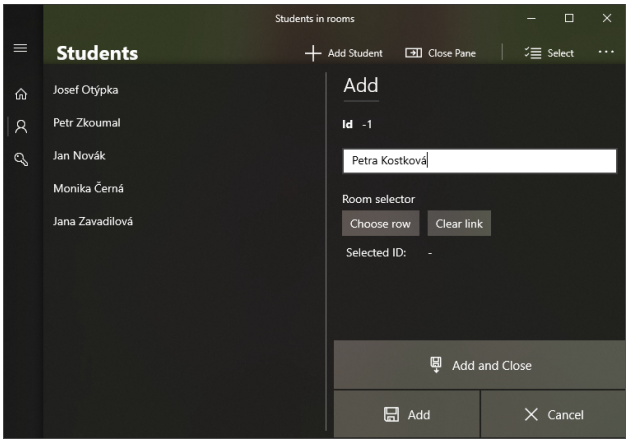
```
public class Student : AtomicItem
{
 public string Name { get; set; }

 [LinkedTable(
  LinkedTableRelation =
  LinkedTableRelation.One,
  LinkedTableType = typeof(Room))]
 public int Room { get; set; }
}

public class Room : AtomicItem
{
 public string Name { get; set; }

 [LinkedTable(
  LinkedTableRelation =
  LinkedTableRelation.Many,
  LinkedTableType = typeof(Student))]
 public List<int> Students { get; set; }
}
```

Tím jsme vytvořili základní moduly. Oba moduly budeme chtít zobrazit v aplikaci, bude tedy nutné k nim vytvořit další třídy, jako jsou hlavní stránka modulu, view modely a komponenty pro zobrazení detailu a přidání záznamu. Nejprve začneme view modely. V ukázce si představíme view model pro modul studenta, který dědí od třídy ModuleVMBase a který očekává jako generický parametr třídu modulu. Poté naimportujeme všechny funkce abstraktní třídy a doplníme funkčnost. Důležité je upravit funkci AddPane, která obstarává volání a vytváření všech vedlejších panelů (přidat/editovat a detail).

```
public class StudentViewModel :
 ModuleVMBase<Student>
```

Další komponenty, které si vytvoříme, budou panely pro práci a zobrazení záznamu. Všechny tyto komponenty mají i design část napsanou v jazyce XAML, kde je nutné nahradit základní třídu UserControl za příslušné třídy odkazující na daný typ. Pro detail je ModuleDetailControlBase a pro práci se záznamem ModuleAddControlBase. Implementace obou tříd je stejná, takže si ukážeme pouze způsob vytvoření komponenty na práci se záznamem.

```
public sealed partial class Add :
 ModuleAddControlBase
{
 public Add(UIModule uiModule,
  Type viewModelType,
  params object[] viewModelArgs) :
  base(uiModule,
  viewModelType,
  viewModelArgs)
 {
 }
}
```



Obrázek 3. Ukázka zobrazení modulu včetně bočního panelu

```
<isuf:ModuleAddControlBase
 xmlns:isuf="using:ISUF.UI.Views"
 x:Class="FITVUTBasicIS.Controls.Add"
```

A samozřejmě k nim vytvořit příslušný view model, zde opět ukázka pro práci se záznamem a pro modul Student. U view modelu pro práci se záznamem (anebo pro zobrazení detailu záznamu) je nutné dědit třídu ModuleAddVMBase (ModuleDetailVMBase), která očekává generický parametr typu daného modulu. Opět implementujeme abstraktní třídu a doplníme nutné chování.

```
public class StudentAddViewModel :
 ModuleAddVMBase<Student>
```

Pro práci se zobrazeným modulem se pak jedná o třídu ModulePageBase. V ukázce implementace pro modul Student.

```
public sealed partial class StudentPage :
 ModulePageBase
{
 public StudentPage() :
  base(typeof(StudentViewModel))
 {
 }
}
```

V případě komponenty pro práci se záznamem a detailem záznamu je nutné implementovat bohatý konstruktor, který určuje jak UI modul, tak view model, který se má použít pro tuto komponentu, a jeho argumenty. Pro zobrazení pak jen stačí určit typ view modelu, který odpovídá danému modulu pro zobrazení a v základu není nutné pro něj přidávat žádné argumenty. Tímto způsobem lze použít jednu komponentu pro všechny moduly a jen specifikovat její chování při inicializaci. V případě UI modulu není nutné speci-

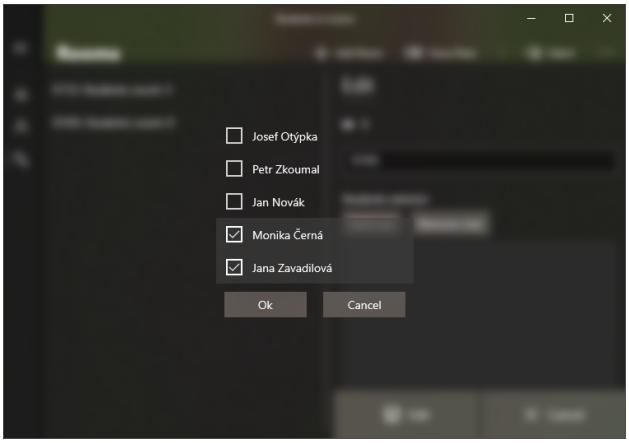
fikovat modul, protože je vyčten z view modelu, který v základu také nepotřebuje specifikovat argumenty.

Nyní máme vytvořenou datovou část, tudíž se můžeme vrhnout na obecné prvky aplikace. Nejprve si vytvoříme view model pro hlavní stránku. Zde stačí pouze implementovat dědičností abstraktní třídu MainPageVMBase. Postup je stejný jako u výše zmíněných view modelů.

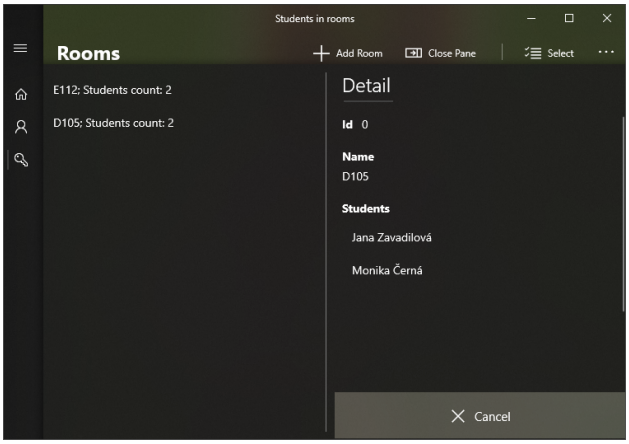
Další zobrazení (view), které je nutné implementovat, je stránka pro hlavní stránku, tedy to, co uvidíme hned po spuštění a co nám zobrazí seznam všech dostupných modulů, se kterými můžeme pracovat v aplikaci, a zobrazení, které zaobalí celou aplikaci. Opět bude nutné implementovat dědičnost abstraktní třídy do obou souborů (xaml a xaml.cs) zobrazení. V případě hlavní stránky pro dědičnost se využije třída ModulePageBase a po implementaci abstraktní třídy vidíme konstruktor, do kterého vložíme typ view modelu, který použijeme pro hlavní stránku. Pro zaobalení aplikace se používá třída ShellBase, kterou stačí implementovat a již ve výchozím stavu je plně funkční.

Nyní máme připravené všechny potřebné části a můžeme provést registraci ve třídě App. Zde budeme dědit třídu ApplicationBase, dědičnost použijeme v obou souborech (xaml a xaml.cs) a implementujeme abstraktní třídu. Dědičnost pro XAML jsme si ukázali již v příkladech výše, proto tedy jen základ implementace v kódu v C#.

```
public sealed partial class App :
 ApplicationBase
{
 public App() :
  base(Package.Current.DisplayName,
  typeof(Shell),
  typeof(MainPage),
  null)
 {
  InitializeComponent();
 }
 public override void RegisterModules()
 {
  ModuleManager =
   new UIModuleManager(
    typeof(XmlDbAccess));

  var studModule = new UIModule(
   typeof(Student),
   typeof(BaseItemManager),
   "Students", (Symbol)0xE13D,
   typeof(StudentPage));

  var roomModule = new UIModule(
   typeof(Room),
   typeof(BaseItemManager),
   "Rooms", (Symbol)0xE192,
   typeof(RoomPage));

  ModuleManager.RegisterModule(studModule);
  ModuleManager.RegisterModule(roomModule);
 }
```



Obrázek 4. Zobrazení propojené tabulky pro přidání do záznamu



Obrázek 5. Zobrazení informací o položce

Můžeme si všimnout konstruktoru, který obsahuje název aplikace, který se zobrazí v horní liště programu, typ třídy, která zaobalí celou aplikaci, dále samozřejmě typ hlavní stránky a jako poslední lze využít typ pro automatické vytvoření stránky pro nastavení aplikace.

Nejdůležitější částí je ale funkce RegisterModules, která slouží pro registraci všech modulů v aplikaci. Nejprve musíme inicializovat správce modulů, který očekává jako argument třídu pro přístup k datům. Dále budeme registrovat dva moduly se zobrazením pomocí třídy UIModule, která očekává v prvním argumentu typ modulu, dále typ třídy, která bude pracovat se záznamy, název modulu v aplikaci, ikonu z fontu Segoe MDL2 Assets a jako poslední je typ třídy pro zobrazení. Nakonec moduly zaregistrujeme do správce a máme hotovou aplikaci pro správu studentů a jejich přítomnost v přednáškových místnostech.

6. Ukázková aplikace

Abychom mohli nějak prezentovat výsledný produkt, bylo zapotřebí vytvořit aplikaci, která využívá možno-

sti frameworku. Z toho důvodu vznikla aplikace BrnoParkingIS. Základem jsou 2 moduly pro reprezentaci auta a parkoviště. Modul auta je reprezentován značkou a SPZ, parkoviště pak názvem. Mezi moduly je vazba 1:N, kdy na parkovišti je N aut, ale auto jen na 1 parkovišti. Aplikace jako způsob ukládání dat používá XML soubor ve složce aplikace.

Zdrojové kódy jsou dostupné na GitHub a pro zprovoznění je potřeba Windows 10 verze 1903.

7. Závěr

Práce se věnuje problematice tvorby frameworků nad operačním systémem Windows 10 a jeho technologií UWP, možnosti zjednodušené tvorby informačního systému a jeho použitelnost v praxi. Navrhované řešení je prezentováno jako koncept a ukázka možného postavení se k problému. Pro potřeby prezentace funkčního prototypu vznikla aplikace pro správu Brněnských parkovišť BrnoParkingIS, která využívá v největší možné míře možnosti frameworku a nechává celé řízení na něm.

Zde představená práce je sice koncept, ale věřím, že mnou navrhnuté řešení je použitelné v komerční sféře, a to i přes aktuální nedokončenost. Když se podíváme na zdrojový kód, můžeme si všimnout množství znovu implementovaných postupů, které bylo nutno vytvořit, protože použitá technologie v kombinaci s řešením je nepodporuje. Na druhou stranu nám ale nabízí jednoduchý způsob aktualizací a jednotnou platformu s obrovským množstvím zařízení, jejichž počet se stále rozrůstá.

Literatura

[1] .net native github readme. Online documentation, March 2019. https://github.com/Microsoft/dotnet/blob/master/releases/UWP/README.md.

[2] Yusuf Mehdi. Windows 10: Powering the world with 1 billion monthly active devices. Blogpost, March 2020. https://blogs.windows.com/windowsexperience/2020/03/16/windows-10-powering-the-world-with-1-billion-monthly-active-devices/.

[3] Microsoft dynamics. Online documentation, April 2020. https://dynamics.microsoft.com/en-us/.

[4] expressapp framework. Online documentation, August 2019. https://docs.devexpress.com/eXpressAppFramework/112670/expressapp-framework.

[5] Mm .net application framework. Product page. https://www.oakleafsd.com/products.

[6] Syncfusion. Documentation. https://help.syncfusion.com/.

[7] Telerik. Documentation. https://www.telerik.com/support.