VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

RÁMEC PRO TVORBU INFORMAČNÍCH SYSTÉMŮ NAD UNIVERSAL WINDOWS PLATFORM

INFORMATION SYSTEMS DEVELOPMENT FRAMEWORK FOR UNIVERSAL WINDOWS PLATFORM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **JAN RAJNOHA**
AUTHOR

VEDOUCÍ PRÁCE — **RNDr. MAREK RYCHLÝ, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Rajnoha Jan**
Program: Informační technologie
Název: **Rámec pro tvorbu informačních systémů nad Universal Windows Platform**
Information Systems Development Framework for Universal Windows Platform
Kategorie: Uživatelská rozhraní

Zadání:

1. Nastudujte technologii Universal Windows Platform (UWP) pro vývoj software ve Windows 10. Seznamte se také s řešením Windows aplikací ve Windows Store. Prozkoumejte a porovnejte existující rámce pro rychlý vývoj informačních systémů.
2. Navrhněte softwarový rámec pro rychlý vývoj informačních systémů pro UWP. Navrhněte také aplikaci demonstrující uvedený rámec a využívající co nejvíce možností UWP.
3. Po konzultaci s vedoucím rámec i ukázkovou aplikaci implementujte a obojí důkladně otestujte. Diskutujte použitelnost rámce.
4. Výsledek popište, vyhodnoťte a zveřejněte jako open-source.

Literatura:

- Microsoft. Universal Windows Platform documentation. [https://docs.microsoft.com/en-us/windows/uwp/]
- Joseph Albahari, Ben Albahari. C# 7.0 in a Nutshell: The Definitive Reference. O'Reilly Media, 2017. ISBN 978-1-491-98765-0. [http://www.albahari.com/nutshell/]

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Rychlý Marek, RNDr., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 16. října 2019

Abstrakt

Tato práce se zabývá analýzou a návrhem vlastního frameworku pro aplikace vytvořené pro operační systém Windows 10. Hlavním přínosem rámce je zjednodušené vytváření informačního systému nad platformou Universal Windows Platform. V práci jsou porovnána a zhodnocena již dostupná řešení a jejich schopnost adaptace na dostupná zařízení. Výsledkem je návrh a realizace vlastního frameworku, který splňuje stanovené požadavky analýzy a využívá výhody konkurenčních řešení. Na závěr práce jsou rozpracovány možné podoby dalšího vývoje.

Abstract

This thesis analyses and designs a custom framework for applications created for operating system Windows 10. The main benefit of the framework is that it simplifies creation of an information system targeting the Universal Windows Platform. The thesis compares and discuss existing solutions and their ability to target available devices. Result of this work is design and implementation of a custom framework, which satisfies requirements from the analysis and preserves strong suits from competitive solutions. At the end of the thesis I explore options for further development.

Klíčová slova

.NET Framework, .NET Core, .NET Native, Entity Framework, Windows 10, Microsoft Store, UWP, WPF, WinForms, Framework, Informační systém, Syncfusion, Telerik, DevExpress, C#, MVVM, Messenger

Keywords

.NET Framework, .NET Core, .NET Native, Entity Framework, Windows 10, Microsoft Store, UWP, WPF, WinForms, Framework, Information system, Syncfusion, Telerik, DevExpress, C#, MVVM, Messenger

Citace

RAJNOHA, Jan. *Rámec pro tvorbu informačních systémů nad Universal Windows Platform.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce RNDr. Marek Rychlý, Ph.D.

Rámec pro tvorbu informačních systémů nad Universal Windows Platform

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením RNDr. Marka Rychlého, Ph. D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Jan Rajnoha
26. května 2020

Poděkování

Rád bych poděkoval vedoucímu práce, doktorovi Markovi Rychlému, za odborné vedení a umožnění vypracování vlastního tématu a jeho nadšení pro něj. Dále pak rodičům a své přítelkyni, kteří při mě stáli a vytvářeli mi vhodné podmínky pro práci. V neposlední řadě děkuji všem přátelům a uživatelům za test frameworku a zpětnou vazbu.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Dnešní doba je plná zařízení různých rozměrů a výkonnostních parametrů. V kapse máme mobil, na ruce chytré hodinky, na stole notebook nebo stolní počítač, na stěně vedle nás může být zavěšené malé zařízení pro část chytré domácnosti a ve druhé místnosti bez problému celý server. Jsme doslova zahlceni moderními technologiemi. Drtivá většina těchto zařízení má ale jednu věc společnou, operační systém, jenž tohle zařízení pohání.

Další nedílnou součástí našich životů jsou informační systémy. Žáci středních škol používají např. systém Bakaláři, který slouží nejen ke klasifikaci studentů, studenti VUT v Brně znají systém Apollo a pro studenty FIT VUT je důležitý systém WIS. Ve sféře podnikové pak najdeme nespočetné množství systémů, které slouží ke sledování zboží na skladech, správě účetnictví, zakázek, lidí a dalších oblastí.

Vytvořit komplexní program nebo aplikaci není otázka pár dnů, ale je to vždy dlouhý vývojový proces, který může trvat i více než rok. Od nápadu, přes první návrhy, prototypy, testování až první prezentovatelný výsledek, je nutné ujít velký kus cesty. Každé ulehčení, které může vývojář využít, proto velice rád využije. Jednou z takových pomůcek jsou různé frameworky, ať už pro prototypování aplikací nebo jejich realizaci.

V této práci se zaměřuji na možnosti tvorby informačního systému pomocí frameworku právě na jednom z nejpoužívanějších operačních systémů, tedy Windows, přesněji jeho poslední verzi označené číslem 10, pomocí univerzálního frameworku, který umožní využít celou škálu zařízení s tímto operačním systémem, a také využít plný potenciál řešení společnosti Microsoft, ať už v oblasti cloud služeb (Azure, SQL, OneDrive, Microsoft 365), tak v oblasti lokální (notifikace, živé dlaždice v nabídce Start, designový jazyk Fluent Design System, a další). Kombinací všech těchto faktorů lze vytvořit řešení, které bude na všech zařízeních vypadat stejně, bude disponovat intuitivním ovládáním a vysokou spolehlivostí.

Dalším důvodem, proč se dané oblasti věnuji, je absence podobného řešení na dané platformě. Pokud se podíváme na konkurenční frameworky pro operační systém Windows, je nutné si povšimnout, že jich není mnoho a navíc ani jeden není schopen pracovat s moderním rozhraním Windows 10.

Cílem této práce je tedy navrhnout framework, který ulehčí tvorbu informačního systému nad moderním rozhraním Windows 10 zvaným UWP (Universal Windows Platform), zrychlí vývoj daného informačního systému a poskytne vývojáři snadnou modifikaci v případě různých adaptací vyžadovaných pro specifická řešení.

Ve 2. kapitole se zaměřím na popis frameworku .NET, platformy UWP, popis a distribuci aplikací přes Microsoft Store a popíšu zde motivaci, která mě vedla k zájmu o tuto problematiku.

Kapitola 3. se bude věnovat stanovení požadavků, označení silných a slabých míst frameworku, výběru možných doplňků speciálně pro Windows 10 a nastínění řešení převzatých funkcí z již hotových a dostupných frameworků.

V kapitole 4. zhodnotím již hotová řešení a jejich použitelnost v obecné rovině tvorby informačního systému, protože jak bylo již výše zmíněno, žádný není schopen pracovat s moderním rozhraním Windows 10.

Další kapitola bude věnována návrhu architektury frameworku, rozdělení do jednotlivých, samostatně funkčních celků a návrhu řešení nedostatků již existujících produktů. Dále zde rozeberu určité klíčové části, popíšu zde problémy spojené s vývojem a jejich řešení a vývoj grafické části frameworku s ohledem na zpětnou vazbu.

6. Kapitola se bude zabývat vývojem testovací aplikace, která byla vyvinuta za účelem prezentace, a uvedením některých problémů a jejich řešení, se kterými jsem se při vývoji setkal.

V 7. kapitole ukáži další možné cesty vývoje mého řešení. Nástin server aplikace s multi-user přístupem, rozšířenými možnostmi týmové práce a tvorbou dashboardů.

Závěr práce je věnován shrnutí vytvořeného řešení, kontrole stanovených požadavků a výběru nejdůležitějších částí pro další vývoj frameworku.

Kapitola 2

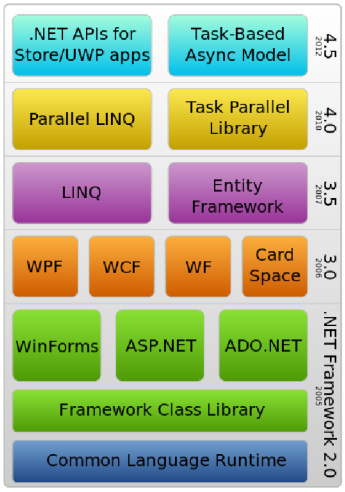
Motivace a popis technologií

V dnešní době je zvláštní pokoušet se psát framework určený pouze pro jednu platformu. Stále více a více sledujeme upřednostnění univerzálních webových řešení, které jsou sice použitelné na mnohem větší škále zařízení, ale nejsou příliš schopné využít všechny možnosti, které systém nabízí. Webový prohlížeč nabídne v mnohých případech jenom základní práci se systémovými prvky, například schopnost zobrazovat notifikace v centru oznámení, ale dál se už nedostane. Problémy mohou tvořit i nestandardní periferie zařízení, se kterými si webové řešení nedovede poradit. Samozřejmě je možnost vytvářet PWA aplikace. PWA, neboli Progressive Web app, je označení aplikace, která kombinuje webový prohlížeč a nativní aplikaci. [25] Tohle řešení však nemusí být pohodlné pro zařízení typu HoloLens anebo Xbox, které vyžadují speciální přístup. Dalším důvodem je nevyspělost PWA aplikací, jsou pomalejší a nemají dokonalý přístup k ostatním částem systému. [10] Z tohoto důvodu jsem se rozhodl pro tvorbu frameworku použít platformu UWP. Pod touto zkratkou se skrývá Universal Windows Platform, které označuje API (Application Programming Interface) pro tvorbu aplikací určených pro nejnovější verzi Windows. Další výhodou využití UWP je .NET Framework, který je v aktuální podobě nepoužitelný pro další multiplatformní vývoj, ale pouze s malými změnami můžeme celou aplikaci přepsat na .NET Core, který požadovanou multiplatformnost nabídne, jehož výsledná aplikace bude fungovat téměř kdekoliv a na téměř kterémkoliv zařízení. [8] K tomu lze využít známé nástroje, mezi které patří například Xamarin.

V mé práci jsem se rozhodl použít programovací jazyk C#, využít framework .NET Core a prostředí Universal Windows Platforms. Pro práci s databází (zkráceně DB) byl vytvořen vlastní code first přístup.

2.1 .NET Framework

.NET Framework, nebo jen zkráceně .NET, je označení pro framework vyvinutý společností Microsoft, který slouží k vytváření a běhu aplikací na něm postavených. [39] Rodina .NET frameworků je velmi rozsáhlá a obsahuje např. frameworky ASP.NET pro tvorbu webových aplikací [6], ADO.NET, který slouží pro databázovou komunikaci [31] a WPF (Windows Presentation Foundation) pro tvorbu grafického rozhraní. [27] První verze tohoto frameworku byla vydána 14. 2. 2002 a od té doby udělal značný pokrok ve vývoji a prošel řadou významných milníků, jako bylo např. přidání podpory WPF, LINQ nebo Entity Frameworku. [36]

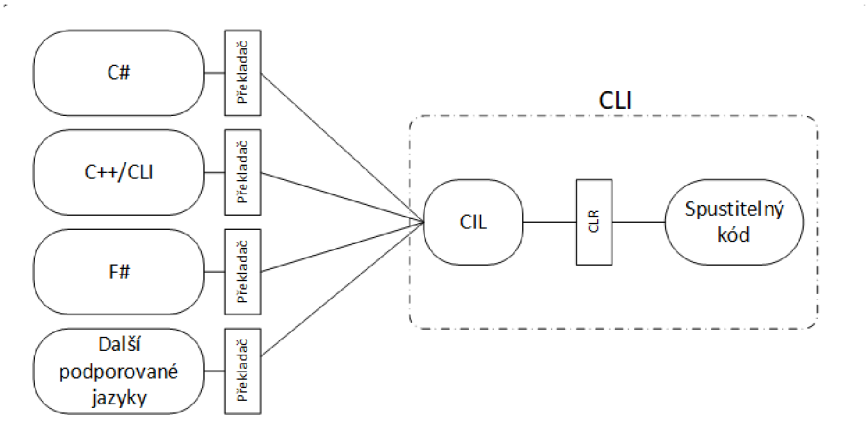


Obrázek 2.1: Struktura .NET Framework [38]

Důležitou částí tohoto frameworku je *CLR* (Common Language Runtime), který byl od první verze součástí frameworku. [9] Jedná se o společný virtuální běhový modul pro jazyky rodiny .NET, který má na starosti překlad a spuštění celého programu. Zároveň poskytuje knihovny, které proces vývoje usnadňují.

Dále je nutné zmínit CIL a CLI. CIL (Common Intermediate Language) slouží spolu k CLR ke generování CLI (Common Language Infrastructure). CLI je stěžejní pro .NET v nezávislosti jazyků, protože takto vygenerovaný kód je stejný pro jakýkoliv jazyk z rodiny .NET. Pro vývojáře to potom znamená, že pokud použijí jakýkoliv jazyk splňující tyto specifikace (CLS - Common Language Specification), mohou ve svých projektech používat i komponenty vytvořené v různých jazycích, které splňují CLS. [30]

Postup překladu je pak následující. Jednotlivé jazyky projdou vlastním překladačem, který vygeneruje CIL. Ten je následně pomocí CLR přetransformován na spustitelný kód. Tato fáze je pak společná pro celý .NET. Schéma 2.2 níže naznačuje postup překladu.



Obrázek 2.2: Schéma překladu pomocí CLI a .NET.

Další součástí je FCL (Framework Class Library), tedy balík knihoven poskytovaných frameworkem, které usnadňují komunikaci se systémem, a které lze používat bez problémů

mezi jazyky rodiny .NET. FCL obsahuje také předlohy datových typů, jako jsou struktury, rozhraní nebo třeba i delegáty. [16]

Tyto knihovny zároveň podporují jazykovou nezávislost. To znamená, že můžeme používat i kód, který je napsaný v jiném jazyce. Aplikace vytvořené v tomto frameworku, vyžadují tento framework ke svému běhu, ale na druhou stranu, při nasazení na systému, který nabízí stejnou verzi v jaké byla tato aplikace napsána, nemá výsledná aplikace problémy s chybějícími knihovnami nebo špatnou kompatibilitou a navíc nabízí systém řízení paměti a ošetřování podmínek.

.NET Framework slouží k tvorbě spoustu druhů aplikací. Mezi nejčastější, podle typu grafického výstupu pro Windows, patří následující.

Mezi nejstarší, a na rychlé testování funkcionalit, patří konzolové aplikace. .NET pro komunikaci nabízí knihovnu *System.Console*, která poskytne čtení a zapisování do konzole. Ve Windows je nejčastěji využívána vestavěná konzole *cmd.exe*. [28]

Dalším velmi typickým výstupem a v dnešní době již zastaralým, jsou aplikace Windows Forms, které využívají knihovnu *System.Forms*. .Net Framework přišel s podporou pro Windows Forms program v roce 2002. [35]

Z důvodu zastaralosti Windows Forms, touze využít další možnosti systému a zkrášlit vzhled aplikací, přišel roku 2006 Microsoft s novou technologií nazvanou *Microsoft Windows Presentation Foundation*. Kromě nových grafických možností přišla s kompletně novým stylem práce s grafickým rozhraním, novým vykreslovacím jádrem *Direct3D*, jenž podporovalo možnost výpočtů na grafickém jádře nebo data bindingy, tedy schopnost vázat modelové hodnoty na grafické prvky.

Poslední rozšířenější desktop technologií a výstupem .NET Frameworku se stalo UWP, Universal Windows Platform, které nahradilo UAP, Universal Application Platform. UAP se velmi pomalu chytalo a nezažilo velký rozmach. Obě tyto technologie, ať už UAP nebo novější a vyspělejší UWP, nabízí distribuci přes Store vestavěný do Windows již od verze Windows 8. Podrobněji se o Windows (Microsoft) Store zmíním v podkapitole 2.4.

2.2 .NET Core

Dalším krokem ve vývoji Microsoft .NET Framework bylo předvedení verze Core, která se od původního .NET v mnoha věcech liší. Asi první a velmi zásadní změnou je multiplatformnost, protože .NET Core je díky tomu možné používat kdekoliv a využívat sílu základu .NET Framework například i na zařízeních s MacOS nebo Linux.

Další velkou změnou, která některým vývojářům způsobuje problém, je již v samotném názvu. Core je označení pro jádro a stejně tak .NET Core vypadá. Jedná se tedy o odlehčenou verzi .NET Framework.

Dále je potřeba zmínit distribuci samotného frameworku, která je modulární. Jednotlivé části nejsou přímo součástí, ale stahují se pomocí NuGet balíčků. Tato změna umožňuje jednodušší vývoj, snížení nákladů a ve výsledku menší a bezpečnější aplikace.

Poslední a neméně důležitou zvláštností je Open source řešení. .NET Core je projektem .NET Foundation a je distribuován pod licencemi MIT a Apache 2. Celý .NET Core balíček lze nalézt na GitHubu a kdokoliv do něj může přispívat. [32]

.NET Core má v našem projektu velmi důležitou roli, protože celý framework bude napsaný právě v této verzi .NET. To nám umožní v budoucnu jednodušší práci s jeho úpravou na další platformy a navíc takto navrhnuté řešení bude menší než jeho obdoba v plném .NET Framework.

2.3 UWP - Universal Windows Platform

Největším problémem zařízení s předcházejícími verzemi Windows byla, do vydání systému Windows 10, nutnost vytvářet pro každé specifické zařízení vlastní aplikaci. Pokud tedy vývojář vytvářel například hru pro Xbox a Windows, musel udělat sestavení pro specifický systém. Windows 8 nabídl prvotní náznak možnosti vytvářet jednu aplikaci pro více systémů. Tento model se jmenoval UAP (Universal App Platform) a představoval možnost vytvářet jednu aplikaci, jak pro Windows 8, tak pro Windows Phone 8. S updatem systému na verzi 8.1 přišla řada změn, která tento model vývoje ještě více sblížila, ale vývoj byl stále dělen na specifické grafické části, které bylo nutné vytvořit pro jednotlivé systémy. Funkční část, kdy vývojář odstínil funkčnost aplikace do jediného projektů, který byl sdílen, bylo možné použít v obou systémech, což umožnilo první velké ulehčení práce vývojáři.

S vydáním verze Windows 10 přišla velká změna, protože vývojář mohl napsat pouze jedinou aplikaci, kterou pak byl schopen spustit na kterémkoliv zařízení s Windows 10, ať už se jednalo o zařízení typu IoT (Internet of Things - jednoduše "chytrá elektronika") nebo chytré konferenční tabule Surface Hub. Zároveň je nutné podotknout, že systémy, které na těchto zařízeních běžely, nebyly vždy stejné a výrazně se lišily ve svém účelu použití. Z toho důvodu měl vývojář k dispozici speciální knihovny pro dané zařízení, které jej obohacovaly o potřebné funkce. Touto kombinací bylo možné docílit správného cílení aplikace s jediným kódem a kompletní funkcionalitou.

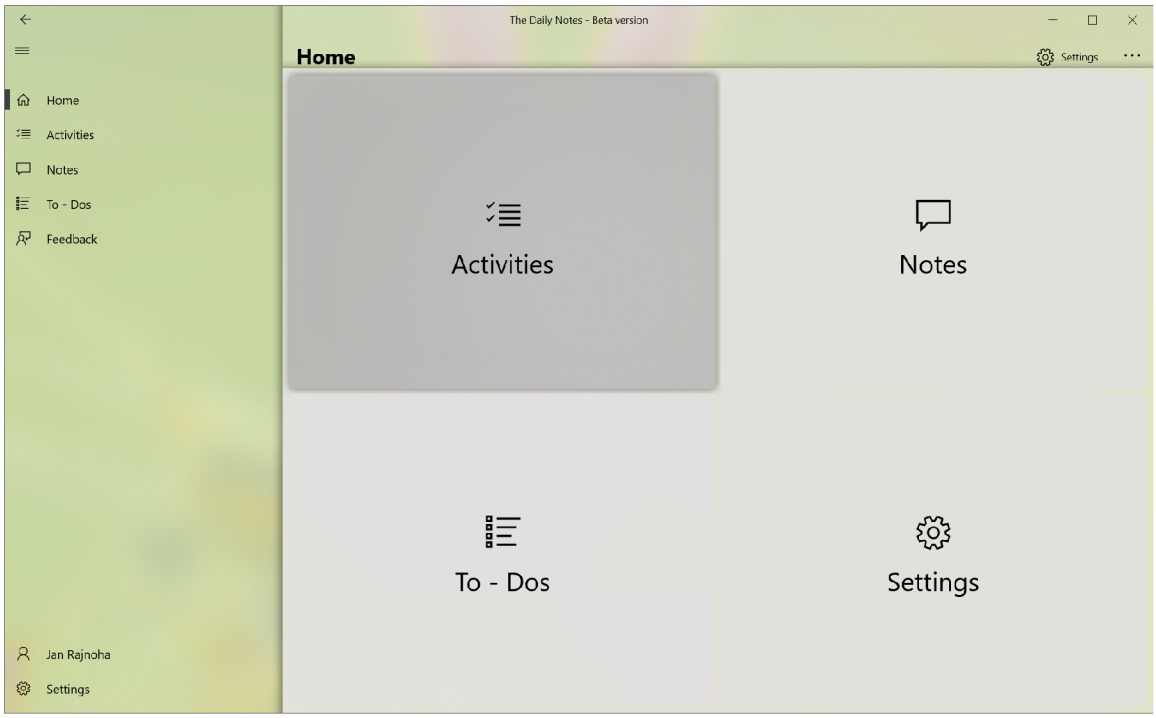
Universal Windows Platform (zkráceně UWP, překlad: Univerzální Windows platforma) je z pohledu vývoje v mnohém podobné již výše zmíněnému frameworku WPF (Windows Presentation Foundation). Oba využívají C# a designový jazyk XAML (Extensible Application Markup Language), který popisuji v sekci 2.6, a k vykreslování používají grafické jádro. Nicméně při bližším zkoumání si lze všimnout, že UWP není pro vývojáře již tak otevřené jako WPF. Největším omezením je množství omezení v dostupných funkcích. Vazby v XAML jsou omezené, konvertory dostupné pouze v jednoduché formě a množství grafických prvků je též značně strohé. Na druhou stranu UWP nabízí rychlejší běh aplikací pomocí .NET Native, nižší náročnost na hardware a optimalizované ovládání pro dotyková zařízení. [37]

2.4 Microsoft Store

Microsoft Store jako aplikace ve Windows, jak již název napovídá, slouží k distribuci software pro Windows 10 a dalších služeb či zařízení. Vývojář zde může nahrávat aplikace pro Windows 10 včetně doprovodných materiálů, jako jsou screenshoty, videa nebo přidávat balíčky k rozšíření své aplikace. Pro uživatele je pak možnost aplikaci hodnotit a posílat v případě problémů zpětnou vazbu.

Microsoft Store, dříve Windows Store, byl uveden jako nedílná součást systému Windows 8. V době Windows 8 sloužil jako místo pro distribuci updatu Windows na verzi 8.1, aplikací a her vytvořených pomocí UAP modelu pro Windows 8. Funkcionalita samotné aplikace se postupem času příliš neměnila, ale doplňoval se obsah, který bylo možné stáhnout, popřípadě koupit. S Windows 10 přišla do Microsoft Store možnost stahovat doplňky systému, jako jsou např. kodeky a kupovat zařízení, filmy a seriály. [12]

Základní předností Microsoft Store je z uživatelského hlediska možnost automatických aktualizací bez nutnosti zásahu vývojáře. Systém si sám hlídá, zda nejsou novější verze aplikace dostupné pro dané zařízení, takže uživatel je od těchto problémů osvobozen a není ani nutná jeho interakce.



Obrázek 2.3: Ukázka UWP aplikace

S Microsoft Store se pojí i distribuce a webové rozhraní pro správu aplikací. Vývojáři zde mají k dispozici celou řadu nástrojů:

- Možnost vytvářet selektivní updaty na základě zařízení a verze systému
- Sledování stavu aplikace
- Základní přehled o pádech aplikace
- Sledování počtu stažení aplikace, doplňků a základní demografické údaje o uživatelích
- A/B testování (cílené testování na základě určených kritérií) a vytváření skupin uživatelů
- Přidávání stahovatelného obsahu
- Správa hodnocení aplikace s možností odpovědi na jednotlivá hodnocení
- Sledování financí získaných z reklam a nákupů
- Tvorba výpisů financí a informací z aplikací

I když Microsoft Store slouží k distribuci univerzálních aplikací, lze distribuovat i aplikace, které tuto podmínku v prvotní fázi nesplňují. Microsoft vývojářům poskytl nástroj *Desktop Bridge*, jenž dokáže zaobalit starší typy aplikací do moderního balíčku, který pak už lze distribuovat přes Windows Store. Zároveň přitom vývojář dostane možnosti přistupovat k prvkům Windows 10, které jsou dostupné pouze moderním aplikacím [26].

Možnost publikovat aplikace v Microsoft Store je placena jednorázovým poplatkem a rozdělena podle typu účtu na soukromé a firemní. Podle toho se liší nejen cena, ale i funkce, které má pak vývojář k dispozici.

2.5 Jazyk C#

Jazyk C# patří do rodiny jazyků C a byl vytvořen společností Microsoft v roce 2000. Jazyk patří v rámci .NET mezi jazyky podporující CLI.

Syntaxe jazyku byla odvozena od jazyků C++ a Java, z nichž převzala objektově orientované použití a typovost. V jazyku je vidět na syntaxi velmi silný vliv jazyka Java, např. možnost dědit pouze jednu nadřazenou třídu, ale nespočet rozhraní, garbage collector nebo různé typy referencí (Phantom reference, soft reference, weak reference, ...). V mnohém jsou si tyto jazyky podobné a méně zkušenému vývojáři mohou na první pohled připadat víceméně stejné.

C# nabízí kromě základních datových typů, také typy nulovatelné (nullable), delegáty, výčty, struktury (podobné třídám, ale bez možnosti dědění), třídy a lambda výrazy. Datové typy se dělí na hodnotové a referenční. Zatímco hodnotové typy samy o sobě obsahují nějakou hodnotu (číselné typy, výčty, struktury), referenční typy pouze nesou odkaz do paměti, kam se program odkazuje a následně vytahuje další hodnoty přímo z paměti, které pak mohou být referenční nebo hodnotové. V případě, že se v programu ztratí odkazy na dané objekty v paměti, garbage collector opuštěná místa v paměti uvolní bez nutnosti zásahu vývojáře. [34]

Dalším stěžejním prvkem jazyka jsou atributy. C# rozlišuje dva typy, vlastnosti (property) a pole (field). Vlastnosti objektu jsou převážně využívána pro vnější komunikaci (zpřístupnění dat), mohou obsahovat vlastní logiku nastavenou pomocí funkcí get a set (v případě, že jsou vynechány, kompilátor je doplní za běhu) a používají především modifikátory public nebo internal. Pole naopak slouží pro ukládání aktuálního stavu objektu, nemají žádné pomocné funkce a pracují s modifikátory private a protected. Atributy jsou stěžejní prvek návrhového vzoru MVVM (Model-View-ViewModel), který je popsaný v sekci 2.7.

Ve své práci využívám další velmi silnou funkci jazyka C# zvanou generika. Jedná se o jakési šablony chování pro třídy pracující s určitým typem datového prvku. Tento datový prvek je sice nutné znát již v době psaní kódu, ale lze jej nahradit za jakéhokoliv jeho potomka, který se dosadí při inicializaci. Jde o další způsob psaní znovupoužitelného kódu, kdy můžeme zobecnit chování a přistupovat k objektům stejně, aniž bychom přímo znali jejich funkcionalitu. Abych blíže přiblížil chování generik, uvedu příklad s různými typy lidí. Pokud budeme u vybraných lidí chtít, aby nám např. zatančili, stačí této skupině lidí jen přikázat, aby začali tancovat. Jak budou tancovat, je už v jejich režii. My jim umíme dát jen příkaz. Stejně tak fungují generika. Nevíme sice, jak je datový typ implementován, ale víme, jak k němu přistupovat. Na první pohled se můžou generika zdát podobná rozhraním, jelikož ve své podstatě jsou, ale jen v oblasti práce s daným generickým prvkem. V jazyce se generika určují pomocí generického parametru, který ve většině případů bývá označován pouze T, a je uváděn za názvem třídy v ostrých závorkách (například `List<T>`). [33]

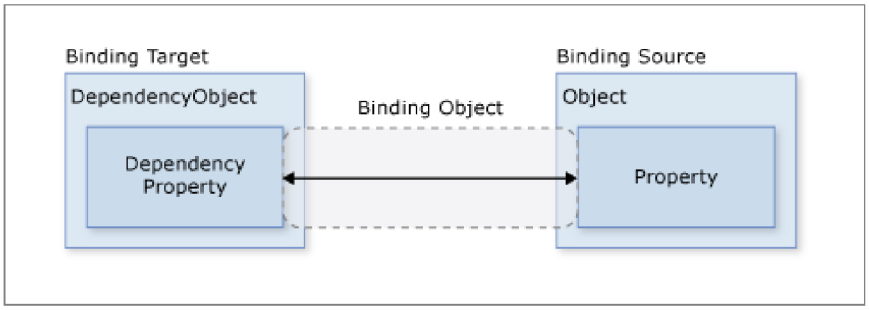
2.6 XAML - Extensible Application Markup Language

Extensible Application Markup Language, neboli zkráceně XAML, je deklarativní XML jazyk vyvinutý společností Microsoft a vydaný v červnu roku 2008. XAML se stal součástí frameworku .NET ve verzi 3.0 spolu s WPF. Zatímco WinForms využívaly pro tvorbu designu aplikace jazyk C#, WPF se přesunulo k jazyku XAML, který slouží k deklaraci uživatelského rozhraní včetně všech vlastností a vazeb. Stejně jako C# u WinForms, stejně tak XAML využívá stromovou strukturu elementů, kde element je jednotlivý grafický prvek. [29]

Každý dokument XAML je zaobalen do kořenového elementu, který označuje typ výsledného designu. Při použití elementu Page se vytváří celá stránka aplikace, na kterou se lze navigovat, UserControl zase vytváří vlastní uživatelské prvky, které lze dále používat, ResourceDictionary pak slouží pro vytváření konstant použitelných při designu, apod.

Jak již bylo výše zmíněno, XAML (v kombinaci s UWP) podporuje tzv. vazby (Binding), tedy provázání designu s daty. XAML nabízí několik typů vazeb podle toho, kam se vazba odkazuje. Vazby se mohou odkazovat jak na data mimo samotný XAML, například do ViewModelu nebo code behindu (například počet kliknutí na tlačítko, kdy v pozadí se po kliknutí zvýší iterátor), tak mezi sebou v rámci XAML (například reprezentace hodnoty posuvníku), pokud splňují kritéria pro uskutečnění vazby. Důvod, proč se vazby v XAML používají, jsou jejich schopnosti různého směru provázání, kterého jsou u UWP 3 typy [17]:

- **One-Time** - Hodnota se přenese z dat do designu při vykreslování a to pouze jednou. Pokud tedy nějaká interakce vyvolá změnu dat, hodnota se již nepropíše do designu.
- **One-Way** - Hodnota se přenese z dat do designu při vykreslování a to pokaždé, když je zjištěna změna hodnoty navázaných dat. Lze tedy updatovat hodnotu v zobrazení bez problémů, ale již nelze propisovat hodnotu z designu aplikace zpět do dat. Tento styl se používá především pro zobrazení dat v režimu pro čtení.
- **Two-Way** - Nejčastěji používaná vazba dat na zobrazení, protože je schopna přenášet hodnoty v obou směrech. Nejčastější použití tedy najde v případě vyplňování formulářů, kdy uživatel zadá údaje, potvrdí je a jednotlivé hodnoty se přenesou např. do ViewModelu, kde s nimi lze pracovat.



Obrázek 2.4: Ilustrace vazby mezi designem (Dependency property) a daty (Property)

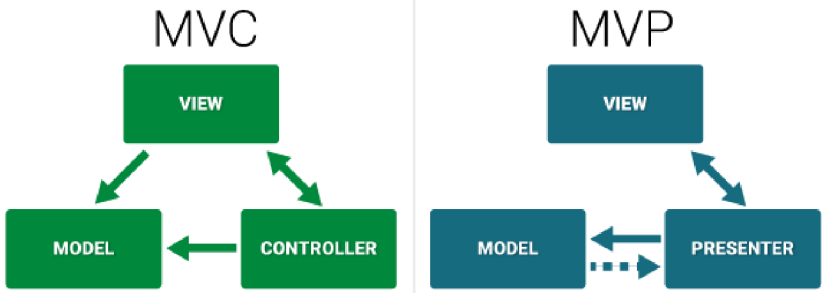
Aktualizace vazeb z dat do designu není vždy automatická a vyžaduje zásah vývojáře, ale zásadně ulehčuje práci s daty, protože ji není nutné navazovat na prvky události. Pokud se aktualizují hodnoty z designu, tak se aktualizace provádí vždy a zásahy nejsou nutné.

Vazby lze rozšiřovat o konvertory, tedy speciální třídy, jenž dokážou přetvořit data na požadovaný formát. Tyto konvertory si vývojář píše sám a slouží k jednoduššímu manipulování s celými objekty a nesourodými daty, protože v konvertoru lze vybrat příslušný prvek objektu a v případě nutnosti jej i konvertovat na požadovaný formát. XAML v dalších aktualizacích přišel i s tečkovou notací pro vazby, takže již není nutné pro vybírání jednotlivých hodnot z objektu používat konvertory, ale stačí pomocí tečkové notace vybrat danou hodnotu přímo ve vazbě.

2.7 MVVM - Model-View-ViewModel

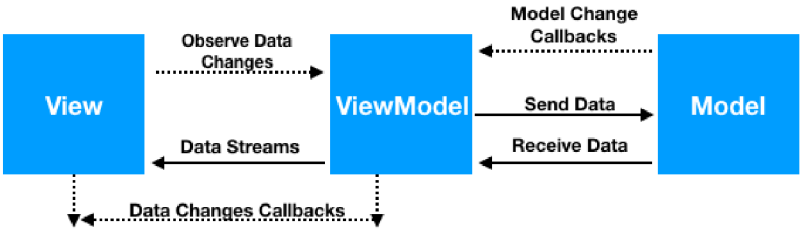
Model-View-ViewModel, neboli zkráceně MVVM, je návrhový vzor vytvořený pro práci s odděleným designem a možností využívat vazby. Byl představen spolu s WPF a XAML a slouží k oddělení designu, logiky a modelu dat od sebe, aby na sobě jednotlivé komponenty nezávisely a byly v případě nutnosti jednodušeji vyměnitelné. [14]

Návrhový vzor MVVM si bere inspiraci z návrhových vzorů MVP (Model-View-Presenter) a MVC (Model-View-Controller). Jak je vidět na obrázku 2.5, architektura obou vzorů je velmi podobná. Je to dané typem návrhových vzorů, kam se tyto vzory řadí. Jedná se o tzv. návrhové vzory implementující uživatelské rozhraní. Základem těchto vzorů je vždy dvojice Model-View, které označuji dva základní prvky implementace, tedy data a zobrazení. Konkrétní vzory pak jen doplňují třetí prvek komunikace a podle toho rozlišuje přístup k datům a zobrazení.



Obrázek 2.5: Ukázka interakcí u vzorů MVC a MVP [22]

Na obrázku 2.6 je vidět schéma interakce mezi komponentami vzoru MVVM. První komponenta, Model, reprezentuje data. V programování je Model reprezentován třídou, která kopíruje vzhled uložených dat, tedy tabulku v databázi nebo třídu pro serializaci. Model neobsahuje žádnou funkcionalitu a slouží pouze jako šablona pro vzor dat. Práci s daty obstarává komponenta ViewModel. Tato komponenta je velmi podobná komponentě Presenter ze vzoru MVP.



Obrázek 2.6: Ukázka interakcí u vzoru MVVM [11]

Hlavní nevýhodou MVP vzoru oproti MVVM byla nevyspělost MVP vzoru, protože nedokázala plně využívat schopnosti vazeb a byla velmi úzce vázaná na design aplikace.

ViewModel tuto závislost odstraňuje tím, že design nezná vůbec. Pokud by jsme vytvářeli obdobu Presenter komponenty v XAML, jednalo by se o code behind řešení.

Třetí komponentou je View, tedy zobrazení aplikace. Zde uživatel vytváří vstupy do aplikace a samotná data, která se pak ukládají pomocí ViewModelu do Modelu, přesněji do databáze, nebo se serializují pro další práci, např. poslání přes API nebo uložení do souboru. Při správné implementaci by neměl za použití tohoto vzoru být vytvářen žádný code behind kód, tedy logika přímo u View. ViewModel komunikuje přes View pomocí vazeb, které se vytvářejí nad položkami mapovaného modelu nebo nad abstrakcí `ICommand`, která slouží pro tvorbu příkazů, jenž se mohou vázat např. na tlačítka.

Jak již bylo výše zmíněno, některé vazby z ViewModelu do zobrazení se neprovádějí automaticky a je tedy nutné je vyvolat ručně. K tomu slouží rozhraní INotifyPropertyChanged, které dokáže při správném implementování notifikovat zobrazení aplikace, jenž má obnovit design. Děje se tak pomocí vazeb při překladu, kdy překladač si tvoří události vázané na jednotlivé datové položky ViewModelu oproti designu, a je schopen při vyvolání funkce PropertyChanged z rozhraní INotifyPropertyChanged správně aktualizovat hodnoty v zobrazení.

Kapitola 3

Analýza požadavků

Když se řekne framework, spousta lidí znalých oboru informačních technologií si představí nějakou knihovnu (nebo množství knihoven) s množstvím obecných funkcionalit zaměřených na danou kategorii. Stejně je to i v případě frameworku na tvorbu informačních systémů. Každý vývojář, který si jej stáhne, bude očekávat řadu nástrojů na jednoduchý návrh systému a jeho možnou přizpůsobitelnost, ať už jde o rozšíření funkcionalit nebo upravení chování navrhnutého řešení.

Proto se chci v této kapitole zaměřit na analýzu požadavků, které by měl framework splňovat, a zhodnotím silné a slabé stránky, které řešení přináší. Výstupem analýzy se stalo samotné řešení, vytvořený framework nad platformou UWP, jehož návrh je popisován v kapitole 5.

3.1 Obecné požadavky

První požadavek, který by měl splňovat každý dobrý návrh a který jsem si převzal z principů programování SOLID, je rozdělení knihovny do menších knihoven. Převzatým principem je "Single responsibility principle"(v překladu Princip jedné odpovědnosti), který popisuje třídu jako objekt jednoho určení. Třída by neměla např. číst z databáze a vytvářet uživatelské rozhraní. Stejně tak by framework měl být rozdělen do částí, kdy každá část se zodpovídá za jednu sadu funkcí, např. knihovna pro práci s databází, knihovna pro práci s uživatelským rozhraním, knihovna pro rozhraní tříd. Tímto přístupem také dáváme vývojáři možnost používat pouze ty funkce, které bude chtít, čímž jeho výsledná aplikace nebude zbytečně bobtnat o nepoužívanou funkcionalitu. [21]

Rozdělení má ještě jeden velmi důležitý důvod, než jen čistotu a snadnější práci s knihovnou. Tímto důvodem je možnost pracovat nezávisle na komponentách. V příkladu jsem zmínil knihovnu na rozhraní tříd. Spolupráce mezi knihovnami, komponentami, by měla fungovat jako posílání mailů. Nevíme, kdo je na druhé straně, ale víme, že pomocí zadané adresy druhá strana zprávu dostane a pokud pošle zprávu na odesílající adresu, dostaneme ji my. Rozhraní zde slouží jako adresář kontaktů, kam lze posílat zprávy. Pokud vývojář shledá, že dostupné řešení je nevyhovující, napíše si vlastní a za pomocí rozhraní jej zapojí do celého řešení.

Vzhledem k faktu, že informační systém má být modulární, tedy jednoduše přidávat a odebírat části programu bez nutnosti provádět velké zásahy do řešení, je nutnost zamyslet se nad vytvářením jednotlivých modulů, prací s nimi, propojování mezi sebou a jejich zapojení do systému a následném spravování na úrovni reprezentace ve frameworku. Modulem není

myšlena pouze část programu, se kterou může uživatel interagovat, ale i ty části, které slouží ke správné funkčnosti aplikace, a tedy není nutné je ukládat, nebo které slouží pouze jako pomocné datové položky. Mohou to být např. historie úprav, které se postupně ukládají, ale v závěru nemají vliv na to, jak se pracuje s aplikací.

Dále je potřeba myslet i na samotné vývojáře, kteří budou framework používat. Geniální knihovna, která je nepraktická k použití, je v konečném řešení zahozena. Z toho důvodu je nutné se zamyslet nad množstvím kódu, které je nutné pro zapojení funkcí do aplikace, tvorbou návrhu entit pomocí kódu, úprava jednotlivých atributů entit a jejich vliv na chod aplikace, ovlivňování aplikace pomocí globálních proměnných, ukládání nastavení a spoustě dalších podobných částí, které utvářejí konečnou formu práce s frameworkem v aplikaci.

Nutná není jen obecná práce, ale také zaměření, které framework má. V návrhu je nutné rozlišovat mezi tím, zda výsledný produkt bude nasazen v neznámých aplikacích nebo v aplikaci s omezeným rozsahem působnosti. Tvorba cíleného řešení má výhodu možného čistšího kódu a vyšší výkonosti, ale zase se velmi těžce transformuje na jiné úkoly. Stejně tak obráceně, některé projekty potřebuji specifické chování, které je nutné do aplikace přidávat a sladit s frameworkem, aby konečná aplikace fungovala jako celek.

Dalším aspektem, jenž je důležitý pro výsledný produkt, je rychlost řešení a celkový výkon, tedy optimalizace. Zde je potřeba správně navrhnout přístup do databáze, načítání uživatelského rozhraní, registraci a práci s moduly, komunikaci uvnitř frameworku a celkový návrh řešení od drobných funkcí po celkové chování (množství provedených operací mezi požadavkem a výsledkem). Do této kategorie spadá i volba technologie, jazyka a řada dalších technických prostředků, které budou pro vývoj použity. Jak již název práce a technologický popis napovídají, bude použita technologie UWP pro Windows 10, jazyk C# a framework .NET.

3.2 Analýza možností Windows 10 a technologie UWP

Jak jsem již zmínil, Windows 10 a prostředí UWP nabízí pro vývojáře velmi příjemné prostředí pro distribuci software a velmi pokročilou optimalizaci práce aplikace se systémovými zdroji, ale v případě vývoje může být značně omezující, protože aplikace vytvořené právě pomocí UWP technologie jsou zaměřeny hlavně na bezpečnost, což v důsledku znamená možné nedostatky a omezení. Výchozí stav zabezpečení UWP aplikací je velmi striktní, např. 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. Podobně je to s přístupem k webkamerám, GPS, mikrofonu a spoustě dalších hardware součástí zařízení. Všechna tato omezení ale lze zrušit povolením přístupu, nicméně je nutné s nimi počítat v aplikaci a správně reagovat na nepovolené součásti.

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). Dále je problém i technologie UWP, která se potýká s nižším zájmem vývojářů, než si společnost Microsoft představovala. I když je designově i implementačně v mnohém pokročilejší, vývojáři se stále drží vývoje na WPF nebo, v horším případě, a to hlavně z pohledu bezpečnosti, WinForms nebo se přiklánějí k multiplatformnímu vývoji PWA aplikací. Z toho důvodu je budoucí vývoj platformy UWP nejasný. Na druhou stranu mu do karet velice silně hraje nový operační systém Windows 10X, který byl určen pro zařízení se dvěmi obrazovkami a stojí převážně na UWP technologii. Další výhodou UWP na systému Windows 10X je jednodušší přizpůsobení aplikace právě pro dvě

obrazovky, takže s větším rozmachem zařízení tohoto typu nebude příliš složité již stávající aplikace přizpůsobit novému formátu.

3.3 SWOT Analýza

Když to shrneme a vytvoříme nad těmito body SWOT analýzu, tedy zhodnocení kladných a záporných stránek v interní a externí analýze, můžeme vidět, že problémy při návrhu budou způsobeny hlavně značnými omezeními použité technologie a její nejasný vývoj, který by mohl celý následný vývoj ukončit. Proto je nutné při vývoji dbát na správné řešení těchto problémů a jejich zaobalení tak, aby uživatel nijak nepoznal omezení funkcionalit. Na druhou stranu unikátnost řešení, možnost jednodušší adaptace na novou formu zařízení (přesněji systém Windows 10X) 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.

Strengths - silné stránky:
- Distribuce aktualizací
- Jednoduchost řešení
- Zabezpečení aplikace
- Možnosti přizpůsobení chování

Weaknesses - slabé stránky:
- Omezený přístup k částem zařízení způsobené bezpečností
- Omezené možnosti při vývoji způsobené technologií UWP
- Pouze Windows 10
- Distribuce přes Microsoft Store vyžaduje licenci

Opportunities - příležitosti:
- Jedinečný koncept v rámci Windows 10 a 10X

Threats - hrozby:
- Aktuálně nejasný vývoj UWP platformy

Kapitola 4

Dostupná řešení

V současné době existují jiné frameworky, které jsou schopny zastoupit funkčnost celého mého řešení nad UWP, ale nejsou vždy dokonalé nebo jsou často drahé, složité anebo na druhou stranu nejsou pro UWP stavěny. Určitě je ale na místě se na ně podrobněji podívat a zjistit jejich výhody a nevýhody.

4.1 Práce s databází - Entity Framework

Jedna z částí, které framework obstarává, je komunikace s datovým úložištěm. Zde za zmínku stojí Entity Framework (zkráceně EF), který v dnešní době hraje velmi velkou roli na poli komunikací s DB. Entity Framework byl vytvořen společností Microsoft v roce 2008 jako součást .NET Framework 3.5. EF patří mezi tzv. ORM, tedy Object-relational mapping, v překladu jednoduše mapper objektů na relační data. V programování to vypadá tak, že v kódu je šablona, předpis pro jednu tabulku v databázi, které se říká entita, a ta věrně zachycuje tabulku 1:1, co se týká atributů použitých v DB. V databázové terminologii je entitou označován jeden záznam, řádek v DB, v programování je entita reprezentována jako instance dané třídy.

Entity framework kromě mapování databázových entit do programu nabízí možnosti tvorby dotazů nad databází. Stále je možnost psát SQL dotazy přímo a posílat je EF ke zpracování, ale EF nabízí podporu dotazovacího jazyku LINQ (Language integrated query), který nabízí snadný způsob psaní dotazů pomocí lambda výrazů. Z toho plyne i další vlastnost, kterou EF disponuje, a to je schopnost jednoduché tvorby tabulek databází nebo připojení k již vytvořeným databázím. [24]

Entity framework je distribuován ve dvou verzích. Starší verze, obsáhlejší a větší, se aktuálně nachází ve verzi 6 a byla vydána v roce 2017. Tento framework je ve všech ohledech ideální k využití v aplikacích pro přístup k databázím, nicméně jeho největším problémem je možnost implementace na technologiích, přesněji v mém případě technologii UWP, kterou nepodporuje. Největší výhodou EF6 je jeho vyspělost, odladěnost a dostupnost obrovského množství návodů a dokumentace použití frameworku a i v dnešní době má dlouhou řadu zastánců.

Druhou a novější verzí je Entity Framework Core, jenž je aktuálně ve verzi 3.1.3 a vyvíjí se její 5. verze (číslování je z důvodu reflekce .NET, který se mění z .NET Framework a .NET Core na .NET). EF Core je jednodušší verze plnohodnotného EF, který má řadu omezení, ať už se jedná o omezení funkcionality nebo dokonce její kompletní odstranění. Největší problém s EF Core je mapování tabulek, jenž obsahují vazby 1:N nebo M:N. Problém

je v reprezentaci samotné vazby v DB, protože využívají mezitabulky na provázání mezi záznamy. I když se může zdát, že 1:N mezitabulku nepotřebuje, protože jeden záznam se jednoduše odkáže na další, ale v případě, kdy chceme vytáhnout celý seznam záznamů provázané na jeden záznam, by to bylo trochu složitější a náročnější na výkon. V případě vazeb M:N je jasné, že tento problém nijak nelze obejít. Na druhou stranu, hlavní výhodou Core verze je možnost jeho využití, která, stejně jako u .NET Core, již není oproti EF6 vázaná na Windows. Lze tedy stavět pomocí něj multiplatformní aplikace za použití jednoho kódu. [23]

4.2 Grafická část

Další část, která hraje velmi důležitou roli, je design aplikace. Ať už se jedná o celkový vzhled aplikace řízený operačním systémem (titlebar, trojice tlačítek minimalizovat, maximalizovat a vypnout a další), jednotlivých komponent až po dialogová a modální okna, notifikace a způsob interakcí se zobrazením. Zatímco úpravy samotných oken jsou velmi vzácné, možnosti mít upravené komponenty a dialogová a modální okna je mnohem častější. Zde je nutné zmínit řadu významných vývojářů frameworků grafických komponent a oken.

4.2.1 Telerik UI

První zmíněný framework pro tvorbu UI je Telerik UI od společnosti Progress Software Corporation. V balíčku najdeme vylepšené komponenty vycházející ze základních prvků UWP, které lze navíc použít na všech zařízeních s Windows 10. I když prvků není mnoho, největší výhodou celého balíčku je jeho cena. Společnost Progress nabízí Telerik ve dvou cenových nabídkách, placenou prémiovou verzi s podporou a zdarma verzi dostupnou pomocí NuGet balíčku a open source kódem na GitHub. [3]

Telerik je nejlepší start pro tvorbu složitějšího designu bez nutnosti vytvářet komponenty. Nepřináší množství převratných možností, ale lze s ním oslnit a zjednodušit si vývoj.

4.2.2 Syncfusion Essential Studio

Dalším velkým hráčem na poli UI komponent pro UWP je společnost Syncfusion Inc., která nabízí balíček Syncfusion Essential Studio. Framework nabízí opravdu bohatou škálu komponent. Od jednoduchých komponent typu combo box nebo progress bar, které lze najít i mezi základními prvky UWP, přes mapové prvky, vylepšené data gridy a kalendářové prvky, až po komplexní řešení typu ribbon panelu nabídek nebo kanban. Balíček komponent pro UWP je nabízen v několika cenových nabídkách, kdy nejlevnější licence vázaná na jednoho vývojáře a rok se pohybuje v řádu několika desítek tisíc, jedná se tedy o velmi drahý nástroj. [2]

Syncfusion je profesionální balíček pro náročnou klientelu hledající množství pokročilých prvků s jednotným designem, ale pro jednotlivce na vlastní nekomerční projekty není příliš zajímavý.

4.2.3 DevExpress

Posledním zmíněným distributorem UI komponent je Developer Express Inc., kteří nabízejí svůj balíček nejen komponent. Tento vydavatel nabízí vlastní řešení s více než 25 prvky pro UWP od základních až po komplexní, jako jsou ribbon panel nabídek nebo hamburger

menu aplikace. Framework je určen pro komerční použití a jeho cena se pohybuje kolem deseti tisíc. Opět jde o roční předplatné pro jednoho vývojáře. [5]

DevExpress je poloprofesionální nástroj vhodný jak pro začátečníky, tak pro větší projekty. Developer Express nicméně nabízí i další produkty. Jedním z nich je i celé řešení pro tvorbu informačních systémů, které popisuji v podsekci 4.3.1.

4.3 Frameworky pro tvorbu informačních systémů

V této části se zaměřím na dostupné frameworky, které řeší celý problém tvorby informačních systémů, od návrhu databází až po tvorbu uživatelského rozhraní. V aktuální době nejsou pro platformu UWP dostupné žádné frameworky, které by řešili tento problém komplexněji, ale slouží jako porovnání, jak si již vytvořená řešení s daným problémem poradily. Vývojář si samozřejmě může postavit řešení z výše zmíněných částí, ale pořád musí zajistit jejich propojení a správnou implementaci.

4.3.1 eXpressApp Framework

První zmíněný framework patří již výše zmíněné společnosti Developer Express Inc. Framework je dostupný v rámci balíku, který obsahuje více než 600 prvků pro různé technologie, mezi které patří i UWP. eXpressApp Framework (zkráceně XAF) stojí na ORM eXpress Persistent Objects (zkráceně XPO), který poskytuje propojení s databází a stejně jako EF nabízí code first přístup a automatické generování databází. [4]

Výhodou celého řešení je automatická podpora generovaných formulářů, tiskových sestav a analytických funkcí. Samozřejmostí je velmi podrobná dokumentace.

Nevýhodou je zde práce s formuláři, které mají velmi omezenou možnost přizpůsobení. Dále je zde problém se zobrazením typu Master-Detail, tedy zobrazení seznamu položek a jejich detailního okna pro vybranou položku. Stejně tak cena celého řešení je značně vysoká z důvodu nutnosti zakoupit celý balík nabízený společností. Další značnou nevýhodou takto univerzálně stavěného frameworku je jeho technická náročnost a nižší výkon, který je znatelný na low end strojích a může znepříjemňovat práci s takto postaveným informačním systémem.

XAF je velmi propracovaný nástroj, ale je určen převážně pro složité aplikace, které jej dokáži v plné míře využít.

4.3.2 MM .NET Framework

Mírně starší framework MM .NET Framework nabízený společností Oak Leaf Enterprise nabízí řešení tvorby informačních systémů pro technologie WinForms a WPF. Přístup k datům řeší pomocí ADO.NET nebo vlastní technologie MM .NET Entity Objects. [1]

Výhodou tohoto řešení je jeho nižší cena a podrobná dokumentace.

Nevýhodou oproti XAF frameworku je absence automaticky generovaných formulářů.

4.3.3 Tvorba vlastního řešení od A do Z

Trochu netradiční možnost, jak vytvořit informační systém, je tvorba kompletně vlastního řešení. Vývojář může ve větší či menší míře využívat výše zmíněných frameworků pro tvorbu uživatelského rozhraní nebo přístupu k datům a kompletně si sestavit vlastní specifické řešení.

Největší výhodou je zde možnost maximální optimalizace, protože není nutné implementovat dodatečný kód, který se nevyužije.

Nevýhodou tohoto postupu je nutnost vlastní implementace již ověřených algoritmů, jako je např. tvorba automaticky generovaných formulářů. Problémem se může stát i následná úprava kódu, která nemusí počítat s navazujícími požadavky a následný kód se začne stávat množstvím úprav nepřehledný. Dále je pak potřeba na projekt asociovat větší množství vývojářů než v případě, kdy se využije specializovaný framework při stejném čase dodání produktu, tudíž i finanční náročnost je vyšší.

Kapitola 5

Návrh řešení frameworku ISUF

Než rozeberu návrh řešení, je nutné si ujasnit základní pojmy, které se v této kapitole budou vyskytovat.

- **ISUF** - Název frameworku, zkratka "Information system universal framework"
- **Modul** - Jednotka aplikace. Obsahuje entitu a způsob, jak s ním aplikace pracuje a na úrovni modulu také přístup do databáze a zobrazení v aplikaci.
- **Entita, datová třída** - Jednotka modulu. Odpovídá částí Model z návrhového vzoru MVVM.
- **Komponenta, formulář** - Část designu. Uživatelský prvek složený z jednotlivých design prvků a z nichž se potom skládají jednotlivé view, zobrazení. Obsahuje převážně vlastní vlastní view model. Odpovídá části View z návrhového vzoru MVVM.
- **Design prvek** - Elementární část prvku, jako jsou například tlačítka a textová pole.
- **View, zobrazení** - Okno modulu, které se skládá ze seznamu záznamů, komponent pro práci se záznamem a chování spojené s modulem. View také obsahuje vlastní view model. Odpovídá části View z návrhového vzoru MVVM.

Jak je již v jednoduchém slovníku pojmů uvedeno, návrh frameworku je postaven na návrhovém vzoru MVVM, který využívá ve velké míře. V kombinaci s návrhovým vzorem Messenger tvoří silnou dvojici, jenž dovoluje posílat množství informací mezi moduly a jejich vzájemnou komunikaci napříč celou aplikací. Podrobnější použití Messengeru popíši v sekci 5.7.

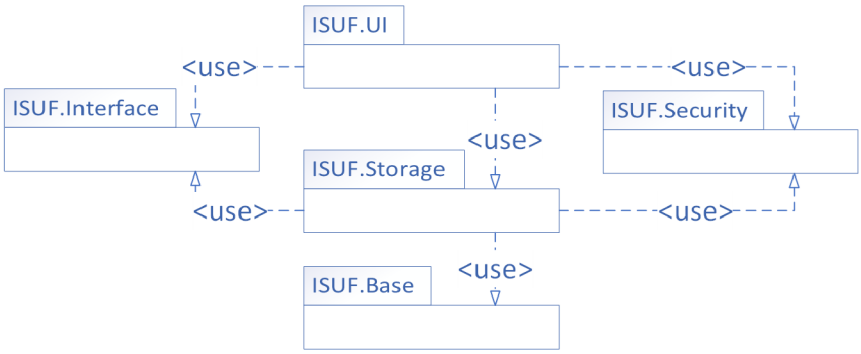
Ještě jednou zde vyjmenuji požadavky na framework, které jsem rozebral v kapitole 3. Z analýzy jsem v konceptu neimplementoval všechny požadavky, ale v kapitole 7 nastíním jejich budoucí vývoj. Pro implementaci jsem vybral následující požadavky:

- Rozčlenění frameworku do více částí
- Jednoduchá tvorba modulů s následnou tvorbou databáze
- Možnost propojovat moduly mezi sebou a ostatní úpravy atributů
- Přednastavené moduly
- Automatické generování UI formulářů

- Využití možností Windows 10 a technologie UWP s možností přenositelnosti na jiné platformy
- Obecný přístup, který si vývojář upraví pro vlastní potřeby
- Optimalizace

5.1 Rozdělení frameworku do více částí

Při návrhu bylo dbáno na možnost rozdělit projekt do více částí. Nejen, že výsledná aplikace, která bude používat framework, nebude muset načítat obrovské množství nepoužitého kódu, ale hlavně vývojář dostane možnost jednoduše nahrazovat celé části frameworku za své vlastní. Dále byla při návrhu zvolena výstavba postupné návaznosti částí, tedy pro využití nejvyšší vrstvy je nutné použít všechny vrstvy nižší, viz obrázek 5.1. Základem ISUF je část `ISUF.Base`, která nese základní operace pro práci s moduly, jejich registraci, implementaci vzoru Messenger včetně zpráv, atributy pro práci s entitami anebo například výčtové typy.



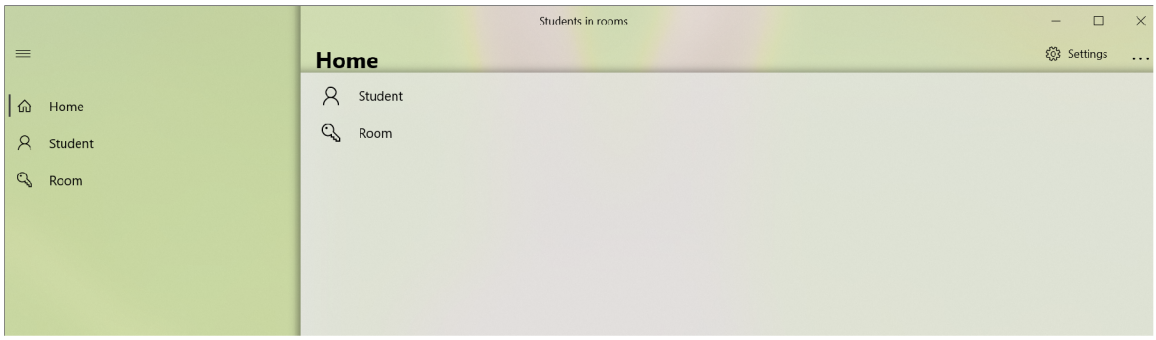
Obrázek 5.1: Vazby mezi částmi ISUF frameworku

Na základní vrstvě stojí `ISUF.Storage`, který navíc implementuje část `ISUF.Interface`. Tato část se stará o povýšení modulů do úrovně práce s databází, obsahuje třídy pro práci se záznamem a nastavuje přednastavené moduly.

Důležitou částí, která se ve frameworku vyskytuje, je část `ISUF.Interface`, tedy taková, která nese pouze rozhraní pro jednotlivé třídy. Vnitřně je rozdělena na část pro datové přístupy (použito v `ISUF.Storage`) a část pro uživatelské rozhraní a view modely (použito v `ISUF.IU`). Díky tomuto přístupu je možné vývojáři dovolit vyměňovat jednotlivé části frameworku.

Poslední úrovní, kterou lze v ISUF použít, je `ISUF.UI`, který obsahuje implementaci uživatelského rozhraní. Tato část obstarává tvorbu aplikace jako celku, od hlavního menu až po jednotlivé komponenty, dále vylepšení modulů pro zobrazení v UI, view modely, zobrazení, komponenty, jednotlivé design prvky a ostatní třídy spojené s tvorbou designu. Základem uživatelského rozhraní je hamburger menu, které obsahuje seznam všech modulů s možností zobrazení v aplikaci. Stejně jako většina prvků v aplikaci, je menu generováno automaticky na základě registrovaných modulů. Podobný seznam modulů je poté dostupný i na hlavním menu. Ukázka jednoduché aplikace o 2 modulech je vidět na obrázku 5.2.

Design jednotlivých modulů je tvořen pomocí Master-Detail vzoru, tedy obsahuje zobrazení seznamu záznamů a boční panel pro práci s jednotlivými záznamy.



Obrázek 5.2: Hlavní menu vytvořené pomocí ISUF

Další důvod, proč je lepší rozdělit projekt do jednotlivých částí, je možnost aktualizací. Pokud by byl použit velký framework pro aplikaci, je nutné při každé aktualizaci jeho celé stažení, tedy v odlehlejších místech s pomalejším internetovým připojením je tento přístup velkou zátěží. Pokud by se výsledná aplikace používala velkou část dne nebo dokonce celý den, tak jakékoliv omezení práce je kritické. Obzvláště pak, pokud aktualizace nese důležitou opravu. Schopnost aktualizace jednotlivých částí ale stojí na předpokladu, že návrh frameworku a jeho odnoží, počítající s tímto přístupem, zachovává pro minoritní aktualizace definice rozhraní pro vnější přístup.

5.2 Jednoduchá tvorba modulů s následnou tvorbou databáze

Jak již jsem zmínil ve výčtu na začátku kapitoly, každý modul je tvořen datovou třídou, entitou, tedy jakýmsi předpisem, jak vypadá tabulka v databázi a jak s ní lze pracovat v programu. Pro využívání entit bylo v ISUF připraveno několik bázových předpisů, které je nutné použít pro implementaci. Tento postup je zvolen také z důvodu jednodušší tvorby znovupoužitelného kódu. V případě potřeby si ale vývojář může jednotlivé bázové třídy entit upravit pro vlastní potřeby. Co se týká povolených datových typů, jsou podporovány všechny základní datové typy a datové typy pro záznamy, takže vývojář má volnou ruku. V případě ukládání složitějších datových typů, tedy třída ve třídě, se používají vazby 1:1.

Tvorba modulů se pak dělá pomocí specifikování názvu modulu a typu entity, kterou modul využívá. Modul, stejně jako entita, využívá základních rozhraní, které implementují na základě použitého typu modulu potřebné funkce, například datový modul obsahuje funkce pro práci s databází a modul pro zobrazení v aplikaci zase funkce pro přidání do seznamu zobrazených modulů v menu. S entitami se v modulu pracuje pomocí reflexe a automatického vytváření instancí za běhu, tudíž framework obstarává potřeby sám a vytváří jednotlivé instance bez zásahu vývojáře. Takto specifikovaný modul se registruje pomocí správce modulů, který jednotlivé entity modulů zkontroluje a podle nich upraví tabulky v databází. Vývojář je tak osvobozen od ruční aktualizace.

5.2.1 Správce modulů

S registrací modulů souvisí i správce modulů, který specifikuje, jakým způsobem se pracuje s moduly. Základní správce obstarává práci s moduly, kontrolu jejich existence a následnou analýzu modulů. Datový správce modulů přidává funkce pro určení typu databáze,

připojovací řetězec pro připojení k databázi (connection string) a automatickou aktualizaci databáze a správce modulů pro aplikaci pak přidává získání výčtu jednotlivých modulů, které se budou zobrazovat v aplikaci. Framework poté s těmito údaji nastaví potřebné operace na pozadí a připraví aplikaci pro práci.

5.2.2 Analýza modulů

V rámci správce modulů je i analyzátor modulu. Tato operace je nutná pro práci s modulem v aplikaci, kdy při získání dat je možné s modulem pracovat i v případě, kdy framework při sestavování nezná přesný popis daného modulu a jeho entity.

Analýza modulu se dělá přes třídu `ModuleAnalyser`. V této třídě se uchovávají všechny rozebrané moduly pro pozdější práci, aby byl možný rychlejší chod aplikace.

5.3 Možnost propojovat moduly mezi sebou a ostatní úpravy atributů

Framework umožňuje práci s ostatními moduly pomocí vazeb jednotlivých entit. Tyto vazby se tvoří pomocí číselných datových typů a datových typů pro seznamy číselného typu, kdy jednotlivé datové typy nesou ID vázaných záznamů z jiné tabulky. V aktuální době ISUF podporuje vazby typu 1:1 a 1:N, vazby typu M:N nebyly implementovány, ale popisuji práci s nimi v kapitole pro možný budoucí vývoj (kapitola 7). Jednotlivé vazby (relace) se tvoří pomocí atributů datových typů.

5.3.1 Atributy entit

Atributy, jak již bylo výše zmíněno, slouží také k tvorbě relací mezi entitami. Další důvod, proč byly implementovány atributy pro entity, byla možnost jejich úprav chování v rámci aplikace, jako jsou např. ignorování položky v databázi, nastavení vzhledu atributu v designu aplikace nebo např. možnosti kontroly při ukládání atributu do databáze.

5.3.2 Analýza entity a jejich atributů

V rámci analýzy modulu (viz. 5.2.2) se provádí i analýza entity a jednotlivých atributů pro datové položky. Stejně jako modul jsou tyto analýzy uloženy a následně i kontrolovány pro korektnost oproti databázi, například správné použití datových typů v kombinaci s atributy nebo atributy pro tvorbu relací. Analýzy jednotlivých datových položek jsou uloženy ve třídě `PropertyAnalyze`. V rámci uložených dat analýzy je uložen název položky, její typ a přidružené atributy.

5.4 Automatické generování UI formulářů

Práce s automaticky generovanými formuláři má 2 režimy. Zobrazení položek entity ve formuláři a práce s nimi a extrahování hodnot formuláře a jejich uložení do entity. Framework všechny takto připravené formuláře ukládá, aby při jejich dalším použití byl běh celého systému rychlejší.

5.4.1 Zobrazení položek entity ve formuláři a práce s nimi

Další specialitou frameworku ISUF jsou automaticky generované formuláře. Tato funkce vychází z již zmiňovaných analyzátorů. Při tvorbě komponent pro práci se záznamem se z již analyzovaných dat vytáhnou příslušné analýzy a algoritmus pak provede jejich vytvoření a vložení do komponenty.

Všechny design objekty, které se v aplikaci používají, mají předpis, který se poté pomocí rozhraní implementuje. Jedna z funkcí, která se musí implementovat, je funkce `AddControls`, která zajišťuje tvorbu jednotlivých prvků. Kromě obecného designu, který se generuje, se generují i prvky spojené s modulem. Ve funkci se nad analyzovaným modulem zavolá pro každou položku vytvoření pomocí třídy `ControlCreator` a její příslušné funkce. Tato třída má 2 typy funkcí pro vytvoření instance prvku. Jednou je `CreateDetailControl` a druhou `CreateEditableControl`. Rozdíl v těchto funkcích je hlavně v typu vytvořeného prvku, protože zatímco `CreateDetailControl` vrací prvek, který je pouze pro čtení, tedy nelze jej editovat, tak `CreateEditableControl` dovoluje uživateli hodnoty do prvku vkládat. Uvnitř těchto funkcí se poté dělá pro specifický datový typ instance specifického design prvku, například pro text textový blok a pro čas a datum prvky spojené s časem, a přidává se k nim speciální přídavek na konci jejich názvu, aby je bylo možné znovu hledat a pracovat s nimi. Takto vytvořené instance se vrací zpět a jsou vloženy do zobrazení nebo komponenty na příslušné místo. Pro umístění v rámci objektu se používá `RelativePanel`, tedy prvek, který adresuje pozici design prvku v závislosti na jiném design prvku. Pozice datové položky v rámci formuláře je pak upravována pomocí UI atributu v návrhu entity.

Takto vytvořený formulář je nutné nastavit pro práci s daty a správně data, v případě již předvyplněného záznamu nebo detailu záznamu, konvertovat na požadovaný výstup. K tomu slouží třída `FormDataMiner`, která obstarává zobrazování a konvertování dat mezi entitou a formulářem. Práce s daty probíhá ve dvou krocích. Prvním krokem je získání seznamu design prvků pro práci s daty z formuláře, k čemuž slouží funkce `GetControlsFromForm`, a druhým krokem je funkce `FillValuesIntoForm`, která tyto hodnoty vloží na příslušná místa ve správném tvaru.

První funkce, `GetControlsFromForm` vytáhne za pomocí funkce `GetControlsByName` z knihovny `Functions` všechny prvky, které mají speciální dodatek na konci svého názvu a vloží je do seznamu,který je pak vrácen zpět. Druhá funkce, `FillValuesIntoForm` tento seznam vezme a spolu s instancí entity začne procházet jednotlivé prvky. Algoritmus poté pro každý prvek shodný s prvkem v entitě vezme hodnotu prvku v entitě a vloží tuto hodnotu ve správném formátování do příslušného design prvku pomocí `SetValueIntoControl`. Stejný algoritmus se používá i pro zobrazení detailu záznamu, takže před vložením hodnoty do prvku je nutné jej formátovat na správný tvar textového řetězce, protože detail položky je tvořen pomocí needitovatelných textových polí.

5.4.2 Extrahování hodnot formuláře a jejich uložení do entity

Stejně, jako jsou pomocí funkcí `FillValuesIntoForm` a `SetValueIntoControl` nastaveny hodnoty do design prvků, tak pro extrahování hodnot z design prvků slouží funkce `FillValuesIntoProperty` a `GetValueFromControl`.

Ve chvíli, kdy uživatel potvrdí data formuláře, se po získání prvků formuláře vyvolává první zmiňovaná funkce `FillValuesIntoProperty`. Jedná se o generickou funkci s parametrem `T`, kde se jako bázový prvek používá základní datová třída entity. Pomocí reflexe se začnou hledat jednotlivé design prvky s vlastnostmi entity a algoritmus je začne plnit daty. K tomu slouží druhá zmiňovaná funkce, `GetValueFromControl`. Funkce vytáhne z jednotli-

vých design prvků hodnoty a zkontroluje jejich typ, jestli odpovídá typu, do kterého budou ukládány. Pokud typy neodpovídají, provede se konverze na požadovaný typ.

5.5 Využití možností Windows 10 a technologie UWP s možností přenositelnosti na jiné platformy

Jak již název práce napovídá, jedním z hlavních požadavků frameworku je pokročilé propojení s Windows 10. Aby bylo toto propojení možné, musí být splněna podmínka vybrané technologie, kterou je UWP. Kromě možností aktualizací přes Microsoft Store je využití technologie UWP výhodné z pohledu moderního designu a multiplatformnosti na zařízeních s Windows 10.

I když je ISUF navržen a vyvíjen pro UWP platformu, díky využití frameworku .NET Core je jeho možnost vytvoření alternativy pro ostatní systémy velmi jednoduchá.

5.5.1 Práce s frameworkem ISUF

Po shrnutí výše uvedených bodů návrhu byl vytvořen systém jednoduché správy modulů a jednotlivých zobrazení.

Vývojář nejprve nastaví parametry aplikace a použité typy jednotlivých základních prvků aplikace, jako je hlavní stránka a menu. Dále se vytvoří entity, které chce použít v aplikaci a registruje je do správce modulů. Správce provede analýzu a připraví seznam modulů pro zobrazení v aplikaci.

Jak již bylo zmíněno v části 2.7, framework pracuje na základě návrhového vzoru MVVM, který byl popsán v sekci 2.7. Z toho důvodu je nutné, aby ISUF disponoval jednotlivými view modely a view pro každou komponentu a zobrazení. I když má framework všechno toto připravené, je nutné vytvořit základní šablony. V oblasti zobrazení a komponent stačí vytvořit prázdný soubor a použít dědičnost. Pro view modely je opět nutné použít dědičnost, ale oproti zobrazením vyžadují upřesnit některé funkce, které nebylo možné implementovat předem. Modely jsou zastoupeny entitami.

Všechny ostatní záležitosti spjaté s tvorbou designů, provazování v rámci funkčnosti frameworku, nastavení zpráv a dalších funkcí si framework provádí sám a není tedy nutný zásah programátora ve výchozím stavu.

5.6 Obecný přístup, který si vývojář upraví pro vlastní potřeby

Framework ISUF byl navržen jako obecný systém, a z toho důvodu umožňuje vývojáři různě ohýbat jeho chování. Při návrhu bylo dbáno na co největší upravitelnost, aby byl ISUF ještě více univerzální.

Až na základní část frameworku, `ISUF.Base`, je možné upravovat stěžejní funkcionality. Mezi tyto funkce patří v datové části přístupy do databáze, práce se záznamem a chování modulů . V části designu vývojář dostane k dispozici možnost upravit view modely, design komponent a zobrazení a jednotlivé další dílčí části funkcí.

5.7 Návrhový vzor Messenger

Framework ISUF stojí na dvou návrhových vzorech. Jedním je již několikrát zmiňovaný vzor MVVM a druhý návrhový vzor Messenger.

Messenger je vzor postavený na zprávách. Třída registruje typy zpráv, které chce přijímat, a jednotlivé funkce, které chtějí informovat ostatní, zasílají tyto zprávy. Zprávy mohou být informativní, bez jakéhokoliv obsahu, nebo mohou nést upřesňující informace, například pro práci se záznamem ID daného záznamu a typ modulu, který zprávu odeslal.

Díky tomuto přístupu je možné propojit nejen komponenty a zobrazení, ale také celé moduly mezi sebou. Jedním z takových propojení pomocí zpráv je i vytváření relací, kdy přes analýzu entit je možné získat jednotlivé vztahy.

Kapitola 6

Vývoj ukázkové aplikace a problémy při vývoji

V tomto textu se podívám na vývoj testovací aplikace, která vznikala zároveň s vývojem celého frameworku, a řadu problémů, které nastaly při vývoji a jejich následné řešení.

6.1 Vývoj testovací aplikace

Při vývoji frameworku vznikala pro potřeby testování základní aplikace se dvěma moduly. Zde je v textu popsána aplikace pro práci se studenty a jejich evidenci v místnostech, v příloze pak aplikace na evidenci aut a parkovišť, na kterých daná auta parkují.

6.1.1 Tvorba základu aplikace

Aplikace nemá žádné pokročilé funkce, vychází pouze ze základu frameworku, a podle toho vypadá i samotná aplikace. Základní implementace je použita u komponent menu a zobrazení hlavní stránky, které implementují třídy `ShellBase` a `MainPageBase`. Implementace je zde prováděna ve dvou souborech, xaml (popis designu pomocí upraveného XML) a cs (C# kód). Implementace v xaml se provádí importováním jmenného prostoru a nahrazením kořenové elementu. V ukázce xaml pro hlavní stránku.

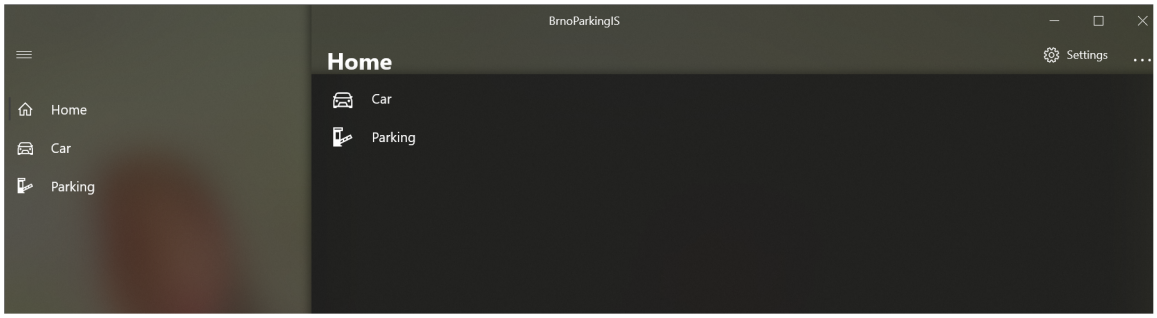
```
1  <isuf:MainPageBase
2      x:Class="StoRIS.Views.MainPage"
3      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
4      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
5      xmlns:local="using:StoRIS"
6      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
7      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
8      mc:Ignorable="d"
9      xmlns:isuf="using:ISUF.UI.Views">
10
11     <Grid>
12     </Grid>
13
14 </isuf:MainPageBase>
```

Importování jmenného prostoru se v XAML dělá pomocí specifikování atributu `xmlns`, kdy se za dvojtečkou doplní název, který budeme v aplikaci používat, a při dosazení hodnoty vkládáme cestu k danému prostoru. Implementace je poté použita pomocí schématu {vlastní jméno jmenného prostoru}:{třída ze jmenného prostoru}.

Další ukázkou je implementace C# kódu pro hlavní stránku.

```
1  using StoRIS.ViewModels;
2  using ISUF.UI.Views;
3
4  namespace StoRIS.Views
5  {
6          public sealed partial class MainPage : MainPageBase
7          {
8                  public MainPage() : base(typeof(MainPageViewModel))
9                  {
10                         InitializeComponent();
11                 }
12         }
13 }
```

Jak je z ukázky vidět, ISUF framework se postará o vše, co je spojené s vytvořením designu aplikace. Výsledek této implementace, včetně zobrazení modulů, lze vidět na obrázku 6.1, který ukazuje přiloženou aplikaci BrnoParkingIS. V ukázce je navíc vidět aplikace v tmavém režimu. ISUF podporuje oba typy motivů, které nabízí Windows 10. Světlou verzi, v provedení zde prezentované aplikace StoRIS, lze vidět na obrázku 5.2.



Obrázek 6.1: Hlavní stránka testovací aplikace BrnoParkingIS

Takto připravené třídy se používají v souboru App.xaml.cs, který slouží jako spouštěcí místo celé UWP aplikace. Jejich implementace (třídy `Shell` a `MainPage`) spolu s frameworkem vypadá následovně. Ukázka je zkrácena.

```
1  public sealed partial class App : ApplicationBase
2  {
3          public App() :
4          base(Package.Current.DisplayName +
5          (Constants.Instance.BetaMode ? " - Beta version" : ""),
6          typeof(Shell), typeof(MainPage), null)
7      {
8          InitializeComponent();
9      }
```

V ukázce je vidět dědičnost třídy `ApplicationBase`, která spadá taktéž pod framework ISUF. Pro správnou implementaci slouží upravený konstruktor, který specifikuje název aplikace (zde v ukázce s možností doplnění informace o beta verzi do názvu pomocí konstant aplikace), typ menu a typ hlavní stránky.

Tímto způsobem připravená aplikace je funkční, ale neplní žádnou funkci. K tomu je potřeba připravit moduly.

6.1.2 Tvorba modulů

Prvním připraveným modulem je správa studentů. Entita modulu obsahuje jméno studenta a volitelný popis ke studentovi. V ukázce je také vidět ignorování vlastnosti Secured, která slouží pro zabezpečení celé entity a její uložení v šifrované podobě.

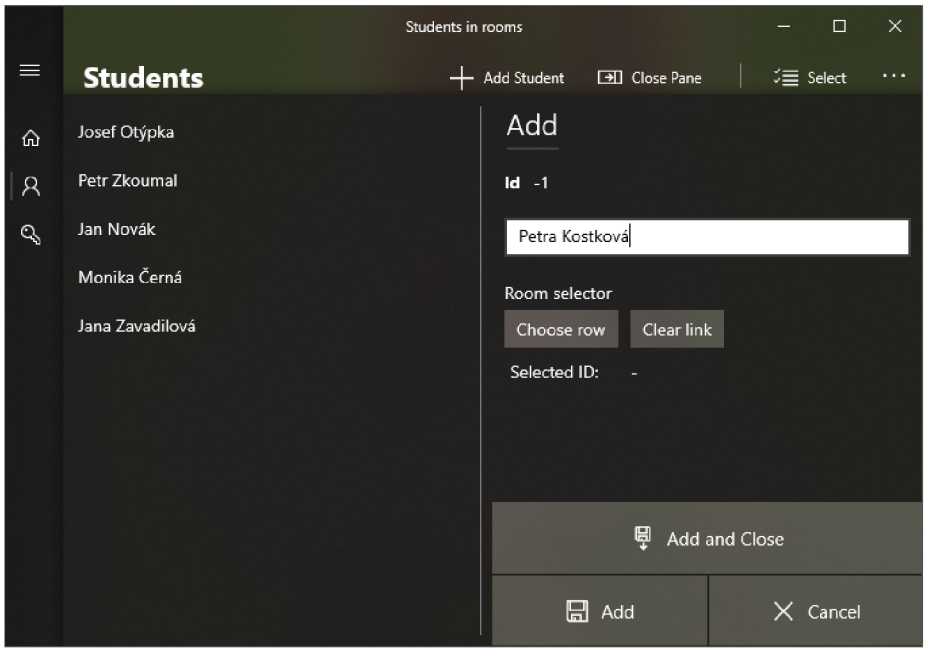
```
1  using ISUF.Base.Attributes;
2  using ISUF.Base.Template;
3
4  namespace StoRIS.Modules
5  {
6          public class Student : BaseItem
7          {
8                  public new string Description { get; set; }
9                  public new string Name { get; set; }
10                 [UIIgnore]
11                 public new bool Secured { get; set; }
12
13                 [UIParams(UIOrder = 5)]
14                 public string Manufacturer { get; set; }
15
16                 [UIParams(UIOrder = 6)]
17                 public string SPZ { get; set; }
18
19                 [LinkedTable(LinkedTableType = typeof(Parking))]
20                 [UIParams(LabelDescription = "Parking selector", UIOrder = 7)]
21                 public int Parking { get; set; } = -1;
22
23                 public override string ToString()
24                 {
25                         return Manufacturer + " - " + SPZ;
26                 }
27         }
28 }
```

Dále byla v entitě vytvořena funkce `ToString`, která se odvozuje od základní funkce `ToString` z třídy `Object`. Důležitost této funkce je vidět v seznamu záznamů, kde se tato funkce volá pro každý záznam pro vytvoření textového popisu záznamu v seznamu.

Stejným způsobem jsou vytvořeny i modul a entita pro místnosti. Ukázka se příliš neliší, není tedy nutné ji popisovat.

Pro jednotlivé moduly je potřeba vytvořit, stejně jako pro menu a hlavní stránku, soubory s předlohou pro design. Mezi tyto soubory patří zobrazení modulu, které nese v sobě

seznam záznamů a místo pro komponenty se záznamem. Mimo tento design se musí vytvořit předloha pro komponenty pro práci se záznamem, nicméně tyto komponenty jsou univerzální a nevyžadují speciální přístup v podobě specifického míření na daný modul díky automaticky generované formulářové části komponenty.



Obrázek 6.2: Zobrazení modulu studentů s bočním panelem pro přidání nového studenta

Tvorba zobrazení modulu je velmi jednoduchá a opět spočívá v dědičnosti. Princip je v popisu designu naprosto stejný jako u hlavní stránky, tudíž není nutné jej zde uvádět. Vývojář není nucen vytvářet design, který lze vidět na obrázku 6.2, tohle vše má na starosti framework. Výchozí design také nabízí možnost připnutí modulu pro rychlejší navigaci do nabídky Start, nastavení viditelnosti bočního panelu a práci s položkami, jako je například hromadné odstranění. Implementace se od hlavní stránky liší v části C# kódu pro zobrazení modulu, který je vidět v následující ukázce.

```
1  using StoRIS.ViewModels;
2  using ISUF.UI.Views;
3
4  namespace StoRIS.Views
5  {
6          public sealed partial class StudentModulePage : ModulePageBase
7          {
8                  public StudentModulePage() : base(typeof(StudentViewModel))
9                  {
10                         this.InitializeComponent();
11                 }
12         }
13 }
```

V ukázce vidíme upravený konstruktor. Parametr konstruktoru, jak název hodnoty parametru napovídá, určuje view model, který daný modul používá. Jedná se tedy o imple-

mentaci návrhového vzoru MVVM a zde view modelu `StudentViewModel`, který obstarává veškerou práci modulu, včetně zobrazení a entit, komunikaci modulu s ostatními moduly, jednotlivými komponentami použitými v modulu i se samotnou aplikací.

Při pohledu na implementaci frameworku ISUF v daném view modelu, je zde vidět opět silné zjednodušení. V konstruktoru je vidět, že každý view model je vázán k danému modulu, nicméně více modulů může využívat stejný view model. Dále je v konstruktoru vidět množství typů, které se, podle jejich názvů, odkazují ke komponentám bočního panelu a jejich view modelům.

```
1  using StoRIS.Modules;
2  using ISUF.Base.Messages;
3  using ISUF.UI.ViewModel;
4  using System;
5  using System.Threading.Tasks;
6  using Windows.UI.StartScreen;
7
8  namespace StoRIS.ViewModels
9  {
10         public class StudentViewModel : ModuleVMBase<Student>
11         {
12                 public StudentViewModel(Type modulePage) : base(modulePage)
13                 {
14                 }
15
16                 public StudentViewModel(Type modulePage,
17                 SecondaryTile secondaryTile, Type addPane,
18                 Type addViewModel, Type detailPane, Type detailViewModel) :
19                 base(modulePage, secondaryTile,
20                 addPane, addViewModel, detailPane, detailViewModel)
21                 {
22                 }
23         }
24 }
```

6.1.3 Tvorba komponent bočního panelu

Po zobrazení modulu je nutné vytvořit komponenty pro práci s položkou a pro její zobrazení v režimu read-only, tedy detail. Implementace designu, opět formou dědičnosti, je zase stejná, tedy není nutné ji dále rozvádět.

K implementaci práce s položkou slouží třída `ModuleAddControlBase`. Její použití lze vidět v ukázce.

```
1  using ISUF.UI.Modules;
2  using ISUF.UI.Views;
3  using System;
4
5  namespace StoRIS.Controls
6  {
7          public sealed partial class AddItem : ModuleAddControlBase
```

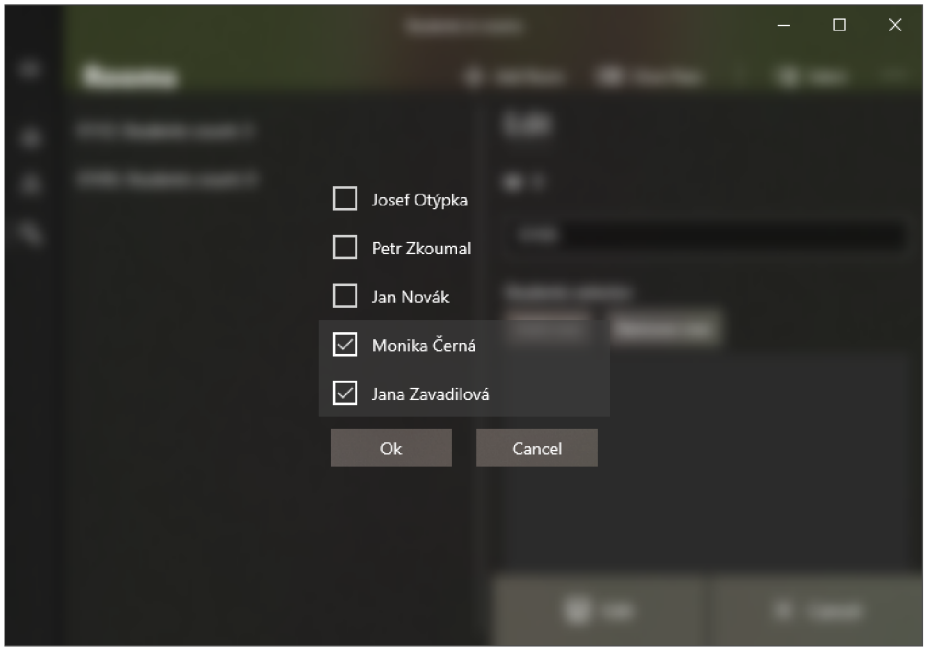
```
8         {
9                 public AddItem(UIModule uiModule, Type viewModelType,
10                params object[] viewModelArgs) :
11                base(uiModule, viewModelType, viewModelArgs)
12                {
13                        this.InitializeComponent();
14                }
15        }
16 }
```

Jak lze vidět, třída vyžaduje konstruktor, který určuje modul, ke kterému je komponenta vytvářena. Ten slouží pro získání dat ohledně modulu, jako jsou např. jeho vlastnosti entity. Dále konstruktor obsahuje informace pro vytvoření view modelu určeného pro danou komponentu.

```
1  using StoRIS.Modules;
2  using ISUF.Base.Classes;
3  using ISUF.UI.ViewModel;
4  using ISUF.UI.Views;
5  using System;
6
7  namespace StoRIS.ViewModels
8  {
9         public class StudentAddViewModel : ModuleAddVMBase<Student>
10        {
11                public StudentAddViewModel(Messenger messenger,
12                Type modulePage, ModuleAddControlBase form) :
13                base(messenger, modulePage, form)
14                {
15                }
16        }
17 }
```

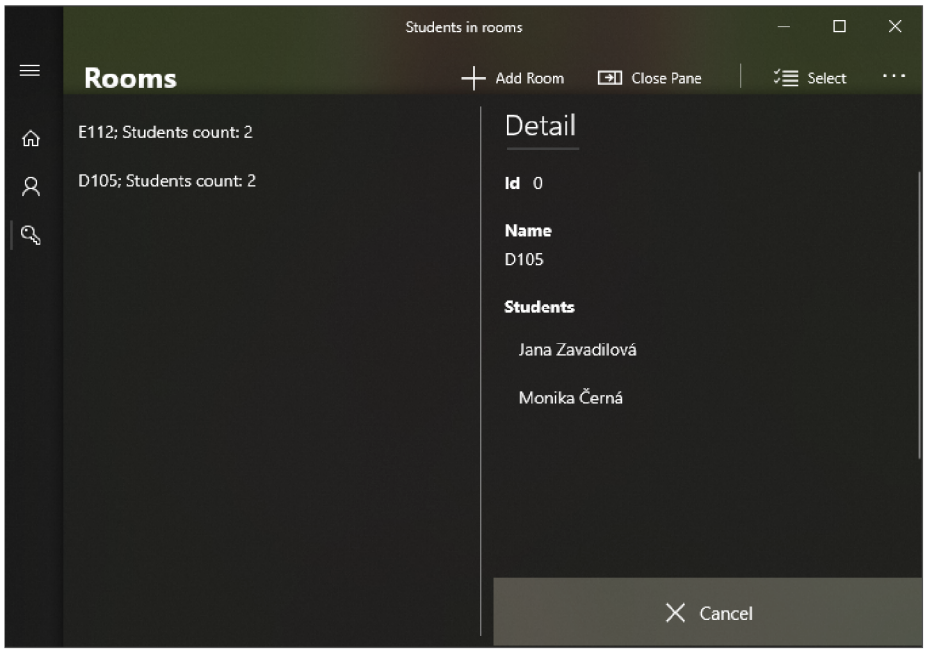
Konstruktor view modelu zde obsahuje kromě messengeru (návrhový vzor Messenger) a stránky modulu pro získání celého modulu, také komponentu, která view model používá. Důvodem je schopnost pracovat s daty z komponenty, jako je například jejich čtení. Tyto data se pak posílají modulu pro uložení do databáze. Výsledný design komponenty pro práci s položkou v režimu přidání je vidět na obrázku 6.2.

Implementace vazeb mezi moduly je vytvořena interně a není nutné ji nijak dodělávat. Výběr záznamů z provázaných modulů se dělá pomocí speciálně upraveného modálního okna, viz obrázek 6.3 s možností výběru N záznamů. Modální okna se dále ve frameworku používají například pro zobrazení poznámek k nově vydané verzi po jejím prvním spuštění. Zobrazení okna je neblokující, tedy aplikace na pozadí pořád funguje bez přerušení, jako se například děje při zobrazování hlášek pomocí třídy `MessageBox`.



Obrázek 6.3: Výběr N záznamů z vázaného modulu

Další komponentou je detailní zobrazení. Jedná se o pouhé zobrazení hodnot v režimu pouze pro čtení. Implementace je stejná jako u komponenty pro práci s položkou. Jediný rozdíl je zde v děděné třídě. Detail komponenty používá pro svou práci třídu `ModuleDetailControlBase`. Výsledný design je možné vidět na obrázku 6.4, kde lze také vidět seznam studentů v místnosti z provázaného modulu studentů.



Obrázek 6.4: Komponenta pro zobrazení detailu položky

6.1.4 Registrace modulů a zprovoznění aplikace

Poslední úpravou, kterou je nutné udělat, je registrace modulů do aplikace. V aplikaci se vytvoří správce modulů a určí se jeho typ přístupu do databáze. Poté se začnou vytvářet a registrovat moduly. Při vytváření se určí typ modulu a jeho název. Dále se podle typu určují další parametry, jako např. správce záznamů nebo ikona pro zobrazení v aplikaci. Registrační funkce se dědí z třídy `ApplicationBase` a nachází se v souboru App.xaml.cs, který tuto třídu dědí.

```
1  public override void RegisterModules()
2  {
3          ModuleManager = new UIModuleManager(typeof(XmlDbAccess));
4
5          var studModule = new UIModule(typeof(Student),
6          typeof(BaseItemManager), "Students", (Symbol)0xE13D,
7          typeof(StudentModulePage));
8
9          var roomModule = new UIModule(typeof(Room),
10         typeof(BaseItemManager), "Rooms", (Symbol)0xE192,
11         typeof(RoomModulePage));
12
13         ModuleManager.RegisterModule(studModule);
14         ModuleManager.RegisterModule(roomModule);
15 }
```

Výsledná aplikace je po těchto implementacích spustitelná a plní svou funkci. Aplikace obsahuje modul studenta a místnosti, jejich vazbu, ukládání dat v podobě XML a výchozí design ve dvou motivech.

6.2 Problémy při vývoji

Vytvořit plnohodnotný framework pro tvorbu informačních systémů nad platformou UWP je velmi náročné. I když tato práce popisuje jeho řešení, jedná se pouze o koncept, který má řadu nedokonalostí. Nicméně i přesto koncept řešil řadu unikátních problémů, které se na první pohled mohou, v jiných situacích, zdát banální. V této kapitole se budu věnovat nejen těmto problémům, ale i problémům ostatním, které sužují koncept ve větší míře a zatím nebyly vyřešeny.

6.2.1 UWP nepodporuje nahrávání kódu za běhu

Problém, který není možné aktuálně řešit z důvodu zabezpečení celé platformy UWP. Win32 programy, tedy desktop aplikace postavené pomocí technologií WinForms a WPF, umožňují dynamické načítání DLL knihoven, tedy funkčního kódu, který ale lze zneužít ke spuštění nebezpečného kódu. Proto byla tato možnost z platformy UWP odstraněna a výsledná aplikace včetně všech přiložených DLL knihoven je neměnná.

Z tohoto pohledu se dá říct, že řešení ISUF není modulární, protože nelze odebrat jeho část, jelikož je napevno vložena do kódu. Modularita řešení ISUF ale spočívá v možnosti vytvořit jednotku aplikace (modul) bez jakýchkoliv návazností na okolní řešení a lze ji bez problémů odebrat (za předpokladu, že neobsahuje návaznost z jiných modulů). Tento přístup byl ve velmi omezené míře otestován při vývoji mé aplikace The Daily Notes, která je

postavená na platformě UWP, dostupné v Microsoft Store a jejíž zdrojový kód je dostupný na GitHub.

6.2.2 XAML design nelze dědit

Design v aplikacích postavených na WPF a UWP se vytváří velmi jednoduše pomocí jazyka XAML. Nabízí spoustu možností, které dřív na WinForms možné nebyly, jako například vazby, ale přináší s sebou jeden malý problém. Nelze na něm tvořit dědičnost. Argumentem může být tvorba uživatelských komponent, jako je v případě komponent pro boční panel, ale ty nenabídnou možnost úpravy a už pak vůbec nenabídnou možnost jednoduchého přepsání chování. Programátor by musel znovu vytvořit celý prvek a pracovat s ní jako s vlastní komponentou, což velice znesnadňuje práci.

Z tohoto důvodu jsem se rozhodl celý design implementovat v kódu C#, který nakonec přinesl omezení. To se ale podařilo vyřešit a popisuji ho v následujících částech. Možnost tvorby designu na základě předlohy jsem vyřešil dědičností, která se v sekci 6.1 několikrát opakuje. Nejedná se o změnu XAML, i když i tam se dědičnost využívá, ale implementování celé nové třídy, která nahrazuje třídy původní.

6.2.3 UWP nepodporuje rozhraní `INotifyPropertyChanged` pro design vytvořený v C#

Design vytvořený pomocí C# kódu a následné zděděný do jiného prvku neobsahuje jednu velmi důležitou a zároveň užitečnou funkci. Rozhraní `INotifyPropertyChanged` patří k základům správné implementace návrhového vzoru MVVM, protože funkce, kterou obsahuje, dokáže notifikovat design, kdy se daná vlastnost změnila a je třeba ji obnovit a znovu načíst do prvku. Ve chvíli, kdy překladač vytváří vazbu (Binding), tak se registrují jednotlivé vlastnosti na vazbu a interně jsou spravovány. Správné implementování rozhraní a jeho navázání na design je velmi jednoduché, ale dohledávání případných chyb je velmi složité. O tom svědčí množství dotazů na webu StackOverflow, kdy podle množství různých dotazů, lze usoudit, že pokazit se může obrovské množství věcí.

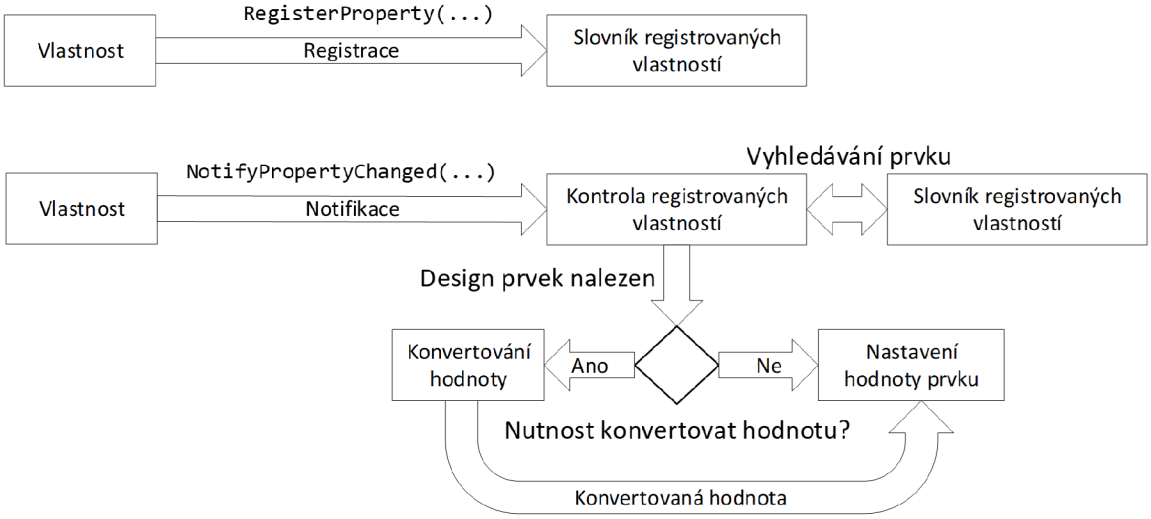
Při zjištění, že rozhraní nefunguje, jsem byl nucen vytvořit vlastní obdobu `INotifyPropertyChanged`, zvanou `PropertyChangedNotifier`, která velmi podobný systém fungování. Při tvorbě vazby se v kódu provádí i registrace vlastností, jak je vidět v následující ukázce. Problém tohoto přístupu je v délce registračního řádku, protože jak je vidět, je nutné specifikovat všechny vlastnosti, stejně jak jsou specifikované u vazeb. V ukázce se registruje design prvek s názvem *AddText*, který je typu `TextBlock` a cílenou vlastností prvku je *TextProperty*. Hodnota se bere z vlastnosti *AddEditItem*. Nutné je specifikovat, který view model vlastnost registruje, aby bylo možné cílit na správnou položku. V případě nutnosti transformovat data jsou připraveny parametry pro specifikaci konvertoru.

```
1 ApplicationBase.Current.PropertyChangedNotifier.RegisterProperty(
2        AddText, TextBlock.TextProperty, "AddEditItem", viewModelType,
3         "Id", converter: new IdConverter(), converterParameter: "text");
```

Notifikace změny vlastnosti se provádí voláním funkce `NotifyPropertyChanged`, která v parametrech nese typ view modelu, ve kterém se vlastnost nachází, a hodnotu vlastnosti. Volitelně lze zadat i název vlastnosti, nicméně ten se získává pomocí reflexe při volání funkce. Následující ukázka zobrazuje volání notifikace.

```
1 PropertyChangedNotifier.NotifyPropertyChanged(GetType(), AddEditItem);
```

Popisovanou funkčnost popisuje obrázek 6.5. Náročnost řešení oproti rozhraní `INotifyPropertyChanged` je nepříjemné pouze v registrování vlastnosti nahrazující vazbu, protože notifikace se musí provádět ručně i v rozhraní.



Obrázek 6.5: Schéma funkcí třídy `PropertyChangedNotifier`

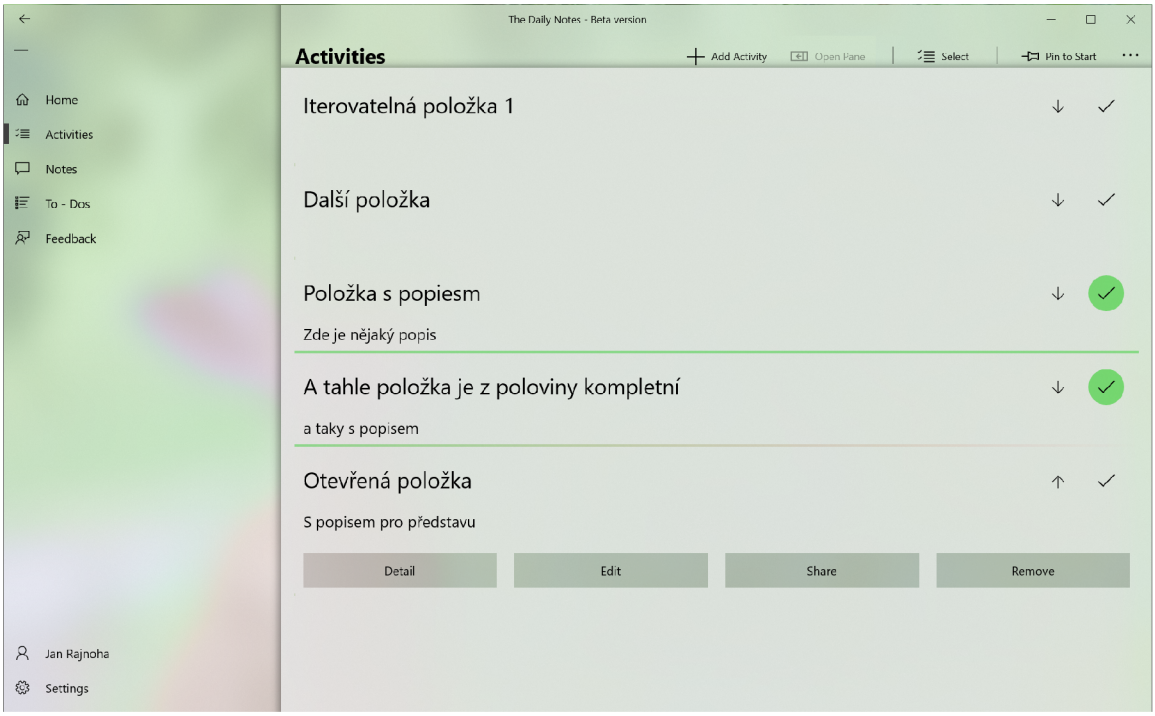
6.2.4 Binding vytvořený v C# nepodporuje TwoWay režim

Spolu s vlastní notifikační infrastrukturou pro vlastnosti souvisí i jeden z režimů vazeb v XAML - TwoWay. Tato vazba dovoluje propagovat změny z designu zpátky do vlastností, která ale vzhledem k designu vytvořeném v C# nefunguje. Nutnost propagovat hodnoty z prvků zpět do view modelu je důležitá při přidávání nebo editování záznamů. Tady se jednalo o problém. Další překážkou ve snadné implementaci byla podmínka automaticky generovaných formulářů, tedy nevědomost výsledného formuláře.

Použité řešení bylo navrženo na základě vědomosti o teoretickém vzhledu formuláře, přesněji názvu jeho prvků. Postup byl již nastíněn v části 5.4.2. Za každý název prvku byl při jeho vytváření vložen speciální textový řetězec, který označoval jeho použití jako pole formuláře. Při ukončení práce se záznamem s jeho následnou prací se provádí ve view modelu volání třídy `FormDataMiner`, jehož funkce `FillValuesIntoProperty` dokáže projít všechny tyto prvky formuláře. Ten se předává pomocí parametru a vkládá se opět do vlastností, a v případě nutnosti hodnoty konvertuje na požadovaný formát.

6.2.5 Kombinace `DataTemplate` a vazeb není v C# možná

Toto je aktuálně nevyřešený problém, který ale nijak zásadně neovlivnil vývoj. Problém spočívá v možnosti tvořit vlastní design položek ve výčtech, jako je např. `ListBox`. Obrovskou výhodou je totiž možnost vytvořit vlastní prvek a k němu vlastnosti s možností vazeb, a tyto vazby poté zapojit do designu, který, v případě výčtových prvků, je schopen iterovat seznam a postupně položky naplňovat, stejně jako je vidět na ukázce 6.6 z aplikace The Daily Notes.



Obrázek 6.6: Ukázka využití `DataTemplate` a vazeb v prvku `ListBox`

Menším problémem se stalo vytvoření tohoto designu, protože ten nejde vytvořit pomocí C# kódu. K tomu je nutné využít třídy `XamlReader` a kódu napsaného v XAML. Pomocí třídy se vezme XAML kód, načte se, a jeho výsledek se uloží na požadované místo. Další problém, prozatím nevyřešený, vznikl při využívání vazeb, kdy reader nebyl schopen XAML kód přečíst, a i když se po několika pokusech čtení povedlo, nebylo možné vazby využívat.

Kapitola 7

Další možné směry vývoje

Jedná se pouze koncept, který ukazuje schopnost platformy UWP zvládnout tak složitý úkol, jako je vytvoření frameworku pro informační systémy. V současné době žádný z vývojářů neví, jak se bude vývoj platformy UWP vyvíjet, protože jeho konkurent PWA nabývá na síle a strhává na sebe čím dál větší pozornost. Nicméně UWP dostává další a další updaty, které jej dělají čím dál tím více použitelným při řešení obtížných úkolů.

V následujícím textu se pokusím nastínit možnosti a schopnosti, které by framework ISUF posunuly z konceptu k reálně použitelnému komerčnímu řešení. Text je rozdělen podle oblastí, do nichž spadá nastíněný vývoj.

7.1 Projekt ISUF

I když je ISUF pouze konceptem, bylo při jeho návrhu dbáno na co nejvyšší autentičnost s reálným projektem. Postupem času se ale začalo ukazovat, že ne všechna rozhodnutí byla správná. Naopak koncept mnohé funkce postrádá, které bude nutné implementovat do konečného řešení, jako je například návrh serverového řešení frameworku.

7.1.1 Lepší rozčlenění na menší projekty

Ačkoliv je v aktuální době vývoj frameworku ISUF rozdělen do 5 částí (Base, Security, Interface, Storage, UI), některé z nich míchají dohromady více funkcí a mohou být matoucí. Příkladem je `ISUF.Interface`, který obsahuje rozhraní jak pro část Storage, tak UI, viz obrázek 5.1. Na první pohled může být zvláštní, proč se využívá ve dvou dalších částech frameworku, a budí tak dojem porušení návrhu, nicméně interně jsou rozhraní oddělena na část Storage a UI.

Stejně tak by asi bylo lepší oddělit z části Storage přístupy do databáze, aby byla možná jejich jednodušší aktualizace a při použití se pak zvýšila celková přehlednost projektu. Stejný přístup by bylo pravděpodobně lepší využít i v části pro tvorbu designu, protože aktuálně je design vytvořený pouze pro Windows 10 a nepodporuje multiplatformní řešení. Vyjmutím Windows 10 závislostí do samostatné části se zvýší přehlednost a možnost vytvářet vlastní balíček pro tvorbu designu, například nad technologií WPF.

7.1.2 Multiplatformní řešení

Koncept frameworku ISUF je napsán pomocí .NET Core, tedy multiplatformního frameworku společnosti Microsoft. Tato možnost otevírá dveře vytvořit další odnože fra-

meworku ISUF pro další systémy, jako je například Linux nebo MacOS. Při návrhu bylo dbáno na odstínění závislosti na systému Windows, aby případný přechod byl co nejjednodušší.

7.1.3 Transformace na server aplikaci

Jedna z možností multiplatformního řešení je také aplikace typu PWA. Tato možnost, i když přináší některá omezení, je velmi příjemná obzvláště při tvorbě řešení pro mobilní platformy při volném přístupu aplikace z internetu. Responzivní design se dokáže přizpůsobit každé obrazovce a využití síly serverového výkonu sníží náročnost na hardwarové vybavení cílového stroje. Problém přichází ve chvíli, kdy by měla být aplikace dostupná pouze z interní sítě. Ne všechny firmy využívají pro pevné připojení stejné podmínky jako pro WiFi a může se pak stát, že PWA aplikace nebude z mobilu dostupná. V tuto chvíli je řešení pouze dostupné na zařízeních podporující VPN. Pokud zařízení splňuje tuto podmínku, je PWA jedna z možných variant posunu frameworku.

7.2 Zabezpečení

Dnešní doba není až tak o lidech, jako spíše o datech. Člověk lehce zapomene, ale data mohou zůstat navždy, anebo naopak mohou být velmi jednoduše smazána. Z toho důvodu se velmi důsledně dbá na zabezpečení aplikací a omezení přístupu řadovým zaměstnancům.

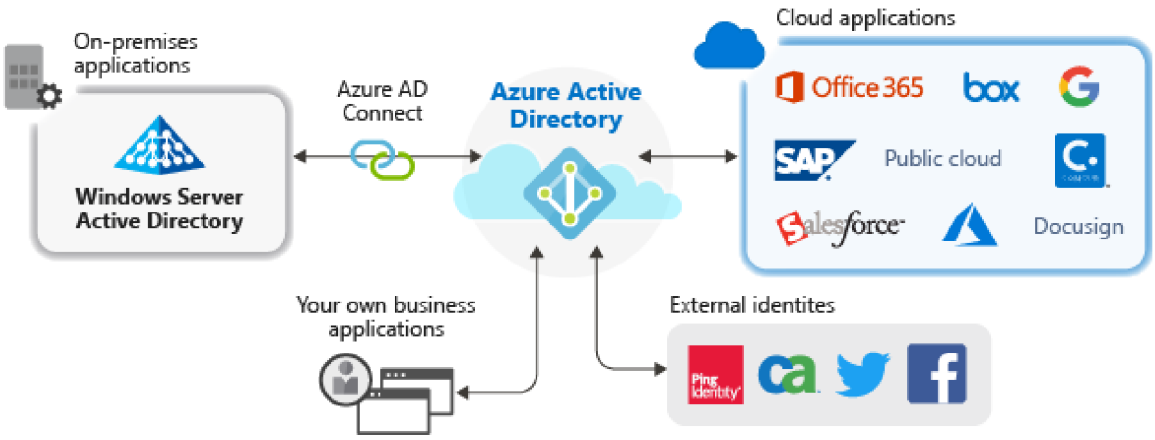
7.2.1 Přihlášení pod heslem

I když framework nabízí možnost základní implementace zabezpečení a UI vstup (tlačítko k přihlášení), chybí mu možnost přihlašování a podmíněného přístupu k položkám.

Jednou z možností, jak implementovat přihlášení, je vlastní databáze uživatelů. Tento postup je sice jednodušší, ale je méně bezpečný, protože správu musíme zajistit sami.

Další možností je využití providerů, poskytovatelů, kteří poskytují ověření identity, nicméně i tak je nutné spravovat databázi uživatelů. Máme také možnost využití volně dostupných ověřovacích služeb, jako jsou účty Outlook (dříve Live.com) od společnosti Microsoft, Google, Facebook, Twitter, LinkedIn, GitHub a další. Do této kategorie spadá i přihlašování pomocí účtu vytvořeného na zařízení s využitím emailové adresy. Systém Windows nabízí i podporu přihlašování bez hesla, tedy možnost přihlášení přes biometrické údaje (scan obličeje nebo oční rohovky a otisk prstu), PIN kód, bezpečnostní klíč nebo pomocí vyznačení bodů na obrázku.

Třetí možnost je pro správu přihlašování využití podnikové řešení, jako je například Office 365 společnosti Microsoft, které nabízí možnost Active Directory [20]. Active Directory vychází z řešení LDAP (celým názvem *Lightweight Directory Access Protocol*) a slouží k centrálnímu uchovávání informací za pomocí stromových struktur. Tento způsob přihlašování lze využít i k přihlašování do PC, kde umožňuje hromadně nastavovat nastavení zařízení, spravovat aplikace a povolovat přístupy různě po síti.



Obrázek 7.1: Schéma využití Azure Active Directory v aplikacích [20]

7.3 Práce s databází

Koncept ISUF v aktuální podobě pracuje pouze s lokálními daty, tedy takovými, která jsou dostupná na stejném počítači, jako samotná aplikace. Tento postup však není příliš vhodný v drtivé většině firem, protože ty používají z pravidla centralizované databáze nebo rovnou databáze uložené v cloudu.

7.3.1 Podpora cloud databází

V současné době koncept ISUF podporuje pouze lokální ukládání dat, což je ve větším měřítku nedostatečné. Většina případů použití frameworku bude právě s centralizovanou databází, k nichž je nutné přistoupit pomocí sítě. Trend dnešní doby ukazuje značný posun směrem k servisovaným službám, ke kterým patří i cloudové databáze, tedy takovým, kdy samotná databáze není v místě společnosti, ale stará se o ni jiná společnost. K takovým řešením patří například i Microsoft Azure, jenž nabízí cloudové řešení databází všech možných velikostí. Mezi další poskytovatele cloud databází lze zmínit Amazon AWS, Google Cloud nebo IBM Cloud.

7.3.2 Zvážení využití Entity Framework

Důvod, proč nebyl využit Entity Framework při tvorbě konceptu ISUF byl popsán v sekci 4.1. Tento problém se ale netýká aplikací postavených pro technologii WPF a WinForms. Nejen, že by využití EF zjednodušilo celý návrh, ale v aktuální době Microsoft zpřístupňuje více a více možností pro vývoj aplikací nad těmito technologiemi s využitím schopností Windows 10, jako je tvorba dlaždic, záznamů v Timeline nebo propracovanější práci s notifikacemi. Koncový uživatel tak není ochuzen o žádnou část moderních funkcí a může aplikaci spustit i na starším systému.

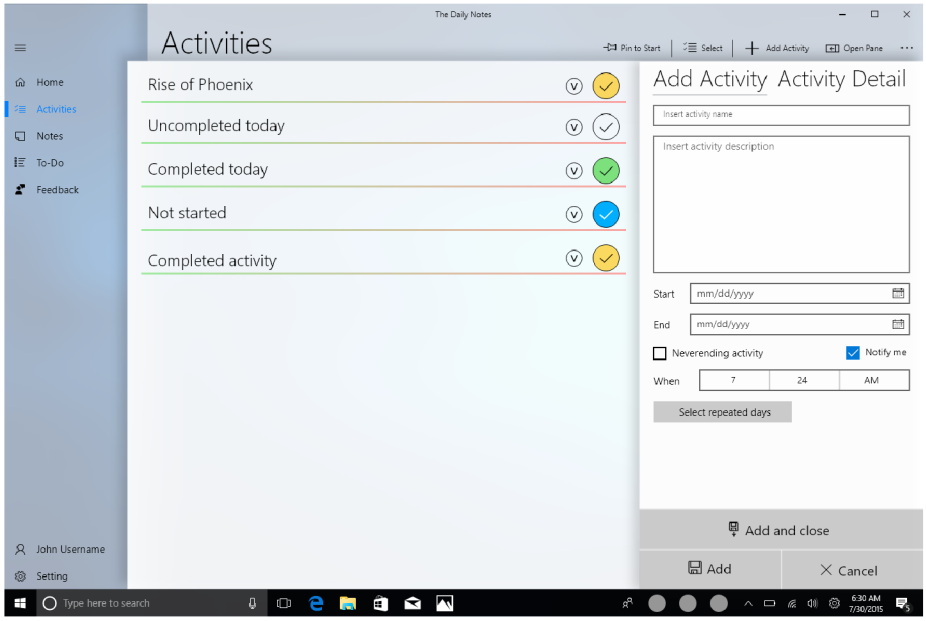
Pokud by jsme zkoumali možnosti využití Entity Framework Core, museli bychom vyřešit problém relací, tedy mezitabulek pro relace. C# nicméně nabízí možnost tvorby dynamických tříd, ale tato praktika není příliš efektivní.

7.4 Design

Každá aplikace je dobrá právě tak, jak dobře a pohodlně se ovládá. Každé ulehčení, každé připodobnění se zajetým standardům a každá funkce, která uživateli pomůže, dělá aplikaci oblíbenější a použitelnější. Proto se také musí dbát na design a jeho celkové zakomponování v oblasti Windows 10 tak, aby uživateli připomínalo známé prostředí, se kterým rád pracuje.

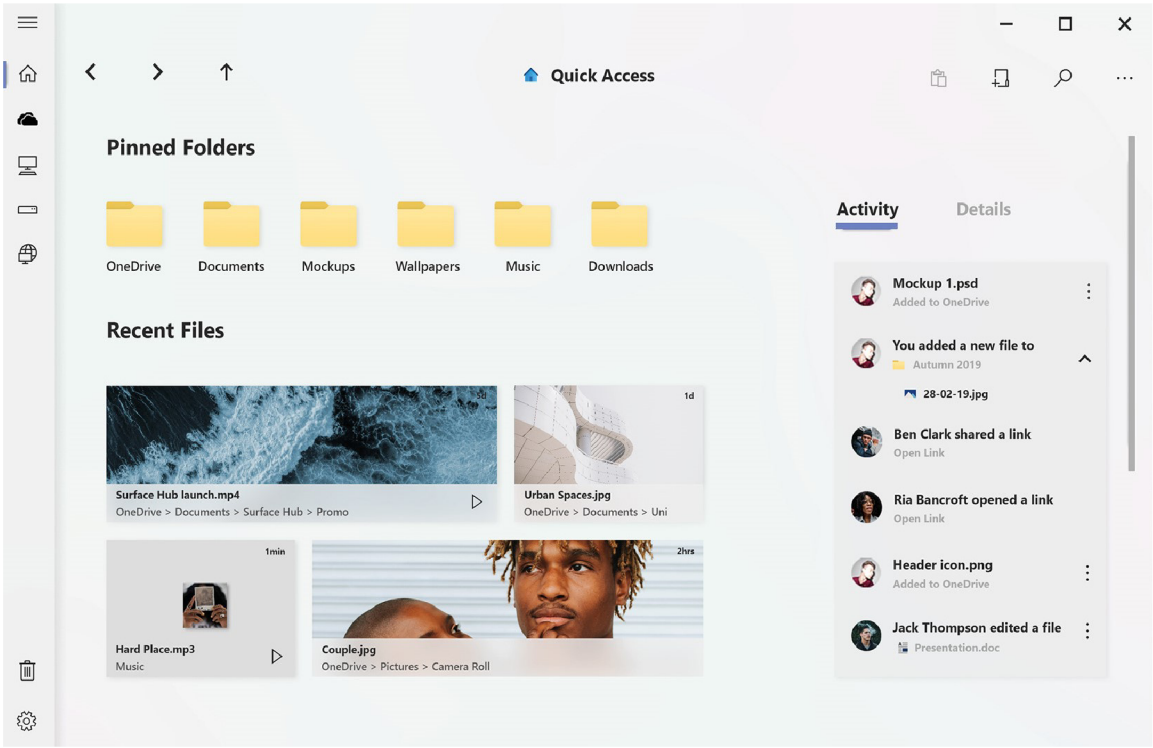
7.4.1 Kompletní implementace Fluent Design System

Windows 10 přišel s novým designem, který nazval Fluent Design System (zkráceně jen Fluent). Tento nový přístup k designu si zakládá na jednoduchém rozhraní zaměřeném uživatelským směrem, použitím osvětlení pro zvýraznění prvků, hloubky zobrazení, které připomíná na sobě poskládané vrstvy, animací jednotlivých prvků a použitím speciálního vzhledu celého designu. [19] Jedním z návrhů použití Fluent Deisgn System je obrázek 7.2 pro aplikaci The Daily Notes, která využívá stejný styl designu jako framework ISUF. Možnost použití Fluent Design System je velmi široké a různorodé, což je vidět i na obrázku 7.3.



Obrázek 7.2: Designový návrh aplikace The Daily Notes za využití Fluent Design System

Fluent Design System se krátce po svém představení (dříve pod názvem *Project Neon*) stal častým cílem designerů, kteří pro něj začali tvořit množství konceptuálních designů aplikací i samotného Windows 10. I z toho důvodu se společnost Microsoft rozhodla, že bude postupně vyvíjen balíček zaobalující design prvky vytvořené v tomto duchu i pro technologie WPF a WinForms pod názvem *Windows UI Library*. [18] Tato knihovna umožňuje využití tvorby stejného designu jak u UWP, tak i u WinForms a WPF, a to i při využití aplikace na staších systémech.

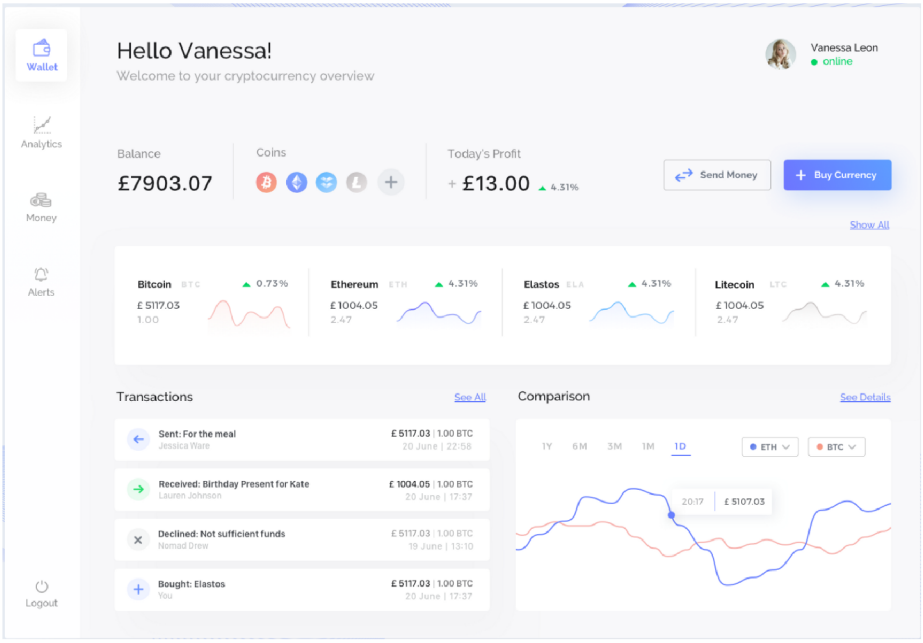


Obrázek 7.3: Koncept aplikace File Explorer ve Windows [7]

7.4.2 Tvorba dashboardů

Tvroba dashboardů je pro každý informační systém důležitou funkcí. Dashboard nejen, že hezky vypadá, ale také nese spoustu informací, např. v jednom obrázku grafu dokáže odkazovat na daná místa v aplikaci, tvořit různé soupisy a v krajním případě i předvídat aktuální vývoj, který pak daný modul podrobně rozebere. Dashboard je jeden velký obsah všech modulů, který je sice stejný, ale při tom se neustále mění. V kombinaci s Fluent Design System z předchozí části vytváří velmi silné spojení při zachování jednoduchého designu a vysoké informační hodnoty. Na obrázku 7.4 je vidět návrh designu dashboardu pro sledování kryptoměn v moderním podání.

Tvorba dashboardů je ze všech navrhovaných funkcí největší a nejsložitější. Nejen, že je potřeba implementovat řadu grafických komponent, které by byly schopny vytvářet grafy a soupisy, ale také nastavit nový přístup k modulům a práci s nimi a implementovat algoritmy pro tvorbu analýz a soupisů. Takto hotový framework pak již bude schopen komerční konkurence ostatním velkým frameworkům a může se rovnat jiným aplikacím z oblasti CRM a ERP, jako je například Microsoft Dynamics. [15]



Obrázek 7.4: Návrh designu dashboardu pro sledování vývoje kryptoměn [13]

Kapitola 8

Závěr

Informační systémy jsou všude kolem nás, a ani si nemusíme být vědomi toho, že je používáme. Každý vývoj nového systému je zdlouhavý proces, který stojí nemalé peníze. Z toho důvodu vznikají pomocné frameworky a aplikace, které následný vývoj urychlí, a to od psaní kódu za využití knihoven frameworku až po vytváření aplikace pomocí grafického rozhraní v jiné aplikaci.

Tato práce se zabývala tvorbou konceptu frameworku pro informační systémy nad operačním systémem Windows 10 za využití pokročilých funkcí, které nabízí právě Windows 10. Součástí práce byl způsob implementace na testovacím projektu (aplikaci) pro evidenci studentů v místnostech (StoRIS) a přiložený projekt pro evidenci aut na parkovištích (BrnoParkingIS).

V kapitole 2 jsem nejprve popsal technologie, které se pojí s vývojem aplikací nad Windows 10, a které jsem při svém vývoji používal. Dále jsem v kapitole 3 rozebral jednotlivé požadavky, které jsou kladeny na framework, zabývající se tvorbou informačních systémů. Na konci kapitoly bylo provedeno pomocí analýzy SWOT zhodnocení, proč dělat nový framework a čím se tento stává ve světě ostatních frameworků jedinečným. Kapitola 4 byla další teoretická kapitolou, která se tentokrát zabývala již hotovými řešeními. Kapitola bbyla rozdělena na části frameworku, které lze nahradit, a pak řešení, která mohou nahradit celý framework.

Další kapitola se zabývala vývojem konceptu ISUF, přesněji jeho návrhem. Kapitola popisovala návrh stěžejních míst, jako jsou práce s entitami, Windows 10 a generování UI formulářů. Vývoj testovací aplikace a problémy při vývoji byly popisovány v kapitole 6. První část popisovala s praktickými ukázkami nejjednodušší implementaci frameworku ISUF a jeho grafický výsledek. Další část se pak zabývala problémy, které při vývoji vznikaly a které buď bylo možné řešit, a nebo které zůstaly z důvodu omezení použitých postupů v aktuální době nevyřešené.

7. kapitola byl věnována budoucnosti, přesněji řečeno vývoje konceptu ISUF v plnohodnotný produkt. Kapitola byla opět rozdělena do oblastí, které by následný vývoj mohl zasáhnout, ať už je to základní navržení projektu nebo až zabezpečení a implementování nových funkcí. Osobně se chci vývoji frameworku ISUF nadále věnovat a ještě více přispívat k automatizaci vývoji informačních systémů.

Literatura

[1] *MM .NET Application Framework* [Product page]. [Online; navštíveno 20.5.2020]. Dostupné z: https://www.oakleafsd.com/products.

[2] *Online documentation Syncfusion Essential Studio for UWP* [Syncfusion Documentation]. [Online; navštíveno 20.5.2020]. Dostupné z: https://help.syncfusion.com/uwp/overview.

[3] *Online documentation Telerik UI for UWP* [Telerik Documentation]. [Online; navštíveno 20.5.2020]. Dostupné z: https://docs.telerik.com/devtools/universal-windows-platform/introduction-uwp?_ga=2.147548535.1549963445.1589154610-1626230954.1588178761.

[4] *Online documentation DevExpress eXpressApp Framework* [DevExpress Documentation]. 2019. [Online; navštíveno 19.5.2020]. Dostupné z: https://docs.devexpress.com/eXpressAppFramework/112670/expressapp-framework.

[5] *Online documentation DevExpress Windows 10 App Controls* [DevExpress Documentation]. 2019. [Online; navštíveno 20.5.2020]. Dostupné z: https://docs.devexpress.com/Win10Apps/212019/windows-10-app-controls.

[6] Anderson, R., Latham, L. a aj., W. P. *Introduction to ASP.NET Core* [Microsoft Docs]. 2020. [Online; navštíveno 14.5.2020]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-3.1.

[7] Ata, C. *Files for Light OS* [Behance]. Květen 2019. [Online; navštíveno 11.5.2020]. Dostupné z: https://www.behance.net/gallery/80605511/Files-for-Light-OS.

[8] Chapman, A. *How to build cross-platform console apps with .NET Core* [OpenSource]. Květen 2017. [Online; navštíveno 13.5.2020]. Dostupné z: https://opensource.com/article/17/5/cross-platform-console-apps.

[9] Chauhan, S. *A Brief Version History of .NET Framework* [dotnettricks.com]. Prosinec 2013. [Online; navštíveno 16.12.2018]. Dostupné z: https://www.dotnettricks.com/learn/netframework/a-brief-version-history-of-net-framework.

[10] Chekalin, D. *What are Progressive Web Apps?* [Quora]. 2018. [Online; navštíveno 12.5.2020]. Dostupné z: https://www.quora.com/What-are-the-disadvantages-of-PWA-Framework.

[11] Chugh, A. *Android MVVM Design Pattern* [Journal Dev]. Květen 2018. [Online; navštíveno 19.5.2020]. Dostupné z: https://www.journaldev.com/20292/android-mvvm-design-pattern.

[12] Decker, J., Lich, B. a aj., G. L. *Windows 10 application management* [Microsoft Docs]. 2019. [Online; navštíveno 16.5.2020]. Dostupné z: https://docs.microsoft.com/en-us/windows/application-management/.

[13] Erfan, K. M. *Web Application Dashboard - Booking Software* [Dribble]. 2020. [Online; navštíveno 10.5.2020]. Dostupné z: https://dribbble.com/shots/11182329-Web-Application-Dashboard-Booking-Software/attachments/2786674?mode=media.

[14] Erickson, K. *Data binding and MVVM* [Microsoft Docs]. 2018. [Online; navštíveno 18.5.2020]. Dostupné z: https://docs.microsoft.com/en-us/windows/uwp/data-binding/data-binding-and-mvvm.

[15] Groes Petersen, S., Dupont, E. a Pedersen, S. W. *Welcome to Dynamics 365 Business Central* [Microsoft Docs]. 2020. [Online; navštíveno 10.5.2020]. Dostupné z: https://docs.microsoft.com/en-us/dynamics365/business-central/.

[16] Harkiran78. *.NET Framework Class Library (FCL)* [Geeks for geeks]. [Online; navštíveno 15.5.2020]. Dostupné z: https://www.geeksforgeeks.org/net-framework-class-library-fcl/.

[17] Jacobs, M., Erickson, K. a aj., M. S. *Data binding in depth* [Microsoft Docs]. 2018. [Online; navštíveno 18.5.2020]. Dostupné z: https://docs.microsoft.com/en-us/windows/uwp/data-binding/data-binding-in-depth.

[18] Jacobs, M., Satran, M. a aj., M. S. *Windows UI Library* [Microsoft Docs]. 2019. [Online; navštíveno 11.5.2020]. Dostupné z: https://docs.microsoft.com/en-us/uwp/toolkits/winui/.

[19] Jacobs, M., Schofield, M. a Hickey, S. *Design and code Windows apps* [Microsoft Docs]. Květen 2019. [Online; navštíveno 11.5.2020]. Dostupné z: https://docs.microsoft.com/en-us/windows/uwp/design/.

[20] Kess, B., Martin, M. a Guzman aj., C. de. *Application management with Azure Active Directory* [Microsoft Docs]. 2019. [Online; navštíveno 19.5.2020]. Dostupné z: https://docs.microsoft.com/en-us/azure/active-directory/manage-apps/what-is-application-management.

[21] LH, S. *SOLID Principles: Explanation and examples* [IT Next]. 2019. [Online; navštíveno 21.5.2020]. Dostupné z: https://itnext.io/solid-principles-explanation-and-examples-715b975dcad4.

[22] Megali, T. *Android MVVM Design Pattern* [Tuts plus]. 2016. [Online; navštíveno 19.5.2020]. Dostupné z: https://code.tutsplus.com/tutorials/an-introduction-to-model-view-presenter-on-android--cms-26162.

[23] Miller, R., Wenzel, M. a aj., B. L. *Entity Framework Core* [Microsoft Docs]. 2016. [Online; navštíveno 20.5.2020]. Dostupné z: https://docs.microsoft.com/en-us/ef/core/.

[24] Miller, R., Wenzel, M. a aj., G. S. *Entity Framework 6* [Microsoft Docs]. 2016. [Online; navštíveno 21.5.2020]. Dostupné z: https://docs.microsoft.com/en-us/ef/ef6/.

[25] Richard, S. a LePage, P. *What are Progressive Web Apps?* [Web.dev]. 2020. [Online; navštíveno 12.5.2020]. Dostupné z: https://web.dev/what-are-pwas/.

[26] Schofield, M., Santos, M. de los a aj., S. W. *Integrate your desktop app with Windows 10 and UWP* [Microsoft Docs]. 2018. [Online; navštíveno 17.5.2020]. Dostupné z: https://docs.microsoft.com/en-us/windows/apps/desktop/modernize/desktop-to-uwp-extensions.

[27] Schonning, N., Warren, G. a aj., S. C. *Introduction to WPF* [Microsoft Docs]. 2016. [Online; navštíveno 14.5.2020]. Dostupné z: https://docs.microsoft.com/en-us/visualstudio/designers/introduction-to-wpf?view=vs-2015&redirectedfrom=MSDN.

[28] Turner, R., Wojciakowski, M. a aj., C. L. *Consoles* [Microsoft Docs]. 2018. [Online; navštíveno 15.5.2020]. Dostupné z: https://docs.microsoft.com/en-us/windows/console/consoles.

[29] Warren, G., Satran, M. a aj., M. J. *XAML overview* [Microsoft Docs]. 2018. [Online; navštíveno 18.5.2020]. Dostupné z: https://docs.microsoft.com/en-us/windows/uwp/xaml-platform/xaml-overview.

[30] Wenzel, M., Dykstra, T. a aj., L. L. *.NET Core CLI overview* [Microsoft Docs]. 2020. [Online; navštíveno 15.5.2020]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/core/tools/.

[31] Wenzel, M., Latham, L. a aj., N. S. *ADO.NET Overview* [Microsoft Docs]. Květen 2020. [Online; navštíveno 14.5.2020]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ado-net-overview.

[32] Wenzel, M., Petrusha, R., Lander, R. a aj.. *.NET Core and Open-Source* [Microsoft Docs]. Březen 2017. [Online; navštíveno 12.1.2019]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/framework/get-started/net-core-and-open-source.

[33] Wenzel, M., Wagner, B. a aj., L. L. *Generics (C# Programming Guide)* [Microsoft Docs]. 2015. [Online; navštíveno 18.5.2020]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/generics/.

[34] Wenzel, M., Wagner, B. a aj., L. L. *Get started with C#* [Microsoft Docs]. 2019. [Online; navštíveno 17.5.2020]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/.

[35] Wenzel, M., Warren, G. a aj., L. L. *Windows Forms overview* [Microsoft Docs]. Květen 2017. [Online; navštíveno 15.5.2020]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/framework/winforms/windows-forms-overview.

[36] Wenzel, M., Warren, G. a aj., L. L. *.NET Framework versions and dependencies* [Microsoft Docs]. 2020. [Online; navštíveno 15.5.2020]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/framework/migration-guide/versions-and-dependencies.

[37] Whitney, T., Jacobs, M. a aj., M. S. *What's a Universal Windows Platform (UWP) app?* [Microsoft Docs]. Květen 2018. [Online; navštíveno 16.5.2020]. Dostupné z: https://docs.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide.

[38] Čápka, D. *Lekce 1 - Úvod do C# a .NET frameworku* [IT Network]. Květen 2012. [Online; navštíveno 15.5.2020]. Dostupné z: https://www.itnetwork.cz/csharp/zaklady/c-sharp-tutorial-uvod-do-jazyka-a-dot-net-framework.

[39] Šeda, J. *J2EE, .NET a vývoj rozsáhlých systémů 2.* [Inrerval.cz]. únor 2003. [Online; navštíveno 16.12.2018]. Dostupné z: https://www.interval.cz/clanky/j2ee-net-a-vyvoj-rozsahlych-systemu-2/.

Příloha A

Obsah přiloženého média

Seznam přiložených adresářů s popisem jejich obsahu.

- **\Thesis** - Zdrojové soubory bakalářská práce
- **\Export** - PDF bakalářská práce
- **\ISUF_src** - Zdrojové soubory frameworku ISUF
- **\BrnoParkingIS_src** - Zdrojové soubory Ukázkové aplikace
- **Readme.pdf** - Informace ke spuštění projektů, jejich aktualizaci, odkazy a další informace