# Zálohování do cloudových služeb

Bc. Roman Jahoda

Diplomová práce
2017



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2016/2017

# ZADÁNÍ DIPLOMOVÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

| | |
|---|---|
| Jméno a příjmení: | **Bc. Roman Jahoda** |
| Osobní číslo: | **A12749** |
| Studijní program: | **N3902 Inženýrská informatika** |
| Studijní obor: | **Informační technologie** |
| Forma studia: | **prezenční** |
| Téma práce: | **Zálohování do cloudových služeb** |
| Téma anglicky: | **Backing-up to Cloud Applications** |

Zásady pro vypracování:

1. **Seznamte se s problematikou cloudových úložišť.**
2. **Vypracujte stručný rozbor technologií, které budou použity k vývoji aplikace.**
3. **Proveďte analýzu požadavků a uživatelských cílů na zvolené řešení.**
4. **Navrhněte vhodné řešení aplikace.**
5. **Realizujte funkční prototyp navržené aplikace.**
6. **Věnujte pozornost zabezpečení aplikace.**
7. **Testujte výslednou aplikaci a vyhodnoťte ji z pohledu výkonnosti.**

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:


1. **TROELSEN, Andrew W. Pro C# 5.0 and the .NET 4.5 framework. Sixth edition. New York, 233 Spring Street: Apress, 2012, lxvii, 1487 pages. ISBN 978-143-0242-338.**
2. **NATHAN, Adam. WPF 4.5 unleashed. 800 East 96th Street, Indianapolis, Indiana 46240 USA: Sams Publishing, 2013, viii, 845 pages. Unleashed. ISBN 06-723-3697-9.**
3. **FARLEY, Marc. Rethinking enterprise storage: a hybrid cloud model. Redmond, Washington 98052-6399: Microsoft Press, 2013, pages cm. ISBN 978-073-5679-603.**
4. **DAVIES, Alex a Allen JONES. Async in C# 5.0. Sebastopol, Calif.: O'Reilly Media, c2012, x, 92 p. ISBN 14-493-3716-3.**
5. **NELSON, Steven. Pro data backup and recovery. New York: Distributed to the book trade worldwide by Springer Science+Business Media, c2011, xiii, 280 p. Expert's voice in data management. ISBN 14-302-2662-5.**
6. **FREEMAN, Eric a Elisabeth FREEMAN. Head first design patterns. 1st ed. Sebastopol: O'Reilly, 2004, xxxvi, 638 s. ISBN 05-960-0712-4.**


Vedoucí diplomové práce: **Ing. Petr Šilhavý, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání diplomové práce: **3. února 2017**

Termín odevzdání diplomové práce: **16. května 2017**

Ve Zlíně dne 3. února 2017

doc. Mgr. Milan Adámek, Ph.D.
*děkan*

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

prof. Mgr. Roman Jašek, Ph.D.
*ředitel ústavu*

Poděkování patří panu Ing. Petru Šilhavému, Ph.D, za odborné vedení diplomové práce. V neposlední řadě bych chtěl poděkovat své rodině za podporu v průběhu vypracování této práce.

**Prohlašuji, že**

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

**Prohlašuji,**

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

Ve Zlíně …….

podpis diplomanta

## ABSTRAKT

Práce obsahuje rešerši na téma zálohování do cloudových služeb. Součástí práce je analýza použitých technologií, popis návrhu a popis implementace aplikace. Výsledkem práce je funkční aplikace pro zálohování do cloudových služeb.



Klíčová slova: *záloha, cloud, analýza, .NET, MVVM, WPF, úložiště, DI*



## ABSTRACT

A thesis contains research of backup to cloud services. Part of this work is to analyze used technologies, describe application design and implementation. The result is functional cloud backup application.



Keywords: *backup, cloud, analysis, .NET, MVVM, WPF, storage, DI*

# OBSAH

# ÚVOD

Pojem cloud je v poslední době velmi moderní pojem, který zaštiťuje širokou škálu služeb a technologií. Jednou z těchto služeb je veřejné cloudové úložiště, které je určené ke sdílení, zálohování a hlavně možnosti přístupu ke svým datům prakticky odkudkoliv, kde má uživatel přístup k internetu. Konkurence v tomto odvětví je veliká a většina společnosti již nabízí široké veřejnosti tarif zdarma, který se pohybuje kolem 2GB. Díky této skutečnosti si mohou i běžní uživatelé efektivně chránit svá cenná data, aniž by museli kupovat externí disky, či jiná média.

Cílem této práce je vytvořit aplikaci pro operační systém Windows, která umožní uživateli jednoduše zálohovat data na některé z cloudových úložišť a umožnit snadnou budoucí rozšiřitelnost o novou funkcionalitu.

Teoretická část této práce obeznámí čtenáře s riziky ztráty dat, vysvětlí co to vlastně ta záloha je a jak se provádí. V další části bude vysvětlen pojem cloudové úložiště a představeny jednotlivé možnosti, které se v dnešní době nabízí.

Problematika zálohování a clodových služeb zahrnuje tak širokou škálu informací, že je nebylo možné zanést do kontextu této práce. Vzhledem k tomu se teoretická část soustředí na základní informace o dané problematice tak, aby čtenář bez předchozích znalostí pochopil praktické chování aplikace a byl ji tak schopen používat.

Součástí práce je i rozbor technologií, které byly použity k vývoji aplikace, zahrnující převážně studium komponent platformy .Net, spolu s návrhovými vzory a doporučenými postupy vývoje aplikace cílené na technologii WPF, které byly při implementaci programu prakticky použity.

Praktická část práce obsahuje stanovené cíle a požadavky na cílovou aplikaci, spolu s komplexním popisem výsledného řešení. Tento popis obsahuje rozbor modulů, ze kterých se výsledná aplikace skládá a praktické příklady zajímavých částí programu. Dále praktická část pokračuje popisem uživatelského rozhraní aplikace a vysvětlí způsob použití jednotlivých dialogů tak, aby tato část mohla sloužit jako uživatelská příručka. V poslední části se nachází informace, které vyplynuly z výsledného testování programu z hlediska výkonnosti.

# I. TEORETICKÁ ČÁST

# 1 ÚVOD DO PROBLEMATIKY

V dnešní době informačních technologii prostupuje digitalizace do našich životů čím dál tím více. Fotky, dokumenty, filmy a data obecně, jsou již dnes převážně ukládány v elektronické podobě. Bohužel je zde celá škála situací, které ohrožují naše data.

- Technické selhání: Porucha úložného zařízení, na kterém jsou data umístěna.
- Viry: Škodlivé programy, které mohou způsobit poškození, nebo ztrátu dat.
- Chyba programového vybavení: Chyba v programu může poškodit data, s kterými manipuluje.
- Chyba uživatele: Uživatel nechtěně zaviní ztrátu, nebo poškození dat. Vztahuje se sem i poškození, nebo ztráta samotného zařízení. [1]

Pro tyto případy je data potřeba zálohovat na jiné úložné zařízení, než na kterém se data nachází. Dříve se i drobná data musela zálohovat na externí pevné disky, flash disky, nebo DVD disky, avšak dnes jsou možnosti podstatně flexibilnější díky cloudovým úložištím. [2]

## 1.1 Zálohování dat

Zálohování je proces, pří kterém se vytváří kopie otisku dat v určitý moment. Provádí se za účelem ochrany vybraných dat před jejich ztrátou. Tento otisk je poté uchováván po určitou dobu užitečnosti.[3]

Nejčastější příčinou ztráty dat je selhání lidského faktoru, kdy se uživatel dopustí nevědomky chyby, která zaviní poškození dat. Data ovšem mohou být poškozena i průnikem zvenčí, který se v mnoha případech dá také zařadit do selhání lidského faktoru.[1] Jako příklad se může uvést kryptovirus, který si uživatel zanese do počítače nedbalým otevřením přílohy emailu. Po inicializaci kryptovirus zašifruje vybraná data a zanechá textový soubor, který uživatele vydírá k zaplacení určité částky.[4]

### 1.1.1 Typy záloh

Pro zálohování existuje několik různých metod. Pro zvolení té správné se je potřeba zaměřit na to, jak často potřebujeme zálohovat a jakou dostupnou kapacitu máme k dispozici. Stejně tak je potřeba zvážit, jak rychle je nutné data obnovit v případě ztráty a poškození. Zvolený typ zálohy spolu s vybraným médiem kam se záloha bude ukládat ovlivňuje rychlost zálohy, stejně jako obnovy, protože některé typy záloh jsou závislé na předchozích iteracích záloh.[2][3]

V praxi je možné navrhnout různé typy záloh a jejich kombinace. V další části bude popis typů záloh jen obecného charakteru, protože konkrétní implementace se může lišit operačními systémy, které zálohování nabízejí, ale také i speciálními zálohovacími aplikacemi, které jsou určeny pro zálohu uživatelských, či firemních dat.[6]

#### *1.1.1.1 Úplná záloha*

Tento typ zálohy označuje zálohu všech zvolených dat. Může se jednat o celý obraz disku, včetně operačního systému, nebo jen o určitou adresářovou strukturu, který obsahuje cenná data. Tento typ zálohy je časově a technicky nejnáročnějším řešením pro zálohu dat, které v sobě nese nevýhodu v podobě časové náročnosti vytvoření zálohy a velkou náročnost na dostupnou kapacitu. Proto se tento typ zálohy provádí ve firemní praxi s většími časovými rozestupy, než u jiných typů záloh. [3]

#### *1.1.1.2 Diferenciální záloha*

Do diferenciální zálohy se zálohují data, která se změnila od poslední úplné zálohy. Pokud úplná záloha ještě neexistuje, je potřeba ji udělat. V případě ztráty dat je pro obnovení nutné mít k dispozici předešlou úplnou zálohu a požadovanou zálohu diferenciální. Výhoda této metody je rychlá obnova a menší závislost na předchozích zálohách, než u zálohy inkrementální, která bude popsána níže. Případné poškození některé z diferenciálních záloh neovlivní jiné diferenciální zálohy. Diferenciální záloha se dá považovat za kompromis mezi zálohou úplnou a zálohou inkrementální.

Nevýhodou tohoto typu zálohy je postupem času narůstající velikost diferenciální zálohy od zálohy úplné, kde se můžeme dostat do stavu, kdy už je diferenciální záloha větší než poslední úplná záloha. Z tohoto důvodu je potřeba vytvářet pravidelně i zálohy úplné. [3][6]

#### *1.1.1.3 Inkrementální záloha*

Tato metoda zálohováni je stejně jako diferenciální záloha závislá na záloze úplné, ale navíc je závislá i na všech předchozích inkrementálních zálohách. Tento typ nabízí nejrychlejší metodu zálohování a nejnižší nároky na velikost úložného prostoru, avšak na úkor rychlosti obnovy a závislosti na předchozích zálohách. V případě obnovy je potřeba obnovit všechny zálohy od úplné, až do požadované zálohy inkrementální. Tento způsob obnovy má nevýhodu v tom, že v případě poškození některé ze zálohy, jsou všechny inkrementální zálohy v řetězci za ní neplatné. Z tohoto důvodu je, stejně jako u záloh diferenciálních nezbytné dělat pravidelně zálohy úplné. [3]

## 1.2 Cloudová úložiště

Cloudové úložiště je druh cloudové služby, ve které jsou vzdáleně spravovány data tak, aby byly přístupné uživatelům přes nějakou síť, typicky internet. Ve výsledku to znamená, že svá data uložíme na server, který je dostupný z internetu. Obecně se cloudové služby dají dělit na 3 hlavní modely nasazení, které budou popsány v následujících kapitolách. [5]

### 1.2.1 Veřejný cloud

Veřejná cloudová úložiště poskytují takzvané multi-tenant úložiště, které je vhodné pro nestrukturované data. Multi-tenant je architektura, kde je jedna instance softwaru, která obsluhuje vícero uživatelů, avšak zajišťuje jejich vzájemné oddělení, aby si uživatelé nemohli navzájem přistupovat k soukromým informacím a prostředkům. Tato architektura je výhodná hlavně v tom, že náklady na vývoj a údržbu jsou sdílené. V cloud computingu je tento výraz ještě širší vzhledem k tomu, že nové modely služeb využívají virtualizaci a vzdálený přístup. [5]

### 1.2.2 Privátní cloud

Privátní cloudové úložiště je dedikované prostředí, které je schováno za firemním firewallem. Tento druh je vhodný hlavně pro klienty, kteří požadují větší kontrolu a flexibilitu nad svými daty. [5]

### 1.2.3 Hybridní cloud

Tento typ je kombinace veřejného a privátního cloudu, který je používán primárně ve firemním prostředí. V praxi to znamená, že veřejná a privátní část je propojena, zpravidla dedikovanou linkou. Ve veřejné části se nachází data, které jsou pro firmu méně důležitá. V privátní části se uchovávají kritická a často používaná data. Pro funkčnost tohoto typu cloudu je potřeba hybridní prostředí, které nabízí například firma Microsoft se svým produktem Microsoft Azure. Mezi další společnosti, které nabízí tuto technologii patří například HP, Rackspace, VMware a další. [7]

# 2 ANALÝZA TECHNOLOGIÍ

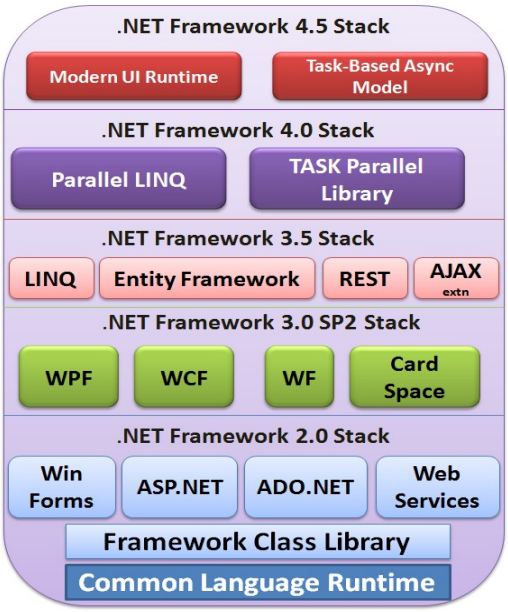
## 2.1 .NET Framework

.NET Framework je softwarová platforma, která je určena pro vývoj a běh různých druhů aplikací. Pomocí této platformy můžeme vyvíjet nejen klasické webové a desktopové aplikace pro Windows, ale i aplikace pro mobilní telefony a služby.

Hlavní částí frameworku je rozsáhlá knihovna tříd a Common Language Runtime, což je běhové prostředí a praktická implementace standardu CLI, který popisuje sadu postupů a specifikací jak implementovat běhové prostředí a náturu generovaného kódu tak, aby umožňoval použití pro více programovacích jazyků na mnoha různých platformách bez nutnosti přepisovat naprogramované aplikace pro různé architektury. Jako další příklad praktické implementace CLI lze uvést Mono CLR, který spadá do opensourcové multiplatformní vývojové platformy Mono.

Výhodou vývoje v .Net Frameworku je rozsáhlost knihoven, relativní jednoduchost a efektivita vývoje. Díky rozsáhlému použití a provázání této platformy s komerční praxí je k dispozici propracovaná dokumentace, spolu s neustálým vývojojem nových verzí této platformy.

Nevýhodou této platformy je slabší výkon oproti nativnímu kódu, který je způsoben správou paměti pomocí Garbage Collectoru a tím, že kompilátor překládá výslednou aplikaci do mezikódu (MSIL). Při následném spuštění aplikace výše zmíněný CLR, mezikód interpretuje do binární podoby za běhu aplikace.[8][10]




Obrázek 1 Diagram .Net Frameworku [12]

## 2.2 Windows Presentation Framework

Windows Presentation Framework (WPF) je grafický subsystém .Net Frameworku, který umožňuje programátorovi relativně snadno vytvářet uživatelský interface pro aplikace směřované na operační systém Windows. Mezi hlavní výhody tohoto frameworku patří možnost snadno oddělit definici prezentační logiky od aplikační logiky. Takovéto oddělení má výhodu v tom, že při změnách v aplikační logice se nemusí přepisovat GUI za předpokladu, že interface zůstal nezměněn. Další předností je možnost paralelní práce návrhářů a vývojářů aplikace, tak aby si nezasahovali vzájemně do práce.

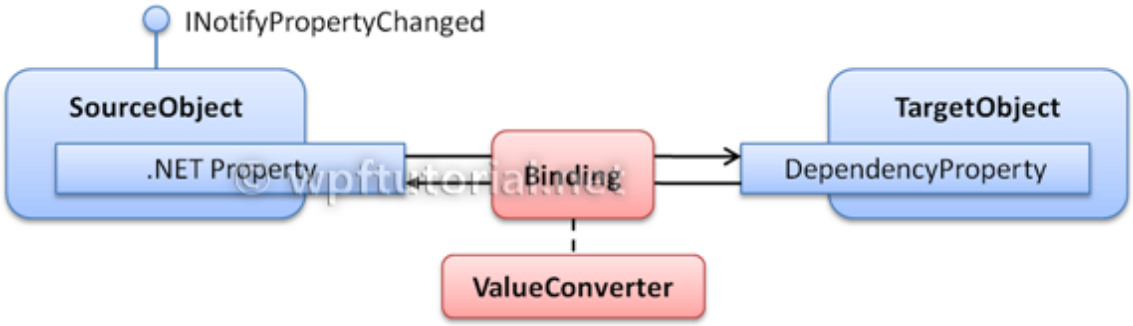
GUI se definuje pomocí značkovacího jazyka XAML, který je založený na XML. Pomocí tohoto jazyka definujeme nejen uživatelské komponenty, ale také jejich styl a chování. Při vývoji ve WPF se doporučuje používat návrhový vzor MVVM, který bude popsán níže.

Součástí WPF je také DataBinding, což je mechanismus, který umožnuje synchronizaci dat mezi datovým zdrojem a GUI. [8][9]

### 2.2.1 DataBinding

Jak již bylo řečeno DataBinding je mechanismus, který umožnuje synchronizovat data mezi GUI (View) a datovým modelem (ViewModel, Model). Předpoklad pro úspěšný DataBinding je ten, že je cílový objekt nastaven jako DataContext komponenty a cílová

vlastnost ve View musí být DependencyProperty. Ve WPF můžeme tento binding specifikovat na cílovou vlastnost vizuálního prvku pomocí syntaxe {Binding}. Tímto způsobem můžeme provést vazbu na jakýkoliv objekt a pokud DataContext implementuje INotifyPropertyChanged, je komponenta notifikována o změnách dat, takže se komponenta dozví, že má například překresit text "Petr" na "Pavel". V případě potřeby je možné nadefinovat konvertory, které mohou převádět data na jiný datový typ, popřípadě formát. Tyto konvertory musí implementovat interface IValueConverter, nebo IMultiValueConverter. Díky těmto konvertorům si můžeme snadno přizpůsobovat data pro potřeby View, aniž by jsme museli dopisovat funkcionalitu do ViewModelu, nebo View. Například, když zdrojový datový objekt obsahuje čas v Unixovém formátu, ale ve View by jsme ho chtěli zobrazit jako klasické datum, tak si vytvoříme konvertor, který dokáže data mezi těmito formáty převádět.[8][9][11]

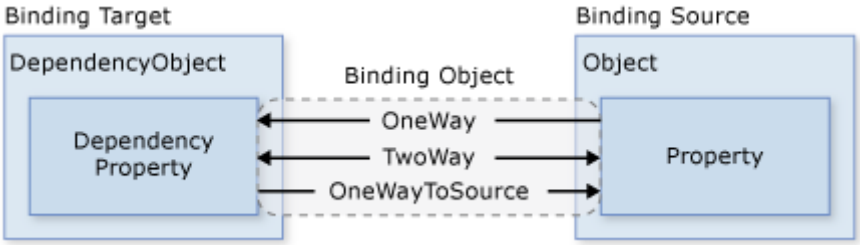



Obrázek 2 Schéma DataBindingu [11]

DataBinding má mnoho parametrů, které ovlivňují jeho chování. Mezi nejdůležitější patří parametr Mode, který určuje směr synchronizace informací. Synchonizační módy jsou následující:

- OneWay- Synchronizace probíhá pouze z datového modelu do View. Tento mód se používá u statických komponent, které pouze zobrazují data, která se můžou na pozadí měnit.
- TwoWay- Obousměrná synchronizace, která se používá u komponent se kterými uživatel nějakým způsobem interaguje a také se stav může změnit na základě nějaké jiné události. Jako příklad si uživatel musí zvolit typ platby.
- OneTime- Stejné chování jako u OneWay, ale synchronizace proběhne jen jednou na začátku. Měl by se používat jako výchozí mód pro data, která se nemění.

- OneWayToSource- Protiklad k OneWay. Pouze View synchronizuje data do datového modelu. [8]




Obrázek 3 Schéma různých typů DataBindingu [8]

### 2.2.2 MVVM

Model-View-ViewModel je návrhový vzor, který má za cíl usnadnit vývojářům rozdělení aplikační vrstvy od prezentační. Při správném použití se i při velmi rozsáhlé aplikací dájí velmi snadno rozšiřovat, modifikovat, či dokonce znovupoužít jednotlivé view.

Jak již z názvu vyplývá, tento vzor se skládá z několika komponent, které jsou od sebe odděleny a proto umožňují těžit z následujících výhod:

- Změna interní implementace komponenty, aniž by ovlivnila jiné.
- Komponenty jsou na sobě nezávislé.
- Vývojáři mohou na komponentách pracovat paralelně za předpokladu, že je dopředu známý interface komponent.
- Při vytváření unit testů se dají snadno izolovat komponenty.
- Možnost vytvořit více View pro jeden ViewModel a dle potřeby je zaměňovat bez zásahu do ViewModelu.[8]

#### *2.2.2.1 Model*

Model reprezentuje data, se kterými pracujeme. Klíčovým prvek je ten, že model by měl obsahovat pouze informace, ale ne chování a služby, které s těmito informacemi manipulují za účelem zobrazení. Například by model neměl být zodpovědný za formátování textu tak, aby na obrazovce vypadal dobře. [8]

#### *2.2.2.2 ViewModel*

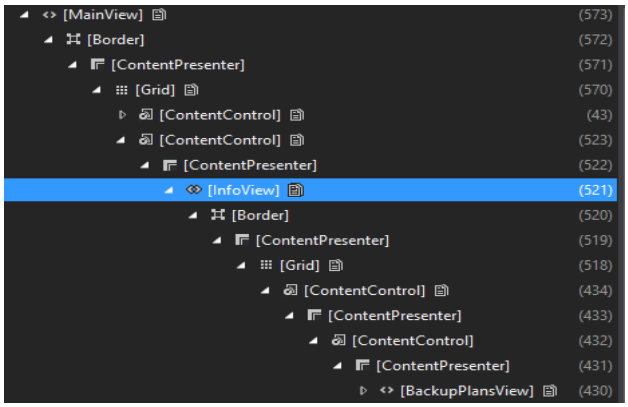
ViewModel slouží jako prostředník mezi View a Modelem. Jeho hlavní zodpovědnost leží v poskytnutí potřebných dat z modelu do View. Ve většině případů ViewModel

implementuje interface IPropertyChanged, který obsahuje pouze jediný prvek, a to event PropertyChanged, pomocí kterého může View notifikovat o změnách dat, na které View může zareagovat překreslením informace na aktuální hodnotu. V některých případech může ViewModel poskytnout View i samotný Model, za předpokladu, že model implementuje interface IPropertyChanged, nebo se data v Modelu již měnit nebudou.

ViewModel se také stará o obsluhu událostí z View, které by se měly správně propagovat přes interface ICommand, který můžeme stejně jako vlastnost DataContext nabindovat a v případě potřeby i nabindovat parametr ve View, například objekt kontaktu v seznamu na který uživatel vykonal dvojklik. [8]

#### *2.2.2.3 View*

View je odpovědný za definici vzhledu, struktury a rozložení toho, co se uživateli zobrazuje na obrazovce. V ideálním případě je celý View definovaný v Xamlu, s limitovaným kódem na pozadí, který neobsahuje bussines logiku aplikace. View si můžeme představit jako stránku, ve stromové struktuře, která v sobě může obsahovat několik dalších stránek. V praxi se View většinou podědí od třídy UserControl a poté nadefinuje vizuální struktura v XAMLu.




Obrázek 4 Ukázka vizuální stromové struktury

Jak již bylo zmíněno, data se do View primárně synchronizují pomocí DataBindingu a pokud je mód synchronizace nastaven jako OneWay, nebo TwoWay, tak se ještě odposlouchává událost PropertyChanged, kterou by DataContext View měl vyvolat při každé změně bindovaných dat. View tyto data aktualizuje automaticky pomocí reflexe v závislosti na nastaveném módu.[8]

# II. PRAKTICKÁ ČÁST

# 3 ANALÝZA POŽADAVKŮ A UŽIVATELSKÝCH CÍLŮ NA ZVOLENÉ ŘEŠENÍ

Tato kapitola se věnuje specifikaci požadavků aplikace. V zásadě se jedná o stručné analytické rozpracování požadavků, cílů a kritérií úspěšnosti implementace prototypu aplikace.

## 3.1 Cíle projektu

Hlavním cílem tohoto projektu je vyvinout aplikaci, které umožní uživateli zálohovat vybraná data na různá úložiště, bez ohledu na to, jestli je umístěno na cloudu, lokálním úložišti, nebo úložišti jiného druhu.

Toto zadání můžeme rozložit na několik dílčích cílů:

- Implementovat proces zálohování.
- Implementovat rozhraní tak, aby se daly snadno doprogramovat další typy úložišť.
- Implementovat logiku aplikace tak, aby nezatěžovala systém.
- Implementovat rozhraní pro uživatele.
- Implementovat stabilní a spolehlivou aplikaci.

Hlavní cílová skupina, která bude tuto aplikaci používat, je uživatel bez větších technických znalostí a proto by rozhraní aplikace mělo být co nejvíce intuitivní vzhledem k požadavkům, které by uživatel od aplikace očekával.

## 3.2 Kritéria úspěšnosti

V rámci základní specifikace projektu je vhodné stanovit také kritéria úspěšnosti celého projektu. Tato kritéria neslouží pouze pro samotné zhodnocení výsledného řešení projektu, ale slouží i jako významná podpůrná informace pro praktický návrh a implementaci projektu, jelikož je z nich možné pochopit, jak uživatel nahlíží na celý projekt a jaké funkční požadavky jsou pro něj více, či méně důležité.

Základní kritéria úspěšnosti projektu jsou stanovena následovně:

- Aplikace by měla běžet na pozadí a nevyrušovat uživatele pokud to nebude nutné pro úspěšné provedení zálohy.
- Aplikace by měla být stabilní a uživatel by měl mít možnost dohledat chybové stavy, které aplikace nemůže ovlivnit (výpadek internetu při běhu zálohy atd.).

- Uživatelský interface by měl být jednoduchý na pochopení a umožnit uživateli snadnou navigaci.
- Propojení úložišť, by mělo být zpracováno ve formě pluginů, tak aby se daly snadno dopisovat další typy možných úložišť.
- Vzhledem k tomu, že by aplikace měla jít spustit i na slabších počítačích, neměla by příliš zatěžovat systémové prostředky.

# 4 SOFTWAROVÉ ŘEŠENÍ

## 4.1 Popis struktury aplikace

Aplikace je napsána modulárně a skládá se z mnoha projektů. Pro tento modulární přístup byl použit framework PRISM, který k tomuto přístupu tvorby aplikace přímo nabádá. Hlavní částí je projekt CloudBackupManager.Shell, který obsahuje hlavní okno aplikace a třídu Bootstraper, jejíž instance se vytvoří ihned po startu aplikace. Bootstraper inicializuje základní služby a části, jako je UnityContainer, RegionManager a EventAggregator, spolu se všemi moduly moduly, které implementují interface IModule a jsou zadefinované v katalogu modulů.

Aplikace se konkrétně skládá z následujících modulů:

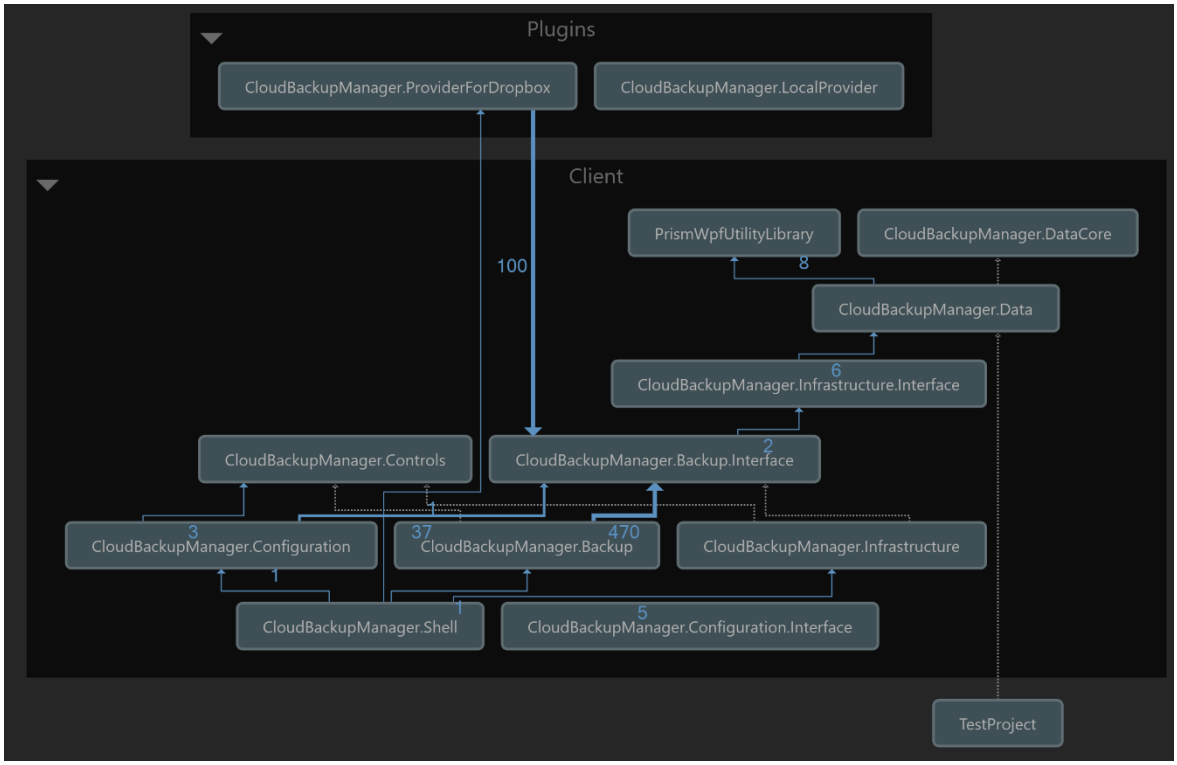
- CloudBackupManager.Backup
- CloudBackupManager.Configuration
- CloudBackupManager.Controls
- CloudBackupManager.Data
- CloudBackupManager.DataCore
- CloudBackupManager.Infrastructure
- CloudBackupManager.Shell
- CloudBackupManager.ProviderForDropbox

Součástí jsou i pomocné knihovny .Interface, ve kterých se nachází pouze rozhraní a třídy potřebné pro práci s danými rozhraními modulů.

- CloudBackupManager.Backup.Interface
- CloudBackupManager.Configuration.Interface
- CloudBackupManager.Infrastructure.Interface

Výhodou takového rozdělení je:

- Obrana proti cyklickým referencím.
- Moduly jsou na sobě navzájem nezávislé.
- Možnost kompletní záměny modulů, za předpokladu, že budou správně implementovány patřičné rozhraní a načteny všechny potřebné služby do DI kontejneru.




Obrázek 5 Diagram vztahů projektů (modulů)

### 4.1.1 Projekt CloudBackupManager.Shell

V tomto projektu se nachází hlavní okno aplikace a třída Bootstraper, který se stará o inicilizaci všech potřebných modulů a služeb aplikace.

### 4.1.2 Projekty CloudBackupManager.DataCore a CloudBackupManager.Data

Projekt CloudBackupManager.DataCore obsahuje třídy potřebné pro správu databáze aplikace. Projekt s databází pracuje pomocí Entity Frameworku, což je takzvané ORM (objektově relační mapování). Pro databázi byla zvolena databáze SQL Server Compact, která pro svůj chod nepotřebuje na cílovém počítači nic instalovat, potřebné knihovny pro práci s touto databází si aplikace nese s sebou.

Struktura databáze se vytváří přístupem Code First, kdy se nejdříve nadefinují datové třídy a poté se pomocí speciálního příkazu vytvoří migrace databáze. Při každé migraci se zanesou rozdíly od posledního stavu. Díky tomuto přístupu si aplikace dokáže sama aktualizovat starší databázi na novější verzi, což umožnuje snadnou distribuci novějších verzí bez obavy ztráty, nebo nekompatibility existujících dat. Ostatní moduly mohou poté s touto databází komunikovat přes třídu CloudBackupManagerContext, kterou tento projekt obsahuje.

SQL Server Compact má pouze limitovaný výběr typů, kterým se mohou reprezentovat data. Proto se v aplikaci datové objekty převádí do typů, které jsou obsaženy v CloudBackupManager.Data a odpovídají více realitě.

```
namespace CloudBackupManager.DataCore.Models
{
    public class BackupPlan
    {
        [Key]
        public int Id { get; set; }
        public string Name { get; set; }
        public int? TimePlanId { get; set; }
        public bool Enabled { get; set; }
        [ForeignKey("TimePlanId")]
        public TimePlan TimePlan { get; set; }
        public int BackupPlanType { get; set; }
        public byte[] SourcePaths { get; set; }
        public byte[] TargetPaths { get; set; }
        public int? BackupChainLength { get; set; }


    }
}

namespace CloudBackupManeger.Data.DataObjects
{
    [Serializable]
    public class BackupPlan : ICopyable<BackupPlan>
    {
        public BackupPlanType BackupPlanType { get; set; }
        public TimePlan TimePlan { get; set; }
        public int Id { get; set; }
        public string Name { get; set; }
        public List<PathElement> SourcePaths { get; set; }
        public List<PathElement> TargetPaths { get; set; }
        public bool Enabled { get; set; }
        public int? BackupChainLength { get; set; }
     . . .
    }
}
```

i

### 4.1.3 Projekt CloudBackupManager.Infrastructure

V tomto projektu se nachází základní služby pro práci s menu a zálohovacími plány. Struktura třídy, která reprezentuje zálohovací plán, je uvedena výše. Tato třída také obsahuje časový plán, který v sobě nese všechny potřebné informace o periodě opakování zálohy. Správu zálohovacích plánů má na starosti singleton, služba BackupPlanService, kterou si mohou ostatní moduly vyžádat přes Unity IoC Container, do kterého je služba při inicializaci modulu vložena pod interfacem IBackupPlanService.

```
    public interface IBackupPlanService : INotifyPropertyChanged
    {
        IEnumerable<BackupPlan> GetBackupPlans();
```

```
        BackupPlan GetBackupPlan(int id);
        bool EditBackupPlan(BackupPlan backupPlan);
        bool DeleteBackupPlan(BackupPlan backupPlan);
        event PropertyChangedEventHandler PropertyChanged;
        BackupPlan SelectedBackupPlan { get; set; }
        BackupPlan GetNewBackupPlan();
        event EventHandler BackupPlansChanged;
    }
```

### 4.1.4 Projekt CloudBackupManager.Backup

V tomto projektu se nachází služby, které mají na starosti správu zálohovacích providerů a samotný proces zálohování. Služba BackupProviderManager spravuje zálohovací providery, které mohou pluginy za běhu aplikace vkládat.

Služba BackupService obsahuje logiku pro samotné zálohování a ukládání metadat o zpracovaných operacích. Stejně jako u ostatních služeb je dostupná pro všechny moduly přes IoC Container pod interfacem IBackupService. Služba v krátkých časových intervalech kontroluje všechny aktivní zálohovací plány a kontroluje jejich časové nastavení. Pokud je čas pro zálohu, služba zařadí zálohovací plán do fronty, která se postupně odbavuje.

```
    public interface IBackupService
    {
        IReadOnlyNotifyCollection<OperationTaskSource> BackupQueue { get; }
        void AddToQueue(int backupPlanId);
        bool EditBackup(CloudBackupManeger.Data.DataObjects.Backup backup);
        bool DeleteBackup(CloudBackupManeger.Data.DataObjects.Backup backup);
        IEnumerable<CloudBackupManeger.Data.DataObjects.Backup> GetBackups();
        IEnumerable<CloudBackupManeger.Data.DataObjects.Backup> GetBackups(int
backupPlanId);
        CloudBackupManeger.Data.DataObjects.Backup GetNewBackup();
        DateTime? CalculateNextBackupTime(TimePlan timePlan);
        void AddToRestoreQueue(int backupId);
    }
```

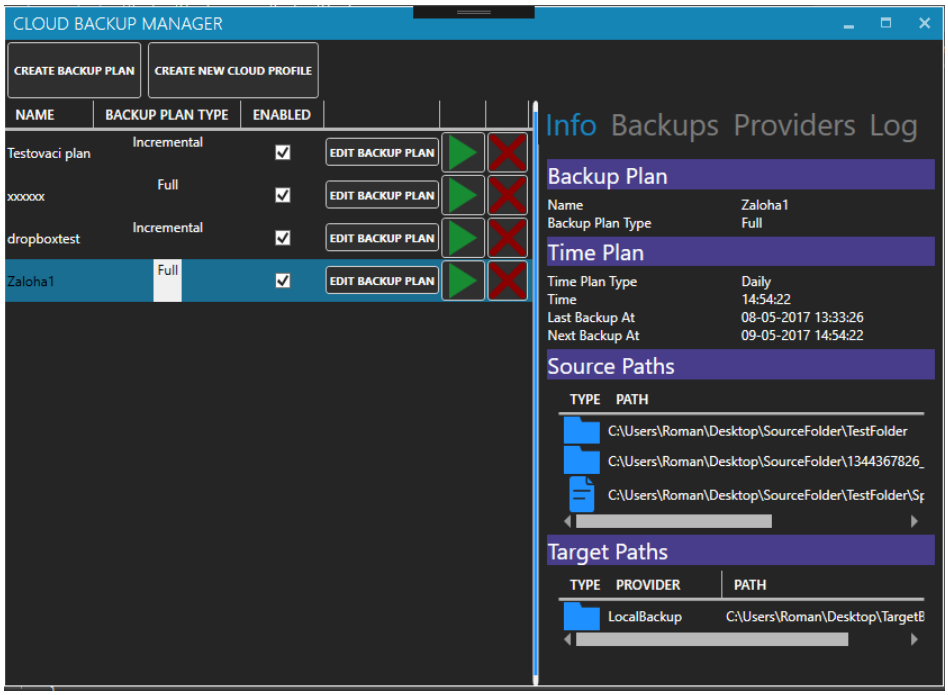
### 4.1.5 Projekt CloudBackupManager.ProviderForDropbox

Tento projekt slouží jako plugin, který se inicializuje při startu aplikace. Obsahuje inicializaci tříd, které jsou nutné pro komunikaci s cloudovým úložištěm DropBox.

# 5 POPIS APLIKACE

V rámci této sekce bude popsáno uživatelské rozhraní s jejich jednotlivými náhledy. Budou také popsány způsoby implementace jednotlivých prvků a příklady užití. V následujících kapitolách budou popsány konkrétně tyto View:

1. BackupPlanDetailView – Detail zálohovacího plánu.
2. MainView – View, které je umístěné v rootu okna aplikace a skládá se z několika dalších View.
3. NewProviderDialogView – View pro přidání nového zálohovacího providera.




Obrázek 6 Hlavní okno aplikace

## 5.1 BackupPlanDetailView

Tento dialog slouží pro editaci zálohovacího plánu. Při kliknutí na volbu Create Backup Plan se vytvoří nový zálohovací plán, který se nachází pouze v paměti do doby, než uživatel klikne na tlačítko save, čímž se záznam uloží do databáze a stává se platným. Při editaci již existujícího zálohovacího plánu se položka v databázi přepíše.

Dialog obsahuje několik záložek:

1. Backup plan settings
2. Backup paths settings
3. Time settings

### 5.1.1 Backup Plan Settings

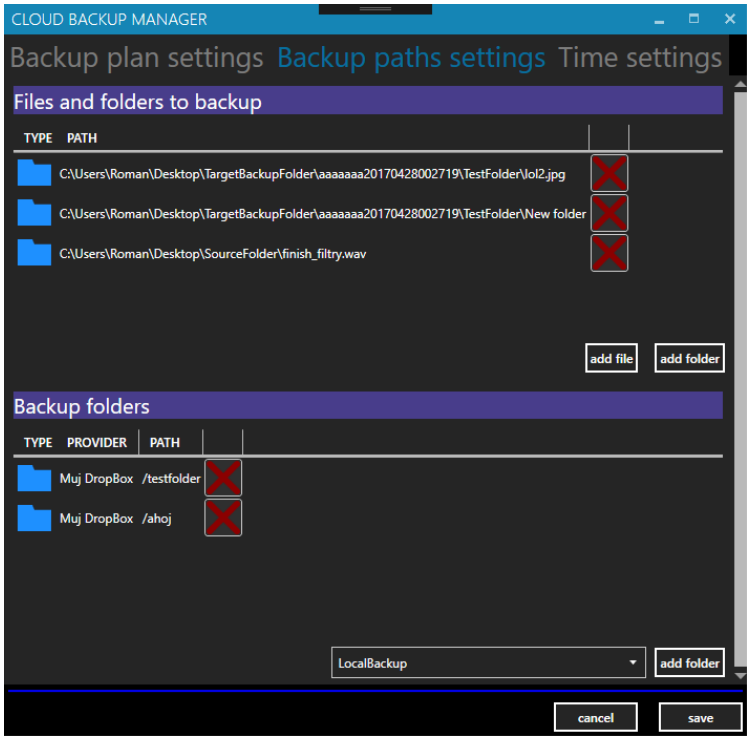
V této záložce se nachází základní volby pro zálohovací plán, kde si uživatel může nastavit základní chování. Všechna možná nastavení této záložky budou popsána v následující tabulce.

Tabulka 1 Popis nastavení záložky Backup plan settings

| Název | Typ kontrolky | Volba | Popis |
|---|---|---|---|
| **Name** | TextBox | | Zde si uživatel libovolně pojmenuje zálohovací plán. |
| **Backup Type** | ComboBox | | Možnost volby typu zálohování. |
| | | Full | Úplná záloha |
| | | Incremental | Inkrementální záloha |
| | | Differential | Diferenciální záloha |
| **Backup chain length** | NumericUpDown | | Nastavení je možné zvolit pouze, pokud je typ zálohovacího plánu nastavený na Incremental, nebo Differential. Určuje po jakém počtu záloh se vytvoří záloha úplná. |
| **Status** | ToggleButton | Enabled/Disabled | Možnost aktivace a deaktivace plánu. U deaktivovaného plánu je možné provést zálohu pouze manuálně. |

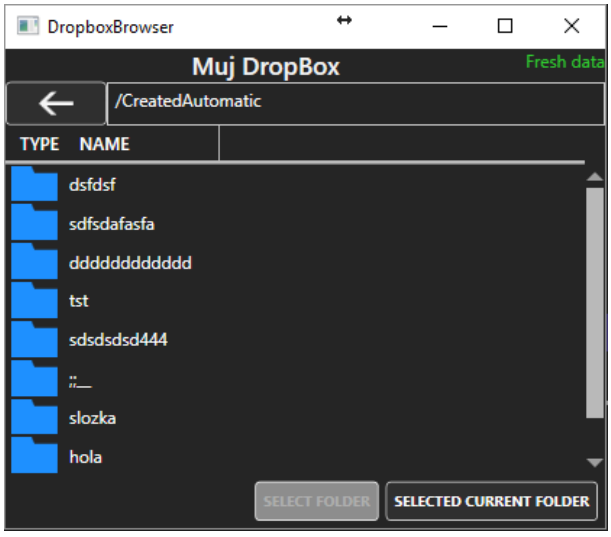
### 5.1.2 Backup paths settings

V této záložce se nachází 2 sekce. V první sekci „Files and folder to backup“ si uživatel zvolí soubory a složky, které chce zálohovat. Uživatel může zálohovat pouze data, ke kterým může přistoupit přes operační systém. V další sekci „Backup folders“, si může uživatel zvolit, kam se budou zálohovaná data kopírovat.




Obrázek 7 Backup path settings

Pro cílové destinace si musí uživatel zvolit takzvaného poskytovatele služby(providera), z comboboxu, který se nachází vedle tlačítka add folder. Po volbě providera a kliknutím na tlačítko add folder, se uživateli zvolí patřičný dialog pro výběr složky. Dialogy se mohou lišit v závislosti na tom, jaký provider je zvolen. Při výběru lokálního providera se například zobrazí nativní dialog pro výběr složky. Při výběru providera pro DropBox se zobrazí dialog, který byl navržen, a implementován speciálně pro službu DropBox.




Obrázek 8 Dialog pro výběr složky v cloudovém úložišti DropBox.

### 5.1.3 Time Settings

Každý zálohovací plán má na sebe navázaný časový plán, který reprezentuje způsob periodického zálohování. Ze všeho nejdříve si musí uživatel zvolit typ požadovaného zálohovacího plánu, který zpřístupní ostatní volby, které jsou závislé na zvoleném typu. Jediný typ, který nemá dodatečné volby je Manuální typ, kde se záloha provádí na podnět uživatele, z přehledu zálohovacích plánů. Veškeré volby u jednotlivých typů budou popsány v následující tabulce.

| Typ časového plánu | Volba | Popis |
|---|---|---|
| **Daily** | | Denní časový plán. |
| | Time | Čas, ve kterém se bude záloha opakovat. |
| **Weekly** | | Týdenní časový plán. |
| | Time | Čas, ve kterém se bude záloha opakovat. |
| | Days in week | Dny v týdnu, kdy se bude záloha provádět. |
| **Monthly** | | Měsíční zálohovací plán. |
| | Time | Čas, ve kterém se bude záloha opakovat. |
| | Days in month | Dny v měsíci, kdy se bude záloha provádět. Vzhledem k tomu, že jednotlivé měsíce mají různou délku, tak je dostupná i volba Last, pro poslední den v jakémkoliv měsíci. |
| **Yearly** | | Roční zálohovací plán |
| | Time | Čas, ve kterém se bude záloha opakovat. |
| | Months in year | Měsíce v roce, kdy se bude záloha provádět. |
| | Days in month | Dny v měsíci, kdy se bude záloha provádět. |
| **Manual** | | Manuální časový plán, uživatel si musí manuálně vyžádat zálohu v programu, přes tlačítko v přehledu zálohovacích plánů. |

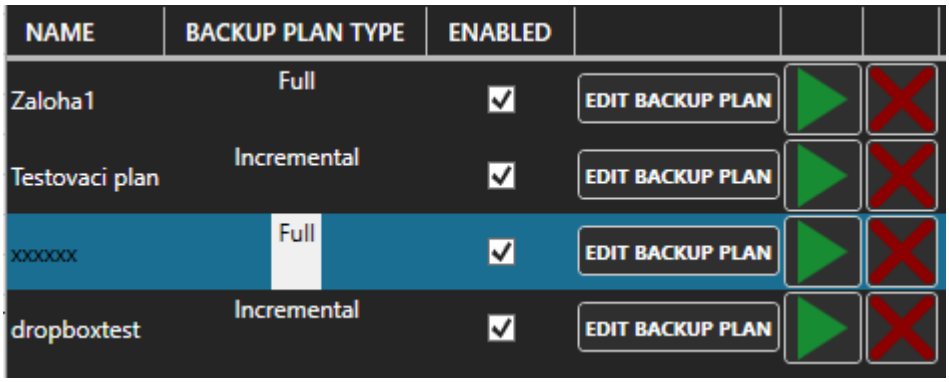
## 5.2 MainView

Toto view se skládá z menu a dvou sekcí. Menu obsahuje dvě tlačítka.

1. Create Backup Plan – Vyvolá dialog pro vytvoření nového zálohovacího plánu.
2. Create New Cloud Profile – Vyvolá dialog pro vytvoření nového cloudového profilu.

V levé části se nachází BackupPlansView, které obsahuje seznam všech vytvořených zálohovacích plánů. Řádek každého zálohovacího plánu obsahuje základní informace a tři tlačítka.

1. Edit Backup Plan – Otevře dialog pro editaci zvoleného plánu.
2. Force Backup – Zařadí zvolený plán do fronty záloh. Toto tlačítko vynutí zálohu, v případě potřeby a je to jediný způsob jak spustit zálohovací plán, který má nastavený manuální typ časového plánu.
3. Delete Backup – Slouží pro odstranění zálohovacího plánu. Před samotným smazáním se ještě dotáže, zda uživatel chce akci opravdu provést.

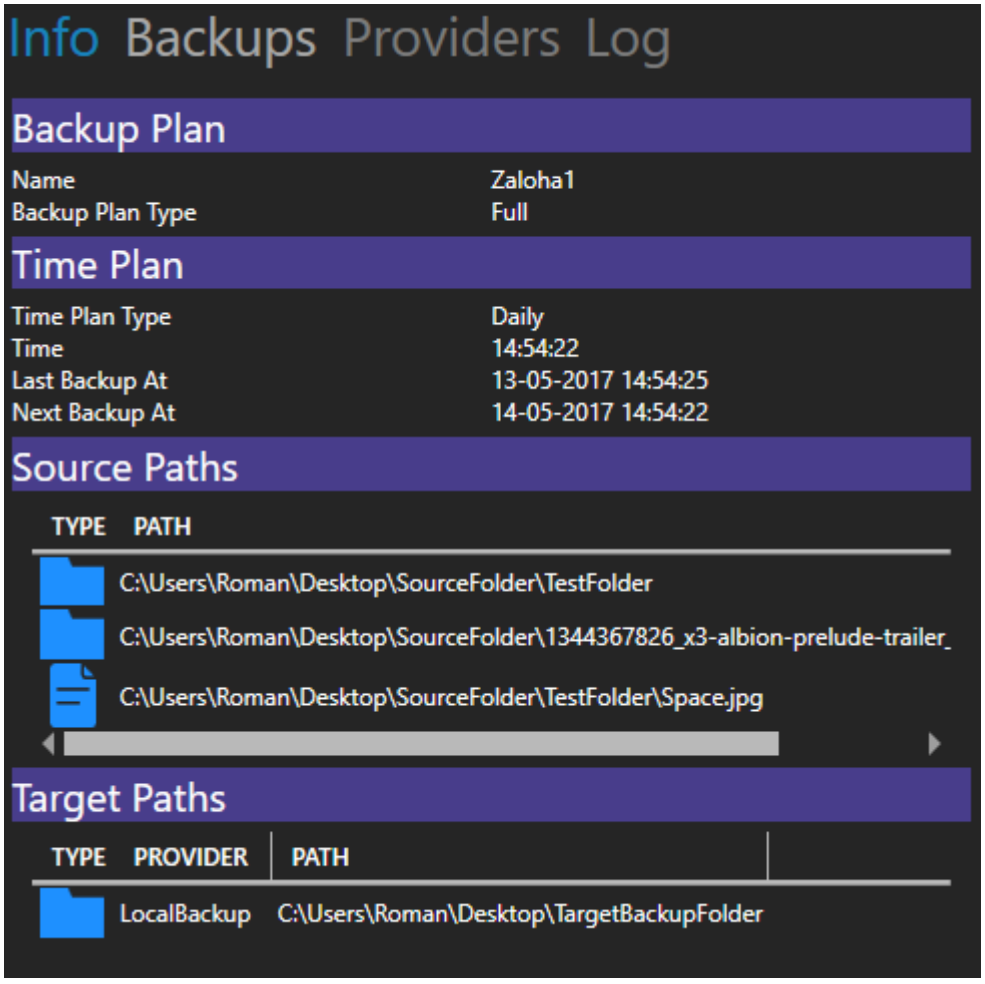



Obrázek 9 BackupPlansView

V pravé části je umístěn TabControl se čtyřmi záložkami, které budou popsány v následujících sekcích.

### 5.2.1 Záložka Info

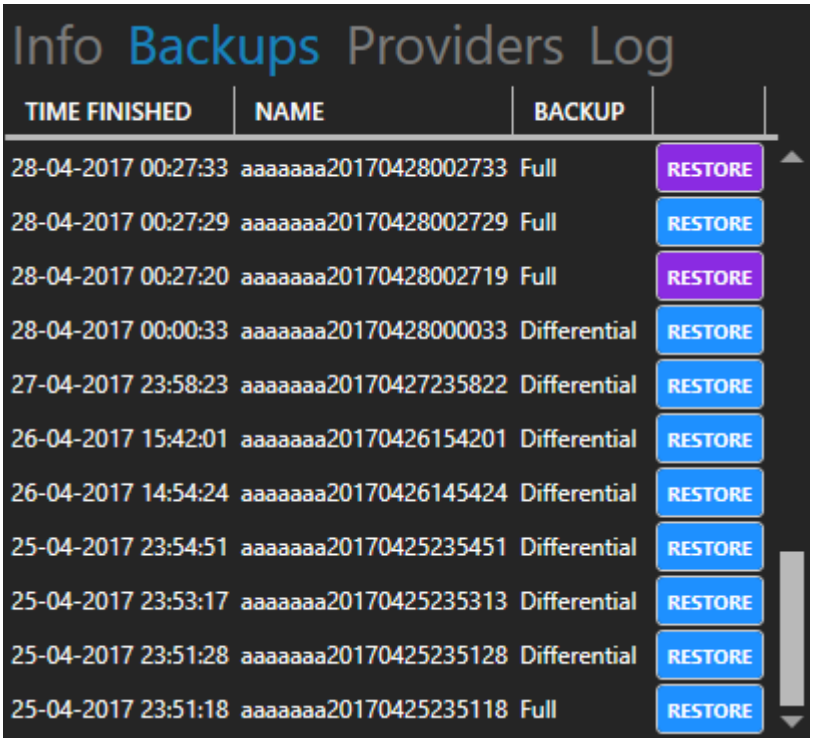
V této záložce se nachází podrobné informace pro zvolený zálohovací plán. Najdeme zde i informaci o tom, kdy se pro zálohovací plán spustí další záloha. Pokud v této době aplikace neběží, tak se zálohovací plán zařadí do zálohovací fronty ihned při příštím spuštění aplikace.




Obrázek 10 Záložka info

### 5.2.2 Záložka Backups

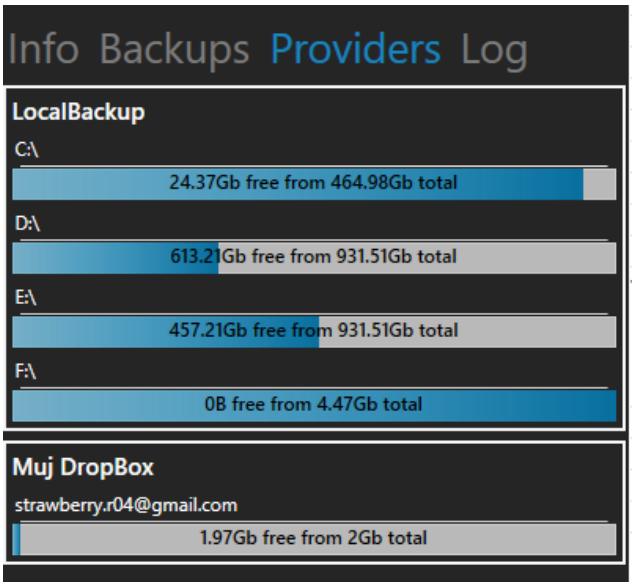
Nachází se zde všechny zálohy pro zvolený zálohovací plán. Tyto zálohy jsou seřazené chronologicky, aby uživatel mohl snadno najít požadovaný bod obnovení. Kromě základních informací o záloze, se na každém řádku nachází tlačítko s nápisem „RESTORE“. Na následujícím obrázku si lze všimnout střídajících se barev. Každá barevná sekvence označuje zálohy, které spadají do stejné skupiny. Tyto skupiny se týkají hlavně záloh inkrementálních, které jsou na sobě vzájemně závislé.




Obrázek 11 Záložka Backups

### 5.2.3 Záložka Providers

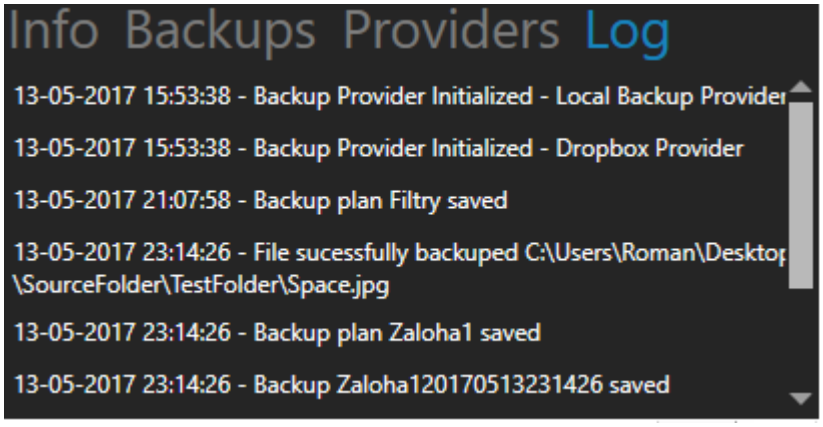
V této záložce se nachází seznam úložišť jednotlivých účtů, u kterých jsou uvedeny i kapacity všech úložišť.




Obrázek 12 Záložka Providers

### 5.2.4 Záložka Log

V této záložce se zobrazují zprávy z provozu aplikace. Uživatel zde může nejen sledovat průběh zálohování, ale i chybové informace a stav editace jednotlivých položek.

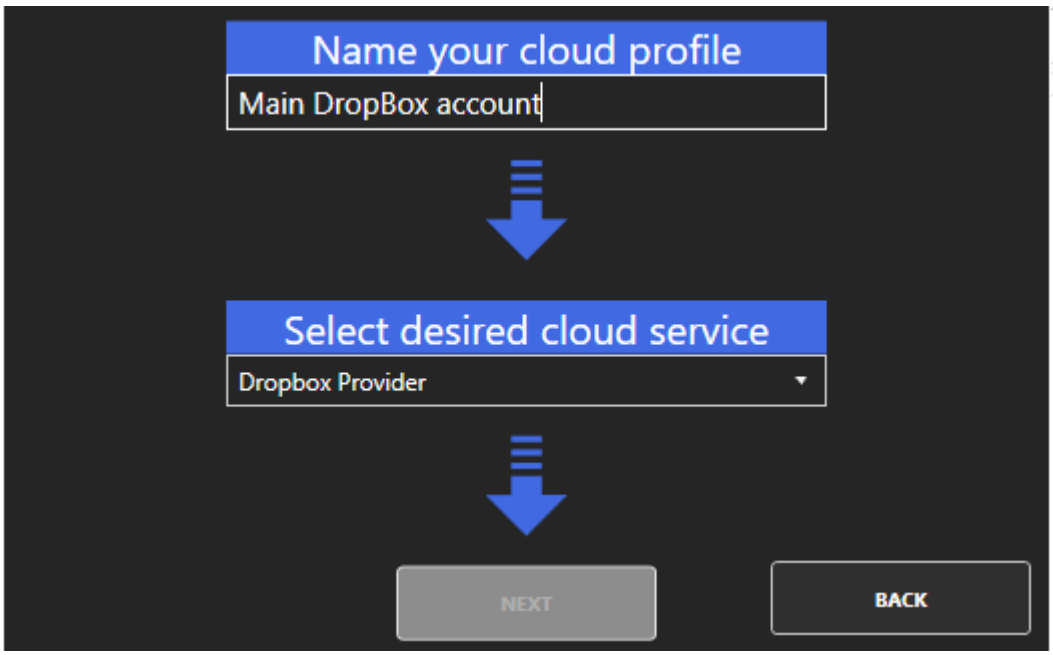



Obrázek 13 Záložka Log

## 5.3 NewProviderDialogView

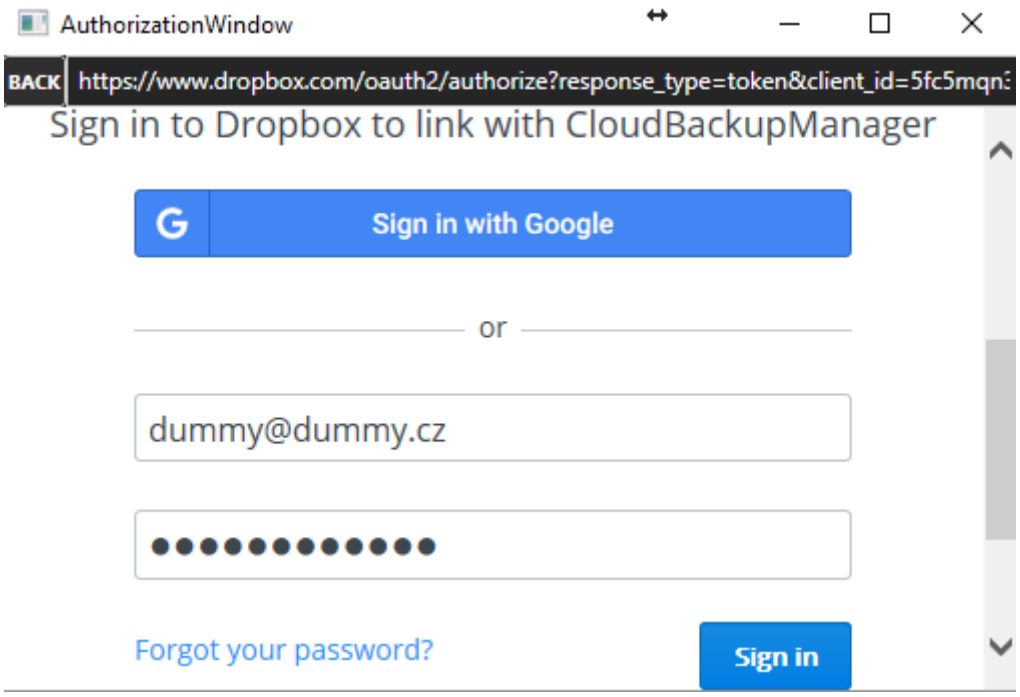
V tomto dialogu si uživatel může vytvářet zálohovací providery. Pro vytvoření nového providera stačí splnit pár kroků, které budou v následujících odstavcích popsány.

Ze všeho nejdříve musí uživatel svého nového providera pojmenovat, poté si zvolí požadovaný typ a klikne na tlačítko další.



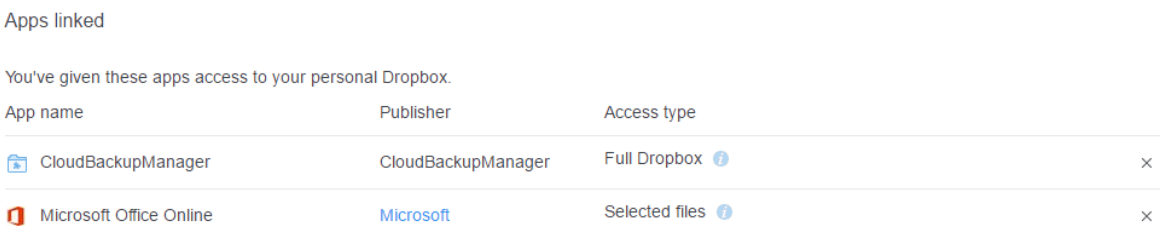

Obrázek 14 NewProviderDialogView

Další postup se může lišit v závislosti na zvoleném provideru, ale většinou spočívá v tom, že se uživatel přihlásí pod svým účtem na požadovanou cloudovou službu a potvrdí aplikaci oprávnění.




Obrázek 15 Autorizační okno pro službu Dropbox

Aplikace si neuchovává použité přihlašovací údaje, ale jen uloží takzvaný token, který má určitou platnost a v žádném případě z něj nelze zjistit přihlašovací údaje uživatele. Uživatel má možnost kdykoliv odejmout přístup aplikaci, z webové správy účtu služby Dropbox.




Obrázek 16 Správa aplikací, ve webovém rozhraní služby Dropbox

# 6 POPIS IMPLEMENTACE

## 6.1 Použité knihovny

### 6.1.1 PRISM

Tento Framework obsahuje pomocné nástroje, které usnadňují vývoj modulárních aplikací ve WPF, UWP a Xamarin Forms. Jsou zde například praktické implementace návrhových vzorů, jako MVVM, DI, Event Aggregator, Commands a další.

### 6.1.2 MahApps.Metro

Tato knihovna obsahuje GUI kontrolky a styly, které se inspirují vzhledem Metra z operačního systému Windows 8 a výše. Nalezneme zde i třídu MetroWindow, která rozšiřuje standardní Window o mnoho funkcí.

### 6.1.3 DropBox.Net

SDK, které obsahuje nástroje usnadňující integraci aplikace s Dropbox API v2.

## 6.2 Základní logika aplikace

V této části práce budou popsány struktury a mechanismy, které jsou využívány pro chod aplikace. Konkrétně se jedná o komunikaci mezi jednotlivými moduly, jejich službami, sdílení dat, a mechanismy propagování stavových informací mezi View a ViewModelem. V každé části bude přiložený výňatek z kódu, který bude demonstrovat praktickou implementaci popisované části.

### 6.2.1 Dependency injection

Napříč aplikací je používán návrhový vzor dependency injection, konkrétně mix constructor injection a interface injection. Pro řešení většiny závislostí důležitých částí programu se používá UnityContainer, do kterého se inicializují sdílené služby pod jejich patřičným rozhraním. Instance tohoto kontejneru se vytvoří při startu aplikace v bootstraperu, a automaticky se s ní řeší všechny závislosti inicializovaných modulů. Pokud je vyžadován objekt, který implementuje rozhraní IUnityContainer, tak kontejner vrátí svoji instanci.

```
        public InfrastructureModule(IUnityContainer container, IRegionManager
regionManager, IEventAggregator eventAggregator)
        {
            Container = container;
            RegionManager = regionManager;
```

```
            EventAggregator = eventAggregator;
        }
```

Všechny služby jsou registrovány jako singleton, což umožnuje jednoduché sdílení dat a služeb napříč všemi moduly aplikace. V následujících odstavcích bude představena ukázka praktické inicializace služby BackupPlanService.

```
public class BackupPlanService : LocalServiceBase, IBackupPlanService {. . .}
```

Tato třída implementuje rozhraní IBackupPlanService, pod kterým bude vystupovat v rámci celé aplikace, což znamená, že si v rámci celé aplikace budeme z kontejneru žádat objekt, který implementuje rozhraní IBackupPlanService, aniž bychom se museli starat o konkrétní typ, jeho konstrukci atd.. Tato metoda předání zodpovědnosti je užitečná v tom, že můžeme změnit parametry konstruktoru za jiné známe závislosti, nebo i nahradit BackupPlanService za jiný typ, který výše uvedené rozhraní implementuje, a nemusíme v ostatních částech kódu dělat žádné dodatečné úpravy. Tato registrace probíhá při inicializaci aplikace v inicializační metodě modulu. Objekt typu ContainerControlledLifetimeManager, který se vkládá jako parametr do metody RegisterType, říká kontejneru, že si má následující instanci ponechat a zacházet s ní jako se singletonem. Bez tohoto parametru by kontejner vytvářel pokaždé novou instanci typu BackupPlanService.

```
public class InfrastructureModule : IModule
    {
. . .
        public void Initialize()
        {
            Container.RegisterType<IBackupPlanService,BackupPlanService>(new
ContainerControlledLifetimeManager());

. . .
```

V dalším kroku si můžeme nechat od kontejneru generovat objekty požadovaných typů pomocí metody Resolve. Prakticky to funguje tak, že pokud kontejner řeší konkrétní typ, tak rekurzivně projde a volá metodu Resolve na všechny parametry konstruktoru, které mohou mít také nějaké závislosti. Toto volání se opakuje, dokud není zkonstruovaný celý strom, až do bodu, kdy je možné vrátit požadovaný objekt. Pokud má metoda Resolve vrátit objekt, který implementuje nějaké rozhraní, tak je potřeba, aby toto rozhraní bylo v

kontejneru předem spárované s konkrétním typem, který toto rozhraní implementuje. Pokud se řešený typ nachází v kontejneru jako singleton, tak je navrácen okamžitě bez dalšího procházení vnitřních závislostí.

```
IUnityContainer container = unityContainer;
BackupPlanDetailViewModel viewModel =
container.Resolve<BackupPlanDetailViewModel>();

IBackupPlanService canResolveInterfaceToo =
container.Resolve<IBackupPlanService>();
```

### 6.2.2 Popis implementace třídy BootStraper

Tato třída dědí od UnityStraper, která se nachází v knihovně PRISM a má na starosti úvodní konfiguraci a registraci modulů za startu aplikace, tomuto procesu se říká také bootstrapping. Nejdříve bylo nutné překrýt virtuální metody z bázové třídy pomocí klíčového slova override a v těchto metodách inicializovat a vytvořit hlavní okno aplikace, které je reprezentováno třídou Shell, která obsahuje bezparametrický konstruktor, proto ji může ještě prázdný unity kontejner inicializovat.

```
class BootStrapper : UnityBootstrapper
{
    protected override DependencyObject CreateShell()
    {
        return this.Container.Resolve<Shell>();
    }

    protected override void InitializeShell()
    {
        base.InitializeShell();

        App.Current.MainWindow = (Window)Shell;
        App.Current.MainWindow.Show();
    }
```

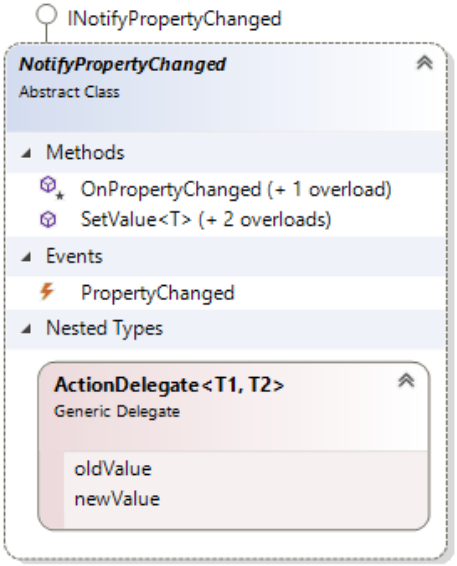
V další části je nutné naskládat do katalogu modulů jednotlivé moduly, které chceme při startu inicializovat. Pořadí v jakém moduly naskládáme do katalogu nemá vliv na pořadí v inicializaci, moduly se inicializují v pořadí podle toho, jaké mají vzájemné závislosti. Tyto závislosti lze definovat pomocí atributů v třídě projektu (modulu), který implementuje rozhraní IModule. Toto rozhraní obsahuje pouze jednu bezparametrickou metodu Initialize, která slouží jako vstupní bod pro inicializaci daného modulu.

```
protected override void ConfigureModuleCatalog()
{
 Type infrastructureModule = typeof(Infrastructure.InfrastructureModule);
 ModuleCatalog.AddModule(
                new ModuleInfo()
                {
                    ModuleName = infrastructureModule.Name,
                    ModuleType = infrastructureModule.AssemblyQualifiedName
                });

 Type configurationModule = typeof(Configuration.ConfigurationModule);
 ModuleCatalog.AddModule(
                new ModuleInfo()
                {
                    ModuleName = configurationModule.Name,
                    ModuleType = configurationModule.AssemblyQualifiedName
```

### 6.2.3 Popis implementace třídy NotifyPropertyChanged

Tato třída implementuje rozhraní INotifyPropertyChanged a snaží se maximálně ulehčit správné notifikování o změnách dat.




Obrázek 17 Diagram třídy NotifyPropertyChanged

Umožňuje například nastavení hodnoty pomocí použití bázové metody SetValue v setru nastavované vlastnosti, kde se v implementaci metody automaticky kontroluje, zda není nastavovaná hodnota rovna té staré. V případě nerovnosti se přiřadí nová hodnota a automaticky vyvolá událost o změně vlastnosti. Tato událost se samozřejmě vyvolá pouze za předpokladu, že má alespoň jednoho subscribera. Do parametrů je možné i vložit instanci delegáta (reference na metodu), která má stejnou signaturu jako ActionDelegate. Do

parametrů referencované metody se po změně dosadí nová i stará hodnota, na které můžeme podle potřeby zareagovat. Například můžeme zakázat použití nějaké funkce v uživatelském rozhraní, pokud nová hodnota není validní.

```
public abstract class NotifyPropertyChanged : INotifyPropertyChanged
    {
        public delegate void ActionDelegate<T1, T2>(T1 oldValue, T2 newValue);

        public void SetValue<T>(ref T field, T value, ActionDelegate<T, T>
action, [CallerMemberName] string propertyName = null)
        {
            var comparer = EqualityComparer<T>.Default;
            if (!comparer.Equals(field, value))
            {
                var oldValue = field;
                field = value;
                action?.Invoke(oldValue, value);
                OnPropertyChanged(propertyName);
            }
        }
```

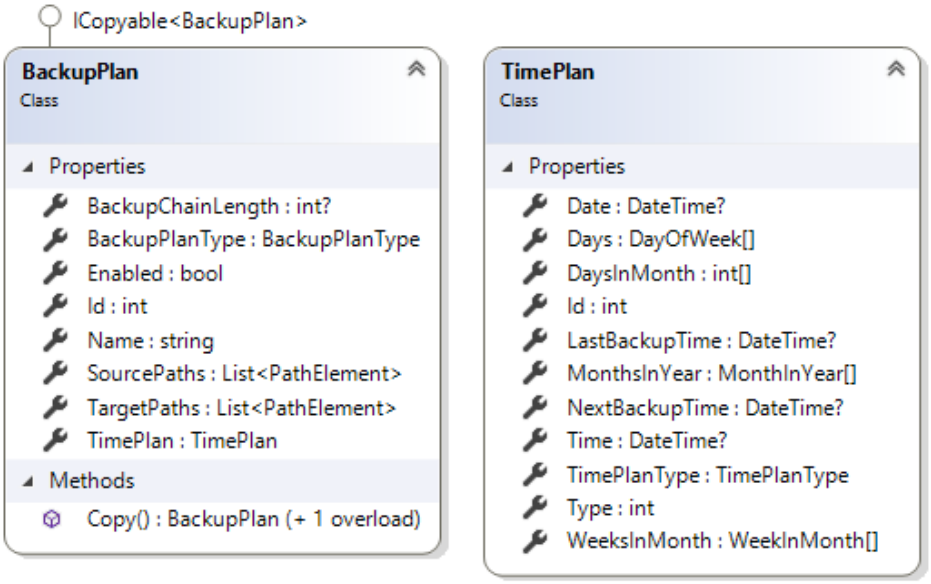
Třída obsahuje i přetížení metody OnPropertyChanged, které umožňuje řetězit vlastnosti, pro které potřebujeme vyvolat událost o změně.

```
        private ulong availableSpace;

        public ulong AvailableSpace
        {
            get { return availableSpace; }
            set => SetValue(ref availableSpace, value, () =>
OnPropertyChanged(nameof(PercentUsed), nameof(SpaceUsageText)));
        }
```

## 6.3 Popis implementace třídy BackupPlan

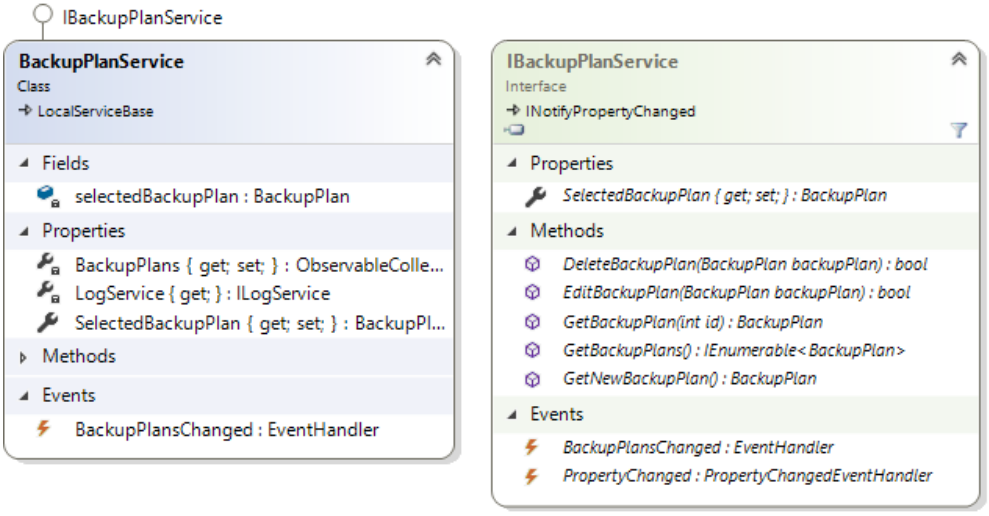
Tato třída reprezentuje zálohovací plán, který obsahuje data, které ovlivňují chování a průběh samotné zálohy. Každý plán také obsahuje vlastnost TimePlan, která obsahuje nezbytná data pro kalkulaci času periodického zálohování.




Obrázek 18 Diagram tříd BackupPlan a TimePlan

## 6.4 Popis implementace služby BackupPlanService

Tato služba implementuje rozhraní IBackupPlanService a slouží jako centrální bod pro čtení a manipulaci s objekty zálohovacích plánů. Je zde zapouzdřena veškerá logika ukládání, čtení a konstrukce nových objektů typu BackupPlan, takže například přechod na jiný typ ukládání těchto dat neovlivní ostatní části programu.




Obrázek 19 Class Diagram třídy BackupPlanService a rozhraní IBackupPlanService

Vnitřní implementace je vláknově bezpečná, což umožňuje editaci a čtení záznamů paralelně, aniž bychom se museli bát nekonzistence dat a obsahuje také cache zálohovacích plánů, která je synchronizována s databází, což umožňuje rychlejší předávání záznamů a menší zátěž na CPU.

Inicializace probíhá po startu programu, kdy se konkrétní typ služby spáruje s jejím rozhraním v rámci DI kontejneru jako singleton. Poté tuto instanci mohou obdržet všechny části programu, které splňují následující body.

1. Projekt má referenci na knihovnu CloudBackupManager.Infrastructure.Interface.
2. Třída, ve které se má implementovat práce s touto službou má v konstruktoru vloženou závislost IBackupPlanService.
3. Instance třídy bude zkonstruována pomocí kontejneru, který do závislosti (parametru) správně dosadí instanci BackupPlanService.

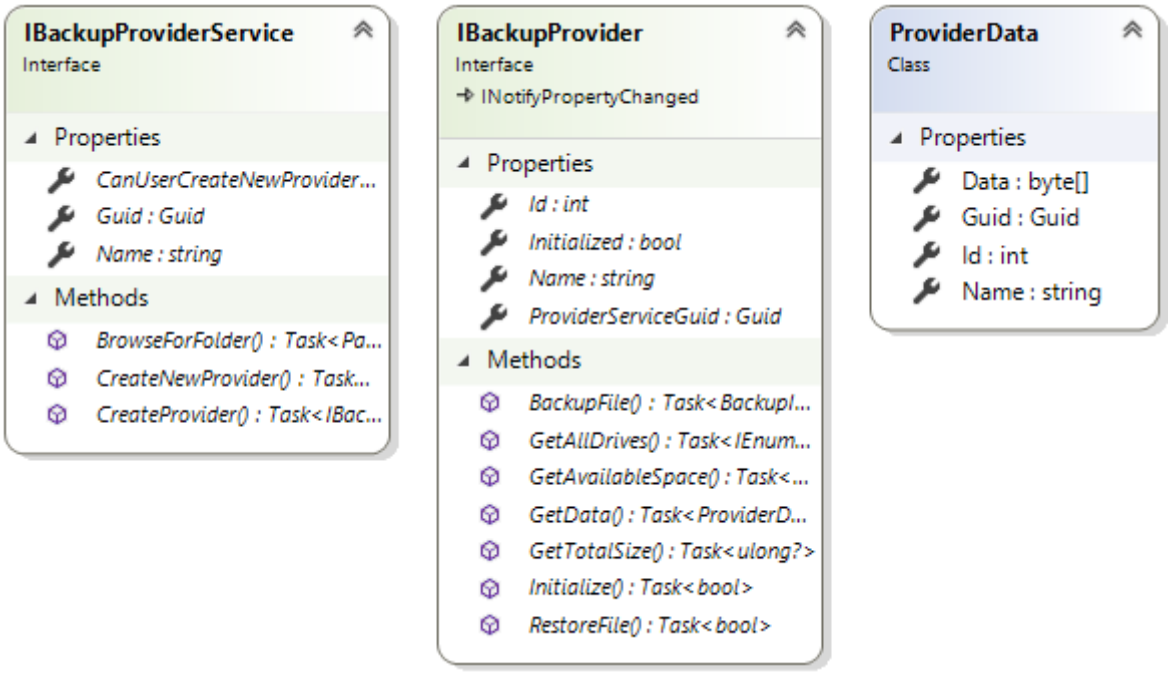
```
        var vm = Container.Resolve<BackupPlanDetailViewModel>();
        . . . .
        public class BackupPlanDetailViewModel : ViewModelBase
        {
        private readonly IBackupPlanService BackupPlanService;
        //konstruktor

public BackupPlanDetailViewModel(IBackupPlanService backupPlanService, . . .
         {
          BackupPlanService = backupPlanService;
```

## 6.5 Implementace služby BackupProviderManager

Tato služba implementuje rozhraní IBackupProviderManger a slouží jako správce konkrétních zálohovacích služeb. Jsou zde spravovány tří typy objektů.

1. Objekt implementující rozhraní IBackupProviderService.
2. Objekt implementující rozhraní IBackupProvider.
3. Objekt typu ProviderData.




Obrázek 20 Diagram tříd spravovaný službou BackupProviderManager

### 6.5.1 IBackupProviderService

Objekty, které implementují tento interface reprezentují centrální službu pro konkrétní typ úložiště (Lokální úložiště, Dropbox úložiště). Pomocí asynchronní metody CreateNewProvider v této službě je možné vytvářet nové zálohovací poskytovatele, kteří se starají o proces přenosu zálohovaných dat na cílové úložiště účtu.

Například, aby služba mohla vytvořit poskytovatele zálohování pro Dropbox, musí nejdříve zobrazit uživateli přihlašovací okno, které obsahuje prohlížeč se zobrazenou přihlašovací stránkou, do které se musí uživatel přihlásit na požadovaný účet. Po přihlášení a autorizaci potřebných oprávnění se v url navrátí přihlašovací token, který bude poskytovatel používat pro komunikaci s účtem služby Dropbox.

```
internal class DropBoxBackupProviderService : IBackupProviderService
{

    public async Task<IBackupProvider> CreateNewProvider()
    {
            var token = await DropBoxHelper.GetAcessToken();
            if (token != null)
            {
                ClientData cd = new ClientData()
                {
                    AccessToken = token.Serialize<string>()
                };

                return new DropboxProvider(new ProviderData()
```

```
                {
                    Id = 0,
                    Data = cd.Serialize(),
                    Guid = AssignedGuid,
                    Name = String.Empty
                }, Container);
            }
            return null;
    }
```

V následují ukázce kódu je předveden postup inicializace služby DropBoxBackupProviderService která implementuje rozhraní IBackupProviderService do správce zálohovacích služeb.

```
public class DropboxModule : IModule
{

   public void Initialize()
   {
    Container.Resolve<IBackupProviderManager>().AddBackupProviderService(
    new DropBoxBackupProviderService(Container));

    Container.Resolve<IBackupProviderManager>().InitProviders(
    Guid.Parse(DropBoxBackupProviderService.AssignedGuid.ToString()))
.   ConfigureAwait(false).GetAwaiter().GetResult();

    var provider = Container.Resolve<IBackupProviderManager>()
                   .Providers.FirstOrDefault(q => q is DropboxProvider);
   }
}
```

### 6.5.2 IBackupProvider

Objekt implementující rozhraní IBackupProvider reprezentuje poskytovatele zálohování, který má na starost komunikaci s konkrétním datovým účtem. Například poskytovatel pro Dropbox zajišťuje komunikaci s úložišti účtu, do kterého se při vytváření poskytovatele uživatel přihlásil a potvrdil patřičná oprávnění. Jeden účet může mít i více úložišť, což platí například pro poskytovatele lokálního zálohování.

Každý poskytovatel generuje objekt typu ProviderData, který je nutný pro zpětné zrekonstruování poskytovatele do posledního stavu. Například pokud poskytovatel potřebuje pro svou práci přihlašovací token, tak ho uloží právě do tohoto objektu. Data se mohou poté libovolně ukládat do databáze pomocí služby implementující rozhraní IBackupProviderManager. Díky tomuto mechanizmu je možné po restartu počítače

poskytovatele znovu obnovit, aniž by po uživateli bylo vyžadováno opětovné přihlášení do svého účtu.

```
public class BackupProviderManager : LocalServiceBase,
Interface.Services.IBackupProviderManager
{

        public async Task<bool> SaveProviderState(IBackupProvider backupProvider)
        {
            var data = await backupProvider.GetData();
            EditProviderData(data);
            return true;
        }
```

Kromě komunikace se poskytovatel také stará o samotnou zálohu a obnovení konkrétního souboru. Pro spuštění zálohy stačí zavolat parametrickou metodu BackupFile. Níže najdeme ukázku metody, která se nachází ve třídě DropboxProvider.

```
public async override Task<BackupItem> BackupFile(OperationTaskSource source,
RelativeFilePath sourceFilePath, PathElement targetPath)
        {
            var fi = new FileInfo(sourceFilePath.FullPath);
            var uploadFolderPath = targetPath + "/" + source.BackupPlan.Name +
$"{source.OperationStart.Value.ToString("yyyyMMddHHmmss")}";
            var targetFilePath = uploadFolderPath + "/" + fi.Name;


            using (var httpClient = GetHttpClient())
            {
                using (var client = await GetDropBoxClient(httpClient))
                {
                    try
                    {
                        await ChunkUpload(client,sourceFilePath.FullPath,
                              targetFilePath);
                        return new BackupItem()
                        {
                            ProviderId = this.Id,
                            PathFrom = sourceFilePath.FullPath,
                            DateEdit = fi.LastWriteTime,
                            PathTo = targetFilePath
                        };

                    }
                    catch (Exception e)
                    {
                        Container.Resolve<ILogService>().AddLog(e.ToString());
                        return null;
                    }
                }
            }
        }
```

## 6.6 Popis implementace třídy BackupService

Tato služba má na starosti logistiku zálohování. Po inicializaci služby se založí Task, který běží na separátním vlákně a periodicky provádí postupně několik operací.

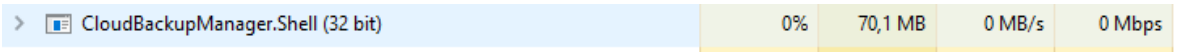
1. Zkontroluje, zda se v pořadníku záloh pro obnovení nenachází záloha, kterou si uživatel vyžádal obnovit. Pokud ano, okamžitě se bude snažit provést operaci obnovení dat ze zálohy.
2. Zkontroluje, zda se v pořadníku zálohovacích plánů nenachází nějaký plán. Pokud ano, okamžitě se bude snažit provést zálohu.
3. Zkontroluje naplánované časy záloh jednotlivých zálohovacích plánů. Je li naplánovaný čas pro zálohu menší nebo roven času momentálnímu, tak se zálohovací plán přesune do pořadníku.
4. Naplánuje budoucí časy záloh zálohovacích plánů.

# 7 ZHODNOCENÍ PROGRAMU Z HLEDISKA VÝKONNOSTI

Program byl naprogramován s ohledem na to, že může být používán na stroji s omezenou kapacitou paměti a slabým dvoujádrovým procesorem.

V klidovém stavu aplikace zabírá kolem 70 MB, v závislosti na tom, kolik zálohovacích plánů a poskytovatelů má uživatel vytvořených. Pro čtení datových struktur se čte a zpracovává každá hodnota samostatně, aby se v paměti neskladovala celá struktura.

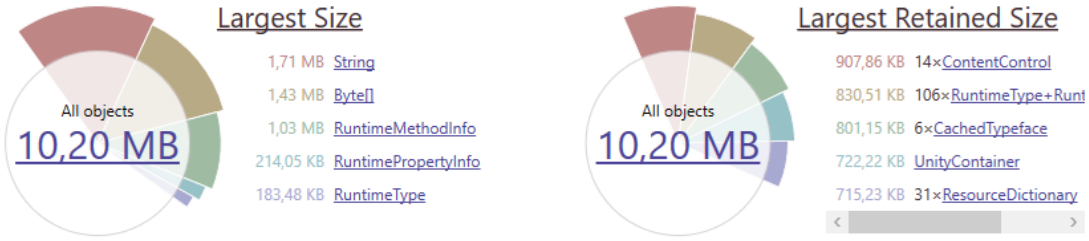
Jednotlivé procesy v aplikaci jsou paralelizované, takže uživatelské rozhraní je responzivní i během procesu zálohování. Aplikace v klidovém stavu nebere téměř žádné prostředky z CPU, takže uživatele neobtěžuje při používání počítače.




Obrázek 21 Vytížení systémových prostředků v klidovém stavu

V případě zálohy spotřeba paměti stoupne přes 100MB a vytížení CPU u 4 jádrového procesoru 3,5 GHz se pohybuje kolem 4 procent.


Obrázek 22 Vytížení systémových prostředků během zálohování




Obrázek 23 Snapshot paměti aplikace během zálohy, pomocí nástroje dotMemory

# ZÁVĚR

Tato práce obsahuje základní poznatky problematiky zálohování a cloudových úložišť. Byly popsány rizika ztráty dat, stručný popis procesu zálohování a jednotlivé typy záloh. Dále byly popsány použité technologie a návrhové vzory, které byly použity při praktickém vývoji aplikace a vysvětlen pojem cloudové úložiště, spolu s popisem základních typů cloudových úložišť.

V rámci praktické části práce byla realizována aplikace, která umí zálohovat jak lokálně, tak i do cloudového úložiště Dropbox. Při tvorbě aplikace byla provedena základní analýza požadavků a stanovení cílů projektu.

Dále praktická část obsahuje stručný popis projektů a závislostí, ze kterých se výsledná aplikace skládá, spolu s popisem uživatelského rozhraní aplikace, který se dá použít i jako uživatelský manuál.

V poslední části práce je obsažen popis praktické implementace klíčových částí, spolu s ukázkami kódu. Tato část je nezbytná pro správné pochopení vnitřní funkcionality aplikace.

Práce na tomto projektu byla přínosnou zkušeností, zejména při praktické implementaci struktur aplikace a návrhových vzorů, mezi které hlavně patří návrhový vzor MVVM a Dependency Injection. Tyto návrhové vzory byly prakticky použity napříč celou aplikací. Dále jsem získal zkušenosti s vývojem pluginu pro platformu Dropbox.

Hlavním přínosem naprogramované aplikace je možnost periodické zálohy na cloudové úložistě Dropbox, kdy si může uživatel vybrat z různých typů záloh a časových nastavení. Vzhledem k tomu, že je aplikace napsaná modulárně, je ji možné v budoucnosti nadále rozšiřovat o novou funkcionalitu a moduly.

# SEZNAM POUŽITÉ LITERATURY

[1] ŠKODOVÁ, Jana. Nejčastější důvody ztrát firemních dat a jak se jich vyvarovat. *CIO* [online]. 2017, , 3 [cit. 2017-05-19]. Dostupné z: http://businessworld.cz/bezpecnost/nejcastejsi-duvody-ztrat-firemnich-dat-a-jak-se-jich-vyvarovat-13412-p14594

[2] JUNEK, Pavel. *Zálohování a archivace dat v datových centrech v podnikovém prostředí* [online]. Ústí nad Labem, 2013 [cit. 2017-05-19]. Dostupné z: http://www.diplomovaprace.cz/2013/9/Bakalarska_prace_Pavel_Junek.pdf. Bakalářská práce. Univerzita Jana Evangelisty Purkyně. Vedoucí práce Ing. Toni Koluch, Ph.D.

[3] NELSON, Steven. *Pro data backup and recovery securing your information in the terabyte age* [online]. Berkeley, CA: Apress, 2011 [cit. 2017-05-19]. ISBN 978-143-0226-635.

[4] LISKA, Allan a Timothy GALLO. *Ransomware: Defending Against Digital Extortion* [online]. 2016. Sebastopol, California: O'Reilly, 2016 [cit. 2017-05-19]. ISBN 978-1-4919-6782-9. Dostupné z: http://shop.oreilly.com/product/0636920054290.do?sortby=publicationDate

[5] MICZKA, Marian. *Zálohování dat a cloudová uložiště* [online]. Brno, 2015, 65 s. [cit. 2017-05-19]. Dostupné z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=106243. Vysoké učení technické v Brně, Fakulta podnikatelská. Vedoucí práce Ing. Jiří Kříž, Ph.D.

[6] MIČAN, Jiří. *Zálohování dat* [online]. Praha, 2015, 65 s. [cit. 2017-05-19]. Dostupné z: https://is.bivs.cz/th/22761/bivs_b/bakalarska_prace_zalohovani_dat__Mican_Jiri.pdf. Bakalářská práce. Bankovní institut vysoká škola Praha, Katedra informatiky a kvantitativních metod. Vedoucí práce Ing. Jiří Kříž, Ph.D.

[7] Hybridní cloud. In: ManagementMania.com [online]. Wilmington (DE) 2011-2017, 10.04.2017 [cit. 19.05.2017]. Dostupné z: https://managementmania.com/cs/hybridni-cloud

[8] Microsoft API and Reference Catalog: MSDN. *Microsoft* [online]. Redmond: Microsoft, c2017 [cit. 2017-05-19]. Dostupné z: https://msdn.microsoft.com/en-us/library

[9] NATHAN, Adam. *WPF 4.5 unleashed*. 1. Indianapolis: Sams Publishing, 2013. Unleashed. ISBN 06-723-3697-9.

[10] TROELSEN, Andrew. *Pro C# and the .NET 4.5 framework*. 6. edition. Berkeley, Calif: APress, 2012. ISBN 14-302-4233-7.

[11] WPF Tutorial | Data Binding Overview [online]. Christian Moser, 2011 [cit. 2017-05-19]. Dostupné z: http://www.wpftutorial.net/DataBindingOverview.html

[12] Components of .Net Framework, CLR, CTS, CLS, Base Class Library. *DeveloperIn* [online]. c2016 [cit. 2017-05-19]. Dostupné z: http://www.developerin.net/a/39-Intro-to-.Net-FrameWork/23-Components-of-.Net-Framework

# SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

| | |
|---|---|
| GUI | Graphical User Inteface |
| WPF | Windows Presentation Foundation. |
| MVVM | Model-View-ViewModel |
| XAML | Extensible Application Markup Language |
| CLR | Common Language Runtime |
| XML | Extensible Markup Language |
| CLI | Common Language Infrastructure |
| CPU | Central Processing Unit |
| DI | Dependency Injection |
| DIC | Dependency Injection Container |

# SEZNAM OBRÁZKŮ

# SEZNAM TABULEK

# SEZNAM PŘÍLOH

# PŘÍLOHA P I: OBSAH PŘILOŽENÉHO DVD

Struktura adresářů přiloženého DVD je následující:

Solution – obsahuje kompletní solution pro Visual Studio 2017

App – obsahuje vytvořenou aplikaci