Jihočeská univerzita v Českých Budějovicích

Přírodovědecká fakulta

Aplikace pro konsolidaci cloudových úložišť

Diplomová práce

Bc. Tomáš Mika

Školitel: Ing. Jan Fesl, Ph.D.

České Budějovice 2021

BIBLIOGRAFICKÉ ÚDAJE

Mika Tomáš, 2021: Aplikace pro konsolidaci cloudových úložišť. [Application for consolidation of cloud drives. Mgr. Thesis, in Czech.] – 74 p., Faculty of Science, University of South Bohemia, České Budějovice, Czech Republic.

ANOTACE

Tématem diplomové práce je vytvoření aplikace pro konsolidaci cloudových úložišť. Práce popisuje nástroje a technologie nezbytné pro vytvoření takovéhoto systému, zároveň uvádí přímé srovnání s již existujícími a dostupnými řešeními. Práce se zabývá problematikou ukládání dat v cloudových úložištích různých poskytovatelů a jejich zabezpečení proti ztrátě, poškození, manipulaci či strojové analýze.

KLÍCOVÁ SLOVA

Cloud, konsolidace, zabezpečení, šifrování, virtuální souborový systém

ABSTRACT

The topic of this thesis is to create an application for consolidation of cloud storage. The work describes the tools and technologies necessary to create such a system, while also providing a direct comparison with existing and available solutions. The thesis provides with data storage problematic in cloud storage of various providers and their security against loss, damage, manipulation, or machine analysis.

KEYWORDS

Cloud, consolidation, security, encryption, virtual file system

Prohlašuji, že jsem autorem této kvalifikační práce a že jsem ji vypracoval pouze s použitím pramenů a literatury uvedených v seznamu použitých zdrojů.

V Českých Budějovicích dne 12. dubna 2021

Podpis: ……

PODĚKOVÁNÍ

Na tomto místě bych rád poděkoval panu Ing. Janu Feslovi, Ph.D. za cenné připomínky a odborné rady, kterými přispěl k vypracování této diplomové práce.

OBSAH

1. ÚVOD

Cloud patří v dnešní době k jednomu z nejskloňovanějších pojmů světa informačních technologií, a to zejména v důsledku tlaku na optimalizace, snižování provozních nákladů a zjednodušení užívání služeb pro koncové uživatele. Pryč jsou doby lokálních úložišť, často nedostatečně zabezpečených, k nimž se přistupovalo skrze více méně nezabezpečený protokol FTP, či s využitím protokolu SMB (CIFS), i když třeba říci, že i tento starší způsob má stále svá uplatnění. Doba pokročila natolik, že dnes je již možné virtuálně v onom magickém cloudu provozovat téměř vše, včetně celých firemních infrastruktur. To samozřejmě vyjma potenciálních výhod, jako je úspora nákladů, přináší i nebezpečí v podobě nárůstu bezpečnostních rizik s nahráváním soukromích, firemních či osobních údajů na servery třetích stran (Microsoft, Google, Amazon) a přenos těchto dat přes internet. Je pak na zvážení pozitiv, rizik a rozhodnout se, co převažuje. Pro menší firmy či soukromé osoby pak většinou provoz vlastní IT infrastruktury nedává z ekonomického pohledu smysl a zůstává zde pouze otázka zabezpečení dat uložených v cloudových infrastrukturách.

Pro fyzické osoby často cloudová úložiště poskytují prostor pro synchronizaci dat mezi více přístroji a zálohování dat. Většina takto fungujících osob využívá bezplatné varianty, které poskytují nemalý prostor zdarma, cloudových úložišť různých poskytovatelů, jelikož bezplatná kapacita jednoho poskytovatele téměř nikdy nestačí. Z toho vyplývá mnoho problémů při práci s takto uloženými daty.

Dnešní cloudová úložiště často umí live synchronizaci, ale už neumí tuto synchronizaci provozovat nad stejnou složkou souborového systému a vzájemně není mnoho aplikací, které by uměly tuto synchronizaci provozovat vůči různým službám. Tyto aplikace jsou téměř vždy úzce zaměřené na jednu službu, OneDrive Sync, BackupAndSync by Google, Dropbox Sync a podobné. Pokud nějaká aplikace umožnuje provoz pro více těchto úložných služeb, většinou nepodporuje live synchronizaci.

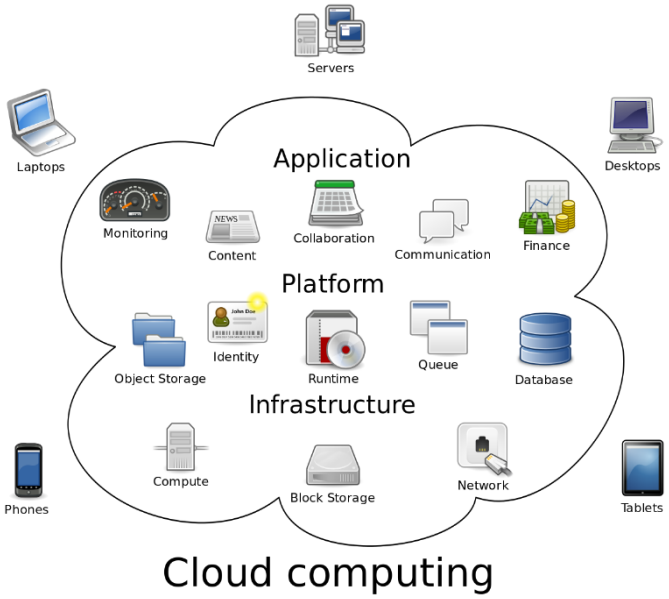
Otázka zabezpečení takto zálohovaných či synchronizovaných dat je velkou problematikou těchto služeb. Data nahraná do úložišť třetích stran jsou často automatizovaně skenována, analyzována a dále zpracována za různými účely. Často se pak stává, že obsah našich dat je využit k predikci priorit a zájmů. Takto analyzovanému uživateli je pak nabízena reklama a další obsah, čímž poskytovatel většinou hrazeného obsahu zvyšuje pravděpodobnost úspěchu, že zaujme daného uživatele, který dostává obsah na míru (často reklamy a nabídky prodeje).

2. CÍL BAKALÁŘSKÉ PRÁCE

Cílem diplomové práce bylo vytvoření funkčního prototypu aplikace pro konsolidaci veřejně dostupných cloudových úložišť jako jsou např. OneDrive, či Google Drive. Aplikace umožní pracovat s uložišti separátně či konsolidovaně dle nastavení. Dále umožní ukládání nezměněných, rozdělených či zašifrovaných dat v cloudových úložištích s realizací ovládání pomocí grafického uživatelského rozhraní, které rovněž umožní přidání úložišť za běhu prostřednictvím modulů/knihoven.

3. CLOUD COMPUTING

Pod pojmem cloud computing se skrývá model vývoje, používání, či poskytování služeb skrze síť internetu bez nutnosti vlastnit fyzickou infrastrukturu na níž tyto služby provozovány. Hlavní myšlenkou je cíl, tedy využití IT infrastruktury či software, namísto technické realizace. Proto se všechny části, tedy infrastrukturu, vývojovou a aplikační platformu i software převádí do formy služby. [1] K takto široce dostupným službám pak lze jednoduše přistupovat pomocí webového prohlížeče, bez nutnosti instalace dalších programů. Taková služba se pak poskytuje na základě měsíčních poplatků.



Obrázek 1: Cloud computing – obecné schéma [10]

Provozovatel poskytuje službu (souborový server, emailový server, CRM aplikace, …) koncovým zákazníkům a nese náklady na provoz fyzické infrastruktury. Tuto fyzickou infrastrukturu pak sdílí více zákazníků a dochází tak k lepšímu využití dostupných prostředků.

Služby mohou být poskytovány až do té míry, že umožnují zákazníkovi pouhé užívání potřebné služby a vše ostatní, včetně instalace, konfigurace, zálohování, údržby (aktualizace) a provozu aplikace, zajišťuje poskytovatel služby.

3.1 Historie Cloud computingu

Začátky tohoto odvětví se datují od 70. letech 20. století. Za duchovního otce myšlenky Cloud computingu je považován John McCarthy, profesor z prestižní americké univerzity MIT. Ten v roce 1961 prezentoval myšlenku sdílení počítačových technologií podobně,

jako již tehdy docházelo ke sdílení elektrické energie (každý dům neměl svou elektrárnu). Zcela geniálně použil analogii se sdílením elektrické energie za pomocí rozvodné sítě v době, kdy veřejně neexistovala hardwarová ani softwarová virtualizace.

Právě díky tomuto více než půl století starému srovnání počítačů, elektřiny a podobných služeb souhrnně nazývaných v angličtině utility se Cloud computingu někdy říká také Utility computing. Samotný pojem „Cloud computing“ se objevil až v roce 1997 v přednášce Ramnatha Chellapa. [1] Ten užil pojem „Cloud“ či oblak pro schématické zobrazení v obrázku nákresu takovéto infrastruktury. Oblak se už tehdy využíval pro názorné zobrazení telekomunikační sítě a Ramnath Challep jej využil, s nápisem internet, jako propojovací segment pro jednotlivé krabičky definující jednotlivé služby.

3.2 Model nasazení

Použitým typem modelu cloudu nám říká, jak je provozovatelem poskytován.

3.2.1 Veřejný cloud (public cloud computing)

Tento model je někdy také označován jako klasický model cloud computingu. V tomto schématu je výpočetní služba nabízena široké veřejnosti. Je charakterizována velmi podobnou či stejnou funkcionalitou napříč všemi klienty. Příkladem veřejné cloud computing služby je například Skype, či Office Online.

3.2.2 Soukromý cloud (private cloud computing)

Cloud je provozován pro účely jediné společnosti nebo organizace, a to organizací samotnou nebo třetí stranou. Příkladem privátního cloudu je emailový server (Microsoft Exchange), který si může organizace provozovat sama na svém hardwaru, popřípadě využít cizí infrastruktury.

3.2.3 Hybridní cloud (hybrid cloud computing)

Hybridní cloudy kombinují veřejný a privátní cloud, které jsou technologicky propojené, aby mezi nimi bylo možné sdílet data a aplikace. Možnost hybridního cloudu přesouvat data a aplikace mezi privátním a veřejným cloudem dává společnosti větší flexibilitu. Další možnosti nasazení pomáhají optimalizovat stávající infrastrukturu, zabezpečení a dodržování předpisů. [2]

3.2.4 Komunitní cloud (community cload computing)

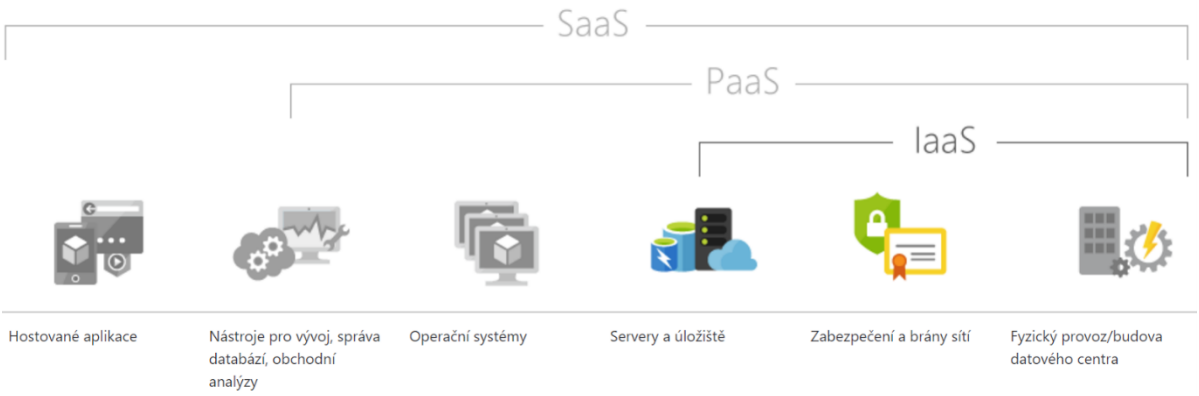
Jedná se o specifický model, kdy je infrastruktura sdílena mezi několika organizacemi, tedy skupinou lidí, kteří ji využívají. Tyto organizace může spojovat bezpečnostní politika, stejný obor zájmu apod. [3]

3.3 Distribuční model

Distribuční model definuje, co je obsahem služby. Zdali jde o službu poskytující software, hardware či celá infrastruktura.

3.3.1 Infrastruktura jako služba (IaaS)

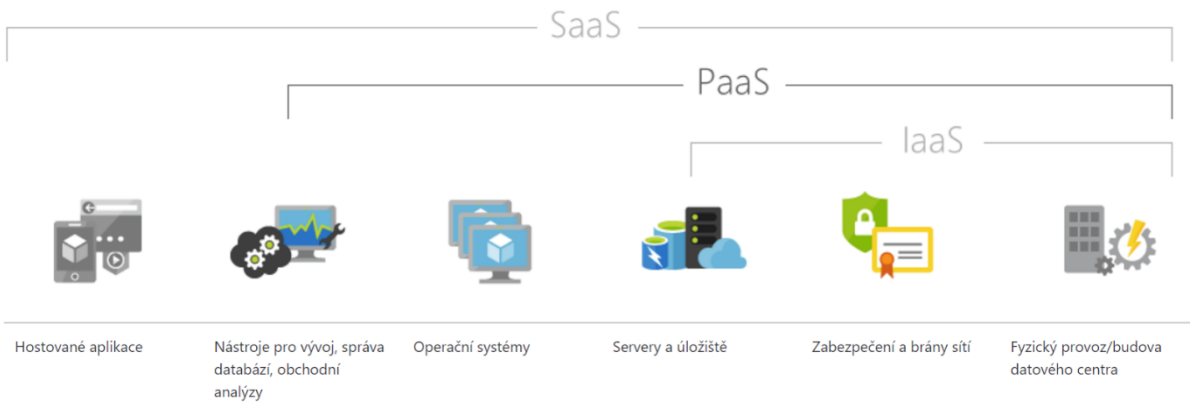
Model „Infrastructure as a Service“ je základním modelem, který uživateli poskytuje základní hardwarové a softwarové prostředky. Také s tímto modelem souvisí nezbytné prostředky pro správu, řízení a nastavování dostupné infrastruktury. Tento koncept říká, že si můžeme pronajmout IT infrastrukturu jako jsou servery, datová úložiště, firewally a další podobné systémy.



Obrázek 2: Distribuční model IaaS [11]

3.3.2 Platforma jako služba (PaaS)

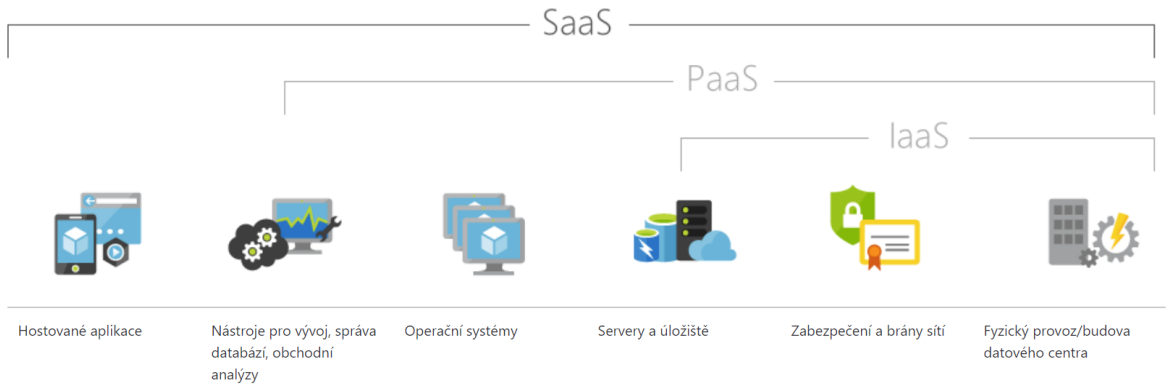
Další úroveň je model Platform as a Service, který je zároveň nejnovější z již dříve definovaných. Platformou se, v tomto případě, rozumí kompletní vývojové nebo aplikační prostředí, do kterého je možné vyvíjet či nahrát vlastní aplikace. Ve chvíli, kdy se do prostředí nahraje finální aplikace, lze takové prostředí označit jako SaaS. Tento model není dle statistik poskytovatelů cloudových služeb (Microsoft, Google, Amazon) příliš rozšířený.



Obrázek 3: Distribuční model PaaS [12]

3.3.3 Software jako služba (SaaS)

Jedná se o nejznámější model, se kterým se setkala většina uživatelů na internetu. SaaS nabízí serverový i klientský software ve formě služby. Mezi tyto aplikace patří například freemail služby (seznam.cz, gmail.com, outlook.com), komunikační programy jako Skype a Messenger, ale také služby typu Office 365, popřípadě alternativa Google Apps. Do této kategorie spadají též služby úložišť dat (OneDrive, Google Drive, DropBox, …). Zde si uživatel kupuje, či zdarma používá, přístup k aplikaci, nikoli aplikaci samotnou.



Obrázek 4: Distribuční model SaaS [13]

3.3.4 Další varianty funkcí jako služby

Dalším příkladem méně používaného modelu distribuce je BaaS, což je Backup as a Service, neboli zálohování jako služba. Tato varianta se používá zejména za účelem archivování a zálohování dat a lze říci, že BaaS je specifická varianta IaaS, kde si zákazník pronajímá diskový prostor.

Svou modelovou kategorii si též zaslouží zmínit monitorovací služby, které jsou nezbytné pro provozovatele služeb garantující nějakou formu dostupnosti (SLA).

Jak lze z předchozích odstavců dovodit, všechny následující zmíněné varianty jsou pouze různé varianty prvních tří zmíněných modelů distribuce, nezapadající přímo do některé z těchto tří modelů.

3.4 Výhody a nevýhody cloud computingu

Nejčastěji zmiňovanou výhodou cloud computingu bývají nízké náklady, oproti pořízení a provozu vlastní infrastruktury. Toto je ale pravda pouze částečně. Hlavní výhodou jsou tedy nulové pořizovací náklady na infrastrukturu a opakované investice do obnovování. Výdaj za licencování zůstává v ideálním případě stejný, ale přibývá zde náklad za konektivitu, resp. náklady, jelikož se většinou platí za konektivitu i za přenesená data. Přibývá nevyhnutelný měsíční poplatek.

Přesunem infrastruktury do cloudu se též eliminují některá rizika, jako je například riziko selhání a s ním spojené výdaje. Jako další výhoda se udává minimalizace výdajů na vlastní IT support na straně firmy. Ten se reálně eliminuje na podporu klientských zařízení.

Výhody:	**Nevýhody:**
Žádné investice do infrastruktury HW	Pravidelná platba za pronájem prostředků
Žádné rizika spojená s infrastrukturou HW	Platba za použitý výkon
Minimalizace výdajů na vlastní IT oddělení	Poplatky za support poskytovatele
Dynamičnost prostředí	Riziko velkých výdajů za výkon
Eliminace rizik fyzické bezpečnosti	Riziko bezpečnosti dat u třetí strany
Rychlost nasazení	Omezení customizace
Široká paleta možností	Data putují internetem (zabezpečení?)

Tabulka 1: Výhody a nevýhody cloud computingu

Velkou výhodou cloud computingu je dynamičnost. Ta umožňuje prakticky, dle finančních možností, manipulovat s dostupnými prostředky, a tak navyšovat či snižovat dostupný používaný výkon v souladu s aktuálními potřebami. Téměř vše je o nastavení, hradí se pouze prostředky, které se spotřebovávají, používají. Další výhodou jsou garantované dostupnosti a profesionální podpora.

Odpovědět jednoznačně na otázku, zdali se vyplatí cloud computing není možné, bez znalosti konkrétního případu firmy či organizace. Tento segment trhu je velice individuální, a tak se to některým organizacím může vyplatit a jiným naopak velice prodražit.

3.5 Cloudová úložiště

Způsob ukládání dat, který aplikacím umožňuje odesílat data na síť vzdálených serverů, kde běží aplikace pro ukládání dat. K výměně dat dochází buď za pomocí aplikace poskytovatele služby, webovým rozhraním, nebo za pomoci webového aplikačního rozhraní pro organizací provozované aplikace. K takto uloženým datům je možné následně přistupovat odkudkoliv a není třeba se starat o jejich životnost a další rizika spojená s ukládáním, zálohováním a zabezpečením dat. Cloudová úložiště jsou ideálním prostředkem pro zálohování dat, jejich sdílení mezi různými zařízeními uživatele a sdílení mezi různými uživateli.

Většina veřejně dostupných cloudových úložišť nabízí varianty zdarma s různou kapacitou. Velké množství cloudových úložišť podporuje několik způsobů přístupu k datům, od základního webového rozhraní, přes mobilní aplikaci, desktopovou aplikaci, až po API rozhraní pro aplikace a služby třetích stran.

Velmi populární součástí některých úložišť se stala live synchronizace složky souborového systému do cloudového úložiště. Uživatel v takovém případě pouze používá počítač a data umístěná v této lokaci se sami, naprosto nerušivě synchronizují s cloudovým úložištěm.

Velkou problematikou poslední doby, vzhledem ke cloudovým úložištím, zejména ve vztahu ke koncovým nefiremním zákazníkům, je otázka geografického umístění serverů pro uložení dat do cloudového úložiště. Není příliš bezpečné ani moudré mít data příslušníků státu uložena na druhé straně planety, ve státě s odlišnou legislativou, než je ta, jež podléhá

uživatel. Mnoho států tuto problematiku řeší nařízeními, která přikazují IT firmám poskytujícím tyto služby zajistit fyzické umístění dat zákazníků na území státu, či unie.

Výhody:	**Nevýhody:**
Data jsou zabezpečená proti ztrátě	Bezpečnost dat a ochrana soukromí
Snadná práce se soubory odkudkoliv	Nezbytné připojení k internetu
Snadná synchronizace	Nahrávání dat při každé změně (live sync)
Možnosti obnovy smazaných dat	Fyzické uložení dat (legislativa státu)
Snadné sdílení	
Snadné verzování souborů	

Tabulka 2: Výhody a nevýhody cloudových úložišť

3.5.1 Box

Jedna z nejstarších cloudových služeb pro ukládání a synchronizaci dat. V bezplatném formátu umožňuje uložení až 10 GB dat s limitem velikosti souboru 250 MB. Za měsíční poplatek 9 eur pak lze získat pro fyzickou osobu 100 GB prostoru s limitem velikosti souboru 50 GB.

Tato služba umožňuje přístup k datům přes webové rozhraní, mobilní aplikace pro platformy Android a iOS, nabízí i aplikaci pro live synchronizaci, byť s prapodivným rozhraním. Pro komunikaci s aplikacemi třetích stran pak Box nabízí API rozhraní s využitím rozhraní REST API.

Box.com je dlouhodobě rozvíjená služba, která bohužel nenabízí lokalizaci ani podporu pro český trh. V současné době se primárně zaměřuje na firemní zákazníky, a tak služby pro běžné zákazníky mohou působit nepropracovaně.

Výhody:	Nevýhody:
10 GB prostoru zdarma	Dražší placená varianta
Za 13,50 eura za uživatele neomezený firemní balíček	Limit velikosti souboru u bezplatné varianty (250 MB)
Možnost nastavovat oprávnění souborů	Verzování jen s placenými tarify

Tabulka 3: Shrnutí Box úložiště [4]

3.5.2 Dropbox

Populární poskytovatel cloudového úložiště s kvalitní podporou a dlouhodobě ustáleným vývojem. Díky stabilitě a popularitě si tato služba mohla dovolit praktickou eliminaci bezplatného režimu používání, kde je pouze 2 GB úložného prostoru, bez jakýchkoliv nástavbových funkcionalit jako je verzování či obnova smazaných dat. Základní hrazená varianta nabízí za měsíční poplatek 8.25 euro 1 TB úložného prostoru, verzování a live synchronizaci.

Služba umožňuje přístup prostřednictvím webového rozhraní, aplikace pro platformy Andorid a iOS, live synchronizaci za pomocí aplikace pro OS Windows a Mac. Pro aplikace třetích stran je nabízeno plně zadokumentované REST API rozhraní.

Dropbox nemá lokalizaci, podporu ani přizpůsobení pro český trh. Služba má mnoho limitů použití a je spíše orientována na business prostředí.

Výhody:	Nevýhody:
Podpora verzování zdarma	Pouze 2 GB prostoru zdarma
Šifrování dat při přenosu i v uložišti zdarma	Drahé řešení pro jednotlivce
Jednoduché rozhraní	Chybějící funkcionalita

Tabulka 4: Shrnutí Dropbox úložiště [4]

3.5.3 Google Drive

Varianta cloudového úložiště od vyhledávacího giganta. Dlouhodobě stabilní řešení s webovým rozhraním zapadajícím k ostatním kancelářským webovým službám společnosti Google. Bezplatná varianta nabízí 15 GB úložného prostoru s limitem velikosti souboru 5 TB. V bezplatné variantě je i verzování, kde se uchovává poslední 100 změn po dobu 30 dní. Pokud uživatel používá i další služby společnosti Google, jako je Gmail, nebo Fotky Google, pak získává bezplatné uložiště a kapacita zdarma je sdílena mezi těmito službami. Základní hrazená varianta nabízí 100 GB prostoru za 60 Kč měsíčně.

Google umožňuje přístup k datům prostřednictvím webového rozhraní jeho služeb souhrnně zvaných G Suite, mobilní aplikace s podporou platforem Android, iOS a nabízí i live synchronizaci s nebývale vysokou propustností pro stahování. Taktéž je nabízena aplikace pro synchronizaci na běžných x86 strojích s operačním systémem Windows. Pro aplikace třetích stran je zde také dostupné plně zadokumentované RESTful aplikační rozhraní.

Cloudový disk od společnosti Google nabízí jak českou lokalizaci, tak i rozhraní uživatelsky přívětivé a použitelné pro běžné uživatele. Disk je na českém trhu oficiálně nabízen a podporován. Služba je vysoce konfigurovatelná, a tak se firemní varianty mohou odlišovat od těch pro běžné uživatele.

Výhody:	**Nevýhody:**
Podpora verzování zdarma	Zdarma pouze 15 GB
Šifrování dat při přenosu i v uložišti	
Limit velikosti souboru 5 TB	

Tabulka 5: Shrnutí Google Drive [4]

3.5.3.1 Drive API

Poskytované aplikační rozhraní je moderní a jednoduché. Existuje v několik verzích, které jsou separátně oddělené, každá majoritní verze má vlastní knihovnu. Verze jsou dokonale popsané, zadokumentované a existují k nim velice kvalitní návody pro práci s nimi včetně ukázkových řešení v mnoha programovacích jazycích.

Primárně úložiště pracuje s identifikátory objektu v podobě GUID. Tento moderní přístup umožnuje vložit do názvu objektu (složka, soubor, odkaz) i tak exotické znaky jako je lomítko či některý ze speciálních znaků. Celé aplikační rozhraní pak poskytuje všechny nezbytné základní metody pro práci nad úložištěm i několik nadstavbových.

3.5.4 OneDrive

Jedná se o jedno z nejznámějších cloudových úložišť, se kterým se dozajista setkala většina uživatelů počítačů díky jeho hluboké integraci do operačních systému Windows od verze 8. Dříve bylo toto úložiště známo pod názvem SkyDrive, který byl v Evropské unii v rozporu s právy britské společnosti Sky. V základním bezplatném tarifu nabízí OneDrive 5 GB diskového prostoru a propojení s online i offline variantami jejich kancelářských balíků. Součástí je i verzování a archivace smazaných souborů pro pozdější obnovení. Toto obnovení je v nespecifikovaným způsobem limitováno a OneDrive sám upozorní, že je uloženo velké množství smazaných souborů. Velikost jednoho souboru je zde limitována na 15 GB. Základní placený balíček datového úložiště nabízí 100 GB za 50 Kč měsíčně.

Tvůrce a provozovatel OneDrive, společnost Microsoft nabízí pro správu úložiště přehledné a uživatelsky přívětivé online webové rozhraní, které je součástí Office online i v bezplatné variantě. Dále jsou k dispozici aplikace pro operační systémy Android, Apple iOS i pro dožívající Windows Mobile 10. Díky velice dobře fungující integraci do operačního systému Windows je nabízena live synchronizace s podobně jako u Google drive vysokou propustností. Dostupné je REST API rozhraní pro použití s aplikacemi třetích stran.

OneDrive je již dlouhou dobu stabilní službou. Díky kvalitní lokalizaci do českého jazyka a podpoře českého trhu i oblíbenou službou. Na rozdíl od konkurence nabízí i alternativní způsoby navýšení základního úložného prostoru, jako například navýšení základního bezplatného úložiště za zapnutí synchronizace fotografií na platformě Windows Mobile 10.

Dále Microsoft nabízí mnoho balíčků, kde je OneDrive součástí za zvýhodněnou cenu či dokonce zdarma.

Výhody:	**Nevýhody:**
Výhodné balíčky a rodinné tarify	Pouze 5 GB zdarma
Zálohuje i nastavení počítače s Windows	Nešifrované úložiště
Hlídání smazaných souborů	Limit velikosti souboru 15 GB

Tabulka 6: Shrnutí OneDrive [4]

3.5.4.1 Graph API

Pro práci s OneDrive poskytuje RESTful aplikační rozhraní. Rozhraní je obstojně zdokumentované, ale bohužel dokumentace je vždy validní primárně vůči nejnovější poskytované verzi. Knihovna pro práci s poskytovaným rozhraním pak není určená pouze pro OneDrive, ale je to univerzální nástroj pro práci s aplikačními rozhraními poskytovanými společností Microsoft. Aplikační rozhraní i soubor knihoven pod stejným názvem provozuje i společnost Facebook pro jimi poskytované služby, ale jedná se o dvě naprosto odlišné aplikační rozhraní.

Část API pro práci s úložištěm OneDrive používá k primární identifikaci identifikátory typu GUID, ale v jednotlivých metodách se nelze vyhnout práci s textovou interpretací názvu a cesty k souboru. To způsobuje mnoho problémů se znaky v názvu souboru a desinterpretaci případů kdy je název zašifrovaný. Rozhraní obsahuje duplicitní metody, které vznikly v průběhu vývoje. Tyto metody pak pracují naprosto rozdílně s danou operací. Informace o tom, kterou z dostupných duplicitních metod použít pro danou operaci lze dohledat v dokumentaci jen velice obtížně. Často je rychlejší uchýlit se ke komunitním webům než prohledávat dokumentaci API.

3.5.5 Mega

Méně známé cloudové úložiště vytvořené zakladatelem legendárního úložiště pirátských souborů Megaupload, které bylo v roce 2012 zablokováno a postupně zlikvidováno americkou vládou. V základním tarifu nabízí 50 GB bezplatného úložiště bez limitu velikosti souborů. Základní myšlenkou tohoto uložiště je soukromí a bezpečnost, proto je postaveno na End-to-End šifrování, kde v teoretické rovině má k nešifrovaným datům přístup pouze vlastník hesla, nebo povolená osoba. Základní hrazený balíček pro nepodnikatele stojí měsíčně 5 eur a obsahuje 200 GB prostoru. Datový přenos je limitován 1 TB.

Mega nabízí kvalitní webové uživatelské rozhraní, aplikace pro mobilní operační systémy Android, iOS i Windows Mobile 10. Dostupná je i live synchronizace pro běžné počítače s velmi dobrou funkcionalitou. Propustnost synchronizace není z nejrychlejších, ale pro běžné použití postačí. Mega nenabízí běžné REST API rozhraní, ale místo toho poskytuje komunikační knihovnu pro vývojáře aplikací.

Toto cloudové úložiště je poměrně mladé, oproti konkurenci, ale svým svébytným způsobem si jistě již našlo své místo na trhu. Služba není oficiálně podporována pro Českou republiku, ale i přesto se v některých částech webového rozhraní setkáme s češtinou.

Výhody:	**Nevýhody:**
50 GB prostoru zdarma	U nižších tarifů dražší cena za GB
Neomezená velikost souboru	Limitace přenesených dat
Absolutní zabezpečení dat	Nemá verzování

Tabulka 7: Shrnutí Mega úložiště [4]

3.6 Přímé porovnání cloudových úložišť

V přímém porovnání vybraných cloudových úložišť, kde byla zvolena ta nejznámější s bezplatnou kapacitou minimálně 5 GB, není na první pohled jasný vítěz. Podle dostupné kapacity zdarma na plné čáře vítězí Mega cloudové úložiště. Druhým je pak, s přičiněním uživatele a využití všech bonusů, OneDrive.

Dostupné veřejné cloudové úložiště není možné přímo srovnávat s užitím metodiky, jelikož výsledky takového srovnání by byli značně zavádějící. Každá ze zmíněných služeb je zaměřena na jiného koncového zákazníka a poskytuje jiné benefity a ceník na základě neveřejných nabídek a služeb. Z tohoto důvodu například Box vypadá v tabulce jako drahý a celkově slabý, což je důsledek zaměření na korporátní sféru.

Název	**Prostor zdarma**	**Cena za GB**	**Limit velikosti souboru**	**Rychlost synchronizace (Up / Down)**	**Verzování**	**Šifrování dat**	**End to End šifrování**
Box	10 GB	2,30 Kč	250 MB (5 GB)	8 Mbps / 30 Mbps	Placené	Ano	Ne
Dropbox	2 GB + bonusy	0,26 Kč	Není	8 Mbps / 30 Mbps	Ano, 30 dní	Ano	Ne
Google Drive	15 GB	0,3 – 0,6 Kč	5 TB	6 Mbps / 70 Mbps	Ano	Ano	Ne
Mega	50 GB	0,1 – 0,64 Kč	Není	8 Mbps / 30 Mbps	Placené	Ano	Ano
OneDrive	5 GB + bonusy (až 30 GB)	0,05 – 1 Kč	15 GB	13 Mbps / 50 Mbps	Ano	Ne	Ne

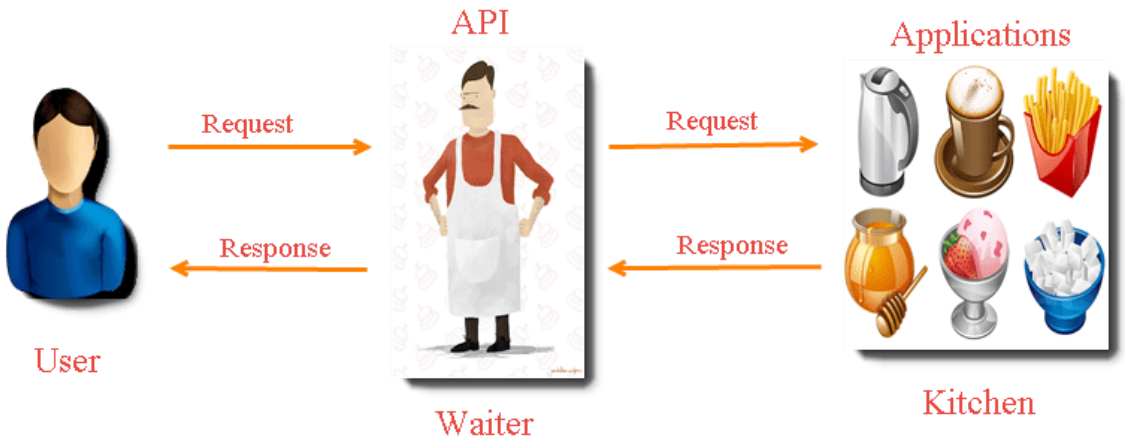
Tabulka 8: Přímé porovnání cloudových úložišť [4]

Z pohledu ceny za uložený GB data je nejvýhodnější použití vyšších balíčků OneDrive. Zde se bohužel nevyhneme kombinaci s Office 365, která jsou neodnímatelnou součástí a prodražují balíček. Když je brán v potaz pouze balíček za úložný prostor, pak se stává nejvýhodnějším z pohledu ceny za GB Mega, ale i ten má onen pomyslný háček. U Mega je limitován přenos souborů, který tím celý balíček znevýhodňuje. S ohledem na použitelnost bez problémů, limitů a ceny za GB tak vítězí Google Drive, jehož nabídky balíčků neobsahují přidané produkty, webové rozhraní je jednoduché a použitelné.

Z pohledu bezpečnosti je absolutním vítězem Mega, jelikož vynucuje použití End-to-End šifrování uživatelských dat. To má za důsledek, že nikdo vyjma majitele či povolené osoby, nemá šanci přečíst uložená data. Má to i odvrácenou stránku v podobě ztráty šifrovacího klíče, což způsobí ztrátu přístupu k uloženým datům. Úložiště ale dostatečně důrazně tuto situaci řeší při vytváření účtu.

4. APLIKAČNÍ ROZHRANÍ

Pod pojmem API, jak jsou zkráceně nazývána aplikační rozhraní, je skryta skupina definicí, procedur, funkcí, tříd a protokolů pro vytváření a integrování aplikací. Jedná se o rozhraní pro programování aplikací. Aplikační rozhraní v nějaké podobě je neoddělitelnou součástí téměř všech dnes vyvíjených programů od operačních systému až po webové aplikace. Hlavní užití je zejména pro oddělení přístupu k prostředku, kdy se pro získání dat nevolá přímo prostředek, ale pouze jeho rozhraní, tedy API.



Obrázek 5: API workflow ukázka [14]

4.1 Historie

Koncept rozhraní pro programování aplikací vznikl v 60. letech, dlouho před vznikem osobních počítačů. API se obvykle používalo pro knihovny operačních systémů. Pojem API vznikl s největší pravděpodobností v roce 2000 v disertační práci s názvem Architektonické styly a designy v návrhu síťových a softwarových architektur pana Roye Fieldinga. Pan Fielding s určitostí vytvořil pojem webového aplikačního rozhraní, ale samotný koncept aplikačního rozhraní, již byl v té době široce znám.

V 70. letech 20. století došlo, díky rozmachu distribuovaných systémů, i k rozšíření aplikačních rozhraní. Objevily se metody, které umožňovaly vzdálený přístup k procedurálnímu rozhraní API a současně obcházely režii typického programátora prostřednictvím zaobalování dat. Obzvláště velkým pokrokem v této oblasti byl Message

Oriented Middleware. Toto řešení se zaměřilo na specifické potřeby Enterprise Application Integration (EAI) tím, že vytvářelo mosty ke starším mainframovým systémům. Průkopníkem v této oblasti byla IBM MQSeries, která zavedla proprietární technologii mainframe messagingu do celého světa. Tyto systémy řešily problémy point to point závislostí jednotlivých systémů, a tak urychlovaly distribuované výpočty.

Současná doba je specifická pro vysokou četnost užití Web API, které se začalo prosazovat postupně od roku 2005 a přibližně od roku 2010 se Web API využívá v masivním měřítku. Velký rozmach aplikačních rozhraní byl umožněn zejména díky rozvoji World Wide Webu a souvisejícího http protokolu.

4.2 Dělení aplikačních rozhraní

Aplikační rozhraní lze rozdělit do mnoha skupin podle různých parametrů. Základní dělení je na veřejné a soukromé. Další možností, jak rozčlenit aplikační rozhraní, je to, jak jsou použita, resp. kde jsou využita.

4.2.1 Dělení podle přístupnosti

Veřejné aplikační rozhraní je ideální způsob, jak poskytnout veřejnosti přístup k datům, k nimž chceme přístup umožnit a vzájemně nevytvářet přímé přístupy do interních systémů a programů společnosti.

Další možností je **partnerské aplikační rozhraní**, kdy je přístup k rozhraní omezen na specifickou skupinu požadovaných lidí. Přístup je zde běžně limitován ověřením vůči serveru identit, či ověření znalosti něčeho co znám (typicky heslo), či mám (např. token).

Předposlední variantou je **soukromé aplikační rozhraní**, které se využívá zejména pro interní účely firem, aplikací a infrastruktur. Tato aplikační rozhraní bývají zřídka kdy vystavena přímo do internetu a většinou jsou dobře chráněna, ať už definicí v zdrojových kódech nebo zabezpečením interní sítě.

Výše zmíněná aplikační rozhraní lze také libovolně kombinovat, a takovému způsobu užití se pak říká **kompozitní aplikační rozhraní**. Takové aplikační rozhraní typicky obsahuje synchronní sekvence volání a zpracování požadavků a dat.

4.2.2 Dělení podle způsobu použití

Nejčastějším způsobem použití aplikačního rozhraní je **knihovna**, popřípadě **framework**. Zde je aplikační rozhraní jako popis očekávaného chování, definuje název, vstupy a výstupy. Programátor jiných částí kódu má znalost, jak volat metodu z knihovny či frameworku, co může očekávat a co musí dodat bez znalosti interní struktury knihovny, pro kterou bylo rozhraní vytvořeno. Jedno aplikační rozhraní může mít více implementací, například lišících se datovým typem vstupních a výstupních hodnot.

Dalším poměrně častým způsobem užití aplikačních rozhraní jsou rozhraní mezi **operačním systémem** a aplikací. Tyto aplikační rozhraní jsou definovány standardem POSIX, popřípadě některými proprietárními standardy. Zde se jedná o komunikaci mezi systémem a aplikací, která jej využívá k vlastnímu chodu.

Samostatnou skupinou jsou volání **vzdálených aplikačních rozhraní** či procedur. Zde se jedná o standardizovanou komunikaci mezi vzdálenými systémy, které tuto komunikaci podporují a umožnují tak spolupráci nezávisle na použité technologii. Příkladem takové kooperace je databázové aplikační rozhraní.

Pro komunikaci skrze prostředí internetu se hojně využívají **webová aplikační rozhraní**, prostřednictvím nichž může docházet k interakcím mezi různými aplikacemi. Tato aplikační rozhraní jsou typicky vystavena do internetu s různým způsobem ověření. Webové aplikační rozhraní je většinou definováno jako sada specifikací v podobě zpráv nad protokolem. Obvykle se k definici struktury takových zprávy využívá XML jazyk, případně modernější varianta JSON.

XML:	**JSON:**
Široká podpora datových typů	Podporuje pouze text a čísla
Zaměřen na dokumenty	Zaměřen na data
Poskytuje vyšší bezpečnost	Je méně bezpečný

Tabulka 9: Porovnání XML a JSON [5]

Většina, v dřívějších dobách vytvořených, webových aplikačních rozhraní byla založena na protokolu SOAP (Simple Object Access Protocol), tedy na služby orientované architektuře (SOA). V současnosti je tendence využívat namísto výše zmíněného jiný protokol a to REST (Representational state transfer), což je architektura orientovaná na zdroje. Existuje ještě jeden starší protokol, než výše zmíněný SOAP, a to XML-RPC. Ten používal specificky definovaný XML formát pro přenos dat a je tak méně náročný na přenesená data a podstatně jednoduší než SOAP. Také umožnuje využití JSON, místo XML pro definici struktury zprávy.

SOAP:	**REST:**
Striktní pravidla a pokročilé zabezpečení	Jednoduchý pro použití
Založen na funkci	Založen na datech
Přenáší více dat	Minimálně zatěžuje síť

Tabulka 10: Porovnání SOAP a REST [5]

4.3 REST

Pojem REST (Representational State Transfer) je architektura rozhraní pocházející z disertační práce Roye Fieldinga, jednoho ze spoluautorů protokolu HTTP, zveřejněné v roce 2000. Příchod REST se často spojuje také s pojmem Web 2.0, které souhrnně popisuje v té době nové technologie využívané pro vytváření webových stránek. Na rozdíl od známějších architektur jako jsou XML-RPC či SOAP je REST orientován na data. Procedury pro přístup k datům jsou definovány na úrovni webového serveru a samotný REST už pouze definuje, jak se přistoupí k datům. REST definuje jednotné a jednoduché způsoby přístupu ke zdrojům. Zdrojem pak mohou být data či daty popsané stavy aplikace. Všechny zdroje mají vlastní identifikátor URI a REST definuje čtyři základní metody pro přístup k nim.

V kontextu tohoto rozhraní je důležité rozlišovat pojmy REST a RESTful. Pod zkratkou REST se skrývá standard, tedy architektonická definice toho, co musí takto označené rozhraní splňovat (architektonický vzor). RESTful je pak implementace onoho vzoru. Pokud hovoříme

o RESTful rozhraní, pak tvrdíme, že má implementované vše minimálně dle REST specifikace.

4.3.1 Metody přístupu ke zdrojům

REST implementuje čtyři základní metody, které jsou obecně známé pod označením CRUD, tedy vytvoření dat (Create), získání požadovaných dat (Retrieve), změnu (Update) a smazání (Delete). Tyto metody jsou implementovány pomocí odpovídajících metod http protokolu. [6]

4.3.1.1 GET (Retrieve)

Základní metoda pro přístup k zdroji dat je metoda GET. Metoda, se níž se setkává každý uživatel HTTP denně, tedy požadavek na stažení stránky.

```
GET /api/user/johndoe
Host: www.mujserver.cz
```

Každý zdroj, na který chceme klást dotaz, musí mít svůj vlastní unikátní identifikátor (URI). Použitím HTTP GET požadavku získáme data z konkrétního přesně definovaného zdroje. Požadavek GET na aplikační rozhraní je shodný s požadavkem na webovou stránku, v případě, že si jej daná služba nemodifikovala. Stejně jako u webové stránky je pak možné přidat různé parametry pro filtrování a další nezbytné dodatečné funkce.

4.3.1.2 POST (Create)

Odeslání dat na aplikační rozhraní je opět shodné s běžným HTTP a jeho metodou POST. U metody POST v době odeslání není znám identifikátor zdroje, jelikož ještě neexistuje. Z tohoto důvodu se zde používá dohodnutý společný identifikátor, tzv. endpoint.

```
POST /api/user/create
Host: www.mujserver.cz
username=bobdoe&footsize=66
```

Po odeslání POST požadavku na aplikační rozhraní by server měl vrátit odpovídající HTTP kód, v případě úspěchu 201 – Created a v případě neúspěchu patřičný chybový kód.

4.3.1.3 DELETE

Smazání zdroje je možné prostým zavoláním metody DELETE. Požadavek vypadá téměř identicky s GET požadavkem, kde za pomoci URI přesně definuji zdroj dat. V praxi bývá volání metody DELETE problematické, a tak se nahrazuje metodou POST s vhodným parametrem, že daný zdroj má být smazán.

```
DELETE /api/user/johndoe
Host: www.mujserver.cz
```

```
POST /api/user/johndoe
Host: www.mujserver.cz
delete=true
```

4.3.1.4 PUT (Update)

Požadavek na změnu zdroje přesně definovaného pomocí URI. Operace změny samotná je podobná operaci vytvoření, s tím rozdílem, že voláme existující záznam a v těle je pouze to co chceme změnit. U metody PUT platí stejná problematika jako u metody DELETE, ale může se stát, že bude nepodporovaná. Pak je nezbytné opět využít metodu POST a specificky upravená data.

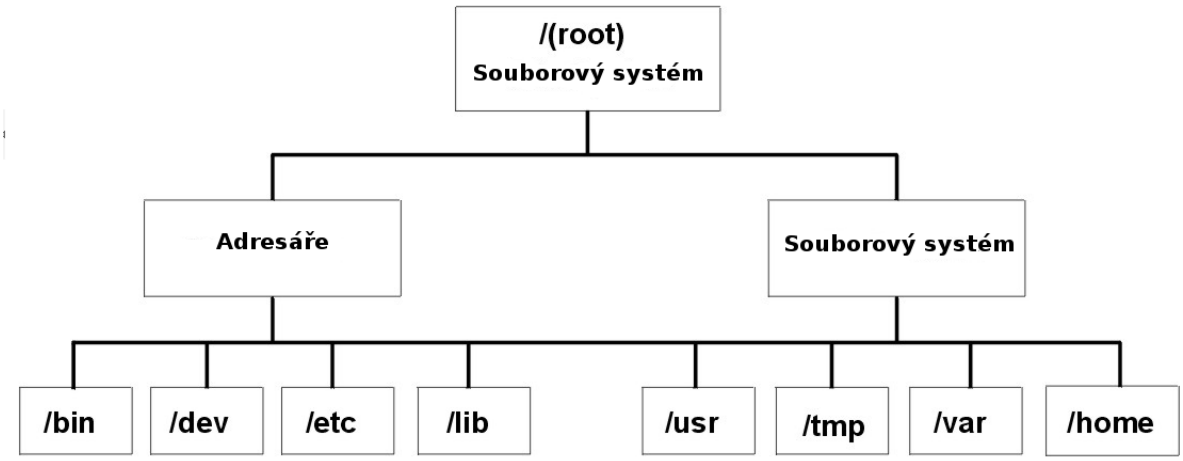
```
PUT /api/user/johndoe
Host: www.mujserver.cz
footsize=45
```

```
POST /api/user/johndoe
Host: www.mujserver.cz
update=true&footsize=45
```

5. SOUBOROVÝ SYSTÉM

Jedná se o způsob organizace informací, tedy souborů a dat o nich, ukládaných na paměťová média (HDD, SSD, CD). Souborový systém definuje způsob vytváření, ukládání a přístupu k souborům a adresářům. Spravuje dostupnou kapacitu na médiu, která je danému souborovému systému přidělena. Údaje o uložených datech jsou soustředěny do několika na sebe navazujících tabulek, tvořících logickou strukturu disku.

Jako **soubor,** v kontextu souborového systému, rozumíme posloupnost bytů uloženou na datovém nosiči. **Adresářem** se pak rozumí oblast disku, která může obsahovat soubory a další podřízené adresáře. **Kořenový adresář** je pak kořenem stromové struktury a je tak nadřazen všem souborům a adresářům. V unixových systémech se kořenový adresář značí znakem lomítko (/) a je společný pro všechna připojená média. V operačním systému Windows je značení kořenového adresáře podobné, ale je zde navíc písmenko daného média a lomítko je zpětné (C:\ ).



Obrázek 6: Znázornění struktury souborového systému [19]

Jedním z nejčastěji používaných souborových systémů dnešní doby je NTFS, což je výchozí souborový systém většiny počítačů s Windows. Je to nástupce dříve hojně využívaného FAT, který má mnohé nedostatky, kvůli kterým se od něho ustoupilo, pro použití jakožto souborového systému pro operační systém. Dalším známým souborovým systémem je EXT, který je v několika verzích využívaný na Linuxových operačních systémech.

Jedním z nejpokročilejších operačních systému dnešní doby je ZFS, který je výchozím souborovým systémem pro operační systém Solaris.

5.1 Virtuální souborový systém

Tento pojmem v závislosti na definici souborového systému lze definovat jako reprezentaci dat a metadat získaných z libovolného zdroje. V operačním systému je virtuální souborový systém použit jako abstraktní vrstva nad různými druhy fyzických souborových systémů. Ty implementují jednotné rozhraní, které virtuální souborový systém používá a očekává. Realizace tohoto rozhraní se ovšem u různých souborových systémů liší. [7]

Virtuální souborové systémy lze vytvářet nezávisle na fyzických souborových systémech. Mohou být vytvářeny nad libovolným typem dat, za předpokladu že má hierarchickou strukturu, v ideálním případě stromovou.

V důsledku oproštění se od fyzické vrstvy ukládání souborů jsme schopni vytvářet nové pohledy na data a zcela nové interpretace. Tímto způsobem můžeme přizpůsobit data na míru určité aplikaci. S takto upraveným systémem práce s daty pak může uživatel pracovat naprosto bez rozdílu od běžných souborových systémů. Virtuální souborový systém lze díky tomu označit nejen za abstraktní vrstvu nad fyzickým souborovým systémem, ale i za abstraktní vrstvu nad daty obecně.

5.2 Souborová oprávnění, bezpečnost a šifrování

Z hlediska bezpečnosti je potřeba upozornit na fakt, že virtuální souborový systém bude vykonávat operace nad interpretovanými daty s oprávněními uživatele, který jej spustil. [8] Z výše uvedeného vyplývá, že je riskantní vytvářet či spouštět virtuální souborové systémy pod privilegovaným účtem operačního systému.

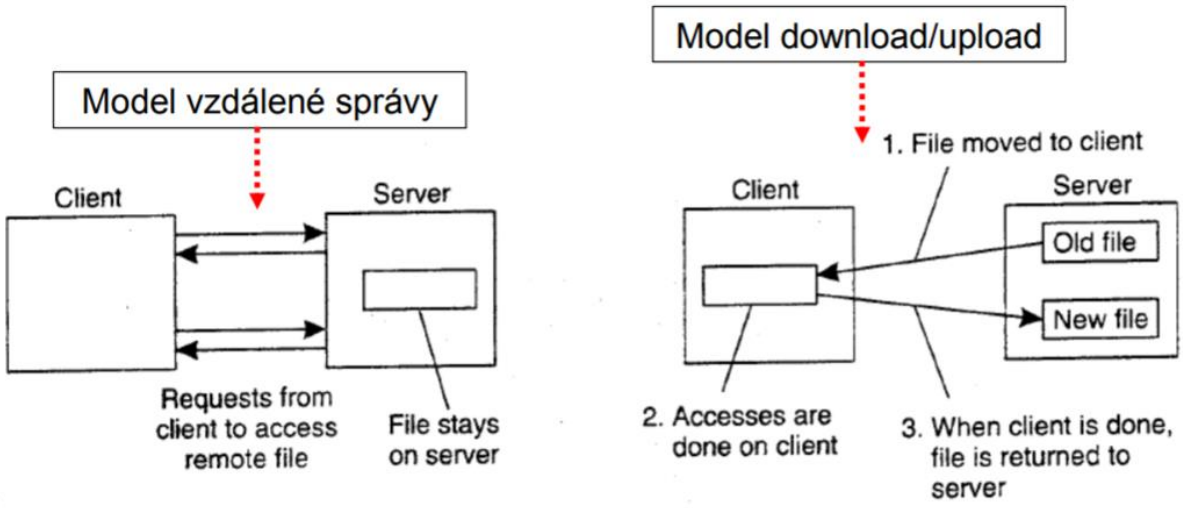
Při použití virtuálního souborového systému, který pracuje s daty uloženými mimo prostředky lokálního stroje, popřípadě lokální sítě, je nezbytné zajistit zabezpečený přenos dat a zabezpečení dat samotných při uložení. Pro tyto účely je vhodné uvažovat hashovací funkce pro ověření integrity a originality dat a šifrovací funkce pro zabezpečení úniku dat.

V kontextu cloudových úložišť je naprosto běžné, že uložená data podléhají hloubkové analýze poskytovatele služby za účelem zlepšení reklamních nabídek a sociálního inženýrství. Mnoho cloudových úložišť definuje, že fyzická data jsou šifrována a v případě úniku nemohou

být rekonstruována, avšak jediné úložiště (Mega), definuje, že jakožto poskytovatel služby nemají žádný způsob, jak přistoupit ke klientem uloženým datům. Z tohoto důvodu se zde vytvoření virtuálního operačního systému nad více cloudovými službami samo nabízí a v kombinaci s šifrováním dat před přenosem samotným se jedná o bezpečnou alternativu. Samotný přenosový kanál, který je dnes zabezpečen pomocí SSL3 nebo TSL, téměř ve 100 % případů, už není vyloženě nezbytný.

5.3 Distribuovaný souborový systém

Distribuovaným systémem souborů je ve své podstatě běžný síťový souborový systém s jedním podstatným vylepšením, a to odolností proti výpadku zdroje dat. Uživateli umožňuje pracovat se vzdálenými daty stejně jako s lokálně umístěnými daty. Hlavní motivací pro tvorbu distribuovaného souborového systému je navýšení kapacity souhrnného úložiště, kdy máme možnost využít všechny dostupné prostředky, které by v případě lokálních úložišť nebyly využity. Nejběžnější architekturou pro distribuovaný souborový systém je klient – server, ale existují i plně distribuovaná řešení.

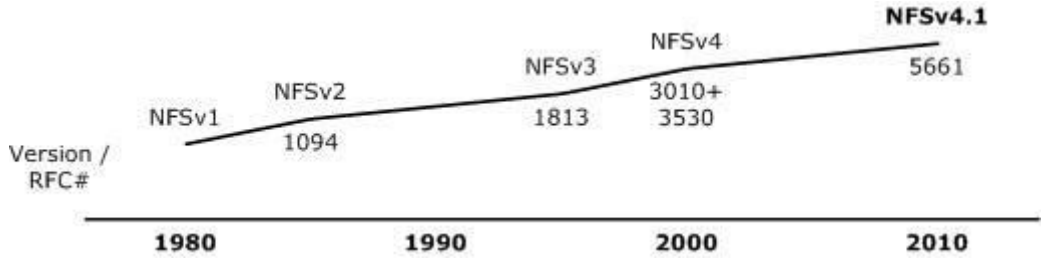


Obrázek 7: Možnosti klient/server implementace [20]

5.3.1 NFS

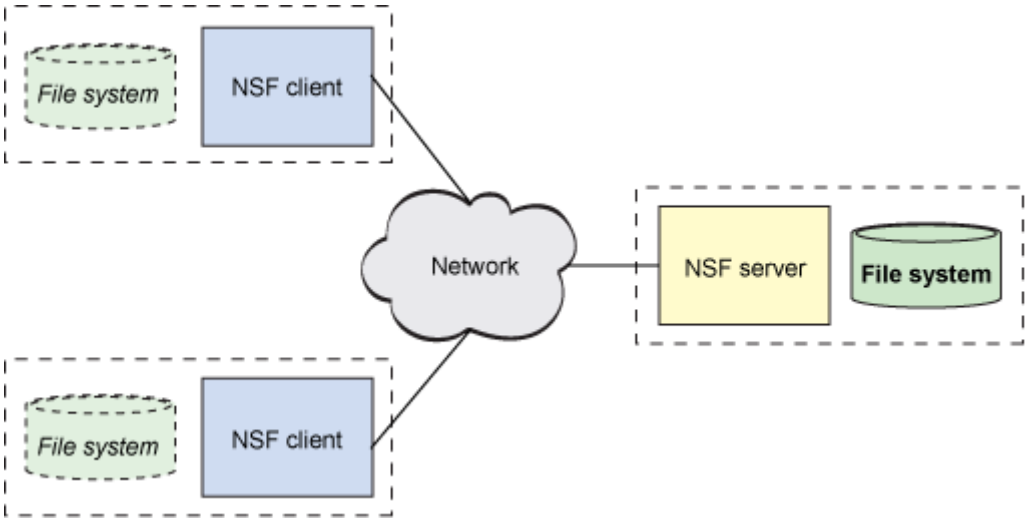
Jedná se o první moderní síťový souborový systém vyvinutý společností Sun Microsystems okolo roku 1980 (NFS = Network File Systém). Poměrně brzy po uvedení, NFS prošel procesem RFC, byla vydána druhá verze NFS, tedy NFSv2. Druhá verze tohoto

síťového souborového systému byla standardizována a její popularita rychle rostla. V současné době se již běžně používá NFSv4.1, který byl publikována pod RFC 5661.



Obrázek 8: Časová osa vývoje NFS [21]

NFS je založeno na architektuře klient – server. Na serveru je nasazen sdílený souborový systém a uložiště, ke kterému se klienti připojují. Klienti používají uživatelské rozhraní do sdíleného systému souborů připojeného v místním souborovém systému klienta. NFS není systém souborů v tradičním slova smyslu, ale protokol pro vzdálený přístup k souborovým systémům.

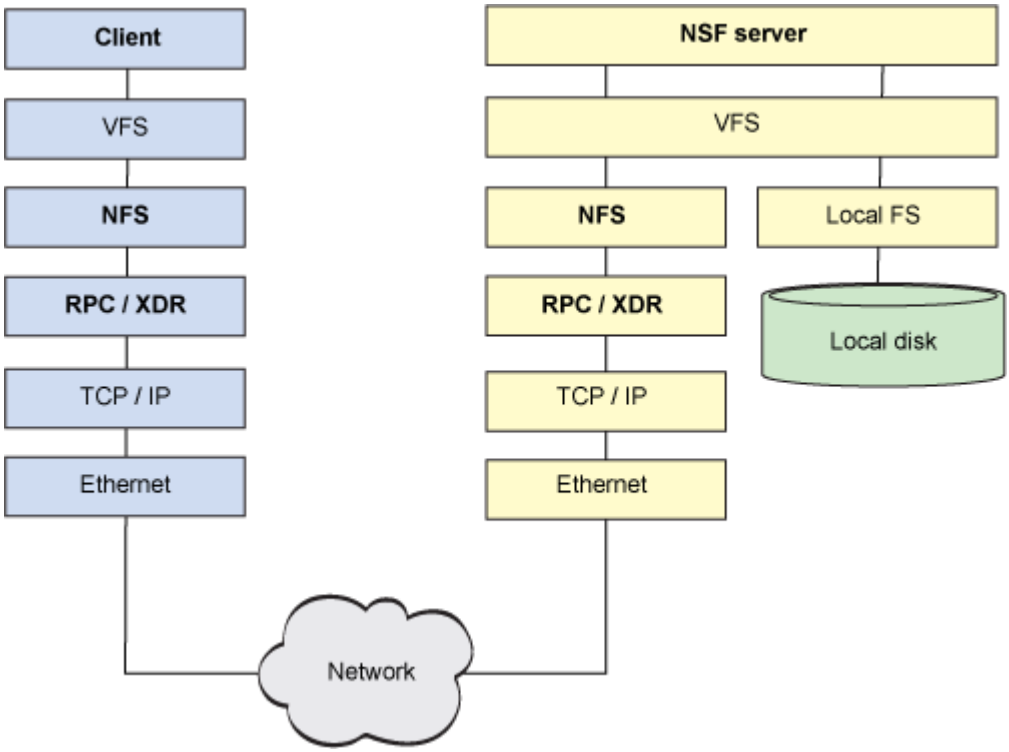


Obrázek 9: NFS klient – server architektura [21]

Moderní počítačové systémy již podporují současnou podporu více virtuálních souborových systémů na jednom hostiteli (standardizování ISO 9660 na CD-ROM a ext3fs). Na Linuxových operačních systémech byl implementován VFS (virtuál filesystem switch), který určuje, pro která úložiště je daný požadavek určen a pro který souborový systém musí být použit, aby vyhověl požadavku. Díky tomu je NFS připojitelný souborový systém

a v systému se tváří jako každý jiný. Jediný rozdíl NFS je v tom, že požadavky na něj nemusí být uspokojeny v místě vytvoření, ale mohou pro dokončení procházet sítí.

Jakmile se zjistí, že je požadavek určený pro NFS, VFS předává instanci NFS a to si převádí I/O operace na NFS procedury. Takto vytvořené proceduře se následně provede vzdálené volání procedury (RPC). Požadavek na NFS se odbaví jak procedura a následně vystaví opět jako I/O operace. Celý proces je obdobný jak na straně serveru, tak i na straně klienta. Pro přenos dat v dřívějších dobách NFS používalo UPD (Universal Datagram Protocol), ale pro zajištění vyšší spolehlivosti se začal používat TCP.



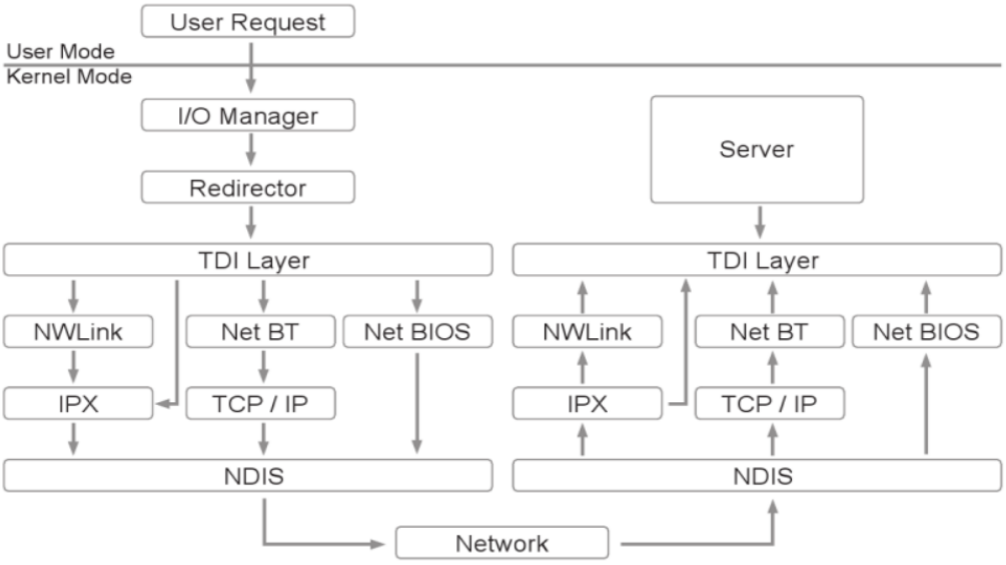
Obrázek 10: UNIX implementace NFS [21]

U sítí s vyšší latencí implementuje NFSv4 tzv. složený přístup, kdy dochází k seskupování více RPC volání do jednoho PRC požadavku. Tím se minimalizuje režie okolo vyřizování jednotlivých RPC žádostí a tím se sníží i využití sítě.

5.3.2 SMB (CIFS)

Proprietární síťový komunikační protokol pro operační systémy Windows od společnosti Microsoft, původně vyvinutý na půdě IBM. Zkratka SMB skrývá název Server Message Block

a zkratka CIFS Common Internet File Systém. Druhé zmíněné označení je synonymum, ale používá se minimálně. Tento síťový protokol neslouží pouze pro přístup k síťovým souborovým systémům, ale umožňuje i připojení tiskáren, sériových portů, autentizaci a další komunikaci. Pro účely zprovoznění protokolu SMB na linuxových operačních systémech byl vytvořen projekt Samba, kdy se za pomoci reverzního inženýrství vytvořila linuxová varianta tohoto proprietárního Windows protokolu.



Obrázek 11: Implementace SMB [22]

Nejčastěji používá protokol SMB pro komunikaci NetBIOS, což původně bylo síťové rozhraní (transportní a relační vrstva). Adresování probíhalo za pomocí 16 znakového jména. NetBIOS byl nesměrovaný a ne hierarchický. V novějších verzích se již pracuje na relační vrstvě nad protokolem TCP/IP (port 139).

6. ZABEZPEČENÍ DAT

Pokud hovoříme o možnostech zabezpečení přenášených a ukládaných dat, pak hovoříme zejména o šifrování, hashování a kódování. Při šifrování se jedná o proces, při kterém dochází k transformaci nezabezpečených dat s využitím kryptografie na zabezpečenou podobu s užitím definovaným šifrovacím klíčem. Míru bezpečnosti a odolnosti takto zabezpečených dat přímo určuje použitá šifra a zejména pak užitý režim v případě blokových šifer. Za zmínění zde stojí zejména symetrická šifra AES a asymetrická šifra RSA.

Symetrické šifry jako je výše zmíněný AES jsou způsobem zabezpečení, kde se užívá pro šifrování i dešifrování stále stejný klíč. AES pak patří mezi blokové šifry, takže je nezbytné volit i vhodný mód, který určuje, jakým způsobem budou data šifrována a samozřejmě i zarovnání, které určí způsob, jak se budou data doplňovat, pokud nepasují na velikost bloku.

Asymetrické šifry jsou specifické tím, že pracují s veřejnou a tajnou částí šifrovacího klíče. Například zmíněný RSA je založen na rozkladu prvočísel a v kombinaci s vhodnou metodou výměny klíčů (DiffieHellman) se jedná o velmi bezpečný a v současné době hojně využívaný způsob zabezpečení.

Je potřeba rozlišovat hashování a šifrování dat. Šifrování dat je obousměrný mechanismus, tedy data zašifrovaná lze dešifrovat a získat tak nezměněná původní data. Naopak, hashování je za ideálních podmínek, tedy pokud je daný hashovací mechanismus bezpečný, jednosměrné. Hashováním zvyšujeme bezpečnost komunikace tak, že vytváříme jedinečný otisk dat a jsme tak schopni ověřit fakt, že daná data nebyla změněna. Příkladem hashovacích funkcí budiž například stařičký MD5 (dnes již překonaný – není jednosměrný) a modernější SHA2, který v současné době poskytuje ideální poměr výkonu a bezpečnosti.

Posledním, na počátku zmíněným, pojmem je kódování. Tento pojem zaštiťuje činnost, kdy se data transformují do jiného formátu. Proces je vždy známý a obousměrný, tady data lze opět dekódovat. Tyto algoritmy se používají zejména při přenosech dat, která obsahují znaky, které by mohly být desinterpretovány. Příkladem takové funkce pro kódování je pak Base64.

6.1 Advanced Encryption Standard (AES)

Tento algoritmus je také znám pod starším označením Rijndael, což je složenina příjmení původních belgických autorů. V současné době je AES podmnožina bloku šifer označovaných jako Rijndael. Jedná se symetrickou a blokovou šifru u níž dochází k šifrování a dešifrování dat za použití shodného klíče.

Pro svou funkčnost AES využívá tzv. substitučně permutační síť, čímž je rychlá jak v softwarové, tak i hardwarové implementaci. Díky tomu je velikost bloku přímo závislá na velikosti klíče (například 128b, 192b, 256b). S délkou klíče pak úzce souvisí další část operace šifry AES a to je počet iterací transformace, neboli kolikrát se aplikuje transformační proces na původní nešifrovaný blok textu.

Počet iterací	**Délka klíče**
10 iterací	128 bitů
12 iterací	192 bitů
14 iterací	256 bitů

Tabulka 11: AES iterace ku počtu bitů

Šifra AES byla vyvinuta v USA a je zde stejně jako v dalších státech světa uznána jako vhodná pro užití při šifrování tajných a přísně tajných dokumentů, dle definice těchto pojmů v jednotlivých státech. Moderní procesory, tedy architektury Intel Core a AMD Ryzen podporují instrukční set zvaný AES-NI. Díky tomu je užití této šifry velice rychlé a bezpečné.

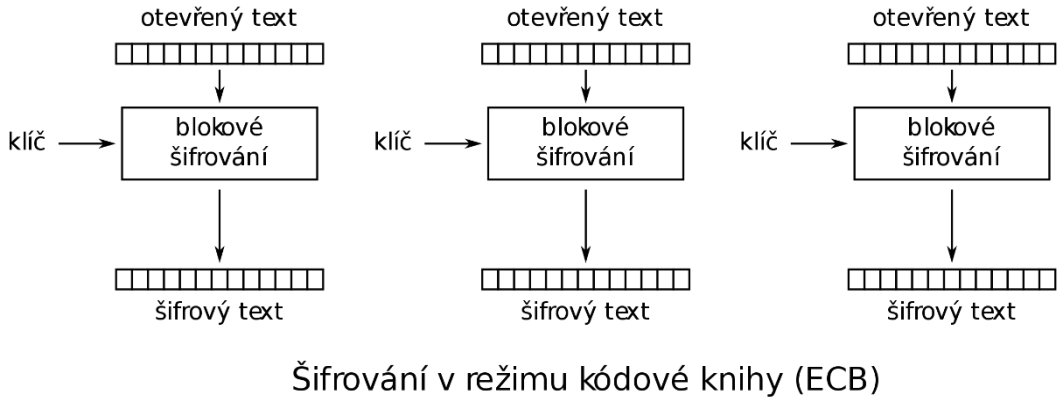
6.2 Provozní režimy/módy blokových šifer

Šifra samotná není využitelná, pokud nemáme definováno, jak ji použít. Tuto informaci poskytuje mód blokové šifry, který udává předpis, jak zvolenou blokovou šifru užít a zašifrovat data tak, aby byla i dešifrovatelná. Většina režimů pro svou funkci využívá kromě klíče i takzvaný inicializační vektor, což je jedinečný náhodně vygenerovaný soubor bajtů,

který v kombinaci s klíčem vytváří výsledný šifrovací klíč. Inicializační vektor si pak sebou data nesou, aby bylo možné užít tuto náhodnou sekvenci při dešifrování. Použitím inicializačního vektoru se předchází situacím, kdy opakované zašifrování totožného nezašifrovaného textu shodným klíčem vede k vytvoření pokaždé shodného šifrovaného textu.

6.2.1 ECB (Electronic Code Book)

Režim, běžně zvaný jako kódová kniha, je jedním ze základních a nejjednodušších. V tomto případě se aplikuje šifra na každý jednotlivý blok nezávisle na ostatních. Tím dochází k situaci, kdy blok otevřeného textu odpovídá bloku šifrovaného textu. To může vést k situacím, kdy se původní informace dat neskryje plně, ale jen rozmělní. Například v případě barevného obrázku dojde při použití tohoto jednoduchého módu pouze k posunu do černobílého spektra a částečnému rozpixelování. Nesená informace je zakryta jen částečně.

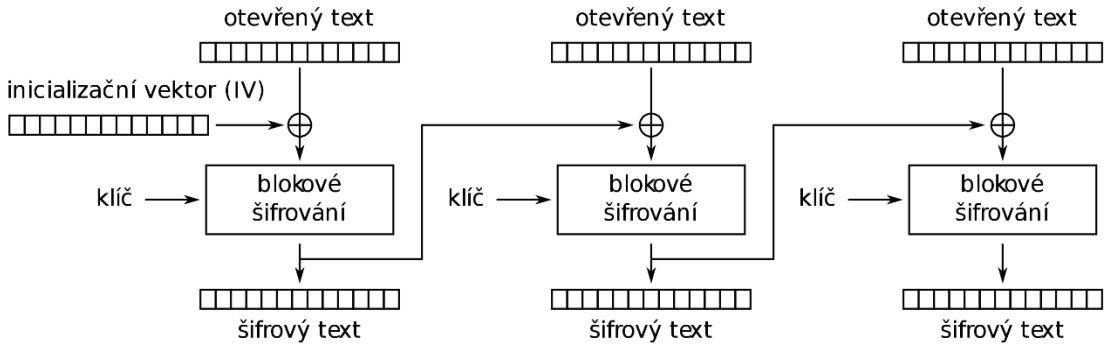


Šifrování v režimu kódové knihy (ECB)

Obrázek 12: Šifrovací mód ECB [24]

6.2.2 CBC (Cipher Block Chaining)

V tomto režimu dochází k zřetězení informace a původní inicializační vektor je použit pouze pro první blok. Výstup prvního bloku se pak používá jako inicializační vektor pro šifrování dalších bloků. Tento postup zřetězení je proveden až do konce šifrovaného souboru bloků dat. To je také jeden z důvodů, proč musí být shodná délka klíče s délkou bloku. Pro toto zřetězení se využívá logické operace zvané XOR.

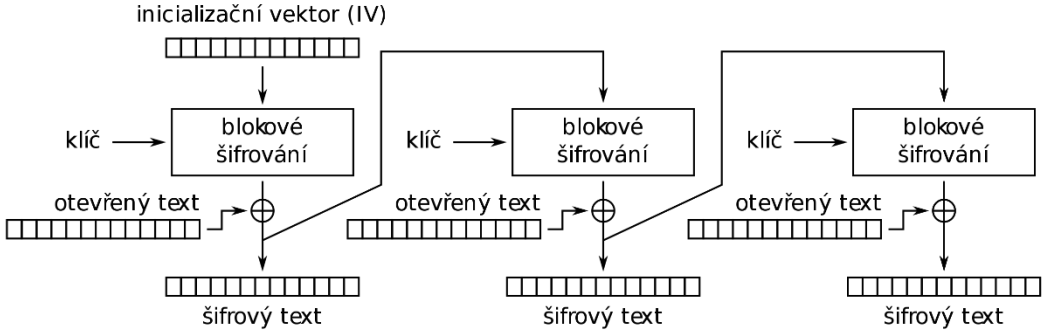


Šifrování v režimu řetězení šifrových bloků (CBC)

Obrázek 13: Šifrovací mód CBC [25]

6.2.3 CFB (Cipher FeedBack)

Tento režim vychází z předchozího, ale mění pořadí prováděných operací. Vlivem této záměny se logická operace XOR aplikuje až na výsledek operace šifrování, zatímco u CBC je tato operace provedena nad nešifrovaným textem. Tato malá změna umožnila decentní zjednodušení dešifrovacího procesu.

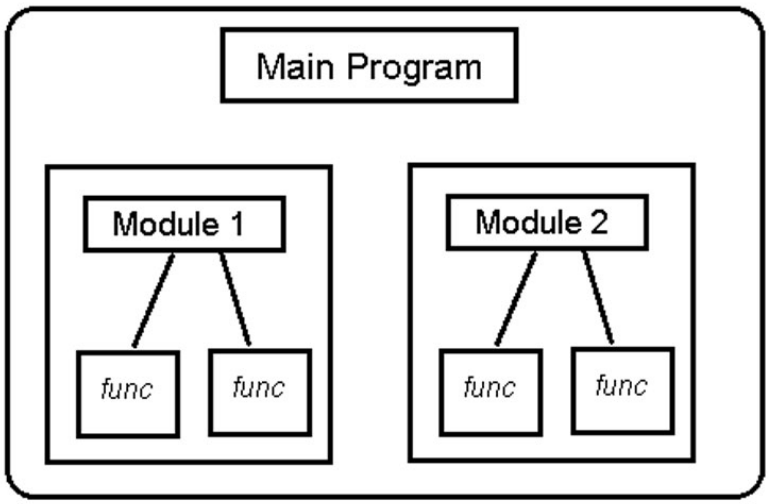


Šifrování v režimu šifrové zpětné vazby (CFB)

Obrázek 14: Šifrovací mód CFB [27]

7. MODULÁRNÍ PROGRAMOVÁNÍ A PLUGINY

Jedná se o techniku návrhu softwaru s důrazem na oddělení nezávislých funkčních částí programu. Každý modul obsahuje vše nezbytné pro provádění jedné určité funkcionality. Modul je popsán rozhraním, které definuje vstupy a v závislosti na nich i výstupy a je detekovatelné ostatními moduly. Implementace obsahuje samotný kód, který odpovídá prvkům deklarovaným v rozhraní.



Obrázek 15: Diagram modulární struktury [23]

Modulární programování úzce souvisí se strukturálním programováním a objektově orientovaným programováním. Všechny mají společný cíl a to rozložit mnohdy složitý a dlouhý kód projektů na malé funkční celky a tím usnadnit konstrukci, úpravu a údržbu. [9]

Knihovnou nazýváme předkompilovanou kolekci rutin, či modulů programu. Má podobu binárního souboru. Umožnuje tak změny, opravy kódu bez nutnosti kompilovat znovu celý projekt.

Plug-In se rozumí softwarové rozšíření programu, které není nezbytné pro jeho chod, ale může doplnit, rozšířit či upravit základní funkcionalitu. Plugin se může skládat z knihoven a tak sám být modulárním prvkem programu.

8. ANALÝZA A NÁVRH PROGRAMU

V praktické části se práce zabývá vytvořením programu pro umožnění konsolidace a zabezpečení cloudových úložišť. V této kapitole je rozebírána prvotní analýza a teoretický popis zvolených prostředků a postupů pro následnou realizaci. Analýza a její výsledky se odvíjí od informacích zpracovaných v praktické části.

Hlavním účelem aplikace je umožnit uživateli veřejných cloudových úložišť spravovat svá data z jednoho místa pro větší množství dostupných služeb, a to s možností tyto úložiště slučovat do společného prostoru. Myšlenka a zpracování této aplikace pracuje pouze s veřejně dostupnými úložišti v jejich bezplatné variantě, tedy ve variantě s velmi omezenou kapacitou, značně omezenými možnostmi, službami a prakticky neexistující bezpečností dat, zejména pak těch osobních.

Na trhu v současné době není dostupné řešení, které by nabízelo slučování úložišť a vytváření nad nimi konsolidovaného úložného prostoru. Dostupná řešení pro práci zcela výjimečně umožnují více než práci s jednotlivými úložišti standardním způsobem jako je stažení, nahrání či přejmenování souboru. I další možnosti navrhované aplikace jsou na trhu poměrně ojedinělé, a to možnosti šifrovat obsah dat. Další možnost v podobě zašifrování názvů souborů žádná z veřejně dostupných aplikací nenabízí. Zrcadlení nahrávaných souborů, či jejich dělení a nahrávání pak opět nenabízí žádná veřejně dostupná aplikace.

Vytvořená aplikace bude plnohodnotnou aplikací postavenou na Windows Forms se všemi pozitivy i negativy tohoto frameworku. Podporováno bude přidávání modulů s implementací podpory jednotlivých úložišť za chodu aplikace s možností postupného povolování a zakazování takto přidaných modulů. Aplikace též umožní zrcadlit uživatelská data mezi vícero veřejných cloudových úložišť a snížit tak riziko ztráty či poškození dat vlivem dlouhodobého skladování či poškození lidským vlivem. Též bude možno zašifrovat odesílaná data, a to nejen jejich obsah, ale i názvy odesílaných složek a souborů.

Pro umožnění postupného přidávání modulů, a to zejména při běhu aplikace bude využito Managed Extensibility Frameworku z dílny společnosti Microsoft, který je součástí .Net Frameworku od verze 4.0. Tento framework umožnuje dynamicky načítat knihovny implementující určité aplikační rozhraní za běhu programu, a to bez potřeby referencování do základního projektu.

Pro realizaci programu byl zvolen programovací jazyk C#, technologie .Net framework a Windows Forms. K vývoji bylo zvoleno vývojového prostředí Visual Studio 2019 Comunity edition. Framework samotný byl zvolen ve verzi nejnovější v době vytváření projektu, tedy ve verzi 4.7.2. Aplikace samotná bude pak vyvíjena jakožto standardní Windows aplikace pod plnohodnotnou variantou .Net frameworku v již zmíněné verzi.

O zabezpečení dat mezi klientem a serverem se, vzhledem k užití webových API, bude starat běžný protokol HTTPS. Samotné šifrování uživatelských dat, konfiguračního souboru uložiště a názvů nahrávaných dat bude implementováno s užitím sdíleného klíče (v našem případě se jedná o tzv. application secret, tedy o neveřejnou informaci uvnitř programu). K šifrování samotnému bude využita standardní knihovna System.Security.Cryptography, která je součástí zvoleného frameworku. Konkrétně zde bude využito algoritmu AES.

8.1 Grafické rozhraní

Pro implementaci uživatelského rozhraní bude využito frameworku Windows Forms. Framework samotný je léty prověřený, stále podporovaný a vyvíjený. Byl zvolen zejména z důvodu autorovy hlubší znalosti a praxe. Grafické rozhraní bude obsahovat všechny standardní ovládací prvky Windows aplikací. Okno se ve výchozím nastavení bude otvírat v zmenšené podobě na středu obrazovky a bude obsahovat všechny nezbytné údaje na výchozím zobrazení.

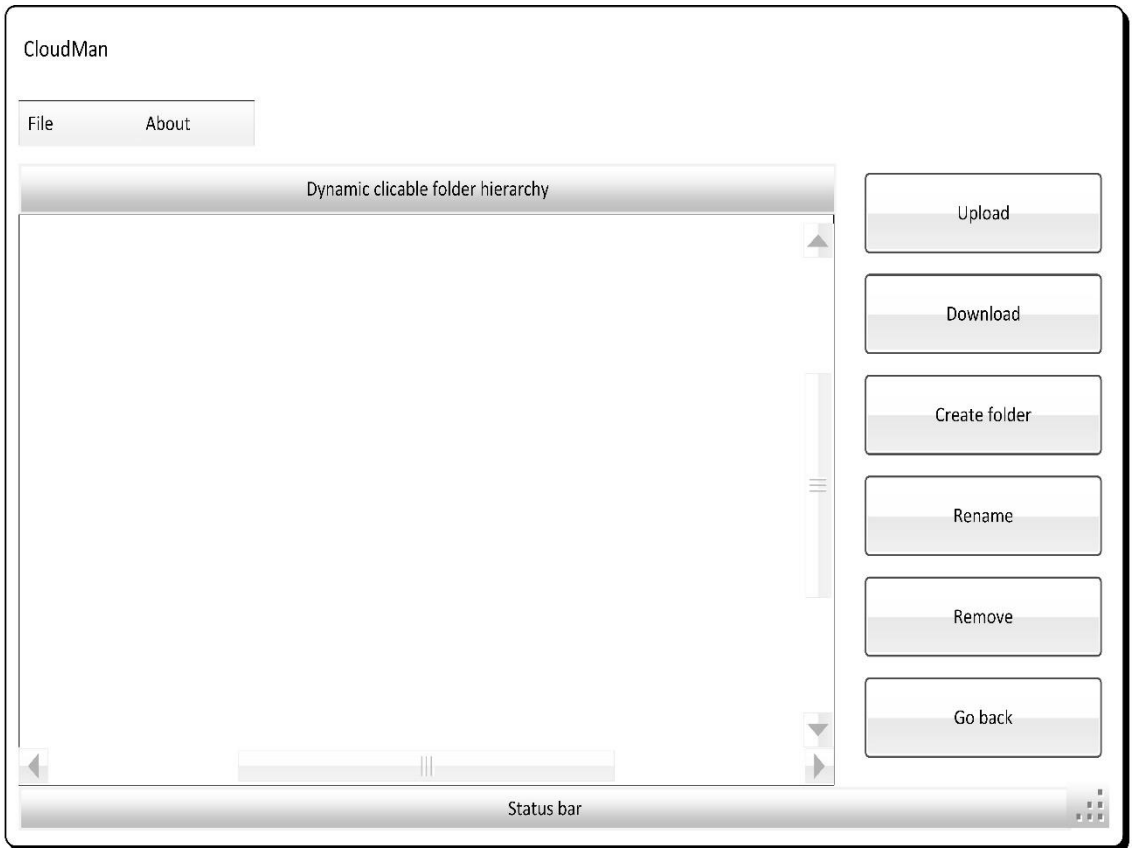


Diagram 1: Wireframe diagram návrhu grafického rozhraní aplikace.

Aplikace se bude otevírat ve standardním okně Windows a bude umožněno měnit velikost okna. Okno aplikace bude možné minimalizovat do oblasti systémových ikon a tato ikona bude mapována na základní menu skrze pravé tlačítko myši. Rozhraní bude obsahovat menu s možnostmi základního ovládání aplikace, nápovědy a informací o aplikaci.

Většinu plochy grafického rozhraní bude zabírat plocha se seznamem dostupných souborů, která bude realizována výčtovým seznamem a zobrazovat základní informace o daném souboru (název, ikona, datum poslední změny, velikost a další). Nad seznamem dostupných souborů a pod hlavním menu aplikace bude realizován prvek zobrazující nadřazené prvky v hierarchii (hierarchii složek od kořenové až po aktuálně zobrazenou).

V pravé části okna budou zobrazena hlavní ovládací tlačítka. Prvním bude tlačítko Nahrát soubor, které inicializuje okno pro výběr nahrávaného souboru a poté zavolá middleware pro nahrání souboru. Druhým tlačítkem bude možnost stáhnutí souboru, které je podmíněné označením libovolného neprázdného řádku v seznamu. Toto tlačítko bude inicializovat výběr umístění ke stáhnutí souboru a následně volání middleware pro stáhnutí souboru. Dalšími ovládacími prvky typu tlačítka jsou vytvoření nové složky, přejmenování a smazání. Poslední dva zmíněné budou podmíněny označením libovolného nenulového řádku v seznamu dostupných souborů a všechny zmíněné budou volat middleware pro zprocesování své funkcionality. Posledním tlačítkem v pravé části bude možnost jít v hierarchii o jednu úroveň zpět. Tuto vlastnost bude mít též seznam hierarchie zobrazený nad seznamem dostupných položek.

Posledním prvkem základního zobrazení aplikace bude proužek zobrazující status aplikace a další užitečné informace, které mohou uživateli osvětlit co aktuálně aplikace dělá. V pravém spodní rohu se bude nacházet prvek grafického rozhraní, který umožňuje snadno, táhnutím myši, měnit velikost okna.

V první položce hlavního menu se budou nacházet položky restartovat aplikaci, obnovit zobrazení seznamu položek, nastavení a ukončení aplikace. V druhé položce pak bude možno nalézt nápovědu a informace o aplikaci. Aplikace samotná bude mít vlastní ikonu a stejně tak prvky v seznamu dostupných položek budou složky a soubory rozlišeny za pomocí ikony.

Důležitým prvkem grafického rozraní bude okno nastavení, které bude přístupné skrze možnost hlavního menu, ale bude také inicializováno při prvním startu aplikace pro úvodní potvrzení nastavení. Toto okno bude obsahovat volbu typu konsolidace, možnosti šifrování a seznam dostupných modulů. Rozhraní samotné bude zakazování vizuálních prvků a zobrazováním upozornění směrovat uživatele k validnímu nastavení.

8.2 Procesní logika

Grafické rozhraní si bude udržovat instanci logiky aplikace, která bude zodpovědná za samotné načítá dat a zprocesování akcí nad daty. Logika bude udržovat instanci virtuálního souborového systému a provádět nad ním operace nezbytné pro práci s daty a udržení konzistence. Logiku samotnou lze vidět na diagramu níže.

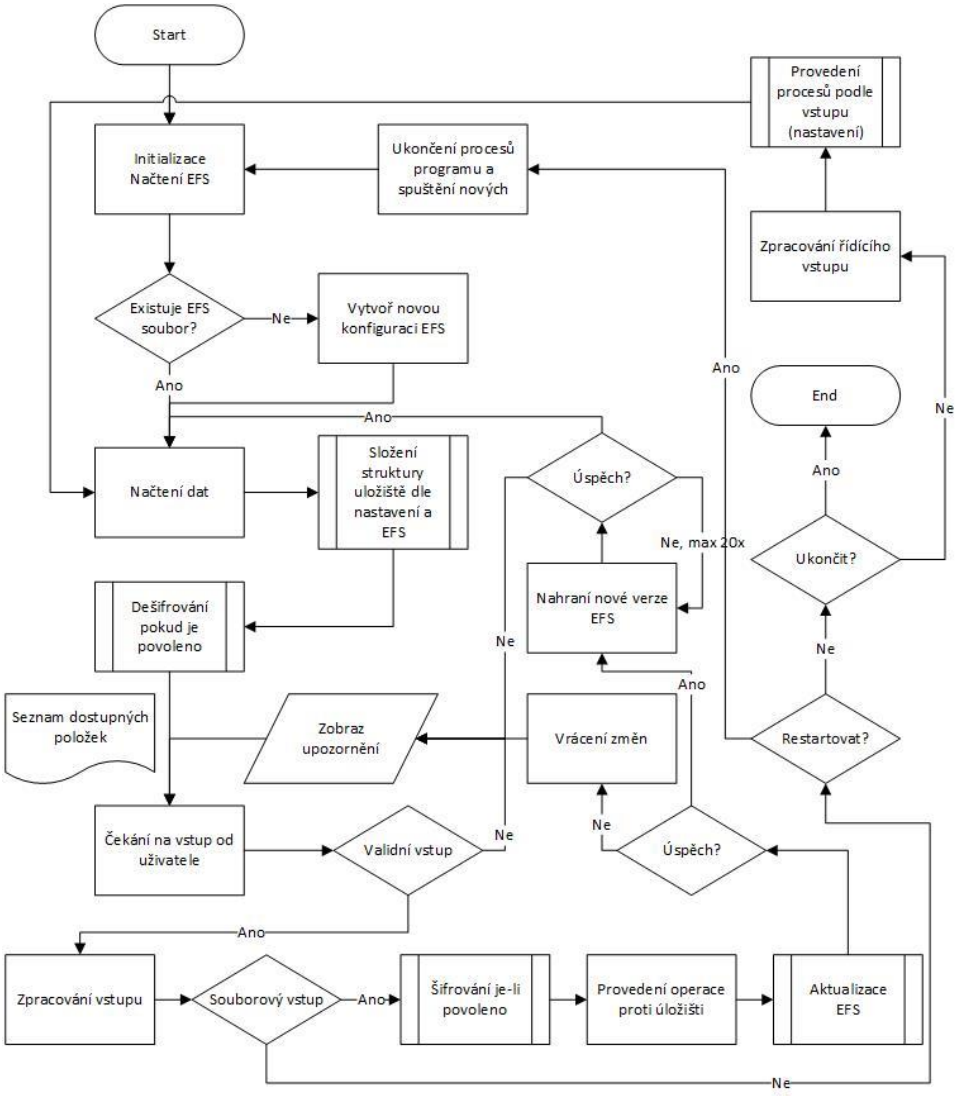


Diagram 2: Procesní logika

Aplikace bude obsahovat dvojí logiku. První bude logika řídící, tedy volby z hlavního menu a nastavení, popřípadě změny nastavení. Druhou bude logika ovlivňující souborové úložiště a procesy nad ním. Jak je na diagramu naznačeno, logika zajišťující šifrování je separována od zbytku souborové logiky.

8.3 Moduly

Aplikace bude definovat rozhraní, které umožní základní operace nad dostupnými úložišti a bude tak definovat operace, které musejí být implementované v každém modulu implementujícím funkcionalitu pro dané úložiště.

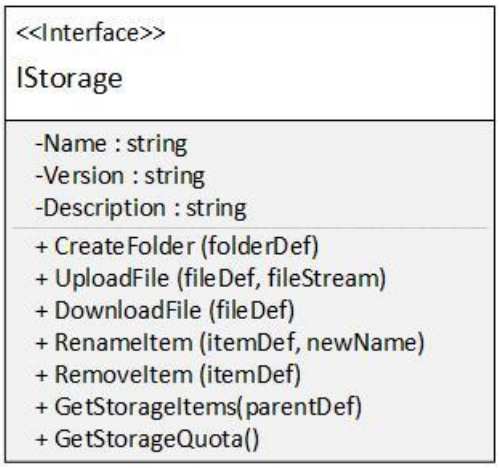


Diagram 3: Návrh rozhraní pro moduly

Takto nadefinované rozhraní bude vytvořeno v separátním projektu tak, aby bylo umožněno jeho referencování jak pro účely aplikace samotné, tak i pro účely užití a implementace v modulech.

8.4 Šifrování

Část aplikační logiky zajišťující šifrování bude zcela separována a řešena nezávisle na zbytku kódu, tak aby byla snadno analyzovatelná a auditovatelná. Tato logika bude implementována s důrazem na jednoduchost a multifunkčnost. Logika šifrování musí být na vstupu schopna přijmou text i datový proud šifrovaného souboru. Na výstupu pak bude vracet stejný datový typ, jaký přijala na vstupu.

Pro šifrování bude využito knihoven integrovaných v .Net Frameworku a bude užit algoritmus AES. Minimální doporučená délka klíče byla stanovena na hodnotě 128 bitů, tudíž velikost bloku bude též 128 bitů a počet iterací minimálně odpovídající této délce klíče. Při šifrování bude využito šifrovacího módu CBC, který patří do kategorie s dostatečnou bezpečností a excelentní rychlostí. Doplnění šifrovaných dat na velikost bloku pak obstará PKCS7 mód.

8.5 Konfigurační management

Práci s konfigurací obstarává integrovaný konfigurační manager v .Net Frameworku. Instanci konfigurace si pak aplikace bude udržovat po celou dobu běhu, jelikož se jedná o kritická data nezbytná pro běh aplikace. Konfigurace se bude ukládat do složky společně s binárními soubory aplikace do souboru s příponou .config. Jedná se o standardizovaný způsob práce s konfigurací v .Net Frameworku.

Prvotní konfigurace proběhne ihned po prvním spuštění, nebo po spuštění, kdy nebude konfigurační soubor aplikace nalezen. Aplikace v případě, kdy nebude dostupný konfigurační soubor nespustí hlavní pracovní okno, ale vyvolá přímo okno nastavení aplikace.

8.6 Virtuální souborový systém

Analýzou problematiky bylo vybráno minimalistické řešení virtuálního souborového systému v podobě serializovatelných objektů, se kterými se za běhu aplikace standardně pracuje jako s objekty jazyka C#. V případě, že bude potřeba konfiguraci nahrát na úložiště, dojde pak k serializaci těchto objektů do textového formátu a uložení do souboru, který bude následně nahrán na dostupná úložiště. Načtení těchto, pro aplikaci kritických dat, pak bude sloužit deserializace, kdy se na základě souboru obsahujícího data původních objektů, vytvoří nová instance objektu pro virtuální souborový systém.

Struktura objektů, symbolizujících virtuální souborový systém, je založena na funkčních módech aplikace. Data pro jednotlivé módy jsou odděleny a nedochází tak v okně aplikace zobrazování dat z nepoužívaných módů. Jediný základní (simple) mód zobrazuje i aplikací

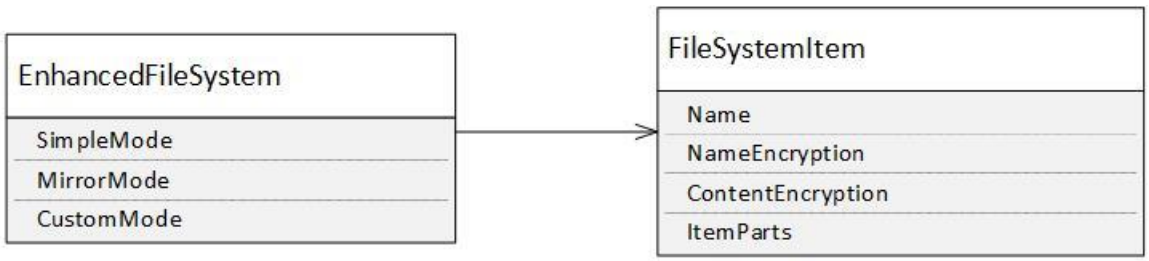


Diagram 4: Návrh tříd virtuálního souborového systému

nespravovaná data na úložišti, a tedy i data z ostatních módů aplikace.

Pro serializaci byl vyhodnocen jako nejlepší dostupný formát JSON, který je minimalistický a neobsahuje zbytečné direktivy a duplicitní data. Tento formát taktéž poskytuje nejlepší výkon pro serializaci a deserializaci. Jedná se o standard, a tak není nutné tuto logiku vytvářet, ale stačí použít některou z dostupných knihoven.

Zvolená struktura souborového systému úzce souvisí se serializací JSONu a optimální strukturou výsledného souboru v poměru s rychlostí a stabilitou takového řešení. Tato varianta je značně čitelnější a decentně rychlejší než v případě užití výčtových typů, což by mohlo způsobit i problémy s nekompatibilními hodnotami.

Objekt popisující položku ve virtuálním souborovém úložišti pak nese pouze nezbytné minimum informací, které slouží k identifikaci objektu a rozklíčování operací nad objektem provedených. Jak lze vidět z diagramu č. 4, objekt obsahuje z reálných položek pouze název, který slouží k identifikaci objektu na úložišti. Další nesené informace slouží k oddělení současného stavu konfigurace se stavem k datu zpracování objektu. Poslední položka pak slouží specificky pro mód aplikace „Custom“, který rozděluje soubor na části a každou část nahrává do jiného úložiště. Tato položka pak nese opět pouze minimální informaci v podobě pořadí dílu a názvu úložiště.

8.7 Definice souboru

Získané informace z úložišť jsou v naprosté většině případů uloženy ve struktuře, která je specifická pro dané úložiště, a tak je nutné pro účely aplikace a její logiky vytvořit obecnou definici souboru, složky či objektu z úložiště tak, aby pokrýval všechny nezbytné informace a kód procesní logiky nemusel obsahovat odbočky závislé na konzumované službě.

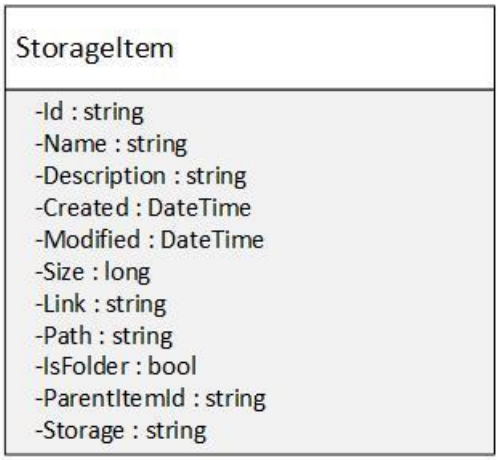


Diagram 5: Návrh třídy pro obecný objekt prvku uložiště

Výše, na diagramu znázorněný objekt, je produktem analýzy výstupních souborů jednotlivých úložišť, která budou implementována. Ne všechna uložiště budou využívat všechny dostupné položky pro ukládání, jelikož je buďto neposkytují, nebo podávají v struktuře a podobě, která je neslučitelná s podáním dat od ostatních poskytovatelů. Ukázkou takového pole je *Path*, které například Google Drive API neposkytuje a místo toho vrací celý rodičovský objekt jako pod objekt žádaného objektu. Naopak OneDrive, tady *GraphAPI*, které je nezbytné pro práci s tímto úložištěm, poskytuje a pracuje s textovou definicí cesty souboru.

8.8 Globální definice a opakovaně užívané hodnoty

Pro účely globálního nadefinování některých proměnných bude vytvořena separátní třída, která bude přístupná napříč aplikací a bude obsahovat všechny nezbytné opakovaně užívané údaje nezbytné pro běh aplikace. Pro tuto třídu nebude možné vytvářet instance, čímž se zajistí stálost údajů takto poskytovaných.

Při vývoji bude užito mnoho opakujících se názvů pocházejících z konfigurace a definovaných v konfiguračním managmentu. Pro eliminování případných problémů s překlepy, špatnou interpretací či nevhodným užitím, budou vytvořeny, pro tyto konfigurační názvy, vlastní objekty výčtového typu. Tyto objekty budou obsahovat všechny, v konfiguračním managmentu i kódu, užité názvy a tím se vyloučí textové definování a všechny problémy s tím související.

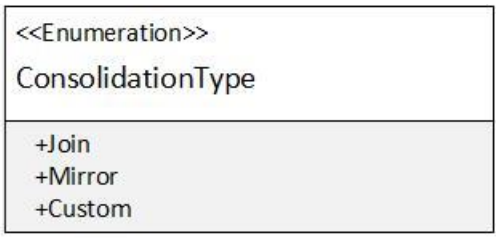


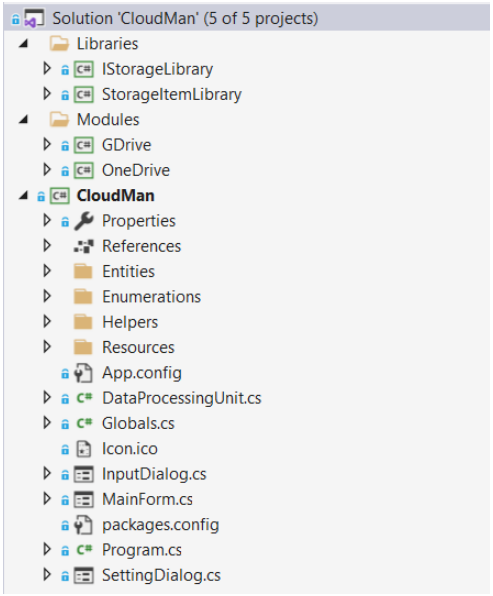
Diagram 6: Výčtový typ pro typ konsolidace

9. IMPLEMENTACE

Tato kapitola práce je věnována popisu řešení, projektů a detailní implementaci jednotlivých částí aplikace. Je zde popsána struktura celého řešení, rozdělení do projektů a následně i detailně rozebrané části implementace včetně vysvětlení co vedlo k takovému řešení, které bylo využito a jaké byly možnosti, či alternativy. K vytvoření implementace bylo využito prostředí Microsoft Visual Studio 2019, programovací jazyk C# v kombinaci s plnohodnotným .Net Framework ve verzi 4.7.2, která byla v době započetí projektu nejnovější a dostupná. Pro tvorbu grafického rozhraní aplikace bylo využito frameworku Windows Forms.

9.1 Základní struktura řešení

Původní řešení (v originále Solution) projektu bylo vytvořeno jako standardní WinForms aplikace ve vývojovém prostředí Visual Studio 2017 Professional, ale bezprostředně poté již bylo pro vývoj aplikace užito nové verze 2019 Community. Základní struktura projektu je tak poplatná tomuto, již letitému frameworku, pro vytváření grafických aplikací pro operační systém Windows. Projekt byl nazván „CloudMan“ jakožto zkratka utvořená ze slov Cloud Manager.



Obrázek 16: Struktura řešení aplikace

Řešení obsahuje celkem 5 projektů, každý se svým specifickým účelem. Prvním projektem je samotný základní projekt s názvem shodným s celým řešením a jeho účelem je definice a základní logika okolo grafického rozhraní. Dále řešení obsahuje projekt obsahující a definující jediné rozhraní (interface) s názvem *IStorageModule*. Účelem tohoto rozhraní je, jak název napovídá, definování objektů a metod, které má následně implementovat projekt každého modulu.

Dalším projektem v řešení je definice objektu nazvaná *StorageItem*. Tento projekt definuje pouze tuto třídu pro užití ve všech částech řešení. Objekt je takto separován především proto, aby bylo možné jej užít napříč řešení bez vzniku nežádoucích referencí, či hůře cyklických referencí, které by znemožnily úspěšné sestavení aplikace.

Výše zmíněné objekty jsou vyseparovány do složky nazvané Libraries neboli knihovny. Název vychází z reálného obsahu této složky po sestavení aplikace, kde se následně nachází DLL soubory (knihovny systému Windows – binární soubory). Další složka v řešení se nazývá Modules. Jak název napovídá, je to složka určená pro jednotlivé implementace modulů, které implementují chování pro jednotlivé cloudové úložiště. Ve výsledném řešení této aplikace obsahuje tato složka 2 projekty. První je implementace modulu pro úložiště OneDrive (dříve SkyDrive) a druhým projektem je implementace úložiště Google Drive.

9.2 IStorageLibrary projekt

Tento projekt, jak již bylo zmíněno, definuje rozhraní pro implementace jednotlivých úložišť a musí být z těchto projektů referencovatelné. Z toho plyne nutnost, aby projekt samotný referencoval pouze nezbytné minimum balíčků a jiných částí z řešení, a to z důvodu eliminace možný problémů při sestavování aplikace. Zejména tím je bráněno vzniku cyklických referencí, které pokud jsou nalezeny způsobí, že vývojové prostředí tuto situaci neumí interpretovat. Visual Studio obsahuje detekce těchto problematických referencí v kódu a při jejich nalezení neumožní spustit sestavení řešení.

Výsledné řešení je mírně odlišné od analýzy popsané v předchozí kapitole. Sktruktura rozhraní byla obohacena o jednu metodu, která ověřuje spojení. Tím se předchází chybě při samotném pokusu o činnost vůči uložišti a snáze se problém nefunkčního spojení identifikuje, zejména pak v kontextu OneDrive, jehož API ne vždy funguje zcela standardně a konzistentně. Dále byly všechny metody, oproti původní analýze, implementovány jako asynchronní, tedy že aplikace nečeká na odpověď modulu a pokračuje v procesu.

Pokračuje až do dokončení, nebo až do chvíle, kdy narazí na operaci vyžadující data od asynchronní operace.

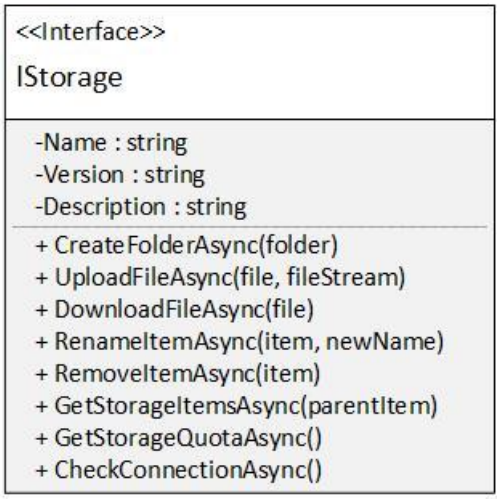


Diagram 7: Interface pro moduly (implementovaná verze)

9.3 StorageItemLibrary projekt

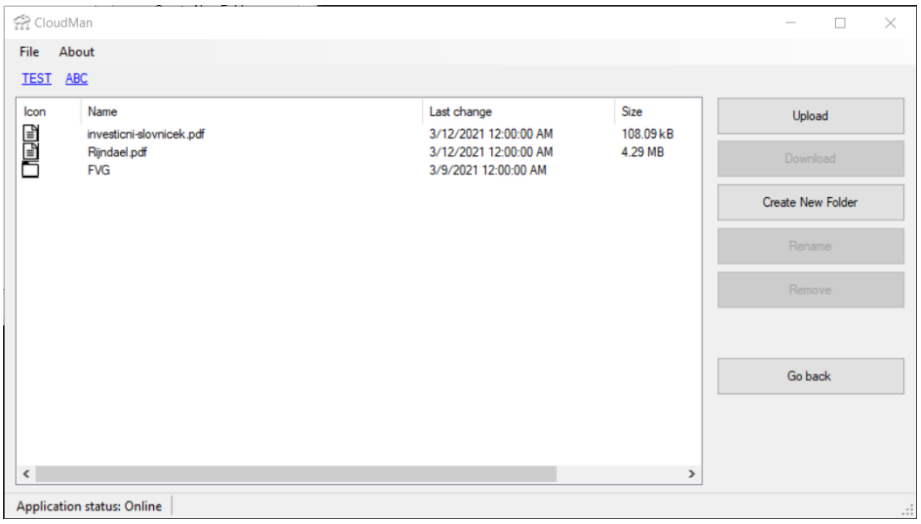
Tato knihovna implementuje jediný objekt, a to třídu s názvem *StorageItem*, který plní roli základní jednotky funkčnosti celé aplikace. Do tohoto objektu se ukládají informace získané z jednotlivých API. Jak již bylo v analýze zmíněno, objekty poskytované napříč aplikačními rozhraními jsou tak odlišné, že bylo zapotřebí je pro vnitřní účely aplikace nahradit vlastní variantou a na úrovni modulů udělat pře-mapování původních objektů na nové. Tímto postupem je možné docílit jednotné podoby objektu symbolizujícího prvek z úložiště napříč aplikací a její logikou.

Navzdory předpokladu nebylo u této třídy možné implementovat referenci na modul tak, aby bylo skrze referenci a identifikátor zjistitelné, z jakého modulu daný objekt pochází. Vznikla by tím cyklická reference, jelikož *IStorageModul* používá *StorageItem* a tudíž je tam reference nezbytná, a tak je zde informace o modulu původu daného prvku, o kterém informuje pouze textová hodnota názvu daného modulu. Tato hodnota musí být unikátní, jelikož slouží jak primární identifikátor napříč aplikací.

Struktura třídy pro objekt prvku úložiště odpovídá zcela původní analýze, která ve zjednodušeném pohledu situaci zobrazovala takto jednoduše, jak to bylo nakonec implementováno.

9.4 CloudMan projekt

Základní projekt celého řešení, který implementuje grafické rozhraní a jeho logiku. Obsahuje hlavní formulář, nazvaný *MainForm*, který implementuje základní okno programu s ovládacím rozhraním tak, jak bylo nadefinováno ve výše popsané analýze grafického rozhraní a s ohledem na vytvořený návrh v podobě wireframe diagramu.



Obrázek 17: Hlavní okno programu

Hlavní okno programu je minimalistické a jednoduché, neobsahuje logiku pro práci s úložištěm, ale pouze logiku pro funkci rozhraní samotného. Tedy logiku, která určuje, co se stane při akcích kliknutí na tlačítka, menu i hierarchické zobrazení složek. Tlačítka pro stáhnutí, přejmenování a smazání jsou z podstaty své funkcionality limitovány povinností mít označený libovolný prvek ze seznamu. Nad polem s výpisem záznamů je implementován event, který hlídá jakoukoliv změnu nad tímto polem a zavolá metodu, která skrze sérii podmínek ověří validitu předchozího nastavení a případně povolí/zakáže části rozhraní.

Instance hlavního okna aplikace si udržuje instance všech nezbytných objektů pro běh aplikace. Tento postarší přístup by bylo možné nahradit užitím IoC a registrací, ale v době počátku implementace tohoto programu mi nebyla existence této technologie známa.

```
public string ApplicationStatusText
{
       get => this.toolStripStatusLabel1.Text;
       set => this.toolStripStatusLabel1.Text = $"{Resources.StatusBarAppStatusName}:
{value}";
}

private Configuration config;
private List<StorageModule> enabledModules;
private AggregateCatalog catalogs;

private StorageItem selectedItem;
private Stack<StorageItem> selectedTree;

[ImportMany(AllowRecomposition = true)]
private IEnumerable<IStorage> availableStorages;
private DataProcessingUnit processingUnit;
private IEnumerable<StorageItem> actualItems;

private readonly ImageList imageList;
```

Ukázka kódu 1: Hlavní instance objektů

Tyto instance jsou užívány napříč celým kódem a například položka *selectedItem* se dostane až do modulu, což je implementace pro jedno z dostupných úložišť. V ukázce jsou také zobrazeny i proměnné užívané pouze pro účely hlavního formuláře. V ukázce lze vidět členění dle skupin užití. První je dle standardu veřejná proměnná (property) a následují kritické běhové soukromé proměnné. Následně je skupina proměnných (fields) pro práci se záznamy. Za nimi následují soukromé proměnné nezbytné pro načtení modulů a mezi nimi i instance procesní logiky, která zpracovává jednotlivé operace nad dostupnými povolenými moduly (úložišti). Instance všech definovaných proměnných se vytvářejí v konstruktoru hlavního okna. Pokud by se zde vyskytla chyba, hlavní okno se nespustí.

Za účelem zpracování výjimek je v základní třídě projektu s názvem Program.cs (základ každé aplikace, který spouští grafické rozhraní) try – cache blok. Všechny zachytnuté výjimky jsou logovány. Nejnižší vrstva aplikace je napsána tak, aby žádná nekritická chyba nezpůsobila ukončení aplikace.

K **logování** chyb, informativních hlášek ale i informací pro debugování je užito frameworku s názvem Log4net. Jedná se o standardní a léty prověřený framework pro logování v .Net aplikaci. Jak je možno vidět v ukázce kódu níže, framework loguje do textového souboru s názvem log.log, který je umístěn v adresáři aplikace. Další nastavení definují, že maximální velikost logu je 5 MB a soubory logů se rotují 5 logů po 5 MB. To znamená, že aplikace udržuje 5 posledních souborů logu o velikosti 5 MB.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
      <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler,
log4net"/>
    </configSections>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2" />
    </startup>
  <log4net>
    <appender name="RollingFileAppender" type="log4net.Appender.RollingFileAppender">
      <file value="log.log"/>
      <appendToFile value="true"/>
      <rollingStyle value="Size"/>
      <maxSizeRollBackups value="5"/>
      <maximumFileSize value="5MB"/>
      <staticLogFileName value="true"/>
      <layout type="log4net.Layout.PatternLayout">
        <conversionPattern value="%date [%thread] %-5level %logger - %message%newline"/>
      </layout>
    </appender>
    <root>
      <level value="ALL"/>
      <appender-ref ref="RollingFileAppender"/>
    </root>
  </log4net>
</configuration>
```

Ukázka kódu 2: Konfigurace log4net

Pro umožnění dynamického načítání modulů za běhu aplikace je zde implementován **Managed Extensibility Framework** (zkráceně MEF), což je soubor knihoven pro vytváření modulárních aplikací. Tento framework umožnuje snadné zapouzdření kódu bez nutnosti definovaných referencí. Aplikace samotná při kompilaci a v prvních fázích spouštění neví žádné informace ani počet dostupných modulů (v kódu nejsou na projekty modulů žádné reference).

MEF pracuje s katalogem dostupných knihoven, v našem případě modulů, který je přímo mapován na konkrétní složku na fyzickém souborovém systému v počítači. V případě této aplikace se jedná o složku *Modules*, která se nachází ve složce aplikace. Sestavení aplikace má nadefinované skripty, které nově zkompilované moduly nakopíruje do této složky

a umožní tak jejich načtení za běhu aplikace. Moduly samotné jsou pak v složce pro moduly umístěny ve vlastních složkách, aby bylo možné rozlišit co patří ke kterému modulu a nedocházelo ke konfliktům knihoven. Různé moduly sebou mohou nést stejné knihovny v různých verzích.

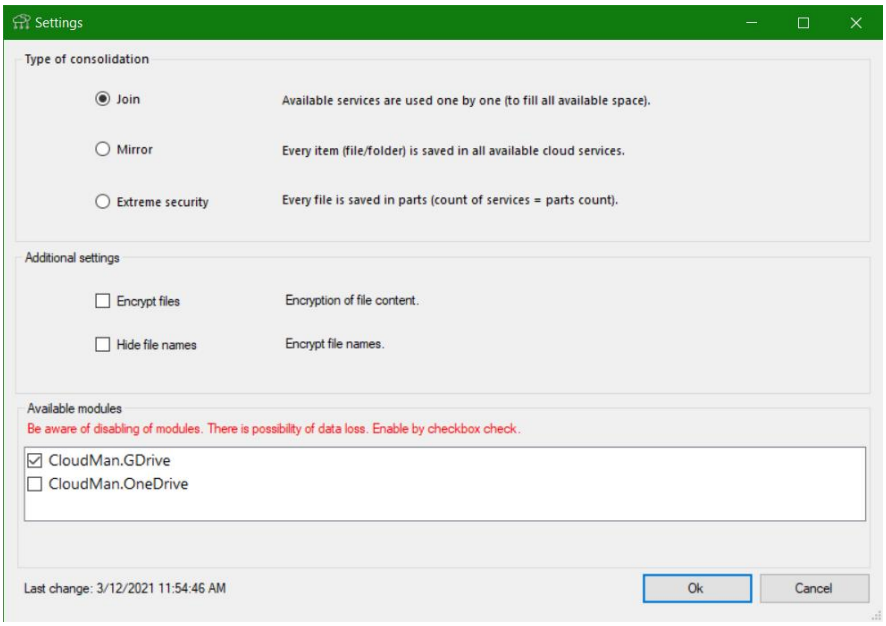
```
/// <summary>
/// Refreshes the catalog.
/// </summary>
/// <param name="sender">The sender.</param>
/// <param name="e">The <see cref="EventArgs"/> instance containing the event
data.</param>
private void RefreshCatalog(object sender, EventArgs e)
{
       catalogs.Catalogs.Clear();
       try
       {
              if (!Directory.Exists(Globals.ModulesFolderName))
                     Directory.CreateDirectory(Globals.ModulesFolderName);

              foreach (var x in Directory.GetDirectories(Globals.ModulesFolderName))
              {
                     var catalog = new DirectoryCatalog(x);
                     catalogs.Catalogs.Add(catalog);
              }
        }
        catch (Exception ex)
        {
              Console.WriteLine(ex.Message);
              MessageBox.Show(ex.Message, Resources.ApplicationName,
MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
}
```

Ukázka kódu 3: Načítání modulů

V ukázce kódu si lze všimnout, že metoda obhospodařující katalog modulů je implementována s parametry podporující event managera. To je z důvodu, že dále v kódu je vytvořen *FileSystemWatcher* (System.IO knihovna), který hlídá změny na souborovém systému v složce pro moduly.

Projekt obsahuje kromě hlavního okna i další okna formulářového typu, která implementují dodatečné sekce nezbytné ke korektnímu chodu aplikace. Prvním takovým dialogovým oknem je **nastavení**. Jedná se o standardní formulář z frameworku Windows Forms a obsahuje kompletní konfiguraci aplikace.



Obrázek 18: Okno nastavení

V první části tohoto formuláře je definice typu konsolidace. Implementována formou výběru jedné z množiny hodnot. Tato volba je přímo závislá na seznamu dostupných modulů, ale volby nejsou zakázány, aby to uživatele nemátlo. Místo toho byla implementována validace, která ověřuje, zda je zvolený typ konsolidace kompatibilní s ostatním nastavením. Zejména pak, zdali je povoleno dostatečné množství modulů pro funkčnost. Pokud není povoleno dostatečné množství modulů pro daný typ konsolidace, automaticky se zakáže tlačítko Ok a je zobrazena hláška o nevalidní konfiguraci.

Druhá část formuláře zobrazuje dodatečná nastavení, která nejsou nezbytná pro základní chod, ale umožnují tak zvýšení bezpečnosti. Nad těmito tlačítky není implementována žádná validace, jelikož jsou plně kompatibilní se všemi typy konsolidace bez ohledu na množství zvolených modulů.

Poslední součástí nastavení je seznam načtených modulů. Pokud by složka s moduly obsahovala nevalidní modul, aplikace vypíše informaci o tom, že se takováto chyba vyskytla a nespustí se. Toto chování eliminuje potenciální práci s neúplnou konfigurací, kdy by mohl zmizet modul, který utrpěl poškození, nebo byl nedopatřením nahrán duplicitní modul.

```
/// <summary>
/// Modeses the by modules re validate.
/// </summary>
/// <param name="sender">The sender.</param>
/// <param name="e">The <see cref="ItemCheckEventArgs"/> instance containing the event
data.</param>
private void ModesByModulesReValidate(object sender, ItemCheckEventArgs e)
{

       if (radioButtonCustom.Checked || radioButtonMirror.Checked)
              if (checkedListBoxEnableModules.CheckedItems.Count < 2 &&
                    checkedListBoxEnableModules.CheckedItems.Count >= 1 &&
                    e != null && e.NewValue.Equals(CheckState.Checked))
              {
                    radioButtonMirror.Enabled = true;
                    radioButtonCustom.Enabled = true;
                    okButton.Enabled = true;
              }
              else
              {
                     radioButtonMirror.Enabled = false;
                     radioButtonCustom.Enabled = false;
                     okButton.Enabled = false;
              }

            if (!okButton.Enabled)
                errorMessage.Text = Resources.SettingsErrorMessageCompatibility;
            else
                errorMessage.Text = string.Empty;
}
```

Ukázka kódu 4: Validace nastavení

Dříve zmíněná validace je inicializovaná při vytváření instance formuláře nastavení a pak změnou typu konsolidace. Hlavní problém nastavení jsou počty dostupných modulů pro chod některých typů konsolidace. Funkcionalita jako taková by například se zrcadlením nad jedním úložištěm neměla žádný problém, ale z hlediska prosté logiky takové nastavení nedává smysl a není povoleno.

Posledním prvkem typu formuláře v toto projektu je prosté dialogové okno jakožto vstup při přejmenovávání a vytváření souboru. Tento formulář je implementováno anonymně. Umožnuje jej užít pro jakoukoliv operaci. Zobrazené texty a vstupy je pak možné definovat při inicializaci instance tohoto formuláře. Jelikož se nejedná přímo o dialogové okno, ale o formulář se vstupem, je zde doplněna výstupní funkcionalita v podobě detekce způsobu ukončení a typu ukončení. Kliknutí na Ok/Ukončit/Křížek je vyvolána specifická událost, která má vracet odlišný typ.

Přesně, dle analýzy, se v projektu CloudMan nachází třída ***Globals***, která definuje globální proměnné pro užití napříč celým projektem. Třída i její proměnné jsou statické, a tudíž nelze vytvářet její instance, ale lze k těmto datům přistupovat přímo. Příkladem dat uložených v tomto souboru je složka, která má být čtena při získávání modulů. Dále se zde nachází kompletní konfigurace šifrování a konfiguračního souboru pro virtuální úložiště.

9.4.1 DataProcessingUnit

Pod názvem této kapitoly se skrývá kompletní implementace logiky pro práci se soubory umístěných ve virtuálním i fyzickém souborovém systému. Jedná se o kritickou instanci pro běh aplikace. Metody této třídy zdánlivě kopírují interface metod pro moduly úložišť, ale logika skrývající se pod nimi je diametrálně odlišná, jelikož umožnuje práci s libovolným množstvím modulů. Také se zde pracuje s uživatelem nastaveným typem konsolidace. Z tohoto důvodu obsahuje každá metoda switch (druh podmínky) a primární členění každé z veřejných metod této třídy je podle typu konsolidace.

Třída si uchovává pouze 2 proměnné, instanci virtuálního souborového systému, nazvaného *EnhancedFileSystem* (EFS), a logger. Obě proměnné jsou nezbytné pro korektní běh, avšak pouze instance v EFS je kritická. Bez korektního vytvoření instance EFS dojde k ukončení aplikace. Instance této proměnné se vytváří v inicializační metodě, jelikož v umístění do konstruktoru vedlo k chybnému vytváření nové prázdné instance ve 100 % případů.

```
/// <summary>
/// Initializes the asynchronous.
/// </summary>
/// <param name="storageModules">The storage modules.</param>
public async Task InitializeAsync(List<StorageModule> storageModules)
{
       var list = (await GetStorageItemsIgnoreConfig(storageModules)).ToList();

       if (list.Where(i => i.Name == Globals.EfsDataFileName && i.Size > 0).Count() >= 1)
          await RefreshLocalEfs(storageModules);
       else
          Efs = new EnhancedFileSystem();

       logger.Debug("DataProcessinUnit initialized.");
}
```

Ukázka kódu 5: Inicializace EFS

Na ukázce výše lze nalézt specifickou asynchronní metodu s názvem *GetStorageItemsIgnoreConfig*, která obstarává stažení konfiguračního souboru virtuálního souborového systému a zároveň se využívá při operacích odvolávání předchozí operace (rollback). Odvolání operace je okrajová činnost, která probíhá pouze ve specifických situacích, kdy operace práce s úložištěm selže v průběhu a zatímco část již mohla být úspěšně

provedena. Změny se zachovávají pouze pokud projde bezchybně celá operace nad úložištěm, a to včetně nahrání nové verze konfiguračního souboru virtuálního souborového systému.

V současné verzi logika obnovování EFS nehodnotí, zdali jsou konfigurace z různých zdrojů shodné, ale použije tu nejnovější. Problematika porovnávání konfiguračních objektů, vyhledávání rozdílů a případné řešení, je námětem na další rozvoj aplikace.

```
/// <summary>
/// Refreshes the local efs.
/// </summary>
/// <param name="storageModules">The enabled storage modules.</param>
private async Task RefreshLocalEfs(List<StorageModule> storageModules)
{
	var list = (await GetStorageItemsIgnoreConfig(storageModules)).ToList();

	var configs = list.Where(i => i.Name == Globals.EfsDataFileName);

	if (!configs.Any()) return;

	var newestCnf = configs.OrderByDescending(i => i.Modified)
				.Where(i => i.Size > 0).First();

	var encryptedStream = await DownloadEfsCfgFile(storageModules, newestCnf);
	var decryptedStream = Cryptography.Decrypt(encryptedStream);
	     Efs =
JsonSerializationFactory.DeserializeClass<EnhancedFileSystem>(decryptedStream);
	encryptedStream.Close();
	decryptedStream.Close();
}
```

Ukázka kódu 6: Obnovení konfiguračního objektu ze souboru

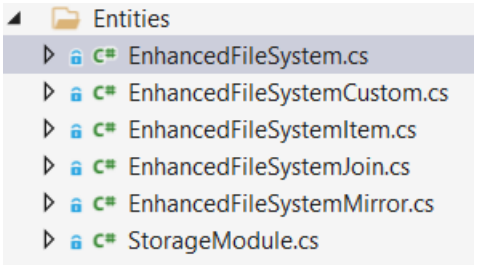
Šifrování a dešifrování jak názvů, tak i obsahu souborů, zpracovávají soukromé metody včetně ověření nastavení v konfiguraci. Zjednodušuje se tím hlavní logika jednotlivých metod pro práci nad virtuálním úložištěm. Mezi hlavní a nejčastější konzumenty šifrovacích metod pak patří samotné udržování konfigurace EFS, které se na cloudová úložiště nahrává pouze v zašifrované podobě.

Napříč procesní jednotkou se hojně využívá sada technologií, která se skrývá pod označením LINQ, jakožto lepší z dostupných variant pro vyhledávání a práci nad kolekcemi implementujícími *IEnumerable* a interfacem samotným. Často opakované úseky, kódy, jsou ve většině možných případů vyseparovány do privátních metod. Taktéž jsou vyseparovány do vlastních metod specifické kusy kódu, například dešifrování a šifrování. Tím se značně zpřehlednil kód uvnitř jednotlivých metod implementujících funkcionalitu nad úložištěm.

9.4.2 EnhancedFileSystem (EFS)

Pod tímto honosným označením se skrývá prostá implementace konfigurace virtuálního souborového systému. Struktura EFS odpovídá potřebě kompatibility se serializátorem do JSON. Konkrétně byla zvolena knihovna *Newtonsoft.Json*, pro implementaci serializace a deserializace. Struktura objektů EFS byla navržena s důrazem na čitelnost souboru, který jeho serializace vyprodukuje.

EFS se skládá ze základního objektu nazvaného *EnhancedFileSystem*, jenž zaobaluje implementace pro jednotlivé typy konsolidace (*EnhancedFileSystemJoin*, *EnhancedFileSystemMirror*, *EnhancedFileSystemCustom*). Na konci tohoto řetězce je *EnhancedFileSystemItem*, který implementuje podobu souboru či složky uložiště. Jedná se o plochou strukturu, která je při čtení úložiště aplikována na strukturu souborů z úložiště. V případě užití konsolidace typu Mirror a Custom se zobrazí v aplikaci pouze data, která byla touto aplikací přidána a nachází se v EFS konfiguračním souboru.



Obrázek 19: Třídy implementace EFS

EFS je kompletně veřejná struktura, jakožto důsledek vytváření instance skrze deserializátor souboru (všechny proměnné jsou veřejné a implementují čtení i zápis metodou – public property). I přes takto implementovanou strukturu, do které lze jakýmkoliv způsobem zasáhnout, jsou implementovány metody s minimální logikou pro práci s EFS. Tím se zjednodušuje kód v procesní jednotce.

Část logiky si sebou nese i *EnhancedFileSystemItem*, který rozlišuje, v jakém typu konsolidace se daný prvek uložiště nachází a podle toho se konstruktor rozhoduje, zda vytvořit strukturu plnou, či redukovanou. Tímto přístupem se šetří paměť při běhu aplikace a zároveň se eliminuje potřeba serializovat prázdné, pouze inicializované, instance.

```
class EnhancedFileSystemItem
    {
        public string Name { get; private set; }
        public bool EncryptedName { get; private set; }
        public bool EncryptedContent { get; private set; }
        public IEnumerable<string> ModulesParts { get; set; } = null;

        /// <summary>
        /// Initializes a new instance of the <see cref="EnhancedFileSystemItem"/> class.
        /// </summary>
        /// <param name="name">The name.</param>
        /// <param name="encryptedName">if set to <c>true</c> [encrypted name].</param>
        /// <param name="encryptedContent">if set to <c>true</c> [encrypted
content].</param>
        /// <param name="strip">if set to <c>true</c> [strip].</param>
        public EnhancedFileSystemItem(string name, bool encryptedName, bool
encryptedContent, bool strip)
        {
            Name = name;
            EncryptedName = encryptedName;
            EncryptedContent = encryptedContent;

            if (strip)
                ModulesParts = new List<string>();
        }
    }
```

Ukázka kódu 7: Zjednodušený kód EFS itemu

9.4.3 JsonSerializationFactory

Pro účely serializace a deserializace byla vytvořena statická třída *JsonSerializationFactory*, která na vstupu přijímá objekt k serializaci a zároveň i jeho typ. Implementace třídy je obecná a umožnuje serializovat instance bez ohledu na datový typ.

Z ukázky kódu metody serializace lze vyčíst, že je použito kódování Unicode a obecný stream. Zde se vyskytla jedna ze závažných nalezených chyb, kdy byl uzavírán lokální stream. Ačkoli se jedná o lokální proměnnou, překladač a optimalizátor ji v polovině případů nahradili referencí. V případě, že je proměnná nahrazena referencí a zároveň je stream ukončen, dojde při prvním následujícím přístupu ke streamu (snaha jej odeslat) k výjimce.

```
public static Stream SerializeClass<T>(T objectForSerialization) where T : new()
{
       var fileStream = new MemoryStream();
       var uniEncoding = new UnicodeEncoding();
       var json = JsonConvert.SerializeObject(objectForSerialization);
       var sw = new StreamWriter(fileStream, uniEncoding);

       sw.Write(json);
       sw.Flush();
       fileStream.Seek(0, SeekOrigin.Begin);

       return fileStream;
}
```

Ukázka kódu 8: Metoda serializace

Deserializace streamu je oproti serializaci podstatně jednoduší operace a nebyly zde žádné problémy. Stream se zde neuzavírá a o jeho instanci se postará automatická správa paměti.

```
public static T DeserializeClass<T>(Stream fileStream) where T : new()
{
	var reader = new StreamReader(fileStream);
	var content = reader.ReadToEnd();

	reader.Close();

	return JsonConvert.DeserializeObject<T>(content);
}
```

Ukázka kódu 9: Metoda deserializace

9.4.4 Cryptography

Statická třída, která obsahuje kompletní logiku šifrování a dešifrování veškerého k tomu určeného obsahu. Třída je statická, a tudíž ji nelze instancovat, volá se přímo. Třída obsahuje sadu interních proměnných, které svůj obsah čtou přímo u definice globálních proměnných. Jedná se o informace jako je heslo (server secret), mód užité šifry, zarovnání, velikost bloku a klíče. Jak vyplývá z analýzy této problematiky, počet iterací aplikovaných na otevřený text při šifrování je závislý na délce hesla a hodnota iterací definovaná zde slouží pro účely vytváření šifrovacího klíče z hesla a soli. Užití soli na heslo zvyšuje jeho bezpečnost a eliminuje situace, kdy užití stejného hesla na stejná data vyprodukuje stejný výstup.

Implementovány jsou zde, vyjma logiky kryptografie, i zaobalující metody. Účelem těchto metod je umožnit šifrování i zdánlivě nekompatibilních datových typů. Metody kryptografie konzumují i produkují pole bajtů. Z tohoto důvodu vznikly metody, které jsou schopny přijmout *stream* i *string* a transformovat tyto datové typy do kompatibilní podoby pole bajtů a následně zavolat logiku kryptografie. Zde je aplikováno pravidlo, že na vstupu metody je stejný datový typ jako na výstupu.

V zaobalujících metodách se vyskytl další z větších problémů, kdy zašifrovaný text obsahoval znaky, které nebyly kompatibilní s Unicode a při jejich vložení do *stringu* docházelo ke ztrátě informace. Tento problém byl vyřešen použitím kódovacího algoritmu Base64. Ten vzápětí vyprodukoval další zvláštní chybu a to, že takto ošetřený text obsahoval lomítka a backslashe. To u dobře implementovaných API nevytvářelo žádné problémy

a byl korektně vytvořen soubor s názvem obsahujícím i tyto problematické znaky (Google drive). Totéž se nedá prohlásit o OneDrive. Aplikační rozhraní této služby pracuje s textovou podobou cesty a hierarchie úložiště, kde název je součástí cesty. Zde ony nežádoucí znaky byly pochopeny jako složky, což vedlo k nežádoucímu vytváření nových složek na úložišti a současně ke ztrátě nahrávaného souboru. V extrémním případě vedlo zašifrování názvu souboru k vytvoření 20 nových složek.

```
/// <summary>
/// Generates a random bytes.
/// </summary>
/// <returns>An array of bytes.</returns>
private static byte[] GenerateRandomEntropyByKeySize()
{
	byte[] randomBytes = new byte[keySize / 8];

	using (var rngCsp = new RNGCryptoServiceProvider())
	{
		rngCsp.GetBytes(randomBytes);
	}

	return randomBytes;
}
```

Ukázka kódu 10: Generování náhodné entropie (salt, iv)

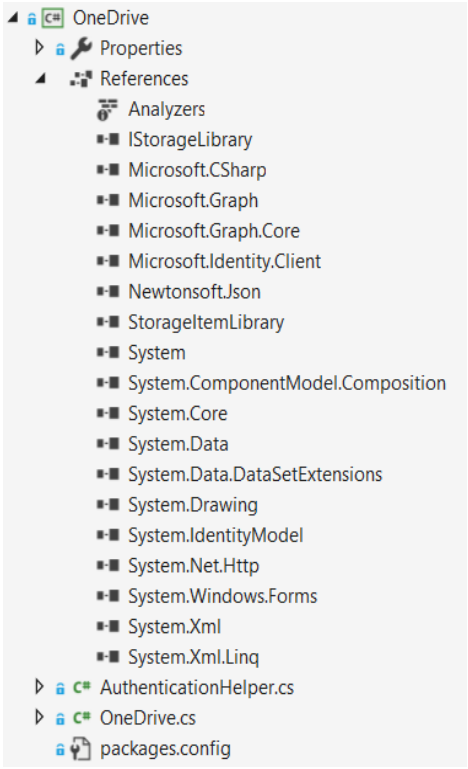
9.4.5 Enumerace

Přesně, dle rozhodnutí z analýzy, jsou z kódu textové identifikátory vyseparovány do výčtových datových typů. Projekt konkrétně obsahuje 3 výčtové datové typy, a to typ konsolidace, názvy prvků v nastavení a v neposlední řadě prosté ano/ne. Užitím výčtových typů namísto textové definice, například klíče v asociativním poli, je možné eliminovat mnoho potencionálních chyb způsobených nepozorností a překlepy. Konzistenci takových údajů pak hlídá kompilátor a není nutná manuální kontrola.

9.5 OneDrive projekt

Tento projekt obsahuje implementaci rozhraní *IStorage* pro úložiště OneDrive od společnosti Microsoft. Projekt obsahuje 2 třídy, kdy první je nazvaná OneDrive a přímo implementuje interface, které obsahují i několik metod navíc. Druhá třída pak implementuje pouze autentizaci a autorizaci.

Z počátku vytváření projektu byla snaha užít co nejnovější verze dostupných API a všech knihoven. To vedlo ke zjištění, že knihovna *Microsoft.Graph*, která je nezbytná pro práci s OneDrive API, musí být užita ve verzi 1.12.0 (nejnovější byla 3.10.0). Počátky implementace tohoto modulu ukázaly, že novější verze této knihovny (NuGet balíčku) nejsou kompatibilní s MEF. Při užití novějších verzí odmítá framework pro načítání modulů tento modul načíst jakožto nevalidní. Během analýzy problému a zjišťování možností se ukázalo, že poslední kompatibilní verze s implementací MEF tak jak je užita v této aplikaci, je verze 1.12.



Obrázek 20: Struktura projektu

Užitím starší verze knihoven pro práci s API byla vytvořena nová vrstva problémů, kdy je část metod implementovaných a povolených v této verzi označena na serveru jakožto zastaralá (Obsolete). API namísto aby vrátila požadovaná data, vrátila pouze informaci, že metoda není povolena. Jednou z metod patřících do této skupiny se po krátké době ukázala i metoda pro autentizaci za pomocí uživatelského jména a hesla. Microsoft tento způsob přihlášení definitivně zakázal, bylo tedy nezbytné tuto část kódu kompletně přepsat. Potřeba práce s lokálním úložištěm přihlašovacích údajů (Windows Credential Manager) se v důsledku toho

ukázala jako zbytečná. Jediný, v současné době povolený způsob autentizace, je skrze WebAuthToken. To v překladu znamená, že při vyvolání přihlášení se otevře webová stránka vyžadující autentifikační údaje. Toto chování se krátce po implementaci změnilo. Vyžádání autentizace začalo vyvolávat Windows OneDrive dialog. Toto chování je důsledkem hluboké integrace služby OneDrive do operačního systému Windows.

Aplikace pro svůj běh vyžaduje vygenerování identifikátoru pro ověření aplikace, což se provádí ve službě Azure Active Directory Admin Center (https://aad.portal.azure.com). V tomto rozhraní existuje záložka nazvaná „Registrace aplikací“, kde je možné přidat svou aplikaci. Tím získáme unikátní identifikátor aplikace pro přístup ke Graph API služby OneDrive. Vygenerování tohoto identifikátoru a vedení záznamu v Azure Active Directory Admin Center je již zdarma (dříve tomu tak nebylo).

```
static readonly string clientId = "   application GUID   ";
public static string[] Scopes = { "Files.ReadWrite" };

public static PublicClientApplication IdentityClientApp = new
PublicClientApplication(clientId);
```

Ukázka kódu 11: Poviné části kódy pro Graph API s OneDrive

Dalším povinným údajem je způsob přístupu (Scope). Tím je definováno, v jakém módu daná aplikace k úložišti přistupuje. V našem případě je nezbytný mód *Files.ReadWrite*, aby byl modul plně funkční. Toto nastavení říká, že daná aplikace (v našem případě tento modul) má povolení provádět nad Graph API jak operace čtení, tak i operace, které mění, popřípadě mažou, data. Prefixem *Files* říkáme, že je povolen pouze přístup na souborové úložiště. Graph API není technologie určená pouze pro práci s OneDrive, ale umožnuje pracovat se všemi službami dostupnými v Azure. Současně je skrze toto API možné měnit nastavení účtu Microsoft, což je v našem případě činnost nežádoucí.

Implementace autentizace, třída *AuthenticationHelper*, je silně inspirovaná návody z webu nápovědami společnosti Microsoft, kde je ne příliš podrobně vysvětleno užití GraphAPI, bohužel bez ohledu na různé verze. Prvotní testování bylo prováděno na poskytnutém vzorovém řešení s názvem QuickStartV2.

```
/// <inheritdoc/>
public async Task<Stream> DownloadFileAsync(StorageItem file)
{
	try
	{
		return await graphClient.Drive.Items[file.Id].Content.Request().GetAsync();
	}
	catch (Exception ex)
	{
		Console.WriteLine($"DownloadFileAsync.Onedrive: {ex.Message}");
	}

	return null;
}
```

Ukázka kódu 12: Stáhnutí souboru v modulu OneDrive

Samotná implementace metod pro práci s daty má mnoho možností. K souborům lze přistupovat za pomocí jejich identifikátoru, což je všeobecně považováno za modernější přístup, ale tato verze GraphAPI umožnuje i starší způsob, kdy se definuje celá cesta (PATH) k souboru, včetně souboru samotného. Obě metody přístupu jsou citlivé na speciální znaky v názvu souboru a neumí tuto situaci interpretovat, validovat ani oznámit. Znaky jsou různě interpretovány, zejména je to vidět na lomítku, kdy si API vytvoří nové složky zcela náhodně.

9.6 GDrive projekt

Jedná se o projekt modulu, který obsahuje třídu s implementací *IStorage* rozhraní obohacenou o vlastní privátní metody. Implementace využívá *Google.Apis* knihovny. Nejdůležitější z nich je *Google.Apis.Drive.v3*, která zajištuje komunikaci s uložištěm Google Drive, respektive s jeho API. Projekt opět obsahuje druhou třídu s názvem *AuthenticationHelper*, která separátně implementuje autentizaci s využitím knihovny *Google.Apis.Auth*.

Implementace a ověřování funkcí modulu pro Google Drive se obešlo bez větších problémů. Knihovny i dostupné WebAPI reagují zcela dle dokumentace, která je validní ku všem dostupným verzím knihoven a API. Pro každou verzi API existuje shodně číslovaná verze knihoven. Nejnovější verze (myšlena verze NuGet balíčku) je kompatibilní s MEF jakožto prostředkem pro dynamické načítání modulů.

Modul pro svůj korektní běh vyžaduje registraci u společnosti Google. Konkrétně je nezbytné navštívit vývojářskou konzoli (https://console.developers.google.com) a zaregistrovat aplikaci. Registrací získáme unikátní identifikátor, který u společnosti Google identifikuje aplikaci a umožní přístup k API. Oproti řešení u společnosti Microsoft je toto rozhraní jednoduší a přístupnější. Google konzole vygeneruje JSON soubor, na který je API připravená a umí s tím automaticky pracovat.

Další povinný parametr je opět přístup (Scope). Zde, na rozdíl od OneDrive, definuje, kam přes Google API přistupujeme. V našem případě je definován *DriveService.Scope.Drive*. Způsob a práva se definují až autentizací uživatele. Posledním potřebným prvkem, bez kterého se neobejdeme, je název aplikace. Bez definování názvu aplikace nelze vytvořit novou instanci služby Google.Api a tudíž nelze volat metody pro komunikaci s úložištěm.

```
public static DriveService GetGetAuthenticatedClient()
{
	UserCredential credential;

	try
	{
		using (var stream =
		new FileStream("Modules\\GDrive\\credentials.json", FileMode.Open,
FileAccess.Read))
		{
			string credPath = "Modules\\GDrive\\token.json";
			credential = GoogleWebAuthorizationBroker.AuthorizeAsync(
			    GoogleClientSecrets.Load(stream).Secrets,
			    Scopes,
			    "user",
			    CancellationToken.None,
			    new FileDataStore(credPath, true)).Result;
		}

		return new DriveService(new BaseClientService.Initializer()
		{
		    HttpClientInitializer = credential,
		    ApplicationName = ApplicationName,
		});
	}
	catch (Exception ex)
	{
		Console.WriteLine($"GetGetAuthenticatedClient.GDrive: {ex.Message}");
	}

	return null;
}
```

Ukázka kódu 13: Google autentifikace a vytvoření Google Service

Google API v případě, kdy se pokoušíme vytvořit službu pro operace nad úložištěm, pro validně registrovanou aplikaci (viz. ukázka kódu 13, soubor credentials), vyvolá webovou stránku s přihlašovacím formulářem a následným povolením pro aplikaci k přístupu k úložišti. Tímto webovým dotazem se následně, v případě úspěchu, vytvoří uživatelský token, který se následně uloží do souboru ve složce s modulem. Tento token má poměrně dlouhou životnost a při běžném užívání nedojde k odhlášení. Při testování tento token jevil známky trvalého přístupu. K odhlášení došlo pouze při inicializaci odhlášení z kódu, anebo ukončení přihlášení v konzolovém rozhraní služeb Google, kde lze odebrat jak povolení programu, tak odhlásit/zablokovat i jednotlivá zařízení a aplikace.

```
public async Task<Stream> DownloadFileAsync(StorageItem file)
{
	try
	{
		var request = driveService.Files.Get(file.Id);
		var stream = new System.IO.MemoryStream();

		request.MediaDownloader.ProgressChanged +=
		(Google.Apis.Download.IDownloadProgress progress) =>
		{
			switch (progress.Status)
			{
			    case Google.Apis.Download.DownloadStatus.Failed:
				stream = null;
				break;
			}
		};
		await request.DownloadAsync(stream);

		return stream;
	}
	catch (Exception ex)
	{
		Console.WriteLine($"DownloadFileAsync.GDrive: {ex.Message}");
	}

	return null;
}
```

Ukázka kódu 14: Stáhnutí souboru v modulu Google Drive

10. TESTOVÁNÍ

Aplikace jako celek byla testována pravidelně v průběhu vývoje. Díky tomu bylo identifikováno a odstraněno mnoho problémů už během vývoje aplikace. Ukázkovým příkladem bylo ukončení podpory základního přihlašování bez web tokenu. Tato skutečnost vedla k nefunkčnosti celé implementace OneDrive modulu.

Následné testovací cykly odhalily problematické metody, jejichž podpora byla ukončena. Další odhalenou chybou byly ony již dříve zmíněné zvláštnosti při zpracovávání speciálních znaků, což bylo vyřešeno.

Původní vývoj a testování aplikace probíhal na dekádu starém 4jádrovém počítači architektury Haswell. Dokončování aplikace již probíhalo na moderním stroji z rodiny AMD Ryzen. Rozdíl ve výkonu hardwaru je extrémní, oproti tomu rozdíl v rychlosti aplikace byl poměrně malý. To byl jeden z nezamýšlených testů, který vedl k poměrně velké optimalizaci kódu aplikace.

V rámci testování, po dokončení implementace, již nebyly nalezeny žádné zásadní funkční problémy či nedostatky. Rozšířen byl především seznam věcí, o které by bylo zajímavé rozšířit funkcionalitu aplikace. Zároveň bylo při testování odhaleno, že mód označený jako vlastní je vzhledem k poměru rizik a bezpečnosti dat nadbytečný, jelikož vysokou míru zabezpečení dat převyšují rizika ztráty dat. Tato skutečnost vedla k jednomu z bodů na seznamu potencionálních rozšíření, a to výpočtu parity nad rozdělovanými daty. Tím by došlo ke kompletní implementaci funkcionality RAID 5 nad sadou cloudových úložišť, což by zajisté nebyl jednoduchý úkol.

Aplikace konzumuje v průměru 45 MB operační paměti. Nejvyšší hodnoty dosahuje při stahování a nahrávání souborů, kdy se datový obsah zpracovávaného souboru, v ideálním případě, nahrává celý do operační paměti. Dosažené hodnoty, nad tuto zmíněnou, jsou poměrně rychle eliminovány, jelikož správa uvolňování paměti (garbage collector) efektivně uklízí v aplikaci již nepotřebné alokace. Co se týče výkonu procesoru, nejnáročnější operací je rozdělování a ověřování souborů, při nejsložitějším módu aplikace a částečně obsluha kryptografie (zde velice záleží na cpu a instrukční sadě).

11. ZÁVĚR

V rámci této diplomové práce byl proveden rozsáhlý průzkum v oblasti cloudových technologií, aplikačních rozhraní a reálné funkcionality poskytovaných dostupných API cloudových úložišť. Tyto znalosti a poznatky byly následně aplikovány při vytváření aplikace pro konsolidaci cloudových úložišť, nazvané CloudMan. Aplikace poskytuje neutrální rozhraní a je tedy možné k ní vytvořit modul kompatibilní s jakýmkoliv dostupným řešením cloudového úložiště poskytujícího webové aplikační rozhraní s užitím standardních webových technologií.

Finální řešení obsahuje aplikaci určenou pro operační systém Windows, která obsahuje jednoduché intuitivní grafické rozhraní. Problematické části jsou schopny poskytnou současně návodná vysvětlení. Řešení obsahuje implementaci dvou modulů pro cloudové úložiště Microsoft OneDrive a Google Drive. Oba moduly implementují stejná rozhraní a je možné je přidávat, odebírat, ale i povolovat a zakazovat za chodu aplikace.

Pro účely uchování informací o nahrávaných souborech, zejména v pokročilých režimech, byl vyvinut virtuální souborový systém reprezentovaný sadou objektů popisujících strukturu souborů a složek uložených na virtuálním úložišti. Tento soubor objektů je interpretovatelný do podoby JSON souboru a v zašifrované podobě je nahráván spolu s daty na dostupná úložiště.

Celý systém byl vytvořen v programovacím jazyce C# a technologie .NET s využitím vývojového prostředí Visual Studio 2019. Řešení nevyžaduje ani neobsahuje žádné aplikace třetích stran potřebné pro použití na operačním systému Windows. Součástí jsou pouze knihovny třetích stran (Microsoft, Google), které jsou nezbytné pro komunikaci s danými službami.

Do budoucna by bylo možné aplikaci rozšířit o komplexnější práci s konfiguračním souborem virtuálního souborového systému, zejména pak řešení nalezené nekonzistence za pomocí slučování a automatického opravování.

12. BIBLIOGRAFIE

[1] M. Zikmund, „BusinessVize.cz,“ Nitana s. r. o., 29 11 2010. [Online]. Available: http://www.businessvize.cz/software/co-je-to-cloud-computing-a-proc-se-o-nem-mluvi. [Přístup získán 05 04 2019].

[2] „Microsoft Azure,“ Microsoft, [Online]. Available: https://azure.microsoft.com/cs-cz/overview/what-is-cloud-computing/. [Přístup získán 04 04 2019].

[3] J. Lavrinčík, „Cloudové systémy,“ 2017. [Online]. Available: https://mvso.cz/wp-content/uploads/2018/02/Cloudové-systémy-studijní-text.pdf. [Přístup získán 04 04 2019].

[4] P. Novák, „Cloudové úložiště 2019: Srovnání Dropbox, OneDrive a Google Drive,“ 29 05 2019. [Online]. Available: https://www.skrblik.cz/telefon/internet/cloudova-uloziste/. [Přístup získán 15 08 2019].

[5] „The RapidAPI Blog,“ The RapidAPI, 07 03 2019. [Online]. Available: https://blog.rapidapi.com/types-of-apis/. [Přístup získán 30 08 2019].

[6] M. Malý, „REST: architektura pro webové API,“ zdroják.cz, 03 08 2009. [Online]. Available: https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/. [Přístup získán 13 07 2019].

[7] M. Samek, „Virtuální souborový systém pro procházení souborových archivů,“ 2012. [Online]. Available: https://dspace.vutbr.cz/xmlui/bitstream/handle/11012/55105/9966.pdf. [Přístup získán 27 08 2019].

[8] J. J. Pfeiffer, „Writing a FUSE Filesystem: a Tutorial,“ 02 04 2018. [Online]. Available: https://www.cs.nmsu.edu/~pfeiffer/fuse-tutorial/. [Přístup získán 28 08 2019].

[9] Wikipedia, „Modular programming,“ 21 03 2019. [Online]. Available: https://en.wikipedia.org/wiki/Modular_programming. [Přístup získán 03 09 2019].

[10] S. Johnston, „Wikipedia,“ Wikimedia Commons, 03 03 2009. [Online]. Available: https://commons.wikimedia.org/wiki/File:Cloud_computing.svg?uselang=cs. [Přístup získán 04 04 2019].

[11] „Co je IaaS?,“ Microsoft, 2019. [Online]. Available: https://azure.microsoft.com/cs-cz/overview/what-is-iaas/. [Přístup získán 04 04 2019].

[12] „Co je PaaS?,“ Microsoft, 2019. [Online]. Available: https://azure.microsoft.com/cs-cz/overview/what-is-paas/. [Přístup získán 04 04 2019].

[13] „Co je SaaS?,“ Microsoft, 2019. [Online]. Available: https://azure.microsoft.com/cs-cz/overview/what-is-saas/. [Přístup získán 04 04 2019].

[14] „API Manual,“ 2018. [Online]. Available: http://testingalert.com/api-testing/api-manual/. [Přístup získán 27 07 2019].

[15] S. Maddox, „API types,“ FFEATHER, 16 02 2014. [Online]. Available: https://ffeathers.wordpress.com/2014/02/16/api-types/. [Přístup získán 29 08 2019].

[16] R. Zumeran, „The History of APIs and How They Impact Your Future,“ OpenLegacy Blog, 07 06 2017. [Online]. Available: https://www.openlegacy.com/blog/the-history-of-apis-and-how-they-impact-your-future. [Přístup získán 11 06 2019].

[17] „What is an API?,“ Red Hat, [Online]. Available: https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces. [Přístup získán 06 07 2019].

[18] M. Kuty, „13. Souborové systémy a logická struktura dat (principy, porovnání, příklady).," Otázky na státnice, 2014. [Online]. Available: http://michaelkuty.github.io/ssz-ai-hk-3/_downloads/131.pdf. [Přístup získán 26 08 2019].

[19] „Jaký je nejlepší souborový systém pro můj Linux?," Linux mint czech, 12 05 2016. [Online]. Available: https://www.linux-mint-czech.cz/2016/05/jaky-je-nejlepsi-souborovy-system-pro-muj-linux/. [Přístup získán 26 08 2019].

[20] J. Fesl, „Přednášky z předmětu distribuované a paralelní algoritmy," 2015.

[21] M. Jones, „Network file systems and Linux," IBM Developer, 10 11 2010. [Online]. Available: https://developer.ibm.com/tutorials/l-network-filesystems/. [Přístup získán 02 09 2019].

[22] Javvin technologies, Network Protocols Handbook, Javvin technologies, 2005.

[23] AOSD, „Importance Of Modularity In Programming," Aspect, 18 01 2018. [Online]. Available: http://aosd.net/importance-of-modularity-in-programming/. [Přístup získán 18 09 2019].

[24] „Šifrování v režimu kódové knihy (ECB)," Wikimedia, 01 06 2013. [Online]. Available: https://upload.wikimedia.org/wikipedia/commons/1/12/ECB_encryption_cs.svg. [Přístup získán 18 06 2019].

[25] „Šifrování v režimu řetězení šifrových bloků (CBC)," 01 06 2013. [Online]. Available: https://upload.wikimedia.org/wikipedia/commons/a/a5/CBC_encryption_cs.svg. [Přístup získán 18 06 2019].

[26] „Advanced Encryption Standard," Wikimedia, 24 02 2021. [Online]. Available: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard. [Přístup získán 11 03 2021].

[27] „Šifrování v režimu šifrové zpětné vazby (CFB),“ Wikimedia, 01 06 2013. [Online]. Available: https://upload.wikimedia.org/wikipedia/commons/a/a0/CFB_encryption_cs.svg. [Přístup získán 18 06 2019].

[28] P. Bouška, „Obecný úvod do šifrování dat,“ Samuraj, 27 05 2019. [Online]. Available: https://www.samuraj-cz.com/clanek/obecny-uvod-do-sifrovani-dat/. [Přístup získán 11 03 2021].

[29] „Quickstart: Acquire a token and call Microsoft Graph API from a Windows desktop app,“ Microsoft, 12 12 2019. [Online]. Available: https://docs.microsoft.com/en-us/azure/active-directory/develop/quickstart-v2-windows-desktop. [Přístup získán 01 02 2020].

[30] T. Mika, „Systém pro vzdálené ovládání virtuálních počítačů,“ Theses.cz, České Budějovice, 2017.

13. SEZNAM OBRÁZKŮ

14. SEZNAM TABULEK

15. SEZNAM DIAGRAMŮ

16. SEZNAM UKÁZEK KÓDU

17. PŘÍLOHY

Příloha 1: CD s PDF verzí práce, zdrojovými kódy systému ve formě projektů pro Visual Studio.