



**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**

BRNO UNIVERSITY OF TECHNOLOGY

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**

FACULTY OF INFORMATION TECHNOLOGY

**ÚSTAV INFORMAČNÍCH SYSTÉMŮ**

DEPARTMENT OF INFORMATION SYSTEMS

**APLIKACE PRO ŘÍZENÝ PŘÍSTUP KE VZDÁLENÝM  
DOKUMENTŮM PRO GNU/LINUX**

APPLICATION FOR CONTROLLED ACCESS TO REMOTE DOCUMENTS FOR GNU/LINUX

**BAKALÁŘSKÁ PRÁCE**

BACHELOR'S THESIS

**AUTOR PRÁCE**

AUTHOR

**JAN BERNARD**

**VEDOUCÍ PRÁCE**

SUPERVISOR

**RNDr. MAREK RYCHLÝ, Ph.D.**

**BRNO 2021**

## Zadání bakalářské práce



Student: **Bernard Jan**

Program: Informační technologie

Název: **Aplikace pro řízený přístup ke vzdáleným dokumentům pro GNU/Linux**  
**Application for Controlled Access to Remote Documents for GNU/Linux**

Kategorie: Operační systémy

Zadání:

1. Seznamte se s požadavky týkající se zabezpečení přístupu k dokumentům v projektu Validované datové úložiště (VDU). Prozkoumejte možnosti virtuálních souborových systémů a integrace aplikací do desktopového prostředí v operačním systému GNU/Linux.
2. Navrhněte klientskou aplikaci pro VDU, která se připojí k úložišti a bude zpřístupňovat obsah úložiště pod zadaným přístupovým klíčem jako soubor virtuálního souborového systému s řízeným přístupem a správou verzí. Navrhněte automatické testy požadovaných vlastností takové aplikace.
3. Po konzultaci s vedoucím navrženou aplikaci implementujte včetně automatických testů.
4. Výsledek popište, vyhodnoťte a zveřejněte jako open-source.

Literatura:

- Interní dokumentace projektu Validované datové úložiště.
- VANGOOR, Bharath Kumar Reddy; TARASOV, Vasily; ZADOK, Erez. To FUSE or Not to FUSE: Performance of User-Space File Systems. In: 15th USENIX Conference on File and Storage Technologies (FAST 17). 2017, s. 59-72. Dostupné z: [https://www.usenix.org/conference/fast17/technical-sessions/presentation/vangoor]
- BURIHABWA, Dorian, et al. SGX-FS: Hardening a File System in User-Space with Intel SGX. In: 2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom). IEEE, 2018, s. 67-72. Dostupné z: [https://doi.org/10.1109/CloudCom2018.2018.00027]
- xdg-utils. *freedesktop.org* [online], 2019 [cit. 2020-10-26]. Dostupné z: [https://freedesktop.org/wiki/Software/xdg-utils/]

Pro udělení zápočtu za první semestr je požadováno:

- Body 1, 2 a rozpracovaný bod 3.

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

Vedoucí práce: **Rychlý Marek, RNDr., Ph.D.**

Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.

Datum zadání: 1. listopadu 2020

Datum odevzdání: 12. května 2021

Datum schválení: 27. října 2020

## Abstrakt

Práce se zaměřuje na synchronizaci a životní cyklus souborů stažených do uživatelského počítače ze vzdáleného validovaného datového uložště. Z analýzy trhu vyplynulo, že na trhu je velký výběr aplikací, ale rozdíly mezi nimi jsou relativně velké. Na základě požadavků a stanovené komunikace s validovaným datovým uložštěm byla navržena a implementována aplikace pro systém GNU/Linux. Testování aplikace probíhalo na dvou vybraných Linuxových distribucích s mock serverem zastupující vzdálené uložště.

## Abstract

The aim of this thesis is synchronization and life cycle of files downloaded to user's computer from validated data storage. Based on market analysis there is large selection of applications in market but there are relatively large differences along them. Application was designed and implemented for GNU/Linux according to requirements and given validated data storage communication. The application was tested on two selected Linux distributions with mock server representing remote storage.

## Klíčová slova

Validované datové uložště, VDU, Linux, souborový systém, HTTP, FUSE, C++, synchronizace, soubor, dokument, oprávnění

## Keywords

Validated data storage, VDS, Linux, filesystem, HTTP, FUSE, C++, synchronization, file, document, permission

## Citace

BERNARD, Jan. *Aplikace pro řízený přístup ke vzdáleným dokumentům pro GNU/Linux*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce RNDr. Marek Rychlý, Ph.D.

# Aplikace pro řízený přístup ke vzdáleným dokumentům pro GNU/Linux

## Prohlášení

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

.....

Jan Bernard  
9. května 2021

## Poděkování

Rád bych poděkoval vedoucímu práce, doktorovi Markovi Rychlému, za vynikající vedení práce a jeho podnětné poznatky a rady. Také bych rád poděkoval všem svým kolegům, se kterými jsem procházel celé bakalářské studium. A v neposlední řadě děkuji všem svým přátelům a rodině za podporu a pochopení.

# Obsah

<b>1</b>	<b>Úvod</b>	<b>3</b>
<b>2</b>	<b>Motivace a existující řešení</b>	<b>4</b>
2.1	Současná řešení . . . . .	4
2.2	Google Drive . . . . .	5
2.3	Dropbox . . . . .	5
2.4	pCloud . . . . .	5
2.5	Syncthing . . . . .	6
2.6	Shrnutí . . . . .	6
<b>3</b>	<b>Návrh řešení a použité technologie</b>	<b>7</b>
3.1	Případy užití . . . . .	7
3.2	Definice REST API . . . . .	8
3.3	Architektura . . . . .	9
3.3.1	Diagram tříd . . . . .	10
3.4	Technologie . . . . .	11
3.4.1	GNU/Linux . . . . .	11
3.4.2	C++/C . . . . .	12
3.4.3	HTTP . . . . .	13
3.4.4	OpenAPI . . . . .	14
3.4.5	Mock Server . . . . .	14
3.4.6	Souborový systém v uživatelském prostoru (FUSE) . . . . .	15
3.4.7	D-Bus . . . . .	16
3.4.8	Keyring . . . . .	16
<b>4</b>	<b>Popis implementace a nasazení aplikace</b>	<b>17</b>
4.1	Využívané cesty v Linuxové adresářové struktuře . . . . .	18
4.1.1	Konfigurační soubor . . . . .	19
4.1.2	SQLite databáze . . . . .	19
4.2	xdg-utils . . . . .	20
4.3	Komunikace s API . . . . .	21
4.3.1	Mock Server . . . . .	21
4.3.2	Autentifikace . . . . .	22
4.4	FUSE démon . . . . .	22
4.4.1	Práce se soubory . . . . .	22
4.5	Shrnutí . . . . .	23
<b>5</b>	<b>Nasazení a testování</b>	<b>24</b>

5.1	Sestavení ze zdrojových souborů . . . . .	24
5.2	DPKG . . . . .	24
5.3	RPM . . . . .	25
5.4	Testování . . . . .	26
5.5	Google Test . . . . .	26
5.6	Testovací sada . . . . .	26
<b>6</b>	<b>Závěr</b>	<b>27</b>
	<b>Literatura</b>	<b>28</b>
<b>A</b>	<b>Seznam použitých knihoven</b>	<b>30</b>
<b>B</b>	<b>Obsah přiloženého média</b>	<b>31</b>

# Kapitola 1

## Úvod

V době vysokorychlostního a stabilního internetu je využívání vzdálených uložišť běžnou záležitostí. Pro většinu uživatelů je to nástroj pro jednoduchou synchronizaci dat napříč několika zařízeními od mobilního telefonu po stolní počítač. Pokud vezmeme v úvahu pouze velká datová centra, jedná se pravděpodobně o nejlevnější a nejspolehlivější způsob uchování dat, protože tyto centra mají velkou kapacitu úložného prostoru a také zálohy na softwarové i hardwarové úrovni. Pod pojmem vzdálené uložení si nemusíme představit jen velká datová centra se stovkami serverů, může se jednat o relativně malé uložení ve firmě nebo dokonce o osobní/domácí řešení. Pro využívání osobních/firemních uložení se lidé uchylují v případech, je-li třeba uložit osobní nebo určitým způsobem citlivá data, která by nemohla být poskytnuta třetí straně. Nabízená řešení také nemusí poskytovat potřebnou funkcionalitu nebo jejich finanční model nesplňuje zákaznická kritéria.

V průběhu let byly vyvinuty protokoly řešící sdílení souborů jako FTP, NFS a mnohé další. Sdílení souborů je komplexní záležitostí a obsahuje velké množství parametrů. Každý protokol se zaměřuje jen na určité parametry jako zabezpečení a spolehlivost přenosu, oprávnění pro jednotlivé uživatele a soubory, synchronizace modifikovaných souborů apod nebo pojme daný parametr odlišným způsobem. V dnešní době je většina komerčně využívaných aplikací vystavěna na proprietárních protokolech nebo na protokolech umožňující obecnější použití. Jedním z nejpoužívanějších aplikačních protokolů bude Hypertext Transfer Protocol neboli HTTP.

Cílem této práce je prozkoumat aktuální nabídku a možnosti na poli vzdálených uložení a navrhnout aplikaci pro řízený přístup ke vzdáleným dokumentům pro platformu GNU/Linux. Aplikace bude zaměřena na řízený životní cyklus souborů s vynucenými oprávněními pro jednotlivé soubory daná vzdáleným uložení. Komunikace skrze HTTP protokol byla definována externím zadavatelem. Zdrojové kódy jsou dostupné na veřejném Github repozitáři.<sup>1</sup>

---

<sup>1</sup><https://github.com/PlayerBerny12/VUT-IBT-Code>

## Kapitola 2

# Motivace a existující řešení

Popularita a využití vzdálených uložišť roste, protože lidé využívají více zařízení a potřebují mezi nimi jednoduchou synchronizaci nebo na jednom souboru potřebuje spolupracovat více lidí. Dalším podnětem využívání vzdálených uložišť je většího množství dat a potřeba tyto data efektivně sdílet a bezpečně uložit. Postupně se digitalizují další systémy například ve státní správě nebo dalších podnikatelských sektorech.

Na trhu je velké množství služeb zaměřujících se na různé klientské potřeby. Diverzita nabídky je až překvapivě velká. Většina se zaměřuje na ukládání a sdílení dat ve svém ekosystému, které jsou určené pro široké použití. Pokud jsou požadavky specifické, tak možnost vlastní konfigurace není možná. Pro částečnou úpravu nebo automatizaci procesů lze využít poskytované API, ve většině případů se jedná o variantu REST API. Poslední možností je vystavení obdobné služby z několika existujících aplikací nebo vytvoření nové.

v kontextu této práce jsou požadavky následující:

- Dodržování oprávnění i na klientském systému
  - read-only
  - write-only
  - read/write
- Životní cyklus souboru (stažení)
  - stažení
  - případná modifikace
  - smazání souboru po vypršení platnosti
- Autentifikace
  - uživatelské jméno a klientský certifikát
  - uživatelské jméno a heslo

### 2.1 Současná řešení

Pro bližší analýzu byl vybrán vzorek služeb a programů obsahující velké ekosystémy služeb po open source aplikace. Zkoumáno bylo několik parametrů, jako poskytovaná funkcionality pro jednotlivce a firmy, cena, integrace s ostatními systémy a aplikacemi apod. Obecně



nelze jednoznačně určit nejlepší uložště, ale je možné poukázat na rozdílné vlastnosti a vydvihnout kladné. Koncový uživatel se následně může rozhodnout dle svých potřeb.

## 2.2 Google Drive

Pro synchronizaci obsahu z Google Drive na osobní počítač se využívá aplikace pojmenovaná Google File Stream. Je možné ji provozovat pouze na systémech Windows a Mac OS. [7] Pro Linuxové distribuce lze využít například neoficiální open source aplikaci Google Drive OCamlFUSE<sup>1</sup> využívající technologii FUSE, která bude popsána v následující kapitole 3.4.6. Google Disk poskytuje plně funkční webové rozhraní, ve kterém je možné dokumenty přímo upravovat bez nutnosti stahování. Problém nenastal ani v případě konkurenčních Microsoft Office dokumentů.

Google Workspace (dříve Google Suite) je možné pořídit v několika balíčcích. Například balíček Business Standard nabízí 2 TB uložště za 10,40 EUR měsíčně. Balíček obsahuje další služby jako firemní email a schůzky až o 150 účastnících na Google Meet. [13]

Každý uživatel má vlastní disk s omezenou kapacitou podle balíčku. Na disku lze vytvářet standardní složkovou hierarchii a je možné sdílet jednotlivé soubory nebo obsah složek s ostatními Google uživateli. Verze Enterprise umožňuje vytváření dalších disků, na které je možné přiřadit seznam uživatelů. Každý uživatel má nastavenou jednu z 6 rolí, které vymezují jeho možnosti. k dispozici je API, která pokrývá veškeré možnosti webového rozhraní jako vytvoření souboru, sdílení atd. [11]

## 2.3 Dropbox

Dropbox má oficiální balíčky své aplikace pro Ubuntu a Fedoru, případně je možné si aplikaci zkompileovat ze zdrojových souborů. Podpora Windows a Mac OS je samozřejmostí. Z práce „Personal Cloud Storage Benchmarks and Comparison“ lze vyčíst, že Dropbox se zaměřuje na efektivitu přenosu a minimální zatížení sítě. Využívá menší množství TCP spojení oproti jeho konkurentům a také stejně jako Google Drive komprimuje data před odesláním. [2] Toto řešení nemusí dosahovat nejvyšší výkonosti, ale nezatěžuje tolik infrastrukturu.

Webové rozhraní je více orientováno na správu souborů a jejich historii. Nabízí jednoduché obnovení smazaných souborů nebo návrat k předchozí verzi. Nenabízí tolik možností jako Google Drive, ale upravovat dokumenty ve webovém prohlížeči lze také. Na výběr je mezi Microsoft Office Online nebo Google Workspacce. Sdílení souborů a složek je možné i s uživateli, kteří nemají Dropbox účet.

Balíček Plus za 9,99 EUR měsíčně dává uživateli přístup ke 2 TB uložšti. Firemní balíčky obsahují rozšíření pro lepší správu uživatelů, vytváření skupin a admin panel/konzole. Dropbox má také API nabízející dostatečnou funkcionalitu pro případnou integraci s jinými systémy apod. [9]

## 2.4 pCloud

Poskytovatel pCloud podporuje nejpoužívanější platformy, a to včetně mobilních. Za 9,99 EUR měsíčně dostane uživatel 2 TB uložště. Jako jediný z poskytovatelů v analýze nabízí balíček s jednorázovou platbou za cenu 350 EUR. Jedná se o identický balíček, pouze forma

<sup>1</sup><https://github.com/astrada/google-drive-ocamlfuse>

platby se liší. Všechna data jsou přenášena přes šifrovaný kanál a všechny kopie souborů na pěti různých serverech jsou šifrovaná 256bitovým AES klíčem. [22]

Webové rozhraní je jednoduché, avšak oproti konkurentům má méně funkcí. Lze zobrazit náhled dokumentů, ale upravovat je nelze. Každý soubor má tzv. revize a je možné obnovit obsah souboru na vybranou revizi. Revize jsou uchovány po dobu jednoho roku. Soubory je možné sdílet i s uživateli, kteří nemají účet u pCloudu.

Pro firemní zákazníky je nabízený rozšířený management uživatelů a monitoring s logy aktivity jednotlivých uživatelů.

## 2.5 Syncthing

Open source aplikace Syncthing synchronizuje soubory peer-to-peer. Nejedná se o čistou peer-to-peer architekturu, protože jeden z uzlů může být nastaven jako server a veškeré data budou synchronizována vůči tomuto uzlu. Syncthing je možné provozovat na většině dnešních systémů jako Windows, Linux, BSD a Mac OS. [24]

Aplikace poskytuje webové rozhraní, které je určené pouze pro nastavení parametrů jako discovery protokol pro objevování ostatních uzlů, jaké složky mají být synchronizovány nebo kolik verzí jednotlivých souborů má být uchováváno a mnoho dalšího. Velká přizpůsobivost umožňuje upravit fungování aplikace vlastním potřebám, na druhou stranu bude náročné udržovat systém s větším množstvím uživatelů vystavěný na této aplikaci. Jedná se tedy spíše o domácí řešení.

## 2.6 Shrnutí

Trendy na poli vzdálených uložišť určují velké firmy jako Google nebo Microsoft. Důkazem tohoto jevu je například integrace aplikací obou těchto firem do webového rozhraní Dropboxu. Aplikace One Drive od Microsoftu nebyla blíže analyzována, protože se jedná o službu pouze pro platformu Windows.

Všechny služby mají webové rozhraní, ke kterému je aplikace synchronizující obsah uložště občas brána jako doplněk. Sdílení nebo obnova předchozí verze souboru je vždy možná ve webové aplikaci, ale ne vždy jsou tyto akce dostupné i na desktopové verzi.

Aplikace na jejíž základě je vytvořena tato práce nebude synchronizovat celé uložště, ale bude pracovat jen s jednotlivými soubory. Uživatel na vyžádání stáhne právě jeden soubor, se kterým bude chtít uživatel v danou dobu pracovat (zobrazit nebo upravit obsah). Synchronizace celého uložště nebo obsahu jedné složky nebude možná. Jedná se spíše o náhradu možnosti zobrazovat a upravovat soubor přímo ve webovém prohlížeči jako to umožňuje například Google Drive.

## Kapitola 3

# Návrh řešení a použité technologie

Prvním krokem navrhovací fáze bylo seznámení se zadáním a požadavky. Externí zadavatel dodal podrobnou dokumentaci REST API pro validované datové uložště, která byla stěžejním dokumentem při návrhu. Jednalo se formální textový popis, který nebyl v žádném standardizovaném formát. API obsahuje funkce pro ověření spojení, autentifikaci a práci se soubory. VDU nebylo při vzniku této práce k dispozici, proto byl pro potřeby vývoje vytvořen Mock Server na základě přepisu poskytnuté dokumentace do standardizovaného formátu OpenAPI.

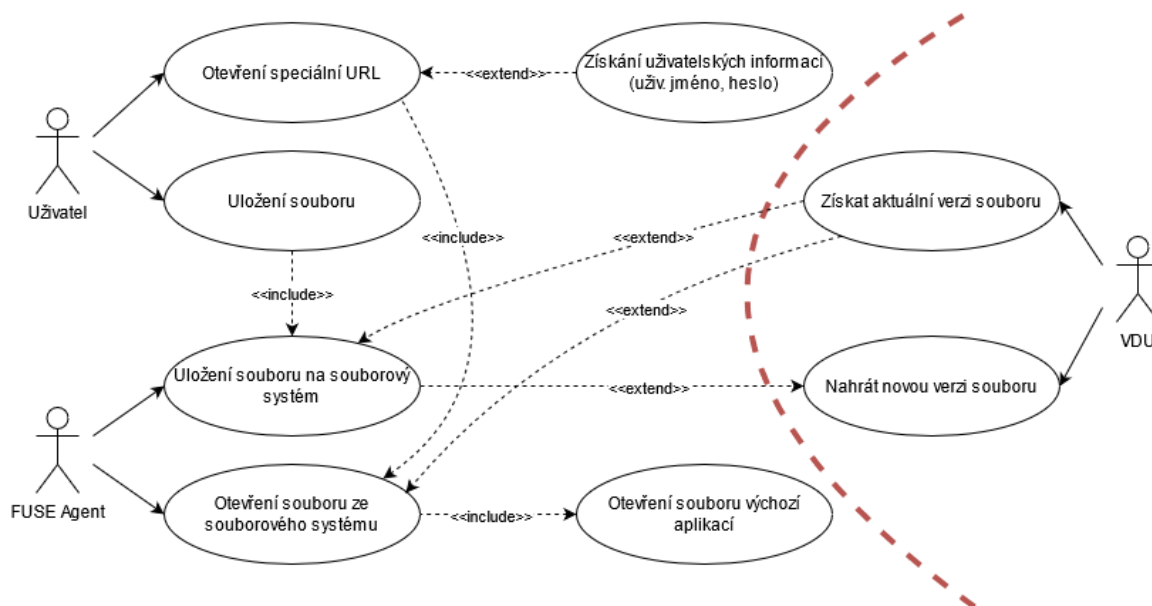
Pro vynucení práv jednotlivých souborů je standardní souborový systém nevyhovující, protože po stažení souboru do počítače uživatele ztrácí VDU kontrolu nad tímto souborem. Po konzultaci s vedoucím práce, doktorem Markem Rychlým, byla zvolena technologie FUSE neboli Filesystem in userspace. FUSE umožňuje vytvořit virtuální souborový systém a implementovat jednotlivé operace jako čtení, zápis atd. Tím je možné dosáhnout dostatečnou kontrolu nad oprávněními a obecně životním cyklem souboru v uživatelské počítači. Znalý uživatel by byl schopen tento systém obejít a přistoupit k souborům z hostovaného souborového systému. k takové operaci jsou však potřeba oprávnění superuživatele `root`.

### 3.1 Případy užití

Na základě požadavků bylo možné přímo vytvořit diagram případu užití. Pouze jedna věc nebyla konkrétně specifikovaná. Webová aplikace musí předat té desktopové přístupový token, pomocí kterého lze volat potřebné API metody VDU. v úvahu přišlo stažení „fake“ souboru obsahující pouze přístupový token. Druhou možností bylo přesměrování na speciální URL z VDU. Prohlížeč by na základě přesměrování rozpoznal, že má otevřít určenou aplikaci a předat ji token jako jeden z parametrů volání. Varianta se speciální URL působí na uživatele mnohem ucelenějším dojmem a nevytváří na souborovém systému dále nepotřebné soubory.

Diagram případů užití zachycuje celý systém jako celek a zobrazuje jednotlivé funkce a jejich návaznosti, které je třeba dále analyzovat a specifikovat v dalších fázích návrhu. v diagramu 3.1 vystupují tři aktéři a to uživatel, FUSE agent/démon a VDU. Implementace VDU není obsahem této práce, proto je oddělena červenou přerušovanou čarou. v diagramu není zanesena například autentifikace, na které závisí veškerá komunikace s VDU. Pro přehlednost byla tato část vypuštěna.

Všechny akce jsou spuštěny uživatelem a většina z nich vyvolá reakci FUSE démona. Jedná se například o uložení, otevření nebo přejmenování souboru ve virtuálním souborovém



Obrázek 3.1: Diagram případu užití

systému. Speciální případ nastává při otevření URL s vlastním protokolem, který má zavolat aplikaci, jež je k danému protokolu v systému přiřazena. Příkladem takového protokolu může být `mailto`, využívaný často u webových stránek. Přesměrování na adresu s tímto protokolem otevře správce elektronické pošty.

K námi definovanému protokolu pojmenovanému `vdu` je třeba mít v systému přiřazenou aplikaci, která pomocí přístupového tokenu uloženého v URL (`vdu://<přístupový token>`) stáhne žádaný soubor z VDU na virtuální souborový systém. Na základě této znalosti byly uskutečněny rozhodnutí při návrhu architektury.

## 3.2 Definice REST API

API obsahuje sedm metod pro ověření dostupnosti, autentifikaci a práci se soubory. Každá funkce má sadu možných návratových hodnot, která je podmnožinou návratových hodnot definovaných protokolem HTTP. Podle tohoto kódu lze rozpoznat, zda bylo dané volání úspěšné, případně jaká chyba při volání nastala. Kódy z rozsahu 200 až 299 označují úspěšná volání a kódy 400 až 599 indikují chybu na straně klienta nebo serveru.

Standardizace definice REST API byla vytvořena v OpenAPI standardu, pro který je dostupných mnoho nástrojů. Jedním z nich je open source nástroj Swagger, pomocí kterého můžete generovat HTML dokumentaci. Vygenerované rozhraní umožňuje i jednoduchou formu testování API. Pomocí standardizované definice bylo také možné rychle vytvořit Mock Server ve webové aplikaci Postman.

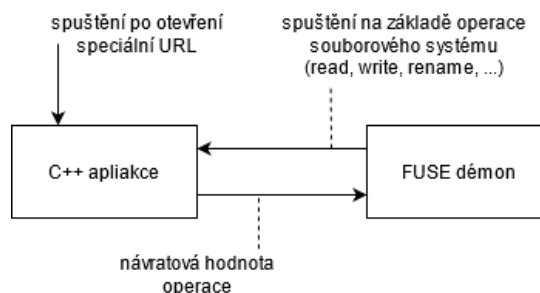
- `/ping`
  - GET `/ping` — pro ověření dostupnosti VDU
- `/auth`
  - GET `/auth/key` — obnova autentifikačního tokenu

- POST /auth/key — získání autentifikačního tokenu
- DELETE /auth/key — zneplatnění autentifikačního tokenu
- /file
  - GET /file/<file-access-token> — stažení obsahu souboru
  - POST /file/<file-access-token> — nahrání obsahu souboru
  - DELETE /file/<file-access-token> — zneplatnění přístupového tokenu

### 3.3 Architektura

Zvažovány byly dvě implementační varianty, jedna verze pracoval pouze s FUSE démonem a druhá rozdělila funkčnost do samostatné aplikace a FUSE démona. První variantu by bylo možné zpracovat dvěma obdobnými způsoby, jež by vedli ke obdobnému řešení. FUSE démon je napsaný v jazyce C, takže jedno z nabízejících se řešení by byla implementace celé aplikace jedním jazyce. Druhým řešením by bylo vytvoření C++ knihovny, která by měla rozhraní pro jazyk C. Takto vytvořená knihovna by mohla být konzumována FUSE démonem. Nejedná se však o nijak kriticky náročnou aplikaci na výkon, takže by varianta s C++ knihovnou byla upřednostňovaná, protože standartní knihovna jazyka C++ a jeho konstrukce umožňují jednodušší vývoj.

Rozdělení na dvě samostatné aplikace, z nichž jedna je FUSE démon by mohlo přinést problémy v udržitelnosti takového systému. Pokud by jedna z aplikací nefungovala, celý systém by nemohl pracovat korektně. Hlavní aplikace by obsluhovala komunikaci s VDU a byla by vždy spuštěna voláním z FUSE démona nebo nepřímo uživatelem. Se znalostí získanou při návrhu diagramu případů užití víme, že je třeba mít registrovanou aplikaci obsluhující otevření speciální URL. i kdybychom vzali místo URL soubor se speciální koncovkou (např. \*.vdu) obsahující přístupový token je stále třeba mít přiřazenou aplikaci pro obsluhu tentokrát otvírání souboru. Démon běžící na pozadí po celou dobu běhu systému nemůže být přiřazen jako aplikace obsluhující takové události. Na základě těchto okolností byla vybrána varianta se dvěma oddělenými aplikacemi k dalšímu zpracování.



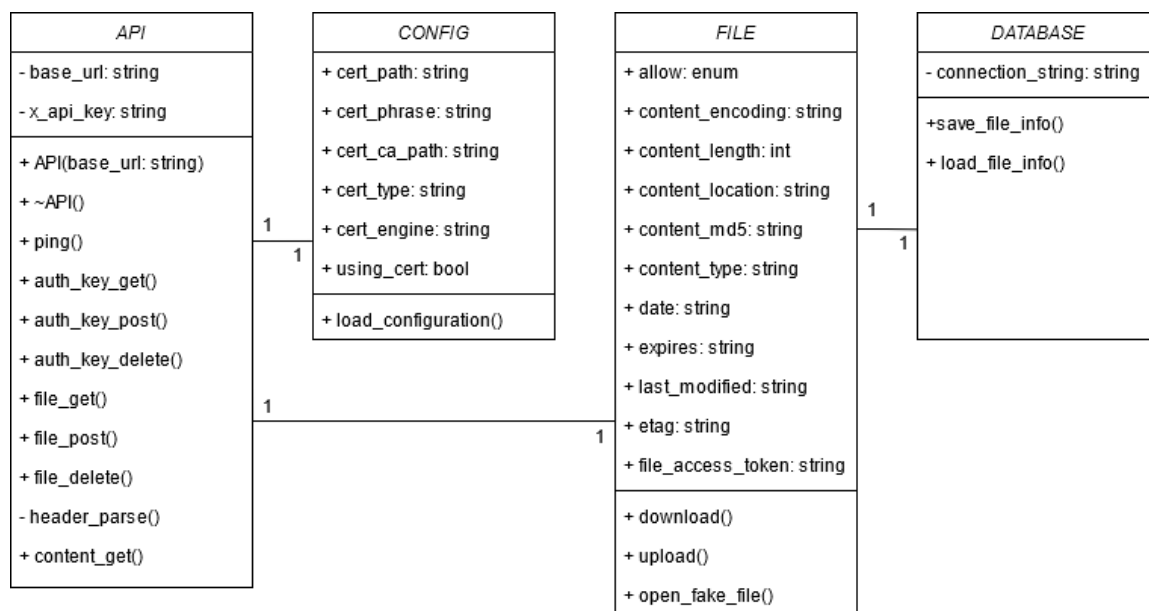
Obrázek 3.2: Architektura

Pokud FUSE démon má vykonat akci, pro kterou nemá dostatečné informace, spustí jako další subprocess druhou hlavní aplikaci a počká na návratovou hodnotu, na základě které se rozhodne, jakým způsobem onu akci dokončí. Může se jednat o situace jako: je soubor určený jen pro četní nebo i zápis, je daný soubor stále validní (nevypršel datum expirace) apod.

V této variantě je FUSE démon velmi minimalistický a zbytek funkcionality je implementován v druhé aplikaci, která běží jen pokud je k tomu vyzvaná, narozdíl od neustále běžícího démona. Démon tedy po celou dobu běhu zabírá méně místa, protože nemusí mít načtené v paměti všechny potřebné knihovny. Tyto knihovny mohou být v paměti i tak namapované, protože je mohou využívat jiné aplikace. Démon musí být spuštěný s oprávněními superuživatele `root` a menší množství kódu běžícího s vysokými oprávněními implikuje menší náchylnost na bezpečnostní chyby. Hlavní aplikace může běžet jen s právy přihlášeného uživatele, což je jedna z dalších výhod oproti druhé zvažované variantě.

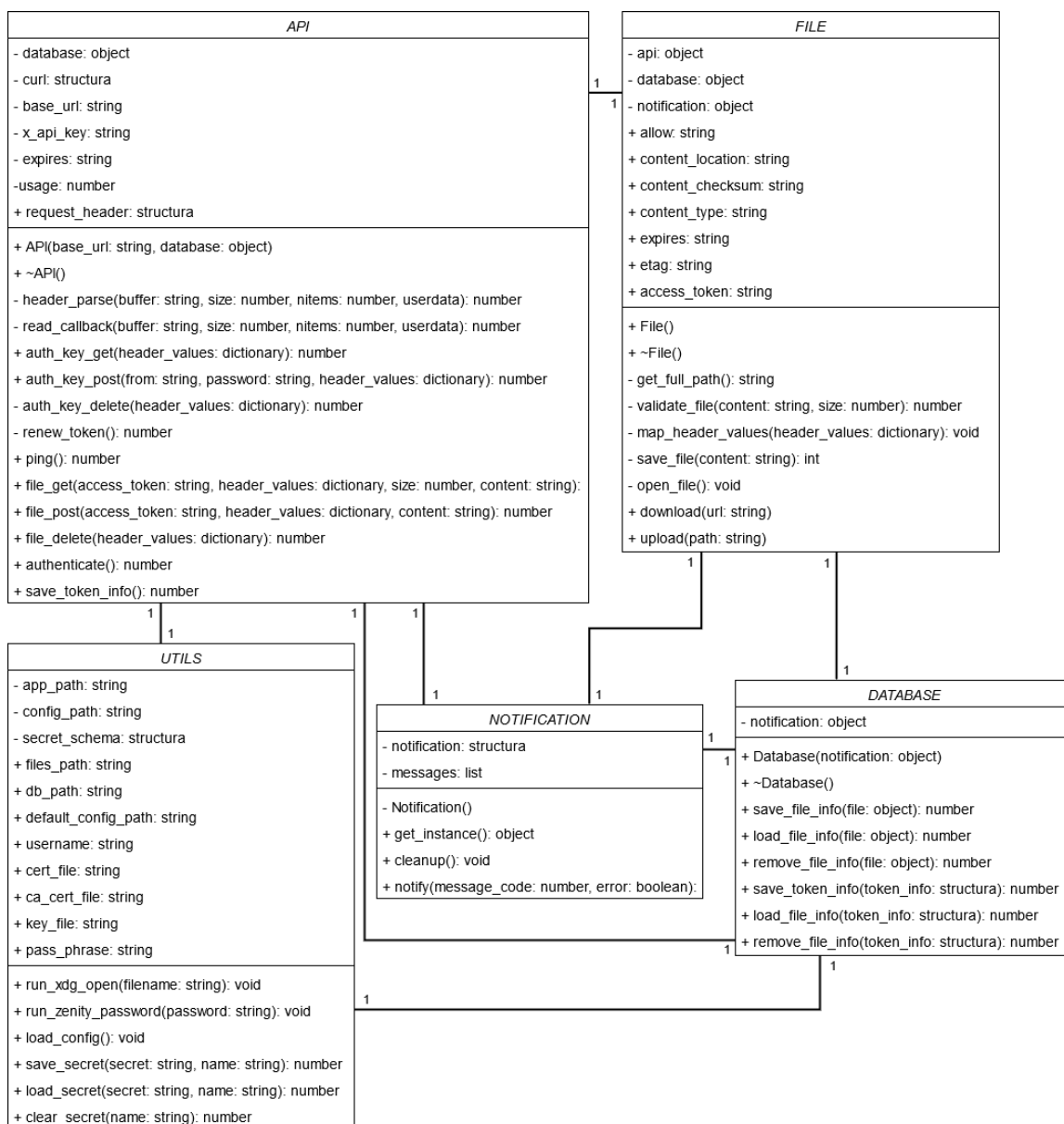
### 3.3.1 Diagram tříd

Aby implementace probíhala s možná nejmenším množstvím komplikací a nebylo třeba již uvažovat nad dalšími rozhodnutími, byl vytvořen diagram tříd 3.3, který se pokouší podrobně vizualizovat třídy s atributy a metodami, jež je třeba implementovat. Jedná se pouze o návrh, který se v průběhu vývoje bude měnit, ale pokud je návrh kvalitní a vývojáři se jej snaží dodržet, neměl by být výsledek zásadně rozdílný.



Obrázek 3.3: Diagram tříd - návrh

Na obrázku 3.4 je možné vidět diagram tříd naimplementovaného programu. Na první pohled vypadá velmi rozdílně, po bližším prozkoumání lze vidět jednu novou třídu oproti návrhu. Jedná se o třídu zobrazující notifikace pro uživatele, která nemá vliv na hlavní funkcionality. Dále přibýlo několik podpůrných funkcí a také přibily některé atributy, případně se přejmenovali. Při implementaci nenastaly žádné zásadní potíže způsobené chybným nebo nepřesným návrhem, tudíž lze návrh označit za úspěšný.



Obrázek 3.4: Diagram tříd - implementace

## 3.4 Technologie

V této sekci budou popsány jednotlivé technologie, které ovlivnili tuto práci. Pro samotný návrh stačí elementární znalosti daných technologií, ale pro implementaci a jejich správné využití je třeba jejich bližší pochopení.

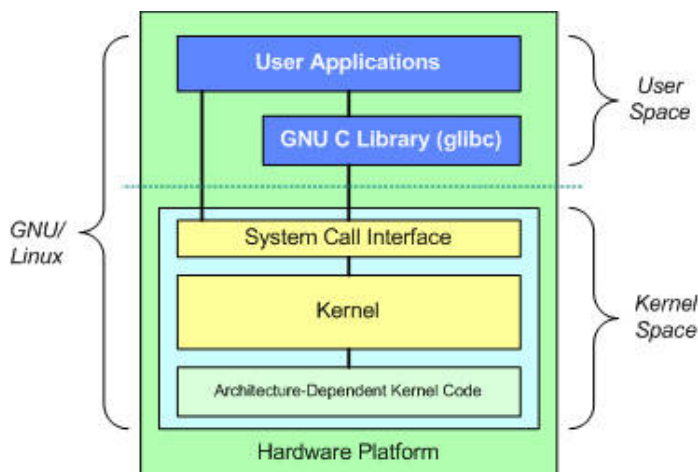
### 3.4.1 GNU/Linux

GNU je rekurzivní zkratka pro „GNU’s Not Unix!“. Jedná se o projekt, který měl za cíl vystavět nový operační systém kompatibilní s Unixem. Všechn kód vytvořený pod tímto projektem je tzv. free software (svobodný software). Jedná se o filosofii vývoje software jejíž

parafrázovaná definice je následující. Svobodný (free) software se vztahuje ke svobodě, ne k ceně. Slovo svobodný (free) by mělo být bráno ve významu svobodný projev (free speech) nikoliv pivo zdarma (free beer) [6].

V projektu GNU postupně vznikali jednotlivé softwarové balíky, které měli ve výsledku poskládat nový operační systém. Pod projektem GNU vznikly aplikace jako překladač GCC, textový formátovač TeX, terminálový shell Bash a mnohé další. Na počátku 90. let byl systém téměř hotový, scházelo jen jádro operačního systému. Ve stejném období dokončoval Linus Torvalds své jádro operačního systému nazývané Linux. [28] [23]

Vznikla tedy myšlenka tyto dva projekty spojit a po vyřešení problémů s kompatibilitou jednotlivých částí projektů vznikl GNU/Linux. Balíčky projektu GNU umožňovali práci se systémem postaveným nad Linuxovým jádrem. Všechny dnešní distribuce jako jsou Debian, Fedora a další jsou distribuce systému GNU/Linux. Linux je název pouze jádra pro operační systém, který zevšeobecněl a je tímto názvem chybně označován celý systém. Projekt GNU dodnes vyvíjí své jádro pojmenované GNU Hurd, se kterým by vznikl operační systém GNU, což byl počáteční cíl projektu. [23]



Obrázek 3.5: Architektura GNU/Linux. Převzato z [16]

Architekturu GNU/Linux vyobrazená na obrázku 3.5 vyznačuje hranici mezi kernelem a uživatelským prostorem. Jádro/kernel běží v tzv. privilegovaném režimu, což je režim s nejvyššími oprávněními a může na CPU provádět jakékoliv operace. Všechny ostatní aplikace běží v uživatelském prostoru a jsou omezeny jen na provádění podмноžiny operací oproti privilegovanému režimu. Pokud aplikace potřebuje vykonat operaci, která vyžaduje vyšší oprávnění, zažádá jádro operačního systému, aby tuto operaci provedlo za místo ní. Tím je docíleno vyššího zabezpečení operačního systému.

### 3.4.2 C++/C

Jazyk C patří do skupiny dlouho používaných programovacích jazyků. Se svojí bohatou historií sahající až do 70. let 20. století velmi ovlivnil další nástupnické procedurální jazyky jako C++, C#, Java atd. v roce 1989 byl jazyk standardizován jako ANSI C a následně přišli ISO standardy označované jako C99, C11, C17. Jazyk C je řazen mezi vysokoúrovňové programovací jazyky, přestože relativně jednoduchý a obsahuje malé množství konstrukcí v porovnání s ostatními jazyky.[14]



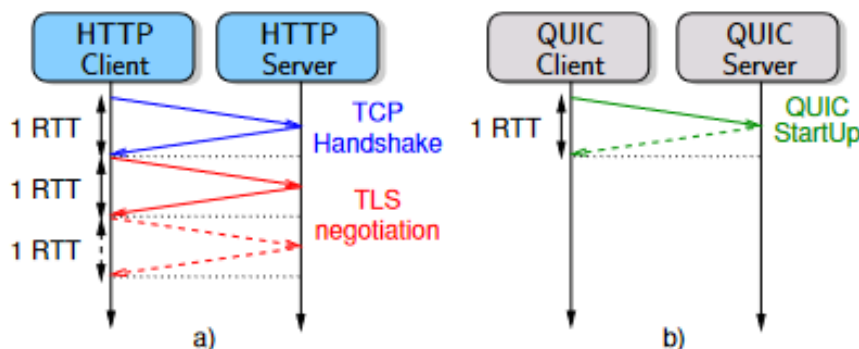
Vývoj C++ je oproti C dynamičtější, což můžeme vidět na jednotlivých ISO standardech C++98, C++03, C++11, C++14, C++17 a C++20. Jazyk C++ se v poslední dekádě velmi proměnil a období od standardu C++11 je označováno za dobu „moderního C++“. v posledním standardu přibyla například podpora konceptů nebo rozsahů.[15]

Při využití překladačů GCC nebo G++ lze využít tzv. GNU rozšiřující standardy. Poté je možné využít některé konstrukce, které nejsou specifikované v oficiálním ISO standardu. V případě jazyka C se jedná o standardy pojmenované GNU99, GNU11 a GNU17, v případě C++ se jedná o GNU++11 a jemu podobné.

### 3.4.3 HTTP

Hypertext Transfer Protokol je protokol aplikační vrstvy TCP/IP stacku. Byl vytvořen pro přenášení hypertextových souborů v 90. letech 20. století a dnes se jedná o nejrozšířenější aplikační protokol. Poslední stabilní verze protokolu označená jako HTTP/2 byla vydána v roce 2015. Aktuálně je ve fázi specifikace nástupce pojmenovaný HTTP/3.

HTTP v čisté formě dnes již téměř není možné potkat, ale využívá se ve spojení s šifrovacím protokolem TLS. Při komunikaci se nejprve vytvoří TCP spojení a poté šifrovaný kanál, skrze který je HTTP bezpečně přenášeno. Toto řešení není nejefektivnější, protože vznikalo postupně a spojovali se technologie, které nebyly od počátku navrženy pro tento účel. Ve většině případů uživatel na stabilním připojení nezaregistruje žádné potíže, kterými tyto technologie trpí. Jedním z problémů je dlouhá časová prodleva, než se kanál ustálí pro přenášení dat. Tento problém má vyřešit právě nová verze HTTP. [4]



Obrázek 3.6: Porovnání HTTP verze 2 (a) a 3 (b). Převzato z [4]

Nová verze využívá nový protokol QUIC, který je přenášen protokolem UDP namísto TCP jako tomu bylo u předchozích verzích HTTP. Na obrázku 3.6 je možné vidět, že navázání komunikace je ze tří výměn informací zredukováno jen na jednu. Formát hlavičky HTTP by měl zůstat nezměněný, a tudíž je zpětně kompatibilní.

Pro tento projekt je důležitá práce s datумы, přesněji formáty umožněné přenášet ve validní HTTP hlavičce. RFC7231 specifikuje tři formáty datumů, z nichž dva jsou označeny jako zastaralé, ale jsou stále validní. Udržované jsou kvůli zpětné kompatibilitě s protokolem HTTP/1.1. [10]

- Upřednostňovaný formát
  - Sun, 06 Nov 1994 08:49:37 GMT (IMF-fixdate formát)
- Zastaralé formáty
  - Sunday, 06-Nov-94 08:49:37 GMT (RFC 850 formát)
  - Sun Nov 6 08:49:37 1994 (ANSI C asctime() formát)

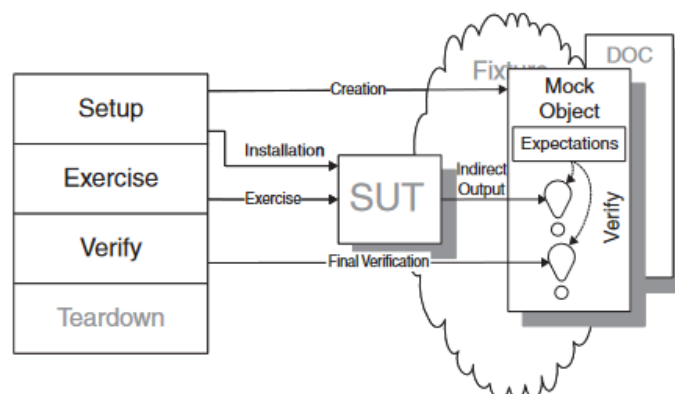
### 3.4.4 OpenAPI

Jedná se o standard pro definování metod REST API, který má v čase psaní této práce poslední verzi 3.0.3. Zápis je možný ve dvou jazycích: JSON a YAML. Oba je možné strojově zpracovat a přitom jsou pro člověka stále čitelné.

Specifikace je velmi obsáhlá a umožňuje popsat API velmi detailně a přitom efektivně. Například je možné vytvářet šablony datových struktur, které jsou používány jako vstupní nebo naopak návratové hodnoty jednotlivých funkcí. v definici je pak možné na tuto šablonu odkázat. Struktura souboru je hierarchická a skládá se z jednotlivých objektů. Každý objekt má povinné a volitelné atributy. Definovat tedy lze hlavičky požadavku, URL query parametry, návratové kódy a k nim přiřazené hlavičky odpovědi a odpovědi samotné, případně příklady možných odpovědí. Toto je jen stručný výpis možností, které standard OpenAPI definuje. [21]

### 3.4.5 Mock Server

Mock Server je speciální případ testovacího vzoru Mock Object, který spadá do kategorie Test Double. Test Double je kategorie testovacích vzorů, do které patří vzory jako právě Mock Object, Fake Object, Test Stub apod. Jednotlivé vzory mohou být mezi sebou zaměnitelné, protože jejich definice a použití je velmi podobné. Tato kategorie vzorů je využívána pro systémy, které pracují s určitou externí komponentou, která není dostupná nebo by testování ovlivnila svými nežádoucími vedlejšími účinky.[20, 522–524] k využití Mock Serveru bylo přistoupeno z důvodu nedostupnosti VDU, při vzniku této práce. Mock Server nebyl využíván jen pro účeli testování, ale i během vývojové fáze. Aby nedošlo k zanesení chyby do software během vývoje, byla napsána sada testů, která ověřovala správnou implementaci Mock Serveru vůči předem specifikované definici.

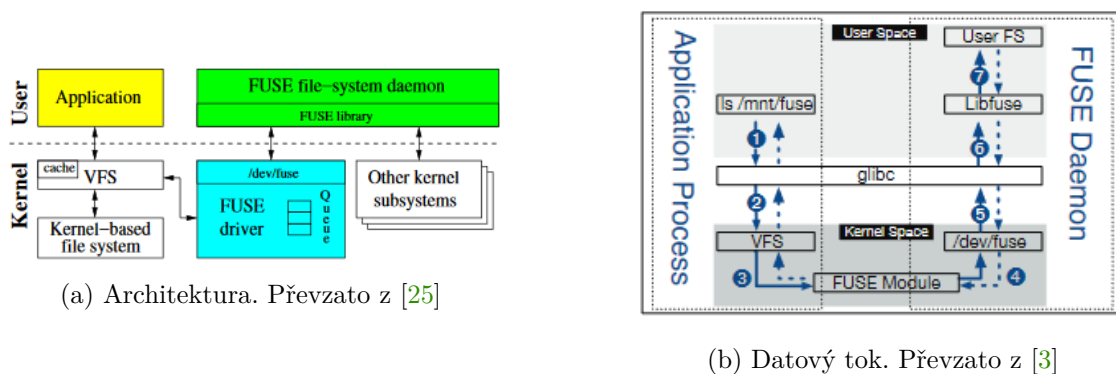


Obrázek 3.7: Test Double schéma. Převzato z [20, 544]

Mock Object pokrývá funkcionalitu Test Stub vzoru a na vývojáři záleží, jakým způsobem využije tyto možnosti. v kontextu této práce by se používaný Mock Server dal klasifikovat jako pouhý Test Stub, protože není využívána verifikující část, kterou definuje Mock Object vzor. [20, 524–525] Verifikovaný je pouze výstup funkcí v testovaném systému.

### 3.4.6 Souborový systém v uživatelském prostoru (FUSE)

FUSE je označení pro File system in userspace a jedná se o framework umožňující vytvořit vlastní virtuální souborový systém v uživatelském prostoru. FUSE se skládá ze dvou částí, démona a modulu pro jádro operačního systému. Je-li modul načtený registruje ovladač v Linuxovém VFS (Virtual filesystem). Tento ovladač poté představuje proxy pro všechny běžící demony. FUSE driver obsahuje několik front pro obsluhu všech potřebných akcí. Nejprioritnější fronta je pro obsluhu přerušení, ostatní fronty jsou pro obsluhu synchronních, asynchronních a tvz. forget požadavků. Priorita zpracování těchto front je daná poměrem 8 ku 16 pro synchronní a asynchronní ku forget požadavkům. [25]



Obrázek 3.8: Vysokoúrovňové diagramy architektury a datového toku pro FUSE

Démon zpracovává postupně požadavky, které se přidávají do front ve FUSE ovladači. Na obrázku 3.8b je vidět jak požadavek prochází jednotlivými částmi systému. Aplikace přistoupí na připojený FUSE souborový systém a požadavek projde skrze VFS a FUSE ovladač až k démonovi. Démon daný požadavek zpracuje, ve většině případů přistoupí na hostitelský souborový systém a vykoná na něm požadovanou operaci. Odpověď je poté navrácena stejným způsobem, jakým byla přenesena k FUSE démonovi.

Démon nemusí implementovat všechny operace, záleží jen na požadavcích. Je-li potřeba vytvořit souborový systém jen pro čtení, implementují se například operace INIT, OPEN, GETATTR, READ a RELEASE. Mohou být implementované další kvůli optimalizacím nebo řešící specifické případy. Více informací k jednotlivým operacím je možné dohledat v dokumentaci libfuse<sup>1</sup>.

<sup>1</sup>[http://libfuse.github.io/doxygen/structfuse\\_\\_operations.html](http://libfuse.github.io/doxygen/structfuse__operations.html)

Skupina	Typ požadavku
Speciální	INIT, DESTROY, INTERRUPT
Metadata	LOOKUP, FORGET, BATCHFORGET, CREATE, UNLINK, LINK, RENAME, RE-NAME2, OPEN, RELEASE, STATFS, FSYNC, FLUSH, ACCESS
Data	READ, WRITE
Atributy	GETATTR, SETATTR
Rozšířené atributy	SETXATTR, GETXATTR, LISTXATTR, REMOVEXATTR
Symbolické linky	SYMLINK, READLINK
Složky	MKDIR, RMDIR, OPENDIR, RE-LEASEDIR, REaddir, REaddirPLUS, FSYNCDIR
Zamíkaní	GETLK, SETLK, SETLKW
Misc	BMAP, FALLOCATE, MKNOD, IOCTL, POLL, NOTIFY-REPLY

Tabulka 3.1: Tabulka typů požadavku pro zpracování FUSE démonem. Převzato z [25]

### 3.4.7 D-Bus

D-Bus je systém pro posílání zpráv mezi jednotlivými aplikacemi. Přesněji se jedná o Inter-Process Communication (IPC) a Remote Procedur Calling (RPC) mechanismus vytvořený pro komunikaci mezi jednotlivými procesy na stejném počítači. D-Bus funguje na Unix-like systémech a port pro operační systém Windows je také dostupný. Objekty jsou identifikovány kombinací tzv. „bus name“ a „object path“. [5]

Existují dva typy „bus name“, jeden je unikátní a je přiřazen každé připojené aplikaci na sběrnici. Pro tyto unikátní jména lze vytvořit alias nazývaný jako „well-known bus name“. Například pro přístup ke keyringu je používán alias „org.freedesktop.secret“. „Object path“ poté identifikuje danou funkci zpřístupněnou pro D-Bus. Komunikace klienta s danou službou poté funguje velmi podobně jako zjišťování IP adresy z doménového jména pomocí DNS. Klient započne komunikaci dotazem na přeložení „well-known bus name“. v odpovědi dostane unikátní „bus name“, který může využít pro komunikaci s cílovým procesem. [5]

### 3.4.8 Keyring

Pro ukládání hesel nebo tokenů pro aktuálně přihlášeného uživatele v systému lze využít službu pojmenovanou keyring. Podle implementace se heslo uloží na disk nebo je jen načtené v operační paměti. Pokud je uživatel přihlášený, je mu heslo k dispozici, při vypnutí systému nebo odhlášení uživatele je heslo z operační paměti odstraněno. Ukládá-li se heslo na disk, je zašifrováno a odšifrovat ho může jen uživatel, který heslo do keyringu uložil. Pokud není heslo k dispozici musí jej aplikace od uživatele vyžádat a do keyringu případně uložit. [27]

GNU/Linux obsahuje keyring aplikaci pojmenovanou linux-keyring. Některé desktopové prostředí obsahují své keyring aplikace, které rozšiřují možnosti základního linux-keyringu. Například desktopové prostředí GNOME obsahuje aplikaci gnome-keyring a desktopové prostředí KDE zase aplikaci kde-wallet. [27]

## Kapitola 4

# Popis implementace a nasazení aplikace

Při vývoji byly využity kromě standardních knihoven jazyka C a C++ také knihovny umožňující práci s databází, notifikování uživatele pomocí systémového notifikačního systému nebo knihovna pro vytvoření automatizovaných testů.

- **libcurl** — Knihovna pro jednoduché přenášení dat po síti. Podporuje velkou velkou řadu protokolů jako FTP, HTTP, IMAP atd. Šifrování je zpřístupněno pomocí externí knihovny (např. openssl). V posledních verzích knihovny je možné pracovat s draft verzí protokolu HTTP verze 3. [17]
- **libnotify** — Jedná se o implementaci Desktop Notification Specification<sup>1</sup>, který specifikuje jednotné rozhraní pro vytváření notifikací v desktopových prostředích. Pro korektní využití libnotify je třeba mít běžící notifikační server, se kterým se komunikuje pomocí D-Bus. Většina desktopových prostředí má vlastní implementaci notifikačního serveru startující při spuštění systému. [8]
- **libfuse** — Nabízí funkce pro vytvoření FUSE démona, komunikující s FUSE modulem kernelu. Implementací jednotlivých operací je možné vytvořit vlastní souborový systém v uživatelském prostoru. [18]
- **libsqlite3** — Knihovna pro práci s jednosouborovou databází SQLite. Jedná se o malou samostatnou a nejpoužívanější databázi na světě. Interakce s databází je vedena v jazyku SQL. Zajímavost týkající se SQLite je plán podpory a udržování do roku 2050. [1]
- **libssl** — Šifrovací knihovna pro obecné použití, ale s primárním zaměřením na protokoly TLS a SSL. V této práci byla použita jen pro výpočet kontrolního součtu algoritmem MD5. [26]
- **libsecret** — Se službou keyring je možné komunikovat pomocí D-Bus sběrnice. Knihovna libsecret zapouzdřuje tuto komunikaci do svého API a vytváří tak jednoduše použitelné rozhraní. [19]
- **libgtest** — Google Test je testovací framework pro psaní automatizovaných C++ testů založený na xUnit architektuře. Zpřístupňuje makra pro ověření výroku nebo podmínky je-li pravdivá. [12]

---

<sup>1</sup><https://developer.gnome.org/notification-spec/>

Další balíčky využitě při vývoji nebo jsou potřeba pro běh aplikace samotné.

- **Zenity** - Aplikace pro vytváření jednoduchých dialogových oken.
- **DB Browser for SQLite** neboli **sqlitebrowser** - Grafický prohlížeč SQLite databází, umožňujícíobrazení obsahu tabulek, vytvoření i úpravu dat nebo struktury databáze.
- **Doxygen** - Nástroj pro generování dokumentace z anotací v kódu. Dokumentace může být ve formátu pdf dokumentu nebo HTML souborů.

## 4.1 Využívané cesty v Linuxové adresářové struktuře

Linuxová hierarchie má své konvence, kde by se jednotlivé soubory měli nacházet. Také existuje specifikace nazývaná XDG Base Directory specification, která řeší například ukládání aplikačních dat pro jednotlivé uživatele. Tím je myšleno, že každý uživatel může mít vlastní konfiguraci aplikace. V tomto případě proto není možné používat globální složky vytvořené k tomuto účelu (např. `/var`). Některé z použitých souborů nebo složek mají název začínající tečkou, jedná se o skryté soubory/složky. Znalý uživatel si tyto složky může ve svém průzkumníku souborů zobrazit a jednoduše k nim přistoupit.

Aplikace je nastavena a byla pro tento případ i vyvíjena, že každý uživatel má vlastní konfiguraci a také ve FUSE souborovém systému vydí pouze své soubory. Toto řešení působilo při vývoji velké potíže, protože FUSE démon musí být spouštěn s právy superuživatele `root` a tím pádem nemůže použít proměnnou `$HOME`. Bohužel totiž neukazuje do domovského adresáře přihlášeného uživatele, ale do domovského adresáře superuživatele `root`.

Po podrobném znovu přečtení dokumentace k FUSE technologii se objevila jedna možnost, jak tento problém vyřešit. V dokumentaci však nebyl tento případ dobře rozveden, ale finální řešení bylo, že binární soubor reprezentující FUSE démona musel mít nastaveného vlastníka uživatele `root` a také tomuto souboru musel být nastaven tzv. suid bit. Díky tomuto nastavení může FUSE démon běžet pod přihlášeným uživatelem, avšak s oprávněními superuživatele `root`.

Seznam souborů a cest využívaných aplikací:

- **/usr/local/bin** — Místo pro programy neboli spustitelné soubory, přístupný pro normálního uživatele. V této složce jsou uloženy oba binární soubory pojmenované **vdu-app** a **vdu-app-fuse**. Druhá jenovaná je FUSE démon, který musí mít vlastníka uživatele `root` a nastavený tzv. suid bit. Aplikace **vdu-app** nevyžaduje žádné specifické požadavky.
- **/mnt/vdu** — Složka **/mnt** je určená pro připojování (mount) externích paměťových zařízení jako USB flash disk, CD apod nebo pro připojení sídlené složky nebo síťového disku. Do této složky bude připojen FUSE souborový systém, který bude využíván pro ukládání stažených souborů z VDU.
- **\$HOME/.local/share/vdu/** — Na této cestě budou umístěny aplikační data jako databáze a složka pro ukládání obsahu z FUSE souborového systému. Každému uživateli, který bude na daném operačním systému využívat **vdu-app** bude vytvořena tato složka.

- **/vdu.db** — SQLite 3 databáze pro ukládání podpůrných informací o souborech stažených z VDU a také metadata potřebná pro běh programu.
- **/fuse** — Složka na hostitelském souborovém systému, kterou využívá FUSE démon pro čtení a ukládání dat, které jsou následně zobrazena ve složce **/mnt/vdu**. Pokud by uživatel chtěl obějit FUSE souborový systém, mohl by k datům přistoupit napřímo skrze tuto složku. Poté by mohl číst soubory, které již vyexpirovali, nebo zapisovat do souborů, ke kterým má pouze práva na čtení. Vlastník této složky je **root**, který jako jediný může z této složky číst nebo do ní zapisovat. Jak již bylo výše popsáno FUSE démon běží s právy superuživatel **root**, takže může nad touto složkou operovat.
- **\$HOME/.vdu.conf** — Konfigurační soubor aplikace, který je také pro každého uživatele jedinečný.
- **/usr/share/applications** — Složka pro ukládání tzv. Desktop Entries, což jsou soubory definující aplikace, které mají být určitým způsobem integrovány do desktopového prostředí. Desktop Entry obsahuje údaje jako jméno, ikonu, popis, přiřazený typ souboru atd.
- **/etc/systemd/user** — Konfigurační složky systému **systemd**, využívaná pro definici jednotlivých systemd služeb (systemd service). V tomto případě pro služby běžící pro každého jednotlivě přihlášeného uživatele. Systémové služby by byla umístěna namísto složky **user** ve složce **system**.

#### 4.1.1 Konfigurační soubor

Jedná se o textový soubor obsahující dvojici klíč a hodnota oddělené znakem rovnítko. Každá tato dvojice musí být na samostatném řádku. Prázdné řádky nebo řádky začínající znakem „#“ jsou přeskakovány. Jedná se o standardní pojetí konfiguračního souboru na systému GNU/Linux.

Konfigurační soubor umožňuje nastavit uživatelské jméno, jména certifikátů nebo klíčů a v neposlední řadě také volitelnou frázi pro zabezpečení certifikátu. Všechny hodnoty jsou volitelné neboli pokud nebudou v konfiguračním souboru nastavené, aplikace zvolí jejich výchozí hodnotu. Případně vyzve uživatele pro doplnění informací, které jsou třeba pro správné fungování.

Pokud není nastaveno uživatelské jméno, tak výchozí hodnotou je uživatelské jméno z operačního systému. Pro certifikáty je situace podobná, nejsou-li nastavené všechny hodnoty jako jméno klientského certifikátu, jméno certifikátu certifikační autority a jméno souboru obsahující privátní klíč, tak je využita autentifikace přes heslo. Bezpečnostní fráze je volitelná, protože ne každý certifikát musí být touto frází zabezpečeny. Nežadává se absolutní cesta k certifikátům, ale pouze jejich jméno, protože certifikáty jsou následně načítány z systémového úložiště certifikátů.

#### 4.1.2 SQLite databáze

Hlavní vyvíjená aplikace potřebuje uchovávat data mezi jednotlivými běhy a SQLite databáze je pro tento případ asi nejvhodnější metoda. Jedná se o jednosouborovou databázi, která zapouzdřuje kompletní SQL engine. Databáze obsahuje dvě tabulky a to **Files** a **Metadata**. Databáze se vytváří automaticky pokud se na dané cestě nenachází.

V tabulce **Files** jsou udržovány informace o jednotlivých souborech, které se právě nacházejí na FUSE souborovém systému. Tabulka **Metadata** má pouze dva sloupce a funguje pro ukládání dvojic klíč a hodnota stejného typu. Taková tabulka by se dala přirovnat slovníku v programovacím jazyce Python nebo C#. C++ ekvivalent by byl kontejner `std::map`. Využívat tabulku takovým způsobem není konvenční, ale nezatěžuje to vývoj projektu dalším odlišným způsobem ukládání dat (např. do strukturovaného souboru).

<i>Files</i>	<i>Metadata</i>
(PK) Location: string	(PK) Key: string
Allow: string	Value: string
Type: string	
Expires: string	
ETag: string	
AccessToken: string	
Usage: string	

Obrázek 4.1: Schéma SQLite databáze.

Pokud SQL dotaz obsahuje parametry využívá se postup předvytvoření dotazu bez parametrů a jejich následné navázání na již zkompileovaný dotaz. Tento postup by měl zabránit technice napadení databázového systému nazývané SQL injection. K tomuto účelu obsahuje libsqlite3 sadu funkcí jako `sqlite3_prepare`, `sqlite3_bind`, `sqlite3_column`. Veškerá komunikace s databází je zabpouzdrěná ve třídě `Database`.

## 4.2 xdg-utils

Projekt `xdg-utils` je soubor nástrojů a shell scriptů, který se snaží vytvořit jednotné rozhraní pro práci na různých desktopových prostředích. Asi polovina nástrojů se zaměřuje na proces instalace aplikace a druhá polovina se zase věnuje integrace apliakcí do desktopového prostředí jako otevření souboru ve specifické aplikaci nebo přiřazení ikony pro určitý typu souborů.

Dále se budeme zabývat jen nástroji `xdg-mime` a `xdg-open`, které jsou v rámci této práce využívány. `Xdg-open` zajišťuje otevření určité URL nebo typu souboru v přiřazené aplikaci. Přiřazovat lze pouze k aplikacím popsáných tzv. Desktop Entries, které jsou specifikové v Desktop Entry Specification<sup>2</sup>. Specifikace také definuje vestavené proměnné pro předávání cesty k souboru nebo vyhledávané URL. Příkaz `xdg-open` je volán v případě, že je potřeba uživateli otevřít soubor ve správné aplikaci po stažení souboru z VDU.

<sup>2</sup><https://specifications.freedesktop.org/desktop-entry-spec/latest/>



```
[Desktop Entry]
Type=Application
Version=1.0
Name=VDU App
Comment=Application for handling remote VDU files in desktop enviroment
Exec=/usr/local/bin/vdu-app --url-handler %u
MimeType=x-scheme-handler/vdu;
```

Výpis 4.1: Příklad struktury Desktop Entry souboru

Druhý nástroj `xdg-mime` umožňuje vytvořit nový MIME type nebo přiřadit k určitému MIME typu Desktop Entry aplikaci. Také umí vyhledávat jaké Desktop Entries jsou přiřazené k určitému MIME typu nebo k jakému MIME typu je přiřazena daná Desktop Entry. Pro registraci speciálního URL protokolu se využívá MIME type pojmenovaný `x-scheme-handler/<název protokolu>`.

## 4.3 Komunikace s API

Pro komunikaci s VDU skrze protokol HTTP byla vybrána knihovna `libcurl` pocházející z projektu `cURL` vyvíjející stejně pojmenovaný nástroj. Knihovna může provádět synchronní i asynchronní volání na základě zvoleného rozhraní. Při vývoji bylo využito tzv. Easy rozhraní umožňující pouze synchronní volání. Využití asynchronního volání nedávalo smysl, protože by aplikace neměla v daném mezechase co vykonávat. Všechny API volání definované externím zadavatelem byly implementované ve třídě pojmenované `API`.

Metoda pro zneplatnění autentifikačního tokenu (`DELETE /auth/key`) nebyla jako jediná využita, protože aplikace implementuje vlastní způsob zneplatnění tohoto tokenu. Blíže je tato implementace popsána v podsekcí 4.3.2.

### 4.3.1 Mock Server

Pro implementaci Mock Serveru byla vybrána aplikace `Postman`. Bohužel to nebyla úplně nejlepší volba. Nedostatky tohoto řešení byly lehce limitující, ale až v pozdějších fázích vývoje projektu a hledat nové řešení postrádalo význam. Jiné řešení by mohlo trpět jinými nedostatky a proto bylo zvolen kompromis.

Aplikace `Postman` totiž využívá některé HTTP hlavičky požadavku pro identifikaci nebo rozhodování jak naložit s daným voláním. Při využití hlavičky `x-api-key`, která podle specifikace API VDU má obsahovat autentifikační token. Aplikace `Postman` toto pole využívá pro ověření přístupu k privátním Mock Serverům. Aby využívaný Mock Server mohl zůstat veřejný, přejmenovalo se pole `x-api-key` na `x-api-key-test`. Toto nastavení muselo být řešeno na úrovni kódu a proto byly do aplikace přidány direktivy preprocesoru pro podmíněný překlad. Pokud je definovaný symbol `VDU_DEV`, tak je využíváno jméno hlavičky s koncovkou `test`.

Definovaný je ještě jeden symbol `VDU_HTTP_3`, který v knihovně `libcurl` vynucuje využití protokolu HTTP/3. Tato funkcionality nebyl otestována, protože aplikace `Postman` tento protokol ještě nepodporuje. Jedná se však o možnost dalšího rozvoje, který je třeba pouze otestovat a odladit případné nedostatky a chyby.

### 4.3.2 Autentifikace

VDU umožňuje autentifikovat uživatele pomocí uživatelského jména a hesla nebo uživatelského jména a klientského certifikátu. Jak bylo zmíněno v podsekcí 4.1.1 tak uživatelské jméno je možné nastavit v konfiguračním souboru nebo je jako výchozí hodnota použito jméno z operačního systému.

Při využívání hesla je uživatel poprvé dotázán za pomoci aplikace Zenity, aby zadal své heslo. Aplikace poté zadané heslo uloží do keyring aplikace a je-li potřeba znovu ověřit uživatelskou identitu je heslo vyhledáno v keyring aplikaci a automaticky využito pro tuto akci. Pokud se nejedná o keyring aplikaci ukládající data perzistentně nebo je heslo manuálně odebráno, tak je uživatel při nemožnosti heslo vyhledat znovu vyzván k zadání hesla.

Využití certifikátů je pro uživatele pohodlnější možností, protože nenastane situace, na kterou by musel uživatel přímo reagovat a vše se děje automaticky. Pro využití certifikátů je třeba klientský certifikát tak i certifikát certifikační autority nainstalovat do systému pomocí nástrojů specifických pro danou distribuci. Jsou-li certifikáty nainstalované, poté stačí nastavit v konfiguračním souboru korektně hodnoty a identita uživatele je následně ověřena za pomoci klientského certifikátu.

Poté co je uživatel ověřen aplikací obdrží od VDU token, který se uloží také do keyring aplikace pro případné znovupoužití. Informace o tokenu jsou uloženy do tabulky **Metadata** v databázi. Jedná se přesněji o datum expirace a počet dalších použití. Aplikace má totiž implementovanou funkcionalitu, že každý autentifikační token může být využit maximálně 5krát. Pokud má být token použit po šesté, tak je místo toho zavolána funkce pro výměnu tokenu a nový navrácený token je použit namísto starého.

## 4.4 FUSE démon

Démon připojuje obsah složky `$HOME/.local/share/vdu/` do `/mnt/vdu` a pokud v připojeném bodu nastane nějaké akce řádně ji zpracuje. Implementace démona je braná minimalisticky a byla inspirována příkladem z dokumentace pojmenovaném **passthrought.c**, který zpřístupňuje kořenový adresář v přojeném bodě. Pro každou funkci implementující určitý typ požadavku byla brána jako základ funkce realizující stejný požadavek z příkladu **passthrought**. Tento příklad byl následně modifikovaný, aby odpovídal stejné stylizaci, poté byl doplněn o komentáře. Nakonec byla přidána další funkcionalita, případně byla modifikovaná ta stávající.

FUSE démon bude registrován jako uživatelská systemd služba, která se spustí pro každého přihlášeného uživatele. Stav této služby bude možné kontrolovat pomocí příkazu `systemctl status vdu.service`.

### 4.4.1 Práce se soubory

Při práci se soubory na FUSE soubrovém systému mohou nastat tři situace, které jsou důležité. Jedná se o čtení, uložení a přejmenování souboru. Při čtení se kontroluje, jestli daný soubor není expirovaný, případně jestli soubor není write-only (VDU může změnit po jendom přečtení práva na write-only). Pokud se zapisuje, může se jednat o první zápis nebo modifikaci stávajícího souboru. Modifikovaný soubor je uploadován na VDU v případě, nejedná-li se o soubor určený jen pro čtení. Přejmenování souboru je velmi podobné jako kdyby byl modifikován, akorát se na VDU nahrává stejný soubor s jiným jménem.

## 4.5 Shrnutí

Vývoj provázelo několik nepříjemností a zdržení, ale veškeré problémy se podařilo kompletně vyřešit nebo minimálně nalézt dostačující řešení. Poslední nezmíněnou věcí jsou notifikace. Uživatel je informován pomocí notifikačního systému desktopového prostředí o chybách nebo informacích, že se na pozadí něco děje.

## Kapitola 5

# Nasazení a testování

Pro úspěšné používání aplikace je třeba ji nejprve nainstalovat do systému uživatele. Někteří schopnější uživatelé by byly schopni zkompilevat ji ze zdrojových souborů, ale pro většinu bude snadnější, bude-li jim distribuován instalační balíček. Tento balíček bude následně možné jednoduše nainstalovat skrze balíčkovacího manažera a při instalaci se provedou veškeré úkony potřebné pro korektní fungování celého systému.

Byly vybrány dva balíčkovací systémy, které pokrývají dvě používané větve GNU/Linux distribucí. Balíčkovací systém DPKG používaný v systému Debian a jeho derivátech. RPM balíčky pokrývají různé operační systémy napojená na ekosystém společnosti Red Hat.

### 5.1 Sestavení ze zdrojových souborů

Pro sestavení ze zdrojových souborů stačí stáhnout zdrojové soubory z git repozitáře a využít připravený Makefile soubor. Samozřejmě je nutné mít nainstalované všechny vývojářské knihovny a nástroje. Protože se jedná C/C++ aplikace je pro sestavení využitý GNU kompilátor GCC respektive g++ wrapper v případě C++ kódů. Jak již bylo výše zmíněno, aplikace umožňuje podmíněný překlad se symboly `VDU_DEV` a `VDH_HTTP_3`, které je případně možné upravit v Makefile souboru.

Makefile obsahuje targety pro sestavení hlavní `vdu-app`, FUSE démona a automatizovaných testů psaných v GoogleTest frameworku. Také je možné vygenerovat HTML dokumentaci pomocí nástroje Doxygen. Celá aplikace je okomentovaná ve formátu, který umožňuje vygenerování dokumentace aplikačního rozhraní.

### 5.2 DPKG

Jedná se o nízko úroňový balíčkovací systém, nad kterým je postavený například balíčkovací systém APT. Pracuje s balíčky ve formátu `.deb`. Vytvoření balíčku je velmi jednoduché a přímočaré. Název balíčku dodržuje formát:

`<název balíčku>_<major verze>.<minor verze>-<revize balíčku>`

Balíček se generuje ze složkové struktury, která reprezentuje Linuxovou adresářovou hierarchii. Podle toho, jak budou umístěné soubory ve adresářové struktuře balíčku, tak do těchto lokací se následně soubory nainstalují. Složka s balíčkem také obsahuje speciální adresář `DEBIAN`, ve kterém jsou umístěné soubory obsahující metadata o balíčku nebo instalační, případně odstraňovací skripty apod.

Příklad struktury DPKG balíčku:

```
vdu_1.0-1
├── DEBIAN
│   ├── control
│   └── usr
│       ├── local
│       │   ├── bin
│       │   │   ├── vdu-app
│       │   │   └── vdu-app-fuse
│       ├── share
│       │   └── applications
│       │       └── vdu.desktop
```

Výše uvedený příklad by nainstaloval soubory `vdu-app` a `vdu-app-fuse` do složky `/usr/local/bin` na počítači uživatele instalující si daný balíček. Soubor `control` obsahuje informace o balíčku samotném jako název, verze, architektura nebo list závislostí. V seznamu závislostí budou vyjmenované veškeré potřebné knihovny a balíčky pro běh aplikace. Následně se balíček už jen zkompile pomocí příkazu `dpkg-deb --build <celý název balíčku>`.

## 5.3 RPM

RPM je založeno na souborech typu `.spec`. Jedná se o soubory specifikující jak má být daný balíček sestaven. SPEC soubor obsahuje hlavičku s údaji jako název, verze, seznam závislostí apod. Dále obsahuje sekce respektive SPEC direktivy jako `%prep`, `%build`, `%install` atd. Každá sekce může obsahovat sérii příkazů, které se vykonají v určitou dobu. Například příkazy v sekci `%install` se vykonají při instalaci balíčku. Příkazem `rpmdev-setuptree` se vytvoří výchozí stromová struktura pro sestavování balíčků.

Příklad struktury RPM balíčku:

```
rpmbuild
├── BUILD
├── RPMS
├── SOURCES
│   └── vdu-1.0.tar.gz
├── SPECS
│   └── vdu.spec
└── SRPMS
```

Pro definici nového balíčku nás zajímají složky `SPECS` a `SOURCES`. Ostatní složky jsou pro odkládání pomocných souborů nebo jako výstupní složky zkompilovaného balíčku. Do složky `source` nahrajme TAR archive obsahující zdrojové soubory. Pomocí příkazu `rpmdev-newspec <název balíčku>` lze vygenerovat základní strukturu SPEC souboru pro nový balíček.

Sestavení balíčku je možné udělat dvěma způsoby. Jeden dělá sestavení ve dvou fázích, což může ušetřit čas, sestavují-li se balíčky složené z mnoha komponent. Příkaz

`rpmbuild --bb <SPEC soubor>` sestaví balíček v jednom kroku a výsledek uloží do složky RPMS.

## 5.4 Testování

Výsledný software je třeba řádně otestovat a ověřit tím, neobsahuje-li velké množství chyb. Napsat bezchybný software je nemožné, takže testování, které neodhalí žádnou chybu by se dalo považovat za nedůkladné testování. Aplikace byla testována manuálně na dvou různých GNU/Linux distribucích s různými desktopovými prostředími. Aplikace by totiž měla být nezávislá na distribuci. Jednalo se o GNU/Linuxové distribuce Debian a Fedora. Použitá desktopová prostředí byla GNOM, KDE nebo Xfce. Při vývoji byla vytvořena sada unit testů ověřující správné fungování Mock Serveru a funkcí třídy API. Nakonec byly také vytvořeny automatické testy pro otestování aplikace jako celku.

## 5.5 Google Test

Google Test je testovací framework pro jazyk C++ a je založen na xUnit architektuře. Podle xUnit architektury se testování software dělí do čtyřech částí pojmenovaných Setup, Excercise, Verify, Teardown. Již podle názvů lze odhadnout, že se jedná o fázi příprav, samotného testování, vyhodnocování a uklid po testech. Google Test je primárně založen na preprocesoru jazyka C++.

Obsahuje dva typy maker, ASSERT a EXPECT. První jmenované při chybě nechá selhat celý test. Na druhou stranu makro EXPECT vypíše chybovou hlášku, ale test neukončí a nechá vyhodnocovat jeho další části. Jedná se o preferovanější variantu, protože daný test již nemůže vyjít kladně, ale projde vyhodnocením dalších podmínek, které mohou objasnit o jakou chybu se jedná a co ji případně způsobuje.

## 5.6 Testovací sada

Jednotkové testy pro třídu API ověřují jednak správnou implementaci funkcí obsažených v této třídě, ale také testovali není-li chybně vytvořená standardizovaná specifikace REST API. Na jejímž základě je vytvořený Mock Server. Testovaná byla každá funkce API třídy na každý návratový kód, který může daná API metoda vyprodukovat. Ověřovala se správnost návratových kódů, obsah HTTP hlaviček a případně tělo zprávy.

Komplexnější testy pracující s aplikací jako celkem a oproti jednotkovým testům ověřují komunikaci mezi jednotlivými komponentami. Tyto testy mohou odhalit chyby typu nekorektního přetypování, špatně nastaveného kódování apod. Jedná se o další fázi při testování software. Jeden z testů simuloval otevření speciální URL a kontroloval, zda data v databázi odpovídají očekávané hodnotě a také jestli je obsah souboru jaký má být. Druhý test ověřoval modifikaci souboru. Bohužel Postman Mock Server v tomto případě funguje jako Test Stub, takže se nedá automaticky ověřit, že zaslaná data jsou správná. Testovat se aspoň dají změny v databázi a jestli byl soubor opravdu modifikován. Poslední test změnil jméno souboru, jedná se však o velmi podobný případ jako je modifikace souboru, takže se ověřují pouze lokální změny.

Všechny testy korektně fungují pouze je-li aplikace připojená k Mock Server, protože testování podobných parametrů by byla na produkčním VDU těžko napodobitelná.

## Kapitola 6

# Závěr

Cílem bylo vytvořit aplikaci zpřístupňující obsah validovaného datového uložiště. Pro aplikaci měla být vytvořena sada automatických testů ověřující její fungování.

Po prostudování problematiky se zdařilo vytvořit návrh aplikace, od kterého se finální software výrazně neoddaluje. Cíle se tedy podařilo naplnit a dokonce vznikli balíčky pro nejpoužívanější GNU/Linuxové distribuce. To i přes fakt, že aplikaci nebylo možné otestovat na reálném VDU.

Nejdříve byla vytvořena analýza trhu a existujících řešení. Byla zkoumaná čtyři řešení od služeb velkých korporátních společností po open source nástroj. Dalším krokem bylo pochopení problematiky spojené s vytvořením FUSE souborového systému a vývoje software pro GNU/Linux. Následně byl vytvořen návrh, ve kterém bylo na výběr několik možností, ale po vyhodnocení požadavků byla vybrána nejvýhodnější varianta. Po konzultaci s vedoucím byla implementovaná i se sadou automatických testů. Postup a výsledky byly popsány v této práci. Vývoj aplikace<sup>1</sup> i tvorba této práce<sup>2</sup> byly od počátku veřejně dostupné v Github repozitářích a tak to také zůstane i po zveřejnění práce.

Jakožto uživatel Windows jsem se při práci naučil velké množství věcí o tom, jak funguje desktopové prostředí v GNU/Linux. Největší přínos pro mě, ale měla technologie FUSE, pomocí které by se dali vytvořit další zajímavé projekty.

Dalším krokem, který je potřeba udělat, je připojení aplikace na funkční VDU a ověření správné funkčnosti v reálném provozu. Práce by se dala nadále rozvíjet, například optimalizací FUSE souborového systému. Stále je zde dost místa pro zlepšení. Osobně neplánuji věnovat další čas rozvoji tohoto projektu, spíše uvažuji o dalším uplatnění nově získaných znalostí v dalších projektech.

---

<sup>1</sup><https://github.com/PlayerBerny12/VUT-IBT-Code>

<sup>2</sup><https://github.com/PlayerBerny12/VUT-IBT>

# Literatura

- [1] *About SQLite* [online]. sqlite.org [cit. 2021-05-07]. Dostupné z: <https://www.sqlite.org/about.html>.
- [2] BOCCHI, E., DRAGO, I. a MELLIA, M. Personal Cloud Storage Benchmarks and Comparison. *IEEE Transactions on Cloud Computing*. 2017, sv. 5, č. 4, s. 751–764. DOI: 10.1109/TCC.2015.2427191. Dostupné z: <https://ieeexplore.ieee.org/abstract/document/7096995>.
- [3] BURIHABWA, D., FELBER, P., MERCIER, H. a SCHIAVONI, V. SGX-FS: Hardening a File System in User-Space with Intel SGX. In: *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. 2018, s. 67–72. DOI: 10.1109/CloudCom2018.2018.00027. Dostupné z: <https://ieeexplore.ieee.org/document/8590996>.
- [4] CARLUCCI, G., DE CICCIO, L. a MASCOLO, S. HTTP over UDP: An Experimental Investigation of QUIC. In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. Association for Computing Machinery, 2015, s. 609–614. DOI: 10.1145/2695664.2695706. ISBN 9781450331968. Dostupné z: <https://doi.org/10.1145/2695664.2695706>.
- [5] COCAGNE, T. *DBus Overview* [online]. pythonhosted.org, červenec 2012 [cit. 2021-05-05]. Dostupné z: [https://pythonhosted.org/txdbus/dbus\\_overview.html](https://pythonhosted.org/txdbus/dbus_overview.html).
- [6] *Co je to svobodný software?* [online]. gnu.org, říjen 2019 [cit. 2021-05-03]. Dostupné z: <https://www.gnu.org/philosophy/free-sw.html>.
- [7] *Deploy Google Drive for desktop* [online]. Google, 2021 [cit. 2021-04-21]. Dostupné z: <https://support.google.com/a/answer/7491144?hl=en>.
- [8] *Desktop notifications* [online]. archlinux.org, květen 2021 [cit. 2021-05-07]. Dostupné z: [https://wiki.archlinux.org/title/Desktop\\_notifications#Notification\\_servers](https://wiki.archlinux.org/title/Desktop_notifications#Notification_servers).
- [9] *Dropbox* [online]. Dropbox, 2021 [cit. 2021-04-21]. Dostupné z: <https://www.dropbox.com/?landing=dbv2>.
- [10] FIELDING, R. a RESCHKE, J. *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content* [Requests for Comments]. RFC Editor, červen 2014 [cit. 2021-04-25]. DOI: 10.17487/RFC7231. Dostupné z: <https://www.rfc-editor.org/info/rfc7231>.
- [11] *Google Drive for developers - API Reference* [online]. Google, 2021 [cit. 2021-04-21]. Dostupné z: <https://developers.google.com/drive/api/v3/reference>.



- [12] *Googletest Primer* [online]. github.io [cit. 2021-05-07]. Dostupné z: <https://google.github.io/googletest/primer.html>.
- [13] *Google Workspace* [online]. Google, 2021 [cit. 2021-04-21]. Dostupné z: <https://workspace.google.com/>.
- [14] *History of C* [online]. cppreference.com, březn 2021 [cit. 2021-05-04]. Dostupné z: <https://en.cppreference.com/w/c/language/history>.
- [15] *History of C++* [online]. cppreference.com, březn 2021 [cit. 2021-05-04]. Dostupné z: <https://en.cppreference.com/w/cpp/language/history>.
- [16] JONES, M. *Anatomy of the Linux kernel* [online]. IBM, červen 2007 [cit. 2021-05-06]. Dostupné z: <https://developer.ibm.com/technologies/linux/articles/1-linux-kernel/>.
- [17] *Libcurl - the multiprotocol file transfer library* [online]. curl.se [cit. 2021-05-07]. Dostupné z: <https://curl.se/libcurl/>.
- [18] *LibfuseAPI Documentation* [online]. github.io [cit. 2021-05-07]. Dostupné z: <http://libfuse.github.io/doxygen/index.html>.
- [19] *Libsecret Library Reference Manual* [online]. gnome.org [cit. 2021-05-07]. Dostupné z: <https://developer.gnome.org/libsecret/>.
- [20] MESZAROS, G. *XUnit Test Patterns*. Addison-Wesley, 2007. ISBN 978-0-13-149505-0.
- [21] *OpenAPI Specification* [online]. SmartBear Software, 2021 [cit. 2021-05-05]. Dostupné z: <https://swagger.io/specification/>.
- [22] *PCloud* [online]. pCloud, 2021 [cit. 2021-04-21]. Dostupné z: <https://www.pcloud.com/eu>.
- [23] STALLMAN, R. *Linux a systém GNU* [online]. gnu.org, červenec 2020 [cit. 2021-05-03]. Dostupné z: <https://www.gnu.org/gnu/linux-and-gnu.cs.html>.
- [24] *Syncthing* [online]. Syncthing, 2021 [cit. 2021-04-21]. Dostupné z: <https://syncthing.net/>.
- [25] VANGOOR, B. K. R., TARASOV, V. a ZADOK, E. To FUSE or Not to FUSE: Performance of User-Space File Systems. In: *15th USENIX Conference on File and Storage Technologies (FAST 17)*. 2017, s. 59–72. ISBN 978-1-931971-36-2.
- [26] *Welcome to OpenSSL!* [online]. openssl.org [cit. 2021-05-07]. Dostupné z: <https://www.openssl.org/>.
- [27] *What is: Linux keyring, gnome-keyring, Secret Service, and D-Bus* [online]. RTFM: Linux, DevOps, and system administration, 12. července 2019 [cit. 2021-05-02]. Dostupné z: <https://rtfm.co.ua/en/what-is-linux-keyring-gnome-keyring-secret-service-and-d-bus/>.
- [28] YALTA, A. T. a LUCCHETTI, R. The GNU/Linux platform and freedom respecting software for economists. *Journal of Applied Econometrics*. 2008, č. 2, s. 279–286. DOI: <https://doi.org/10.1002/jae.990>. Dostupné z: <https://onlinelibrary.wiley.com/doi/abs/10.1002/jae.990>.

# Příloha A

## Seznam použitých knihoven

- libcurl
  - Dokumentace: <https://curl.se/libcurl/>
  - Repozitář: <https://github.com/curl/curl>
- libnotify
  - Dokumentace: <https://developer.gnome.org/libnotify/0.7/>
  - Repozitář: <https://gitlab.gnome.org/GNOME/libnotify>
- libfuse
  - Dokumentace: <http://libfuse.github.io/doxygen/index.html>
  - Repozitář: <https://github.com/libfuse/libfuse>
- libsqlite3
  - Dokumentace: <https://www.sqlite.org/capi3ref.html>
  - Repozitář: <https://www.sqlite.org/cgi/src>
- libssl
  - Dokumentace: <https://www.openssl.org/docs/man1.1.0/man3/>
  - Repozitář: <https://github.com/openssl/openssl>
- libsecret
  - Dokumentace: <https://developer.gnome.org/libsecret/0.20/>
  - Repozitář: <https://gitlab.gnome.org/GNOME/libsecret>
- libgtest
  - Dokumentace: <https://google.github.io/googletest/primer.html>
  - Repozitář: <https://github.com/google/googletest>

## Příloha B

# Obsah přiloženého média

```
/
├── bin
├── home
│   ├── jeancome
│   │   ├── texmf
│   │   │   └── tex
│   ├── jeancomeson - asdasda .3 jeancomedaughter - asdasd .2 usr
│   └── bin
```

**Složka 1** - XXX

**Složka 2** - XXX

**Složka 3** - XXX