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 — **JAN BERNARD**
AUTHOR

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

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2020/2021

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 uložiště. 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 uložištěm byla navržena a implementována aplikace pro systém GNU/Linux. Testování aplikace probíhalo na dvou vybraných Linuxových distribucích s mock serverem zastupující vzdálené uložiště.

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 represententing remote storage.

Klíčová slova

Validované datové uložiště, 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
10. 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

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 uložiště si nemusíme představit jen velká datová centra se stovkami serverů, může se jednat o relativně malé uložiště ve firmě nebo dokonce o osobní/domácí řešení. Pro využívání osobních/firemních uložišť 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 zákazníkova 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 Protokol neboli HTTP.

Cílem této práce je prozkoumat aktuální nabídku a možnosti na poli vzdálených uložišť 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 uložištěm. Komunikace skrze HTTP protokol byla definovaná externím zadavatelem. Zdrojové kódy jsou dostupné na veřejném Github repozitáři. [1]

[1] 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 větší 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 případu se jedná o variantu REST API. Poslední možností je vystavění 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 funkcionalita pro jednotlivce a firmy, cena, integrace s ostatními systémy a aplikacemi apod. Obecně

nelze jednoznačně určit nejlepší uložiště, ale je možné poukázat na rozdílné vlastnosti a vyzdvihnout 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. [11] Pro Linuxové distribuce lze využít například neoficiální open source aplikaci Google Drive OCamlFUSE[1] 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 Bussiness Standard nabízí 2 TB uložiště za 10,40 EUR měsíčně. Balíček obsahuje další služby jako firemní email a schůzky až o 150 účastnících na Google Meet. [18]

Každý uživatel má vlastní disk s omezenou kapacitou podle balíčku. Na disku lze vytvářet standartní 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. [16]

2.3 Dropbox

Dropbox má oficiální balíčky své aplikace pro Ubuntu a Fedoru, případně je možné si aplikaci zkompilovat 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. [4] 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 Worksapce. 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 uložišti. 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. [13]

2.4 pCloud

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

[1] 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. [27]

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 veškerá data budou synchronizována vůči tomuto uzlu. Syncthing je možné provozovat na většině dnešních systémů jako Windows, Linux, BSD a Mac OS. [30]

Aplikace poskytuje webové rozhraní, které je určené pouze pro nastavení parametrů jako discovery protokolu 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 Micorsoft. Důkazem tohoto jevu je například integrace aplikací obou těchto firem do webového rozhraní Dropboxu. Aplikace One Drive od Microsoftu nebyla blíže analyzovaná, protože se jedná o službu pouze pro platformu Windows.

Všechny služby mají webové rozhraní, ke kterému je aplikace synchronizující obsah uložiště 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 jejímž základě je vytvořená tato práce nebude synchronizovat celé uložiště, ale bude pracovat jen s jednotlivými soubory. Uživateli na vyžádání stáhne právě jeden soubor, se kterým bude chtít uživatel v danou dobu pracovat (zobrazit nebo upravit obsah). Synchronizace celého uložiště 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 uložiště, která byla stěžejním dokumentem při návrhu. Jednalo se o 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.

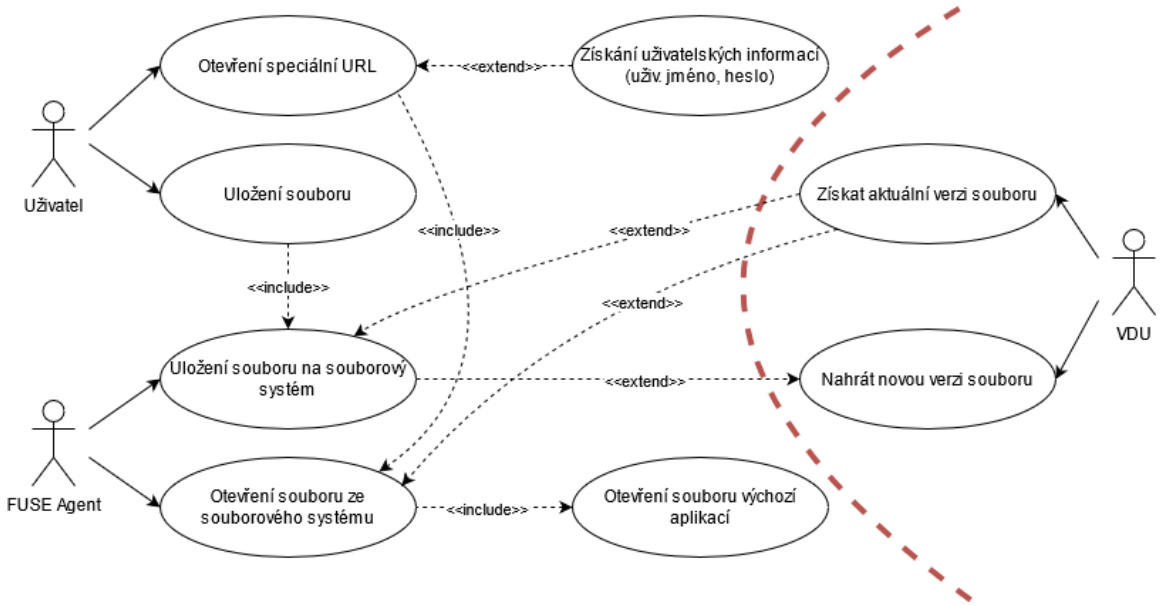
Pro vynucení práv jednotlivých souborů je standartní souborový systém nevyhovující, protože po stažení souboru do počítače uživatele ztrácí VDU kontrolu nad tímto souborem. Po konzultaci s vedoucím práce, doktorem Markem Rychlým, byla zvolena technologie FUSE neboli Filesystem in userspace. FUSE umožňuje 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 uživatelově 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 systém 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. Autentifikace, na které závisí veškerá komunikace s VDU, není zanesená do diagramu. Díky této operaci byla zvýšena přehlednost diagramu.

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 pojmenovaném `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 volaní 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.

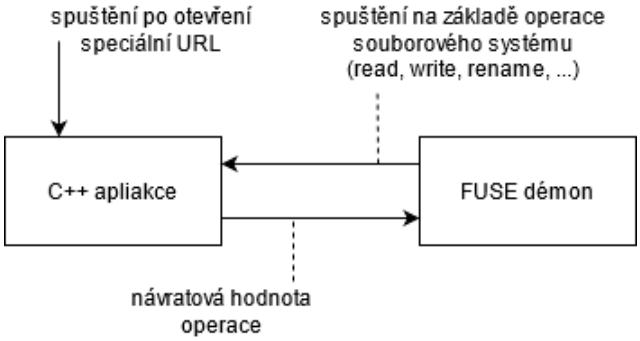
- /ping
  - GET /ping – pro ověření dostupnosti VDU
- /auth
  - GET /auth/key – obnova autentifikačního tokenu

  - 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 pracovala 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 vedly k obdobnému řešení. FUSE démon je napsaný v jazyce C, takže jedno z nabízejících se řešení by byla implementace celé aplikace jednom 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.



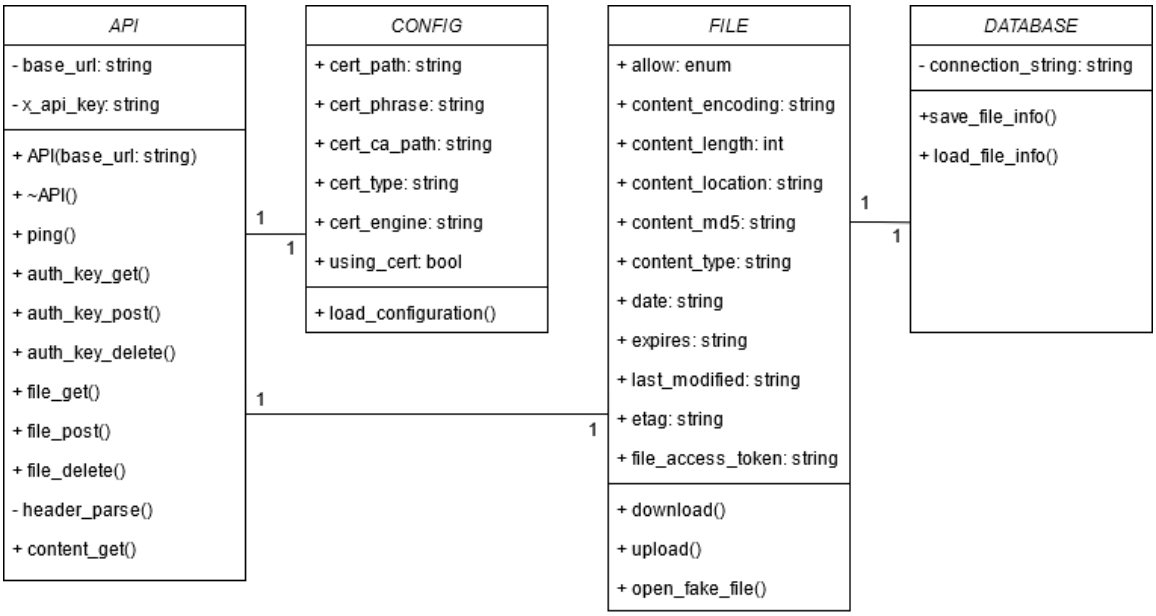
Obrázek 3.2: Vysokoúrovňový návrh architektury aplikace.

Pokud FUSE démon má vykonat akci, pro kterou nemá dostatečné informace, spustí jako další subproces 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. Na druhou stranu mohou být tyto knihovny i tak namapované, protože je můžou 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.

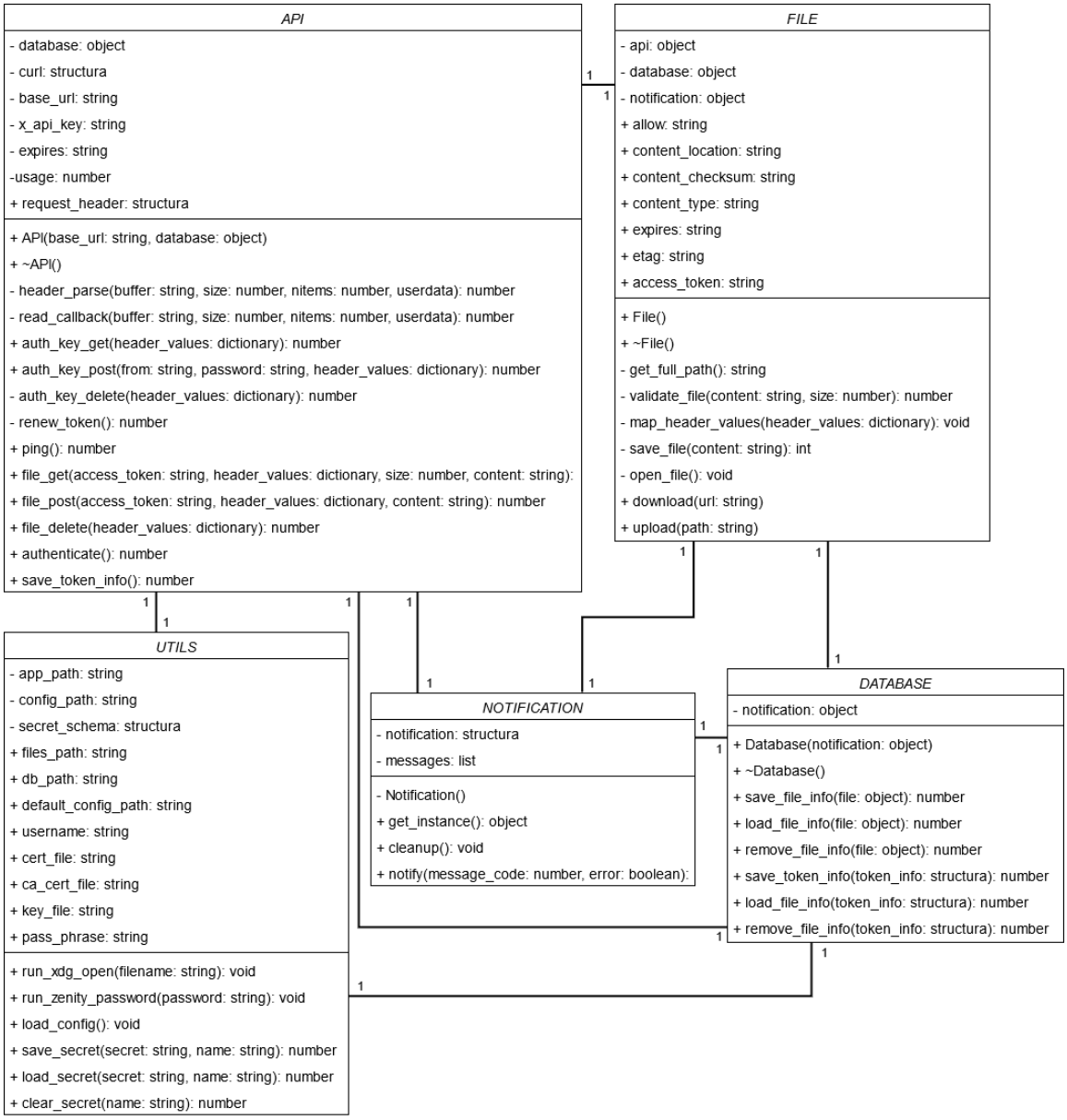
3.3.1 Diagram tříd

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 prozkoumáním lze vidět jednu novou třídu oproti návrhu. Jedná se o třídu zobrazující notifikace pro uživatele, která nemá vliv na hlavní funkcionalitu. Dále přibylo několik podpůrných funkcí a také přibily některé atributy, případně se přejmenovaly. Při implementaci nenastaly žádné zásadní potíže způsobené chybným nebo nepřesným návrhem, tudíž lze návrh označit za úspěšný.



Obrázek 3.4: Diagram tříd - téměř hotová implementace.

3.4 Technologie

V této sekci budou popsány jednotlivé technologie, které ovlivnily tuto práci. Pro samotný návrh stačí elementární znalosti daných technologií, ale pro implementaci a jejich správné a efektivní využití je třeba jejich bližší pochopení. Za klíčovou technologii by se dala označit technologie FUSE, bez které by řešení nemohlo splňovat zadaná kritéria.

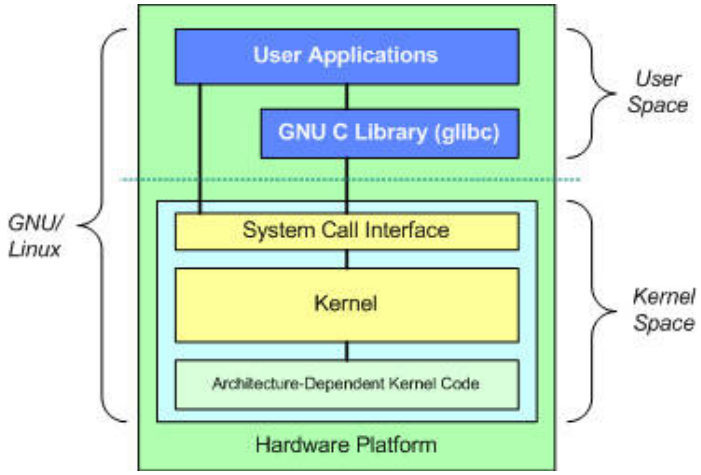
3.4.1 GNU/Linux

GNU je rekurzivní zkratka pro „GNU's Not Unix!“. Jedná se o projekt, který měl za cíl vystavět nový operační systém kompatibilní s Unixem. Všechen 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) [9].

V projektu GNU postupně vznikaly jednotlivé softwarové balíky, které měly ve výsledku poskládat nový operační systém. Pod projektem GNU vnikli 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. [36] [29]

Vznikla tedy myšlenka tyto dva projekty spojit a po vyřešení problémů s kompatibilitou jednotlivých částí projektů vnikl GNU/Linux. Balíčky projektu GNU umožnovaly 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. [29]



Obrázek 3.5: Architektura GNU/Linux. Převzato z [21].

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

Desktopová prostředí

V GNU/Linxovém prostředí existuje velké množství desktopových prostředí. Každé z nich je založeno na tzv. display serveru. Jedná se o program zodpovědný za koordinaci komunikace mezi grafickým prostředím a kernelem. Se svými klienty komunikuje pomocí proprietárního protokolu. V GNU/Linux systémech jsou dnes rozšířené dva display servery X.org a Wayland. Wayland je novější implementací display serveru a postupně v jednotlivých distribucích nahrazuje X.Org.

Každé desktopové rozhraní je něčím specifickým a vývoj software schopný běžet na všech desktopových rozhraních může vyžadovat větší úsilí. Mezi běžně používaná desktopová rozhraní patří například GNOME, KDE, Xfce, MATE a mnohé další.

3.4.2 C++/C

Jazyk C patří do skupiny dlouho používaných programovacích jazyků. Se svou 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 přišly ISO standardy označované jako C99, C11, C17. Jazyk C je řazen mezi vysokoúrovňové programovací jazyky, přestože je relativně jednoduchý a obsahuje malé množství konstrukcí v porovnání s ostatními jazyky.[19]

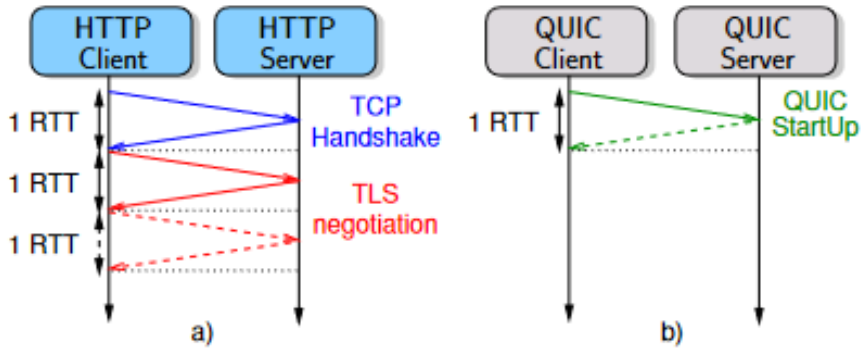
Vývoj C++ je oproti C dynamičtější, což můžeme vidět na jednotlivých ISO standardech C++98, C++03, C++11, C++14, C++17 a C++20. Jazyk C++ se v poslední dekádě velmi proměnil a období od standardu C++11 je označováno za dobu „moderního C++“. V posledním standardu přibyla například podpora konceptů nebo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

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 vydaná 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 spojovaly se technologie, které nebyly od počátku navržené 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. [7]



Obrázek 3.6: Porovnání HTTP verze 2 (a) a 3 (b). Převzato z [7].

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 datumy, přesněji formáty umožněné přenášet ve validní HTTP hlavičce. RFC7231 specifikuje tři formáty datumů, z nichž dva jsou označené jako zastaralé, ale jsou stále validní. Udržované jsou kvůli zpětné kompatibilitě s protokolem HTTP/1.1. [14]

- 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

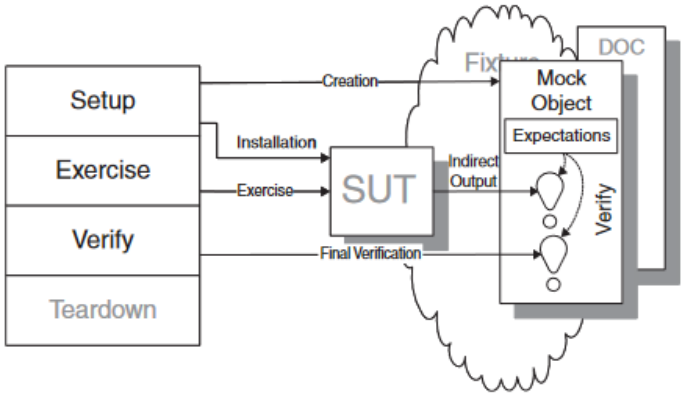
Jedná se o standard pro definování metod REST API, který má v čase psaní této práce poslední verzi 3.0.3. Zápis je možný ve dvou jazycích: JSON a YAML. Oba je možné strojově zpracovat, a přitom jsou pro člověka stále čitelné.

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 odpovědí a odpovědi samotné, případně příklady možných odpovědí. Toto je jen stručný výpis možností, které standard OpenAPI definuje. [26]

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 vrozů je využívaná pro systémy, které pracují s určitou externí komponentou, která není dostupná nebo by testování ovlivnila svými nežádoucími vedlejšími účinky.[25, 522–524] K využití Mock Serveru bylo přistoupeno z důvodu nedostupnosti VDU, při vzniku této práce. Mock Server nebyl využíván jen pro účely 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.

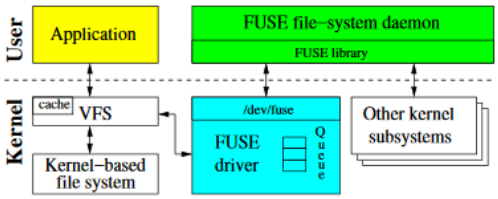
Mock Object pokrývá funkcionalitu Test Stub vzoru a na vývojáři záleží, jakým způsobem využije tyto možnosti. V kontextu této práce by se používaný Mock Server dal klasifikovat jako pouhý Test Stub, protože není využívaná verifikující část, kterou definuje Mock Object vzor. [25, 524–525] Verifikovaný je pouze výstup funkcí v testovaném systému.



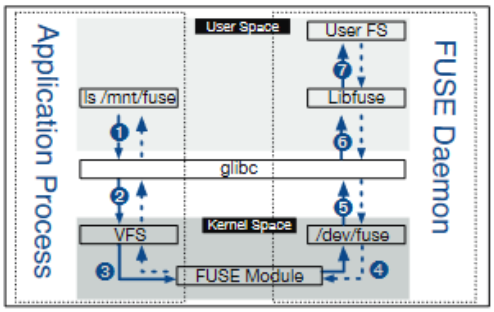
Obrázek 3.7: Test Double schéma. Převzato z [25, 544].

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 démony. FUSE driver obsahuje několik front pro obsloužení všech potřebných akcí. Nejprioritnější fronta je pro obsloužení přerušení, ostatní fronty jsou pro obsluhu synchronních, asynchronních a a tvz. forget požadavků. Priorita zpracování těchto front je daná poměrem 8 ku 16 pro synchronní a asynchronní ku forget požadavkům. [32]



(a) Architektura. Převzato z [32].



(b) Datový tok. Převzato z [6].

Obrázek 3.8: Vysokoúrovňové diagramy architketury 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[1].

Skupina	**Typ požadavku**
Speciální	INIT, DESTROY, INTERRUPT
Metadata	LOOKUP, FORGET, BATCHFORGET, CREATE, UNLINK, LINK, RENAME, RENAME2, 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, RELEASEDIR, READDIR, READDIRPLUS, FSYNCDIR
Zamikání	GETLK, SETLK, SETLKW
Misc	BMAP, FALLOCATE, MKNOD, IOCTL, POLL, NOTIFY_REPLY

Tabulka 3.1: Tabulka typů požadavku pro zpracování FUSE démonem. Převzato z [32].

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. Existuje také reimplementace D-Bus protokolu pro další jazyka jako je Java, C# nebo Ruby. Objekty jsou identifikovány kombinací tzv. „bus name“ a „object path“. [8]

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 používaný 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. [8]

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.

[1] http://libfuse.github.io/doxygen/structfuse__operations.html

Pokud není heslo k dispozici musí jej aplikace od uživatele vyžádat a do keyringu případně uložit. Pokud aplikace využívající keyring nemusí s uživatelem tolik interagovat a práce bývá pro uživatele příjemnější, přitom heslo je velmi dobře chráněno. Utočník napadající systém z vnějšku není schopen heslo zjistit bez jeho dešifrování. [34]

GNU/Linux obsahuje keyring aplikaci pojmenovanou linux-keyring. Některá desktopová 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. [34]

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 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. [22]
- **libnotify** – Jedná se o implementaci Desktop Notification Specification[1], 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. [12]
- **libfuse** – Nabízí funkce pro vytvoření FUSE démona, komunikující s FUSE modulem kernelu. Implementací jednotilivých operací je možné vytvořit vlastní souborový systém v uživatelském prostoru. [23]
- **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. [33]
- **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. [24]
- **libgtest** – Google Test je testovací framework pro psaní jednotkových testů pro programy psané v jazyce C++. Je založen na architektuře xUnit. Zpřístupňuje makra pro ověření výroku nebo podmínky je-li pravdivá. [17]

[1] https://developer.gnome.org/notification-spec/

Další balíčky využité 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, umožňující zobrazení obsahu tabulek, vytvoření i úpravu dat nebo struktury databáze.
- **Behave** – Software pro psaní testů v přirozeném jazyce. Je využíván pro automatické testování software.
- **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 měly nacházet. Také existuje specifikace nazývaná XDG Base Directory specification[2], 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 vidí 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 speruživatele `root`.

Seznam souborů a cest využívaných aplikací:

- **/usr/local/bin** – Místo pro programy neboli spustitelné soubory, přístupný pro normálního uživatele. [15] V této složce jsou uloženy oba binární soubory pojmenované `vdu-app` a `vdu-app-fuse`. Druhá jmenovaná 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 sídílené slokžy nebo síťového disku. Do této složky bude připojen FUSE souborový systém, který bude využíván pro ukládání stažených souborů z VDU.

[2] https://specifications.freedesktop.org/basedir-spec/basedir-spec-latest.html

- **$HOME/.local/share/vdu/** – Na této cestě budou umístěny aplikační data jako databáze a složka pro ukládání obsahu z FUSE souborového systému. Každému uživateli, který bude na daném operačním systému využívat `vdu-app` bude vytvořena tato složka.
  - **/vdu.db** – SQLite 3 databáze pro ukládání podpůrných informací o souborech stažených z VDU a také metadata potřebná pro běh programu.
  - **/fuse** – Složka na hostitelském souborovém systému, kterou využívá FUSE démon pro čtení a ukládání dat, která jsou následně zobrazena ve složce `/mnt/vdu`. Pokud by uživatel chtěl obejít FUSE souborový systém, mohl by k datům přistoupit napřímo skrze tuto složku. Poté by mohl číst soubory, které již expirovali, nebo zapisovat do souborů, ke kterým má pouze práva na čtení. Vlastník této složky je `root`, který jako jediný může z této složky číst nebo do ní zapisovat. Jak již bylo výše popsáno FUSE démon běží s právy superuživatel `root`, takže může nad tomuto složkou operovat.
- **$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. [5]
- **/etc/systemd/user** – Konfigurační složky systému `systemd`, využívané pro definici jednotlivých systemd služeb (systemd service). V tomto případě pro služby běžící pro každého jednotlivého přihlášeného uživatele. Systémová služba by byla umístěná namísto složky `user` ve složce `system`. [31]

4.1.1 Konfigurační soubor

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

```
Username = username
ClientCertFile = client.pem
CACertFile = ca.pem
KeyFile = client.key
#Passphrase = SuperSecretPassPhraseXX12@
```

Výpis 4.1: Příklad obsahu konfiguračního souboru.

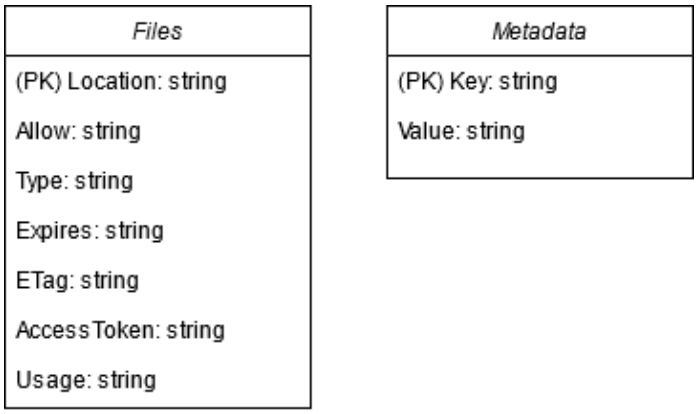
Konfigurační soubor umožňuje nastavit uživatelské jméno, jména certifikátů nebo klíčů a v neposlední řadě také volitelnou frázi pro zabezpečení certifikátu. Všechny hodnoty jsou volitelné neboli pokud nebudou v konfiguračním souboru nastavené, aplikace zvolí jejich výchozí hodnotu. Případně vyzve uživatele 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 zabezpečený. Nezadává se absolutní cesta k certifikátům, ale pouze jejich jméno, protože certifikáty jsou následně načítány z systémového uložiště 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).



Obrázek 4.1: Schéma SQLite databáze.

Pokud SQL dotaz obsahuje parametry, využívá se postup před vytvoření dotazu bez parametrů a jejich následné navázání na již zkompilovaný 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`. [2] Veškerá komunikace s databází je zapouzdřená 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 integraci aplikací do desktopového prostředí. Integrace spočívá ve využívání instalovaných aplikací desktopovým prostředím 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 využívané. Xdg-open zajišťuje otevření určité URL nebo typu souboru v přiřazené aplikaci. Přiřazovat lze pouze k aplikacím popsaných tzv. Desktop Entries, které jsou specifikové v Desktop Entry Specification[3]. Specifikace také definuje vestavěné 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. [35]

```
[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 4.2: 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>`. [35]

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 mezičase 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 popsaná v podsekci 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 pozdnějších 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 přidané direktivy preprocesoru

[3] https://specifications.freedesktop.org/desktop-entry-spec/latest/

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. Tuto funkcionalitu nebylo možné otestovat, 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 podsekci 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 pomocí aplikace Zenity, aby zadal své heslo. Aplikace poté zadané heslo uloží do keyring aplikace a je-li potřeba znovu ověřit uživatelovu 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 aplikace 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 funkcionalita, že každý autentifikační token může být využit maximálně 5krát. Pokud má být token použit po šesté, tak je místo toho zavolaná 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 minimalistiky a byla inspirovaná příkladem z dokumentace pojmenovaném `passthrought.c`, který zpřístupňuje kořenový adresář v přípojném 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 kódu jako zbytek projektu a následně byl doplněn o komentáře. Nakonec byla přidaná další funkcionalita., případně byla modifikovaná ta stávající.

```
static const struct fuse_operations fuse_opers = {
    .init = fuse_init,
    .getattr = fuse_getattr,
    .access = fuse_access,
    .readdir = fuse_readdir,
    ...
};
```

Výpis 4.3: Část struktury předávané FUSE démonovi.

Při spouštění démona je jako jeden z parametrů předáván ukazatel na strukturu typu `struct fuse_operations`. V této struktuře jsou ukazatele na jednotlivé funkce pro daný typ požadavku. Na ukázce struktury 4.3 je požadavek typu GETATTR obsluhován funkcí `fuse_getattr`, která je ve zdrojovém souboru definované před definicí této struktury.

```
char return_val[1];

// Create command
char command_vdu[512] = "vdu-app --real-file save '";
strcat(command_vdu, path);
strcat(command_vdu, "'");

// Execute command
FILE *f = popen(command_vdu, "r");

// Read value from stream
fgets(return_val, 1, f);
printf("%s\n", return_val);

// Close stram crated by popen
pclose(f);
```

Výpis 4.4: Příklad volání vdu-app z FUSE démona.

Aplikace `vdu-app` je volaná z FUSE démona pomocí funkce `popen`, jak je možné vidět na příkladu 4.4. Tato funkce vytvoří tzv. pipe a novou větev programu, ve které se spustí volaná funkce. Z pipe lze buďto pouze číst nebo do ní zapisovat, protože je jednosměrná. Skrze pipe dochází k předání návratového kódu z `vdu-app` zpět do `vdu-app-fuse`. Funkce `pclose` čeká na dokončení volaného příkazu a poté uvolní daný stream. V případě úspěšného provedení příkazu vrátí funkce `pclose` hodnotu rozdílnou -1.

FUSE démon bude registrován jako uživatelská služba v `systemd`, 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 souborovém systému mohou nastat tři situace, které jsou důležité. Jedná se o čtení, uložení a přejmenování souboru. Při čtení se kontroluje, jestli daný soubor není zastaralý, případně jestli soubor není write-only (VDU může změnit po

jednom přečtení práva na write-only). Pokud se zapisuje, může se jednat o první zápis nebo modifikaci stávajícího souboru. Modifikovaný soubor je nahráván 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 Notifikace uživatele

Uživatele je třeba notifikovat o běhu programu, aby nebyl zmatený v případě nastane-li chyba. Komunikace aplikace s notifikačním serverm probíhá přes D-Bus. Většina desktopových prostředí obsahuje svoji implementaci notifikačního serveru, která se spustí při zapínání operačního systému. Pokud daný systém notifikační server neobsahuje je nutné nějaký nainstalovat a spustit, aby bylo možné uživateli zobrazovat informace.

Obrázek 4.2: Notifikace v desktopovém prostředí GNOME.

Aplikace sdružuje vše potřebné pro notifikování uživatele ve třídě `Notification`. Třída je implementovaná návrhovým vzorem jedináček (singleton). Tento vzor v projektu také využívá třída `Utils`. K tomuto rozhodnutí bylo přistoupeno, protože třída potřebuje alokovat některé zdroje, které nemusí být alokovány při každém volaní pro zobrazení notifikace. Přitom je třeba třídu využívat v různých částech kódu. Na výběr bylo tedy mezi statickou třídou a třídou implementující návrhový vzor singleton. Mezi těmito variantami nebyl dosatečný rozdíl, a tudíž byla zvolana implementace podle návrhového vzoru.

4.6 Shrnutí

Vývoj provázelo několik nepříjemností a zdržení, například nastavení suid bitu nebo problémy spojené s Mock Serverem. Veškeré problémy se však podařilo kompletně vyřešit, nebo minimálně nalézt řešení, které bylo dostatečné a splňovalo potřebná kritéria. Aplikace je plně funkční s Mock Serverm a dalším krokem by bylo připojit ji k reálému VDU. Byl by zde i určitý prostor pro vylepšení a optimalizaci, ale nejedná se o nic kritického, co by znemožňovalo nebo znepříjemňovalo práci s aplikací.

Kapitola 5

Nasazení a testování

Pro úspěšné používání aplikace je třeba ji nejprve nainstalovat do systému uživatele. Někteří schopnější uživatelé by ji byly schopni zkompilovat 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.

Vybrány byly dva balíčkovací systémy, které pokrývají dvě velmi používané větve distribucí GNU/Linux. Balíčkovací systém DPKG používaný v systému Debian a jeho derivátech. Druhým vybraným je systém RPM pokrývající různé operační systémy napojené na ekosystém společnosti Red Hat.

5.1 Sestavení ze zdrojových souborů

Pro sestavení ze zdrojových souborů stačí stáhnout zdrojové soubory z git repoziráře 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.

Po sestavení je nutné nastavit všechny náležitosti jako přiřazení `vdu-app` k otevírání speciální URL nebo spustit FUSE démon jako systemd službu. Upřednostňovanou variantou je využití vytvořených balíčků.

Makefile obsahuje targety pro sestavení, nebo spuštění hlavní `vdu-app`, FUSE démona a automatizovaných testů psaných v GoogleTest, nebo Behave 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 úrovňový 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 scripty apod. [10]

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 zkompiluje pomocí příkazu `dpkg-deb --build <název balíčku>`.

Takto vytvořený balíček je možné nainstalovat na určený GNU/Linux systém pomocí příkazu `dpkg -install <.deb balíček>`. [10]

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. [28]

Příklad struktury pro sestavování RPM balíčků:

```
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 ba-

líčku. 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. Nainstalovat balíček je potom možné pomocí příkazu `rpm -i <.rpm balíček>`. Stejně jaké u DPKG balíčku je možné aplikaci začít ihned používat, protože se při instalaci provede veškerá konfigurace automaticky. [28]

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 a 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.

5.4.1 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`, `Excersise`, `Verify` a `Teardown`. Již podle názvů lze odhadnout, že se jedná o fázi příprav, samotného testování, vyhodnocování a úklid 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. [17]

5.4.2 Behave

Behave je software pro spouštění a psaní testů v přirozeném jazyce. Testy je možné popsat tzv. features obsahující jednotlivé scénáře. Takto popsané testy je potom nutné implementovat v jazyce Python. Scénář se může skládá až ze tří částí `Given`, `When` a `Then`. [3]

```
Scenario: Download file form VDU
    Given Open URL "vdu://123456789"
    Then file is downloaded to FUSE file system
    And metadata are stored in database
```

Výpis 5.1: Scénář pro testování stažení souboru z VDU.

Výše pospaný scénář 5.1 využívá pouze dvě ze tří možných forem popisu scénáře. Na začátku scénáře je pospána akce otevření specifické URL a v druhé části jsou testovány

dvě podmínky pro úspěšné vyhodnocení test. Implementace by nepřímo spustila `vdu-app` a ověřila by, zda je soubor korektně uložený na souborovém systému. V dalším kroku by ověřila data v SQLite databázi. Takovýto test ověřuje funkčnost aplikace jako celku a testuje jednotlivé části jako sttahování souboru, autentifikaci apod.

5.4.3 Testovací sada

Jednotkové testy pro třídu API ověřují jednak správnou implementaci funkcí obsažených v této třídě, ale také testovaly 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. Byly vytvořený tři scénáře simulující běžné operace, které by dělal běžně uživatel. 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 vytvořená sada automatických testů ověřující její fungování.

Po prostudování problematiky se zdařilo vytvořit návrh aplikace, od kterého se finální software výrazně neoddaluje. Při samotném vývoji byla vytvořena sada jednotkových testů a na závěr byly také implementovány automatické testy. Systém byl testován jako celek a ověřovalo se jestli, se chová podle vytvořeného návrhu. Cíle se tedy podařilo naplnit, a dokonce vznikly 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[1] i tvorba této práce[2] byly od počátku veřejně dostupné v Github repozitářích a tak to také zůstane i po zveřejnění práce. Zdrojové soubory aplikací jsou zveřejněné pod MIT licencí.

Jakožto uživatel Windows jsem se při práci naučil velké množství věcí o tom, jak funguje desktopové prostředí v GNU/Linux. Největší přínos pro mě mělo studium a práce s technologií 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.

[1] https://github.com/PlayerBerny12/VUT-IBT-Code
[2] 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] *An Introduction To The SQLite C/C++ Interface* [online]. sqlite.org [cit. 2021-05-07]. Dostupné z: https://www.sqlite.org/cintro.html.

[3] *Behavior Driven Development* [online]. github.io [cit. 2021-05-09]. Dostupné z: https://behave.readthedocs.io/en/stable/philosophy.html#bdd-practices.

[4] 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.

[5] Brown, P., Blandford, J. et al. *Desktop Entry Specification* [online]. freedesktop.org, duben 2020 [cit. 2021-05-08]. Dostupné z: https://specifications.freedesktop.org/desktop-entry-spec/latest/.

[6] 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.

[7] Carlucci, G., De Cicco, 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.

[8] Cocagne, T. *DBus Overview* [online]. pythonhosted.org, červenec 2012 [cit. 2021-05-05]. Dostupné z: https://pythonhosted.org/txdbus/dbus_overview.html.

[9] *Co je to svobodný software?* [online]. gnu.org, říjen 2019 [cit. 2021-05-03]. Dostupné z: https://www.gnu.org/philosophy/free-sw.html.

[10] *Debian New Maintainers' Guide* [online]. debian.org, prosinec 2020 [cit. 2021-05-08]. Dostupné z: https://www.debian.org/doc/manuals/maint-guide/.

[11] *Deploy Google Drive for desktop* [online]. Google, 2021 [cit. 2021-04-21]. Dostupné z: https://support.google.com/a/answer/7491144?hl=en.

[12] *Desktop notifications* [online]. archlinux.org, květen 2021 [cit. 2021-05-07]. Dostupné z: https://wiki.archlinux.org/title/Desktop_notifications#Notification_servers.

[13] *Dropbox* [online]. Dropbox, 2021 [cit. 2021-04-21]. Dostupné z: https://www.dropbox.com/?landing=dbv2.

[14] 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.

[15] *Filesystem Hierarchy Standard* [online]. linuxfoundation.org, březen 2015 [cit. 2021-05-07]. Dostupné z: https://refspecs.linuxfoundation.org/FHS_3.0/fhs-3.0.html.

[16] *Google Drive for developers - API Reference* [online]. Google, 2021 [cit. 2021-04-21]. Dostupné z: https://developers.google.com/drive/api/v3/reference.

[17] *Googletest Primer* [online]. github.io [cit. 2021-05-07]. Dostupné z: https://google.github.io/googletest/primer.html.

[18] *Google Workspace* [online]. Google, 2021 [cit. 2021-04-21]. Dostupné z: https://workspace.google.com/.

[19] *History of C* [online]. cppreference.com, březen 2021 [cit. 2021-05-04]. Dostupné z: https://en.cppreference.com/w/c/language/history.

[20] *History of C++* [online]. cppreference.com, březen 2021 [cit. 2021-05-04]. Dostupné z: https://en.cppreference.com/w/cpp/language/history.

[21] Jones, M. *Anatomy of the Linux kernel* [online]. IBM, červen 2007 [cit. 2021-05-06]. Dostupné z: https://developer.ibm.com/technologies/linux/articles/l-linux-kernel/.

[22] *Libcurl - the multiprotocol file transfer library* [online]. curl.se [cit. 2021-05-07]. Dostupné z: https://curl.se/libcurl/.

[23] *LibfuseAPIDocumentation* [online]. github.io [cit. 2021-05-07]. Dostupné z: http://libfuse.github.io/doxygen/index.html.

[24] *Libsecret Library Reference Manual* [online]. gnome.org [cit. 2021-05-07]. Dostupné z: https://developer.gnome.org/libsecret/.

[25] Meszaros, G. *XUnit Test Patterns*. Addison-Wesley, 2007. ISBN 978-0-13-149505-0.

[26] *OpenAPI Specification* [online]. SmartBear Software, 2021 [cit. 2021-05-05]. Dostupné z: https://swagger.io/specification/.

[27] *PCloud* [online]. pCloud, 2021 [cit. 2021-04-21]. Dostupné z: https://www.pcloud.com/eu.

[28] *RPM Packaging Guide* [online]. github.io, září 2020 [cit. 2021-05-08]. Dostupné z: https://rpm-packaging-guide.github.io/.

[29] 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.

[30] *Syncthing* [online]. Syncthing, 2021 [cit. 2021-04-21]. Dostupné z: https://syncthing.net/.

[31] *Systemd/User* [online]. archlinux.io, květen 2021 [cit. 2021-05-07]. Dostupné z: https://wiki.archlinux.org/title/Systemd/User.

[32] 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. Dostupné z: https://www.usenix.org/conference/fast17/technical-sessions/presentation/vangoor.

[33] *Welcome to OpenSSL!* [online]. openssl.org [cit. 2021-05-07]. Dostupné z: https://www.openssl.org/.

[34] *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/.

[35] *Xdg-utils* [online]. freedesktop.org, červen 2019 [cit. 2021-04-20]. Dostupné z: https://freedesktop.org/wiki/Software/xdg-utils/.

[36] 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

src – zdrojové soubory obou aplikací a testů

text – zdrojové soubory tohoto dokumentu

docs – podpůrné dokumenty jako definice API, zdrojové soubory použitých diagramů

packages – zabalené zdrojové složky pro sestavení balíčku

build – sestavené/spustitelné soubory, instalační balíčky

README.md – návod k instalaci a korektnímu používání

xberna18.pdf – tento dokument