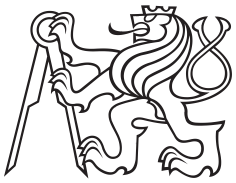
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická

Distribuované úložiště dat - Java knihovna

Bogdan Grigorian

Vedoucí: Ing. Macejko Peter
Květen 2019

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Grigorian** Jméno: **Bogdan** Osobní číslo: **452746**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Distribuované úložiště dat - Java knihovna

Název bakalářské práce anglicky:

Distributed data storage - Java library

Pokyny pro vypracování:

Navrhněte a implementujte Java knihovnu pro přístup k funkcím daného distribuovaného datového úložiště (viz literatura).
Knihovna musí poskytovat alespoň tyto funkce:
1) Manipulace s daty (čtení, zápis, modifikace).
2) Manipulace s metadaty (přejmenování, přesouvání, počet kopií, nastavení sdílení).
3) Zabezpečit poskytnutý klíč k dešifrování dat (nesmí opustit knihovnu).
V případě potřeby upravte funkci zbývajících částí systému sdílení dat (viz literatura).
Své řešení otestujte na pilotní implementaci datového úložiště pomocí jednoduché implementace klienta (například pomocí FUSE wraperu, nebo konzolové aplikace), která prověří alespoň základní operace s daty (viz bod 1).

Seznam doporučené literatury:

[1] Dyntar, T.: Distribuované úložiště dat - správa metadat. [Diplomová práce]. Praha: ČVUT FEL, Katedra počítačů, 2014. 71 s.
[2] Janura, J.: Distribuované úložiště dat - klientská část. [Diplomová práce]. Praha: ČVUT FEL, Katedra počítačů, 2014. 66 s.
[3] Kudrnáč, M.: Distribuované úložiště dat - správa dat. [Diplomová práce]. Praha: ČVUT FEL, Katedra počítačů, 2014. 63 s.
[4] Tuček Petr, Distribuované úložiště dat, diplomová práce FEL ČVUT, Praha, červen 2012
[5] Volf Martin, Distribuované úložiště dat, diplomová práce FEL ČVUT, Praha, leden 2014
[6] George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair, Distributed Systems: Concepts and Design, Addison-Wesley, 2011, 978-0132143011

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Peter Macejko, katedra telekomunikační techniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **26.03.2019** Termín odevzdání bakalářské práce: **24.05.2019**

Platnost zadání bakalářské práce: **19.02.2021**

Ing. Peter Macejko podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Poděkování

Chtěl bych poděkovat Ing. Petru Macejkovi, vedoucímu bakalářské práce, za časté konzultace, cenné rady a ochotu pomáhat. Dále bych chtěl poděkovat své rodině za podporu během studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, ...

.....

Bogdan Grigorian

Abstrakt

Tato práce se zabývá návrhem a implementací knihovny v Javě, poskytující prostředky k uložení a načtení dat z distribuovaného datového serveru.

Klíčová slova: distribuované úložiště, DHT, datové centrum, klientská knihovna, Java knihovna, JNA

Vedoucí: Ing. Macejko Peter
Katedra telekomunikační techniky,
Technická 1902/2,
160 00 Praha 6

Abstract

This work deals with design and implementation of Java library, providing tools for storing and retrieving data from distributed data server.

Keywords: distributed storage, DHT, data center, client library, Java library, JNA

Title translation: Distributed data storage - Java library

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

1.1 Specifikace problému

Tato bakalářská práce navazuje na tři předchozí diplomové práce, jejichž cílem byl návrh a implementace pilotní verze distribuovaného systému pro ukládání dat. Jedna práce [1] se týkala klientské části aplikací a byla napsána v jazyku C s použitím knihovny FUSE. Kvůli tomu tento klient není vhodný pro většinu platforem, jako například Windows. Navíc jazyk C není objektově orientovaný a proto není čitelný, není znovupoužitelný a obtížněji se v něm hledají chyby.

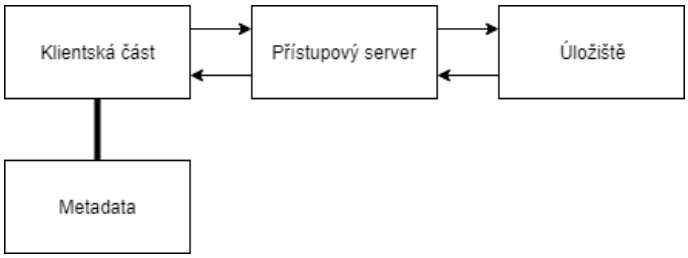
1.2 Cíle práce

Cílem této bakalářské práce je implementace knihovny v jazyku Java, která by splňovala všechny funkce plnohodnotného klienta pro již existující distribuovaný systém. Knihovna musí poskytovat alespoň následující funkce: manipulace s daty (čtení, zápis, modifikace), manipulace s metadaty (přejmenování, přesouvání, počet kopií, nastavení sdílení) a zabezpečení klíče k šifrování a dešifrování dat, který by neměl opustit knihovnu. Zároveň knihovna musí být vyvinuta s ohledem na požadavky projektu [2] pro vývoj uživatelského rozhraní klientské části pro již zmíněný distribuovaný systém.

Kapitola 2

Analýza

Před začátkem vývoje byla potřeba provést analýzu celého distribuovaného systému. Jeho zjednodušené schéma je zobrazeno na obrázku 2.1:



Obrázek 2.1: Schéma distribuovaného úložiště

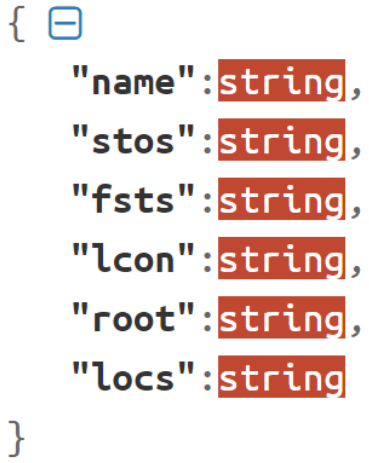
Na obrázku 2.1 lze vidět, že celý systém je rozdělen na čtyři subsystémy. Dále pro každý subsystém bude uvedena jeho analýza.

2.1 Metadata

Subsystém Metadata byl implementován v rámci diplomové práce "Distribuované úložiště dat - správa metadat" [3], autorem je Tomáš Dyntar. Cílem projektu byl návrh a implementace zabezpečeného systému pro správu metadat uživatele.

V textu dále budu pro soubor s metadaty používat pojem chunk (česky kus). Pod pojmem metadata se myslí sada souborů, které obsahují informace o příslušném uživateli a popisují adresářovou strukturu souborů, uložených na datovém serveru. Počet těchto souborů se může lišit, ale minimálně existují dva soubory pro jednoho uživatele. První chunk je chunk s uživatelskými informacemi. Tento soubor udržuje informace o uživateli: jeho jméno, kapacitu prostoru na serveru, poslední datum připojení a hash aktuálního uživatelského kořenového chunku, kterému se říká uživatelský chunk metadat. Hash uživatelského chunku metadat ukazuje na jméno souboru. Tento soubor obsahuje kořenový strom reprezentující adresářovou strukturu patřící uživatelům souborů na datovém serveru.

Všechna metadata jsou uložena ve formátu JSON a na obrázcích 2.2 a 2.3 jsou zobrazené jejich příklady.



Obrázek 2.2: Chunk s uživatelskými informacemi

Obrázek 2.3: Uživatelský chunk metadat

Celá myšlenka správy metadat spočívá v tom, že v paměti se nachází strom, který reprezentuje adresáře a soubory příslušného uživatele. Každý element tohoto stromu se nazývá node, který udržuje mnoho informací, jako například jméno nodu, jeho velikost, hash atd. Při volání funkcí z knihovny MetaManagement se prochází celým stromem, dokud se nenajde potřebný node. Po provedení nějaké změny se uloží celý strom do nového uživatelského chunku metadat a pak se vytvoří chunk s uživatelskými informacemi, obsahující hash již vytvořeného uživatelského chunku metadat.

O struktuře metadat se podrobněji pojednává v diplomové práci “Distribuované úložiště dat - správa metadat” [3].

2.2 Server

Subsystém server byl navržen a implementován v rámci diplomové práce Martina Kudrnáče "Distribuované úložiště dat - správa dat" [4]. Pro účely této bakalářské práce nebylo třeba důkladně analyzovat danou část systému, stačilo vědět, že je to distribuovaný server, který se stará o ukládání a poskytnutí uživatelských dat. Několik spuštěných serverů je při správné konfiguraci zapojeno do jednoho distribuovaného systému. Každý uzel tohoto systému ukládá jednotlivé chunky do vlastní databáze SQLite3 [5], která má následující strukturu 2.4:

NAME	KEY	TIME	REP	REPL	UID	REPCOMP	FLAG

Obrázek 2.4: Struktura databáze

Dále je uveden příklad záznamu o chunku v databázi, všechny hodnoty jsou rozdělené pomocí symbolu '|':

```
fc199a25e07a5|39e567bb2b708|2019-05-14 14:55:05|0|0||0|1
```

Podrobněji o DHT serveru v diplomové práci "Distribuované úložiště dat - správa dat" [4].

2.3 Přístupový server

Výstupem diplomové práce "Distribuované úložiště dat - klientská část " [1], jejímž autorem je Jan Janura, jsou dvě aplikace. První z nich je přístupový server.

Přístupový server je "brána" mezi klientem a datovým serverem, jejímž hlavním úkolem je zajištění bezpečné komunikace a zpracování požadavků klientů. Pro každého nově připojeného klienta se spustí vlákno, které bude plnit frontu požadavků, až klient datový server o něco požádá. Až server bude mít odpověď, pošle ji na přístupový server, který po jejím zpracování přepošle tuto odpověď klientovi.

Kód přístupového serveru nebylo třeba analyzovat, pouze byl během vývoje nalezen bug, který způsoboval nedefinované chování. Nakonec byl tento bug opraven a nahrán do příslušného repozitáře (podrobněji o bugu viz. 4.4).

2.4 Klientská aplikace

Knihovna implementovaná v této bakalářské práci bude plnit většinou stejné funkce jako klientská aplikace diplomové práce "Distribuované úložiště dat - klientská část" autora Jana Janury [1], a proto je analýza této části systému nejdůležitější.

Před začátkem implementace bylo potřeba důkladně zanalyzovat kód stávající klientské aplikace, aby bylo jasné, jak probíhá připojení, handshake, odesílání a příjem souborů.

Kód je rozdělen do tří modulů:

1. Modul ClientConnection, který se stará o navázání spojení a komunikace s přístupovým serverem.
2. Modul Configuration pro načítání dat z konfiguračního souboru.
3. Modul Klient obsahuje funkci main, která se zavolá při spuštění aplikace a nejdřív načte konfiguraci pomocí metod modulu Configuration. Pak, voláním funkce init, provede inicializaci celého systému a na konec spustí vlákno thread_function.

Frontu požadavků po inicializaci budou naplňovat funkce knihovny FUSE [6], které jsou detailně popsané v diplomové práci klientské části [1].

2.4.1 Funkce init

Funkce init provádí inicializaci celého systému v následujícím pořadí:

1. Volání funkce clientStart z modulu ClientConnection, tím se připojí k přístupovému serveru, nastaví TLS session a spustí vlákno pro příjem zpráv a souborů recvMessage.
2. Odstranění dočasných chunků v pracovní složce.
3. Vygenerování nenáhodného šifrovacího klíče na základě obsahu privátního klíče, cesta k němuž je uvedena v konfiguračním souboru. Ten se pak používá pro šifrování a dešifrování souborů.
4. Vygenerování nenáhodného hashe uživatele jen na základě privátního klíče, kterým pak budou pojmenované chunky s uživatelskými informacemi.
5. Volání funkce getChunkFromDS, která přidá do fronty požadavků zprávu v následujícím tvaru: RECV:hash_uživatele, kde RECV je flag pro příjem a po dvojtečce sleduje hash požadovaného souboru, v tomto případě hash uživatele, vygenerovaný v předchozím kroku. O frontu požadavků se stará vlákno thread_function, které ji během práce plní.
6. Když vlákno thread_function přidá požadavek do fronty a příslušná funkce modulu ClientConnection ji odešle, server odpoví zprávou ve tvaru RECV:hash_uživatele:velikost_souboru, kde hash_uživatele je název souboru a velikost_souboru je jeho velikost.
   a. Pokud tento soubor na datovém serveru existuje, velikost_souboru bude celé číslo, soubor následně bude stažen ze serveru a po úspěšném stažení klient odpoví zprávou REOK a nastaví proměnnou IS_USER_CHUNK_CREATE na hodnotu 1.
   b. Pokud soubor neexistuje, bude v odpovědi místo velikost_souboru hodnota “NOT_EXIST” a proměnné IS_USER_CHUNK_CREATE přiřadí hodnotu 0.

7. Pokud se proměnná IS_USER_CHUNK_CREATE rovná 0, zavolá se funkce meta_firstLaunch, která vytvoří chunk s uživatelskými informacemi a uživatelský chunk metadat, následně je přidá do fronty požadavků na odesílání.

8. Pokud má proměnná IS_USER_CHUNK_CREATE hodnotu 1, budou zavolány následující funkce:

   a. Funkce meta_user_loadChunk načte uživatelský hash chunk, který byl stažen ze serveru v kroku 5.a.

   b. Funkce meta_user_getRootMetaChunkHash vrátí hash aktuálního uživatelského chunku metadat, který byl nastaven v předchozím kroku.

   c. Funkce meta_loadRootChunk načte uživatelský chunk metadat do paměti.

2.4.2 Komunikace s přístupovým serverem

Klient komunikuje s přístupovým serverem prostřednictvím socketu. Existuje několik druhů zpráv:

1. Klientský požadavek o soubor s hashem

2. Odpověď serveru, jestli soubor existuje a jakou má velikost

3. Klientská zpráva o začátku odesílání souboru

4. Serverová zpráva o ukončení odesílání souboru

5. Klientská zpráva o ukončení příjmu souboru

Podrobněji o zprávách se dočtete v diplomové práci “Distribuované úložiště dat - klientská část” [1].

2.4.3 Příjem a odesílání souboru

Příjem a odesílání souboru se provádí po chuncích. Před odesláním se soubor rozdělí na jednotlivé kousky o maximální velikosti 4 MB. Každý kousek je následně zašifrován algoritmem AES-128 s využitím šifrovacího klíče, který je vygenerován v metodě init v kroku 3 (viz. 2.4.1). Šifrovací metody poskytuje knihovna MetaManagement. Jednotlivé kousky jsou poté odeslány na server.

Příjem souboru probíhá v opačném směru: na začátku se stáhnou všechny kousky souboru. Pak se jednotlivé kousky dešifrují uživatelským šifrovacím klíčem a jsou sloučeny do jednoho souboru.

2.4.4 Aktualizace metadat

Skoro po každé FUSE operaci se vytváří nový uživatelský chunk metadat a chunk s uživatelskými informacemi, který má v sobě hash nového uživatelského chunku metadat (viz. 2.1). Pak jsou oba uživatelské chunky odeslány na přístupový server. V případě sdílení chunku se odesílá navíc chunk metadat tohoto sdíleného souboru.

Kapitola 3

Návrh

3.1 Funkční požadavky UI

Aby knihovna byla použitelná GUI klientem (z anglického Graphical User Interface, tedy grafické uživatelské rozhraní) musí poskytovat následující metody operací se soubory:

1. Metoda kopírování souboru nebo složky včetně jejího obsahu
2. Metoda přesouvání souboru nebo složky včetně jejího obsahu
3. Metoda přejmenování souboru nebo složky
4. Metoda pro změnu počtu replikací souboru
5. Metoda pro změnu nastavení sdílení souboru
6. Metoda pro nahrání souboru ze serveru
7. Metoda pro stažení souboru a vrácení odkazu na něj
8. Metoda pro odstranění souboru nebo složky včetně jejího obsahu

Zmíněnou klientskou GUI aplikaci implementuje v rámci bakalářské práci [2] Artem Grigorian.

3.2 Nefunkční požadavky

Mezi nefunkční požadavky této knihovny patří následující:

1. Knihovna musí mít zabezpečený klíč k šifrování/dešifrování dat
2. Tento šifrovací klíč nesmí opustit knihovnu
3. Komunikace mezi serverem a klientem musí probíhat přes zabezpečený kanál

3.3 Architektura

Volba architektury je velmi důležitá část návrhu. Při správném výběru architektury bude zdrojový kód mít logickou a přehlednou strukturu. Navíc správný výběr architektury umožní snadnou rozšiřitelnost, když bude potřeba přidat nějakou funkcionalitu.

Po analýze distribuovaného systému bylo rozhodnuto, že knihovna bude mít vícevrstvou architekturu a kód bude rozdělen do následujících vrstev:

1. Vrstva, které patří metody pro práce se soubory a metadaty. Tato vrstva bude využívat funkce modulu MetaManagement.
2. Do druhé spadají metody pro komunikaci s přístupovým serverem, tzn. připojení, handshake, posílání a příjem zpráv a odesílání a příjem chunků.
3. Třetí vrstva bude mít metody a třídy, které bude volat GUI klient. Tyto metody využívají funkce dvou předchozích vrstev.

3.3.1 Komunikace

Pro komunikaci s přístupovým serverem bude vytvořena třída, která bude poskytovat metody pro hlavní případy užití:

1. Odeslání požadavku na získání souboru
2. Příjem požadavku
3. Příjem souboru
4. Odeslání souboru

3.3.2 Metadata a práce se soubory

Před odesláním je potřeba rozdělit odesílaný soubor na chunky, stejně jako po příjmu chunků patřících jednomu souboru bude potřeba je spojit. O to se budou starat metody třídy, která bude sloužit pro práci se soubory na disku a práci s metadaty. Metody této třídy budou volat funkci modulu MetaManagement, implementovanou v rámci diplomové práce “Distribuované úložiště dat - správa metadat” [3].

Kapitola 4
Implementační část

Implementace probíhala ve třech fázích. V první fázi byla realizována analýza kódu klientské aplikace a přístupového serveru. Druhá fáze se týkala implementace třídy pro práci se soubory a metadaty. Ve třetí fázi byla implementována komunikace s přístupovým serverem.

4.1 Použité technologie

Během vývoje se pracovalo se dvěma programovacími jazyky, proto byly použity dva operační systémy a vývojová prostředí.

4.1.1 Platforma a vývojová prostředí

Vývoj probíhal na dvou strojích - fyzickém a virtuálním. Jako fyzický stroj byl použit notebook firmy DELL, který má následující charakteristiky: Intel(R) Core(TM) i7-8850H CPU @ 2.60 Hz 2.59 GHz, 16 GB RAM.

Java knihovna byla implementována pomocí Intellij Idea 2018.2.7 Community Edition [7] v operačním systému Windows 10 Pro.

Pro práci a vývoj v knihovnách v C/C++ bylo použito Eclipse IDE for C/C++ Developers [8], ver. 2019-03 (4.11.0) v operačním systému Debian GNU/Linux 9 běžícím na Oracle VM VirtualBox [9].

4.1.2 Java Native Access

JNA je zkratka pro Java Native Access [10]. Jedná se o Java knihovnu, která umožňuje volat nativní funkce knihoven, napsaných v jazyce C/C++ (více informace v kapitole 4.3.1).

4.1.3 Gradle

Jako nástroj pro automatizaci sestavování programu byl zvolen Gradle [11]. Jedná se o projekt původně vyvíjený pro Javu. Tento nástroj je analogem Mavenu [12] a umožňuje jedním tlačítkem stáhnout všechno potřebné pro vývoj knihovny, spustit build nebo psát vlastní úkoly.

4.1.4 Guava

Knihovna Guava [13] je velmi užitečná Java knihovna od firmy Google. Obsahuje hlavně nástroje pro práci s kolekcemi a vlákny. V této bakalářské práci se knihovna používá pro počítání hashe, generování šifrovacího klíče a práci s bytovými proudy.

4.1.5 Crypto

Pro šifrování a dešifrování souborů byla použita knihovna Crypto, která je součástí standardních knihoven Java JDK.

4.1.6 Apache Common-io

Apache common-io [14] je jedna z nejpopulárnějších Java knihoven [15] obsahující velké množství užitečných pomocných tříd. V této práci byla Apache common-io použita pro sloučení souborů.

4.2 Seznam datových tříd

Většina datových tříd je reprezentací odpovídajících struktur z knihovny MetaManagement a má metody pro načtení řádku a vrácení instance třídy.

1. CChunk. Třída reprezentující strukturu CChunk knihovny MetaManagement. Má atributy id - identifikátor chunku souboru, size - velikost chunku souboru a hash - vygenerovaný hash chunku. Tato třída poskytuje metody pro vytváření instancí z řádku a převod kolekce CChunku do řetězcu.

2. FolderContent reprezentuje strukturu FolderContent z knihovny MetaManagement. Obsahuje atributy type - typ nodu, size - jeho velikost, sharedHash - sdílený hash (prázdný pro obecné soubory), sharedKey - sdílený klíč (prázdný pro obecné soubory). Poskytuje metodu pro načtení řádku a vracení instancí této třídy.

3. HashChunk je kopie struktury HashChunk knihovny MetaManagement, má atributy ttype - typ nodu, size - jeho velikost, sharedHash - sdílený hash (prázdný pro obecné soubory), sharedKey - sdílený klíč (prázdný pro obecné soubory) a metody pro vytváření objektů z řádku.

4. InitData je třída pro ukládání šifrovacího klíče a hashe uživatele. Plní se při inicializaci knihovny.

5. Třída Record obsahuje dva atributy: seznam objektů typu HashChunk a seznam objektů typu CChunk. Tato trída se používá při odesílání pro seskupení uživatelských metadat a chunku souboru v rámci jedné operace.

4.3 Použití modulu MetaManagement

O správu metadat se starají funkce modulu MetaManagement, proto je potřeba zvolit framework, který by povolil tyto funkce volat z přímo z Javy a zároveň by byl co nejjednodušší na použití. Pro tyto účely byla zvolená knihovna JNA.

4.3.1 Java Native Access

JNA Java Native Access je Java knihovna, která umožňuje volání funkcí C/C++ modulu v Javě. Hlavními výhodami této knihovny je to, že je snadno použitelná (stačí psát jenom Java kód) a nevyžaduje použití JNI. Dalšími výhodami jsou:

1. Velmi jednoduché mapování funkcí a datových typů
2. Stále se aktualizuje
3. Má otevřený kód, který je uložen v repozitáře na stránkach github [10].

Příklad použití

Níže je uveden příklad použití funkce printf z jazyku C v Javě.

```
public interface MojeKnihovnaC extends Library {
    void printf(String format, Object... args);
}
public class Priklad {
    public static void main(String[] args) throws
        ↪ InterruptedException {
        MojeKnihovnaC libraryMapping = (MojeKnihovnaC) Native.
            ↪ loadLibrary("NativeLibrary",
            ↪ YourOwnLibraryMapping.class);
        libraryMapping.printf("Jna␣%s", "Example");
    }
}
```

Jak je vidět, tak pro fungování stačí vytvořit Java interface se stejným názvem metody a změnit datové typy podle tabulky 4.1:

4.3.2 Úpravy v modulu MetaManagement

Během vývoje vznikala spousta problémů, které se týkaly hlavně mapování pole struktur na Java třídy. Proto bylo potřeba upravit logiku některých funkcí knihovny MetaManagement.

Native Type	Size	Java Type	Common Windows Types
char	8-bit integer	byte	BYTE, TCHAR
short	16-bit integer	short	WORD
wchar_t	16/32-bit	character char	TCHAR
int	32-bit integer	int	DWORD
int	boolean value	boolean	BOOL
long	32/64-bit integer	NativeLong	LONG
long long	64-bit integer	long	__int64
float	32-bit FP	float	
double	64-bit FP	double	
char*	C string	String	LPTCSTR
void*	pointer	Pointer	LPVOID, HANDLE, LPXXX

Tabulka 4.1: JNA - Mapování primitivních typů

Úpravy v modulu CObject

Následující úpravy byly provedeny v modulu CObject:

1. Funkce

```
HashArray convertVectorToHashArray(vector<FileHash*> hashes
    ↪ )
```

   která vrací objekt HashArray byla nahrazena funkcí

```
char* convertVectorOfHashArray(vector<FileHash*> hashes)
```

   která vrací ukazatel na char, což v Javě je String. Místo toho aby byl v těle funkce vektor struktur FileHash přeměněn na strukturu HashArray, mění se na řetězec, kde hodnoty parametru FileHash jsou rozdělené znakem '|' a jednotlivé elementy jsou rozdělené znakem ','. Příklad výstupu takové funkce pro 3 objekty ve vektoru:

```
typ1|size1|key1|hash1,typ2|size2|key2|hash2,typ3|size3|key3
    ↪ |hash3
```

   kde typ je typ objektu, size je velikost objektu, key je sdílený klíč a hash je hash objektu.

2. Funkce

```
HashChunk convertFileHashToHashChunk(FileHash* _hash)
```

   která vrací strukturu HashChunk byla nahrazena funkcí

```
char* convertFileHashToHashChunk(FileHash* _hash)
```

   která taky vrací ukazatel na char. V těle funkce byl místo převodu FileHash objektu na objekt HashChunk vytvořen vektor obsahující pouze

jeden element typu FileHash a pak zavolána funkce convertVectorToHashArray.

3. Stejným způsobem byla nahrazena funkce

```
CChunkArray convertVectorToCChunkArray(vector<Chunk*>
    ↪ _chunks)
```

která pracuje s objekty CChunk funkcí

```
char* convertVectorToCChunkArray(vector<Chunk*> _chunks)
```

4. Aby se dala použít funkce

```
vector<Chunk*> convertCChunkArrayToVector(CChunkArray*
    ↪ _cchunkArray)
```

bylo potřeba změnit typ parametru z ukazatele na CChunkArray na ukazatel na char. Teď tato funkce vypadá následovně:

```
vector<Chunk*> convertCChunkArrayToVector(char* cchunkArray
    ↪ )
```

Na vstupu musí být řádek, který má uvedené hodnoty objektu CChunk rozdělené symbolem '|'. Jednotlivé elementy jsou rozdělené symbolem ','. Dále je uveden příklad tohoto objektu: id|size|hash, kde id je identifikátor chunku, size je jeho velikost a hash je jeho vlastní hash, který je zároveň jménem souboru na disku.

5. Funkce

```
FolderContentArray convertVectorToFolderContent(vector<Node
    ↪ *> _content)
```

byla nahrazena funkcí

```
char* convertVectorOfFolderContent(vector<Node*> _content)
```

která převede vektor struktur Node do řetězců, s použitím atributů type, size, name, shared hash a shared key. Type je typ nodu, size je jeho velikost, name je jméno. Další dva atributy shared key a shared hash byly přidány proto, aby při získávání obsahu adresáře se (funkce meta_getFolderContent modulu MetaManupulation) vracel i sdílený hash a sdílený klíč tohoto souboru. U normálních souborů a složek atributy shared hash a shared key mají prázdnou hodnotu.

Úpravy v modulu MetaManagement

1. Všechny funkce mající jako návratový typ strukturu HashChunkArray po úpravě vrací ukazatel na char. Uvnitř těchto funkcí před návratem hodnoty byly zavolány odpovídající funkce, určené pro proměnnou z polí struktur na ukazatel na řádek.

2. Funkce meta_updateFile a meta_createNewFile teď místo struktury CChunkArray přijímají ukazatel na char (String v Javě) a v těle ho převádějí do vektoru struktur pomocí funkce convertCChunkArrayToVector 4.3.2.

3. Stejným způsobem byla upravena funkce meta_error.

4. Funkce

```
FolderContentArray meta_getFolderContent(const char* _path)
```

byla nahrazena funkcí

```
char* convertVectorOfFolderContent(vector<Node*> _content)
```

a teď vrací o dva atributy navíc: sdílený klíč a sdílený hash.

Úpravy v modulu MetaManipulation

Ve funkci loadSharedMetaChunk se při načítání obsahu objektu teď načítá i sdílený klíč voláním newNode->setKey(_key). Tato úprava umožňuje po uvolnění celé paměti a spuštění inicializace dostat klíč tohoto přidaného sdíleného souboru.

Více o uvedených typech a funkcích se dá přečíst v diplomové práci "Distribuované úložiště dat - správa metadat" [3].

Třída MetaManagementWrap

Pro mapování funkcí knihovny MetaManagement byla vytvořena třída se stejným názvem. V aplikaci však se používá třída MetaManagementWrap, která je obálkou pro třídu MetaManagement a deleguje všechny její metody. Nicméně kvůli tomu, že po úpravách popsaným v předchozích kapitolách většina funkcí vrací řádky místo struktur, bylo potřeba udělat převody do odpovídajících Java tříd.

Nepoužité funkce knihovny metaManagement

Funkce pro šifrování, dešifrování, vytváření hashe a získávání času, obsažené v modulu sfuncs, nebyly použity kvůli tomu, že způsobovaly chyby při volání z Javy:

1. int encryptFileAES128CTR(const char* _readFilePath, const char* _writeFilePath, const char* _key)

2. int decryptFileAES128CTR(const char* _readFilePath, const char* _writeFilePath, const char* _key)

3. void genHashSha256_64ext(char* _output, char* _filePath, char* _username, char* _timestamp)

Subsystém, modul	Funkce	Opravený bug
Metadata, MetaManagement	meta_loadSharedChunk	Místo toho, aby do proměnné key byla kopírována hodnota proměnné _key, kopírovala se hodnota proměnné _fileName což působilo špatné chování při sdílení souborů
Metadata, MetaManipulation	loadSharedMetaChunk	Tato funkce po úspěšném přidání sdíleného chunku do kořene vracela 0, místo toho aby byl vytvořen nový uživatelský chunk metadat a aktualizován chunk s uživatelskými informacemi. Byla opravena přidáním volání funkcí createJSONMetaFiles a userCreateJSONFile na konec funkce a následným vrácením vektoru struktur FileHash. Navíc teď, když nastane nějaká chyba, vrací tato funkce vektor s jediným elementem FileHash, který místo hashe obsahuje kód chyby.
Přístupový server, NewUserConnection	run	Chyba byla nalezena ve funkci pro posílání zprávy klientu o ukončení odesílání souborů. V těle této funkce v bloku if se řádek "SEND" porovnával s hodnotou proměnné opCode, která by měla obsahovat kód zprávy. Chyba spočívala v tom, že tato proměnná byla objevena ale nebyla inicializována a proto neměla žádnou hodnotu. Chyba byla opravena tím, že proměnná opCode byla nahrazena proměnnou msg_opCode.

Tabulka 4.2: Nalezené bugy

4. void getCurrentTime(char* _output)

5. void genPrivateKeyHash(char* _output, char* _filePath)

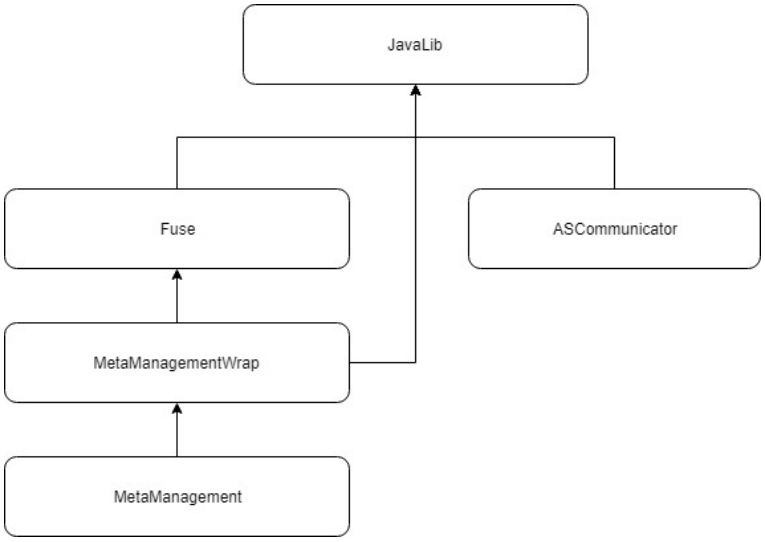
Tyto funkce byly znovu implementovány a nachází se v pomocné třídě CryptoUtils.

4.4 Nalezené bugy

Během vývoje bylo nalezeno několik bugů, které jsou popsané v tabulce 4.2.

4.5 Implementované třídy

Podle návrhu byly implementovány tři hlavní třídy: Fuse, ASConnector a JavaLib. Na obrázku 4.1 je schematicky zobrazena mapa tříd a jejich vzájemné použití:



Obrázek 4.1: Schematická mapa tříd

4.5.1 Třída Fuse

Pro účely správy metadat a práce se soubory byla vytvořena třída s názvem Fuse. Tato třída poskytuje metody pro operace se soubory:

1. List<HashChunk> fs_mkdir(String metaFolderFullPath).
   Tato metoda vytváří cílový adresář. Na vstupu má proměnnou type String, která musí obsahovat plnou cestu k přidávané složce. Na výstupu bude vrácen seznam chunků (chunk s uživatelskými informacemi a uživatelský chunk metadat). Každý tento chunk pak bude odeslán na datový server.

2. Record fs_write(String metaFullFilePath, String fullFilePath, String encryptionKey, boolean shared).
   Metoda pro zápis souboru do metadat. Na vstupu má plnou cestu k vytvářenému souboru (včetně jména), plnou cestu k souboru na disku, který je potřeba zapsat, klíč pro šifrování chunku souboru, který v případě nahrávání sdíleného souboru bude mít hodnotu sdíleného klíče a v případě nahrávání normálního souboru - šifrovací klíč uživatele. Pak metoda rozdeli soubor na chunky, spocita pro kazdy chunk jeho hash, zasifruje jednotlive chunky a prejmenuje na predem spocitany hash.

3. List<HashChunk> fs_rename(String metaFromNodeFullPath, String newNodeName, Integer numberOfReplications)
   Metoda provádí přejmenování souboru nebo složky a navíc, když se parametr numberOfReplications nebude rovnat null, změní se i počet replikací pro soubor.

4. List<FolderContent> fs_readdir(String metaFolderFullPath)
   Tato metoda přečte cílovou složku a vrátí seznam objektů FolderContent, reprezentující obsah tohoto adresáře.

5. List<HashChunk> fs_copy(String metaNodeFullPathFrom, String metaNodeFullPathTo, boolean deleteSource)
   Metoda zkopíruje node na cestě metaNodeFullPathFrom do složky metaNodeFullPathTo. Pokud se kopíruje soubor, bude zkopírován i jeho obsah použitím rekurzivního volání metod.

6. List<HashChunk> fs_rm(String metaNodeFullPath)
   Metoda smaže cílový node.

7. File fs_read(List<File> fileChunks, String cipherKey)
   Metoda určená pro přečtení souboru. Na vstupu je seznam chunků cílového souboru, které budou za prvé dešifrované klíčem cipherKey a pak sloučené do jednoho souboru. Na konci bude cesta k tomuto souboru na disku vrácena jako objekt třídy File.

8. Record fs_update(String metaFullFilePath, String fullFilePath, String encryptionKey)
   Metoda určená pro aktualizaci souborů po provedení nějaké změny. Nejdřív bude soubor, nad kterým je změna prováděna, rozbít na jednotlivé chunky, které se pak šifrují klíčem encryptionKey a vytváří se seznam chunku. Tento seznam pak bude přidělen aktualizovanému souboru v metadatech.

4.5.2 Třída ASCommunicator

Tato třída poskytuje metody pro odesílání a příjem zpráv a souborů:

1. void sendRECVMessage(String hash)
   Metoda má na vstupu hash souboru, který bude požadován po odeslání zprávy na server.

2. void sendOKMessage()
   Tato metoda odešle zprávu o ukončení příjmu souboru.

3. void sendMessage(ClientSendMessage clientSendMessage)
   Volání této metody převede objekt ClientSendMessage do řádku a pak ho odešle na server. Objekt třídy ClientSendMessage může obsahovat flag USCH pro odesílání chunku s uživatelskými informacemi nebo flag SEND pro odesílání chunku souboru nebo uživatelského chunku metadat.

4. ServerMessage receiveMessage()
   Metoda určená pro příjem zprávy od přístupového serveru ve formátu řádku, který pak bude převeden do objektu třídy ServerMessage. Tento objekt má atributy hash souboru a size - jeho velikost. Pokud server odpověděl, že soubor neexistuje, bude atribut size mít hodnotu null.

5. File sendFile(String fileHash, long fileSize)
   Metoda přijímá hash souboru, který musí být odeslán, a jeho velikost, a po ukončení příjmu vrátí odkaz na tento soubor na disku.

6. File receiveFile(String fileHash, long fileSize)
   Tato metoda má stejné parametry jako ta předchozí. Přijme soubor o velikosti fileSize od přístupového serveru a po ukončení příjmu ho přejmenuje na fileHash a vrátí odkaz na tento soubor na disku.

TLS handshake

Transport Layer Security (TLS) je protokol, který určen pro zabezpečení komunikace mezi dvěma aplikacemi [16]. Při vývoji se TLS spojení nepoužívalo a bylo nahrazeno nezabezpečným TCP spojením. Hlavním důvodem byl nedostatek času. Navíc, u Java knihovny, která zajišťuje TLS spojení, chyběla dokumentace s příklady, což by hodně zpomalilo vývoj. Proto se rozhodlo, že bude využito nezabezpečené spojení.

4.5.3 Třída JavaLib

Metody třídy JavaLib budou použité přímo uživatelem, v tomto případě GUI klientem. Kvůli tomu, že některé případy užití jsou zaměřené na práce s několika soubory najednou (například když uživatel bude chtít zvolit několik souborů nebo složek a přesunout je nebo smazat), byla vytvořena metoda execute. Tato metoda je určená pro odesílání seznamu uživatelských metadat a chunku souboru po zpracování funkce, která je vstupním parametrem metody execute(). Tuto funkci v parametru musí uživatel implementovat tak, že v těle zavolá všechny potřebné metody třídy JavaLib (například přesunutí všech zvolených souborů) a pak vrátí objekt třídy Record, obsahující všechny poslední změny. Dělá se to, aby pokaždé při volání jakékoliv metody se zbytečně neposílala všechna data na server. Pokud metoda execute() nebude zavolána, neuloží se provedené změny na server.

4.6 Přidání složky

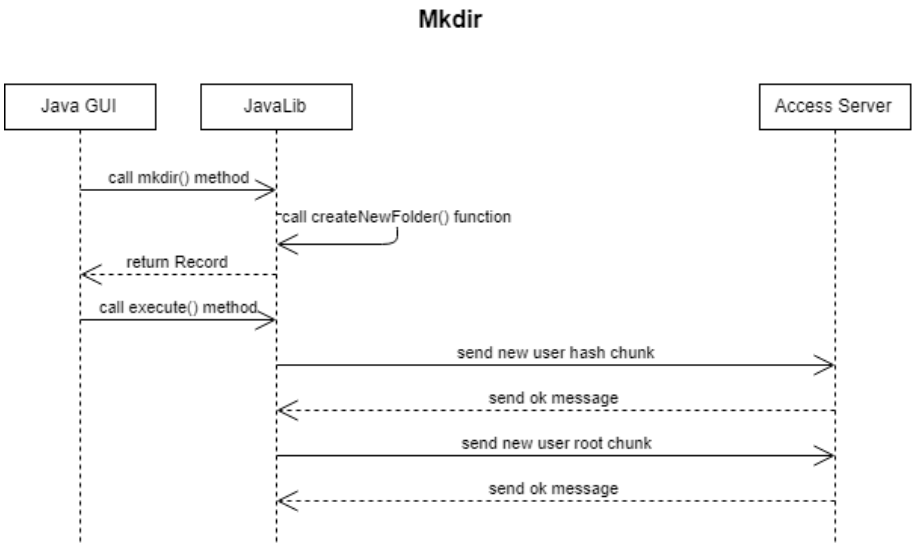
Pro přidání složky byla implementována metoda mkdir(String path) třídy JavaLib. Níže je zobrazen sekvenční diagram pro tyto události 4.2:

Tato metoda volá metodu fs_mkdir třídy Fuse, která použitím metody meta_createNewFolder třídy MetaManagementWrap vytvoří prázdnou složku v metadatech a vrati nove uzivatelske chunky. Path je cesta k souboru v metadatech, například: “/slozka1/slozka2”.

4.7 Přidání normálního a sdíleného souboru

Metoda write(String metaFullFilePath, String fullFilePath, boolean shared) nahraje soubor na server. Níže je uveden sekvenční diagram průběhu přidávání souboru 4.3:

Když se zavolá metoda write, vytvoří se prázdný soubor. Pokud uživatel zvolil přidávání sdíleného souboru, bude nad ním zavolána metoda



Obrázek 4.2: Sekvenční diagram - mkdir

meta_setFileShared, čímž se mu změní typ, spočítá se a přidělí sdílený hash a klíč. Pak se změří velikost nahrávaného souboru a provede se jeho rozdělení na jednotlivé chunky. Pak pro každý chunk bude spočítán jeho hash, kterým bude po šifrování přejmenován a zašifrován buď sdíleným nebo uživatelským šifrovacím klíčem. Potom se voláním metody meta_updateFile na začátku vytvořený soubor aktualizuje a bude mu přidělen sdílený šifrovací klíč a nedávno vytvořené chunky. Na konci se na server odešlou oba uživatelské chunky a chunky nahrávaného souboru.

4.8 Úprava souboru

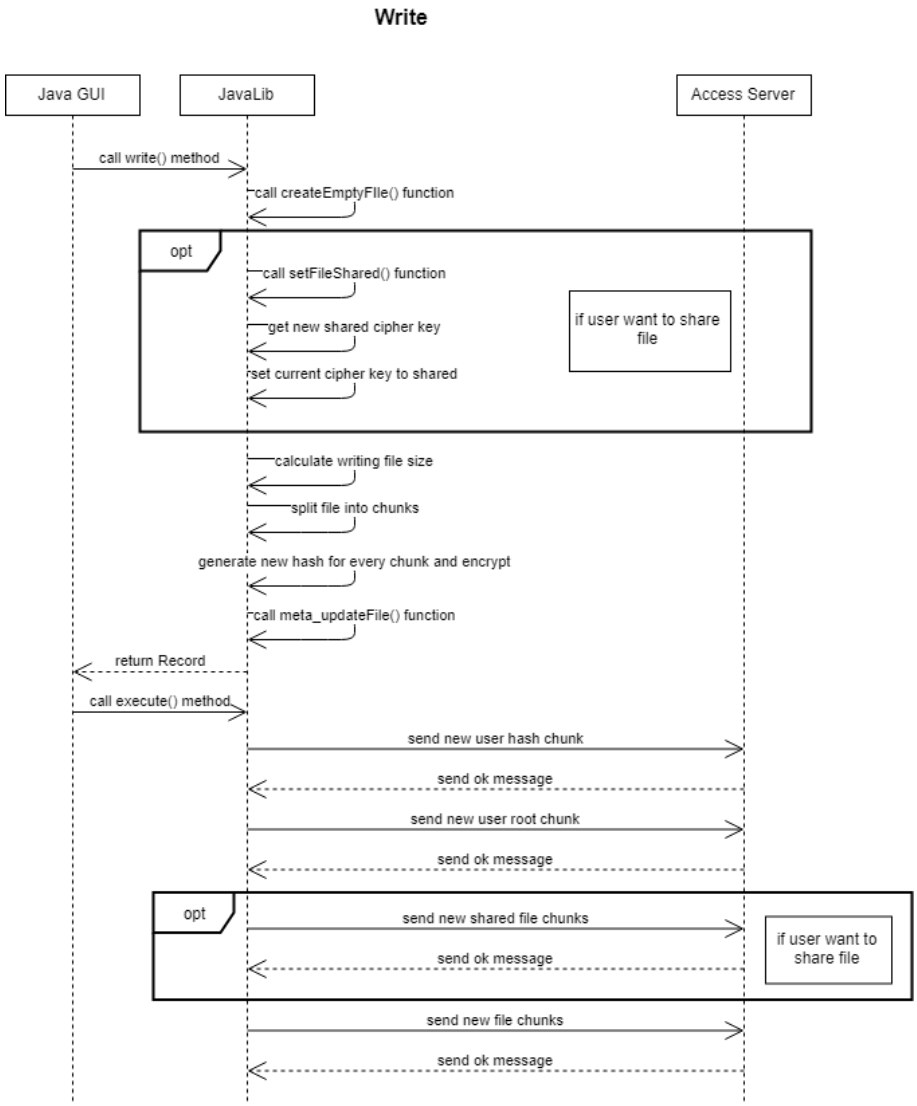
Uživatel může po provedení změny nějakého souboru soubor aktualizovat zavoláním metody update(String metaFullFilePath, String fullFilePath). Níže je uveden sekvenční diagram pro aktualizaci souboru 4.4:

Aktualizace souboru se provádí skoro stejně jako vytváření, pouze v případě aktualizací se nebude vytvářet nový soubor, ale bude se měnit již existující.

4.9 Mazání souboru nebo složky

Mazání souboru nebo složky metoda remove provádí pouze v metadatech, to znamená že žádný chunk na serveru v rámci této operaci smazán nebude. Více o mazání nepoužívaných chunku na serveru je v diplomové práci “Distribuované úložiště dat - správa dat” [4]. Níže je zobrazen sekvenční diagram pro tuto událost 4.5.

Metoda remove volá metodu List<HashChunk> fs_rm(String metaNodeFullPath) třídy Fuse. Tato metoda zjistí, jestli uživatel se snaží smazat soubor nebo složku. Pokud metaNodeFullPath je cesta k souboru, zavolá se metoda



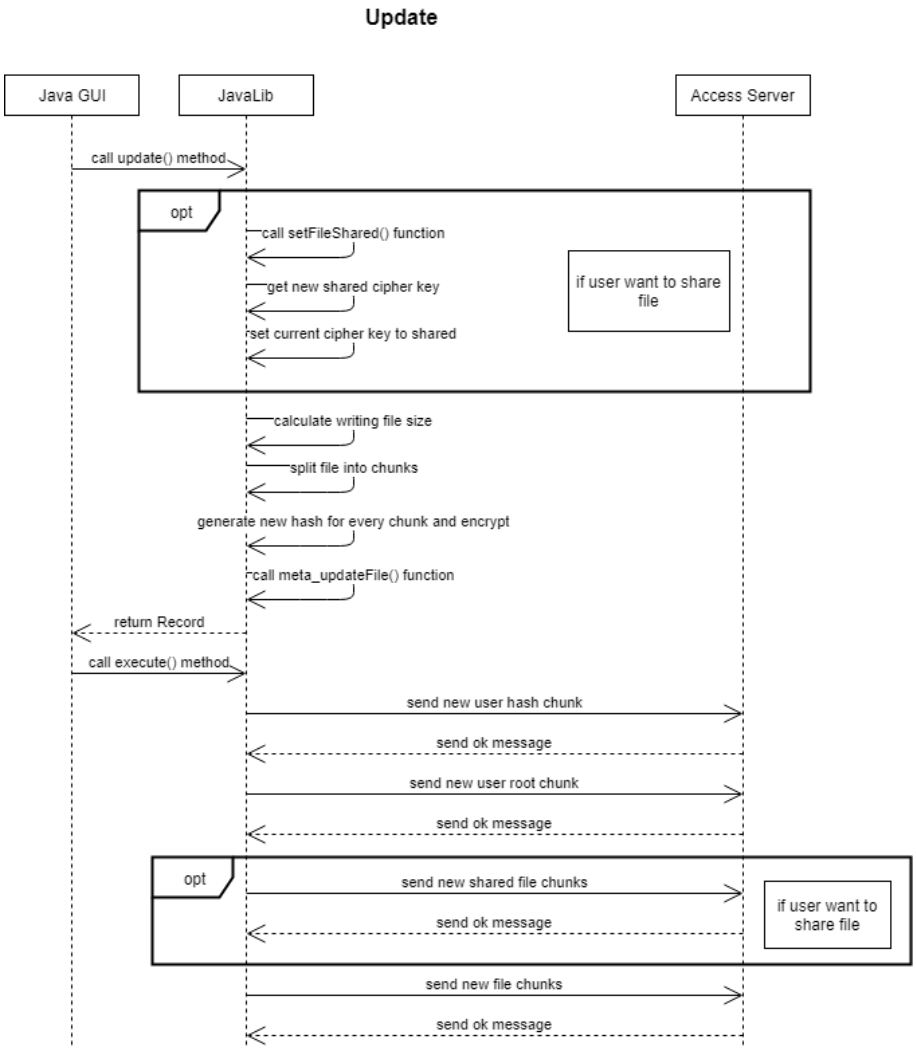
Obrázek 4.3: Sekvenční diagram - write

meta_deleteFolder třídy MetaManagementWrap, pokud metaNodeFullPath je cesta k souboru, zavolá se metoda meta_deleteFile.

4.10 Čtení souboru

Metoda read(String metaFileFullPath) umožňuje uživateli přečíst již existující soubor. Přečíst znamená stáhnout chunky souborů, sloučit je a vrátit odkaz na výsledný soubor pro další použití. Níže je s uveden sekvenční diagram pro tento případ užití 4.6:

Metoda read, pokud požadovaný soubor je sdílený, stáhne sdílený chunk metadat a přidá ho do paměti pomocí metody meta_loadSharedChunk. Pak voláním metody meta_getFileChunks bude získán seznam chunku požado-

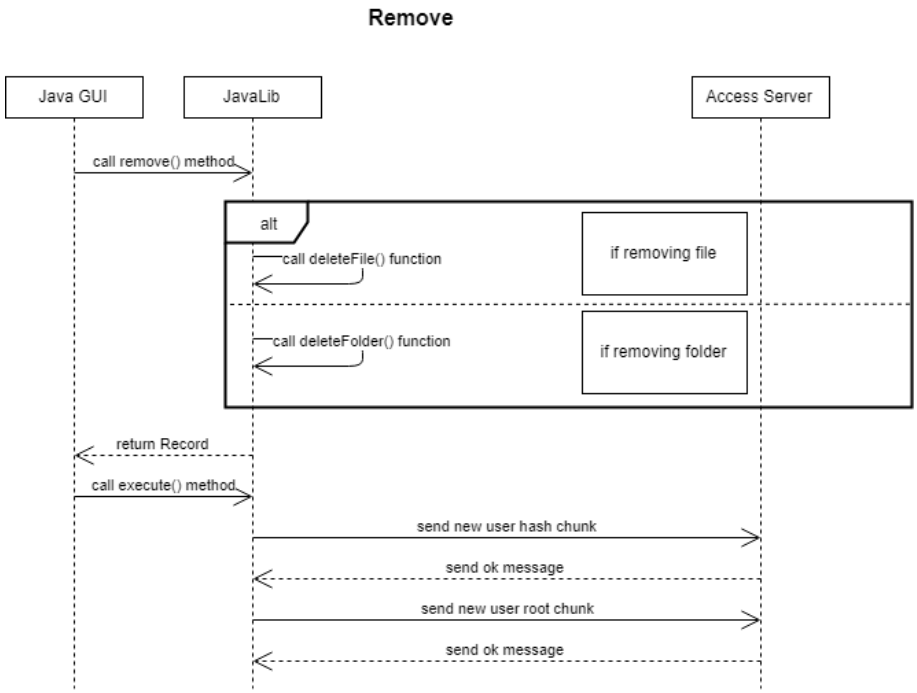


Obrázek 4.4: Sekvenční diagram - update

vaného souboru. Dále všechny chunky souboru budou staženy z serveru, dešifrovány sdíleným nebo uživatelským klíčem a sloučeny do jednoho souboru, odkaz na který bude vrácen uživatelů. Tuto metodu není nutné volat uvnitř metody execute.

4.11 Čtení sdíleného souboru

Metoda readShared(String sharedHash, String sharedKey) stáhne a přidá do uživatelského chunku metadat požadovaný soubor. Níže je uveden odpovídající sekvenční diagram 4.7:



Obrázek 4.5: Sekvenční diagram - remove

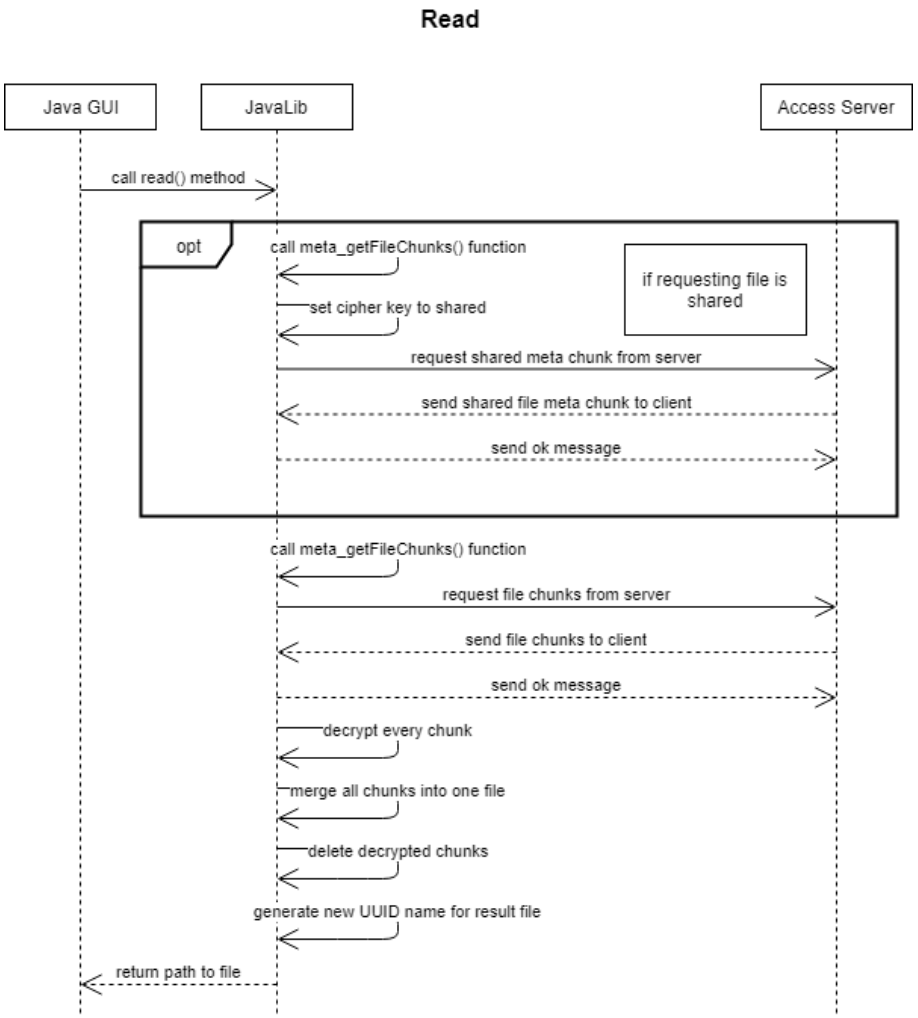
4.12 Přesouvání a kopírování souboru nebo složky

Pro kopírování a přesouvání složek nebo souborů se používají metody move(String metaFullFilePathFrom, String metaFullFilePathTo) a copy(String metaFullFilePathFrom, String metaFullFilePathTo). Jak lze vidět na sekvenčním diagramu 4.8 metody jsou identické, akorát při přesouvání se výchozí node maže a při kopírování - ne.

Při volání metody copy nebo move se používá stejná metoda třídy Fuse: fs_copy(String metaNodeFullPathFrom, String metaNodeFullPathTo, boolean deleteSource). Rozdíl je v tom, že hodnota parametru deleteSource v případě kopírování je pravda (true) a při přesouvání je nepravda (falše). Tento parametr rozhoduje jestli se bude výchozí soubor nebo složka smazána. Navíc je třeba zmínit, že při práce s neprázdnou složkou bude zkopírován / přesunut i jeho obsah voláním rekurzivním voláním metody fs_copy.

4.13 Přejmenování souboru nebo složky

Jméno souboru nebo složky se mění jenom v metadatech. Pro tyto účely slouží metoda rename(String fromFullPath, String toNodeName, Integer replications). Pokud uživatel mění jméno souboru, dá se i změnit jeho počet replikací. Pokud na vstupu výchozí node má stejné jméno jako cílový - žádná změna se neprovede. Když se jedná o přejmenování souborů a uživatel měnit počet replikací nechce, předá do parametru numberOfReplications nulovou



Obrázek 4.6: Sekvenční diagram - read

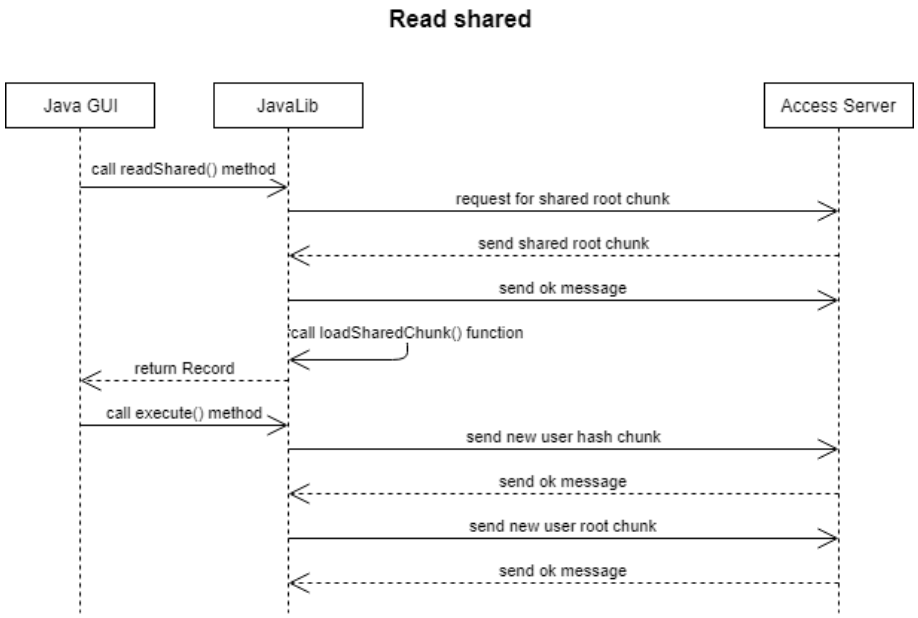
hodnotu. Podrobnější přehled fungování této metody je zobrazen v sekvenčním diagramu níže 4.9:

4.14 Ukončení práce

Po ukončení prací není třeba ručně uzavírat spojení s přístupovým serverem, nicméně se velmi doporučuje zavolat metodu exit(), která použitím funkce meta_exit modulu MetaManagement provede správné uvolnění paměti.

4.15 Šifrování a hashování

Pro šifrování a hashování chunku souboru byly implementovány speciální metody obsažené v pomocné třídě CryptoUtils. Důvod, proč byly přepsané do Javy, ale ne zavolané z modulu MetaManagement je ten, že se při použití



Obrázek 4.7: Sekvenční diagram - readShared

JNA z nějakého důvodu vyskytovaly chyby a celá aplikace padala.

4.16 Vlastní výjimky

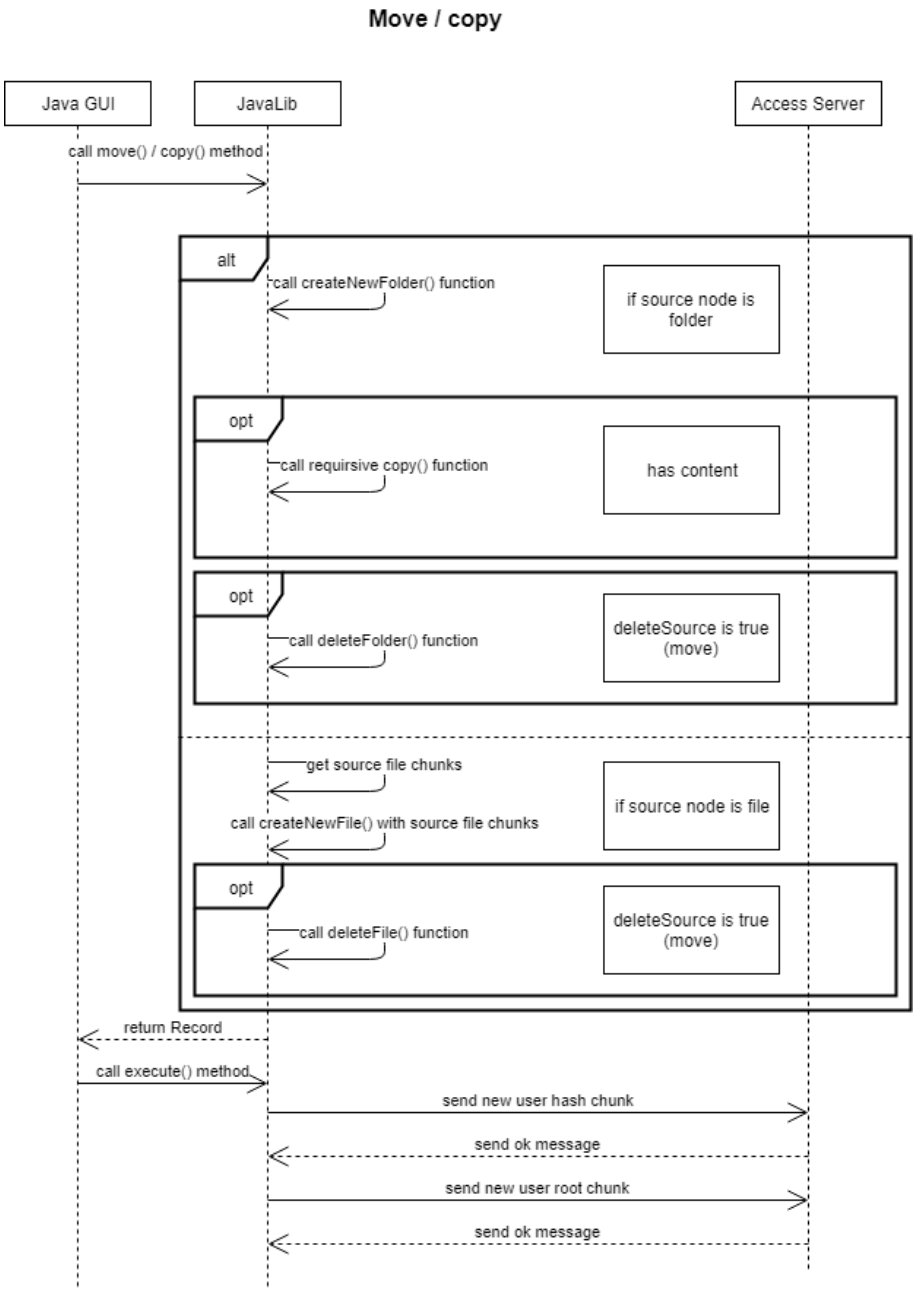
Během implementace byly třeba vytvořit několik výjimek, které uvedené a popsané v seznamu níže:

1. CryptoException - tuto hlášku hází metody třídy CryptoUtils v případě, když se nepodařilo šifrování nebo dešifrování chunku.
2. FileAlreadyExistsException - hláška určená pro případy, když soubor nebo složka v cílovém adresáře uživatelských metadat už existuje.
3. FileDoesNotExistException - tato výjimka se vrací v případě, když požadovaný soubor nebo složka neexistuje v metadatech uživatele.

4.17 Konfigurace a konstanty

Třída Configuration je určena pro načítání konfiguračních dat ze souboru. Pro vytváření instancí třídy JavaLib je potřeba mít instanci této třídy. Obsah konfiguračního souboru je následující:

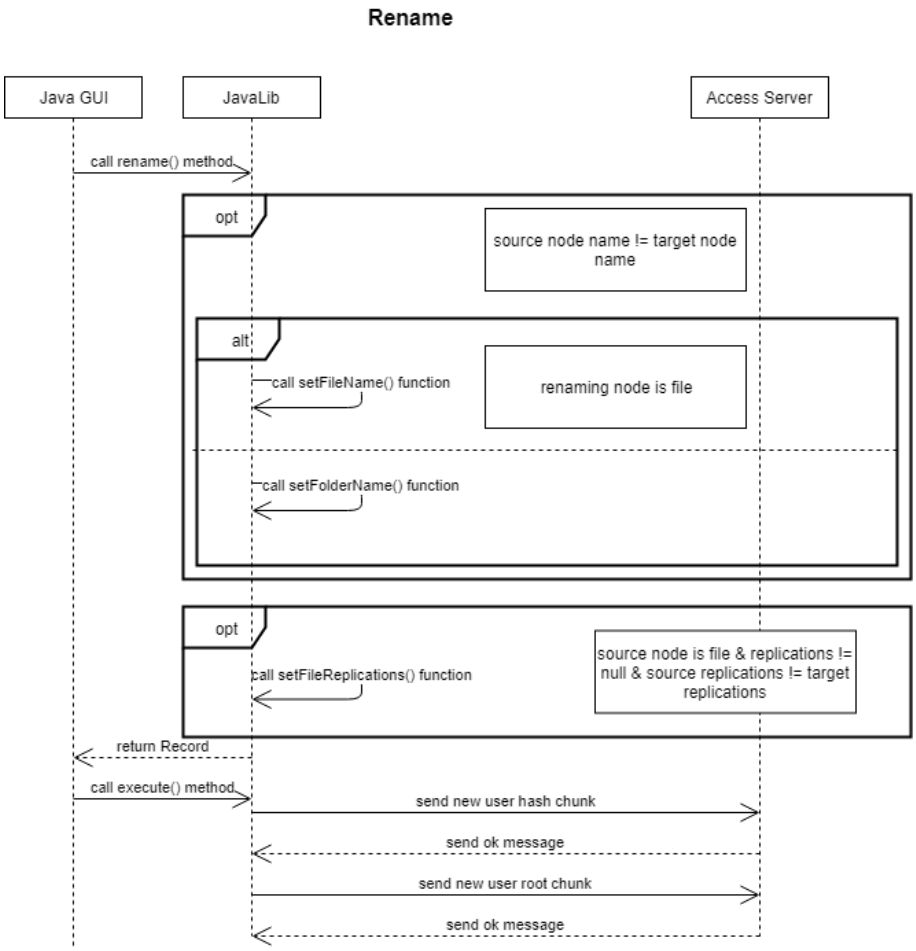
```
username=jméno uživatele
privateKeyPath=plná cesta k privátnímu klíči
cachePath=cesta pro ukládání dočasných souborů
numberOfReplication=výchozí počet replikací pro soubory
accessServerIp=ip adresa přístupového serveru
```



Obrázek 4.8: Sekvenční diagram - move a copy

```
accessServerPort=port přístupového serveru
storageSize=velikost úložiště
```

Třída Constants obsahuje konstanty důležité pro fungování aplikací, jako například jméno zkompilované sdílené knihovny (shared library) MetaManagement (pro JNA).



Obrázek 4.9: Sekvenční diagram - rename

Kapitola 5

Testování

Testování implementované knihovny probíhalo ve třech fázích.

5.1 Testování knihovny JavaLib

První fáze je testování s použitím jednotkových testů, které byli napsané během vývoje skoro pro každou třídu. Aby testy byli nezávislé na ostatních třídách a metodách, bylo použité tak zvané mockování. Mockování znamená nahrazení původní třídy za její testovací verzi, která imituje její práci. Všechny testy byly napsané s použitím knihovny JUnit [17].

5.2 Testování v rámci celého systému bez GUI Klienta

Druhá fáze testování se týkala testu knihovny v rámci distribuovaného systému bez použití GUI Klienta. Pro testování byla napsaná třída JavaLibIntTest, metody které testují celou knihovnu, stejně jako by jí používal GUI Klient. Pro spuštění těchto testů je potřeba mít běžící přístupový a datový server.

5.3 Testování v rámci celého systému s GUI Klientem

Poslední fáze testování byla s použitím GUI Klienta, kterého implementoval Artem Grigorian v rámci bakalářské práci "Distribuované úložiště dat - uživatelské rozhraní". Testování probíhalo, jak ručně, tak i s použitím testovacích metod na straně GUI Klienta.

Kapitola 6

Co zbývá / Možnosti rozšíření

Zůstává mnoho možností pro zlepšení knihovny a celého systému. Za prvé se dá zabezpečit spojení mezi knihovnou a přístupovým serverem pomocí protokolu TLS, což bylo z této práce vynecháno z důvodu nedostatku času. Pak je možné přepsat celou knihovnu MetaManagement do jazyka Java, aby celý klient byl nezávislý na platformě. Třetí možnost je implementace logiky změnového chunku a historie změn, což se z důvodu nedostatku času nedostalo do diplomové práce "Distribuované úložiště dat - správa metadat" a proto ani do knihovny JavaLib. Dá se řešit logika komunikace přístupového a datového serveru: v současném stavu se pokaždé otevírá nové spojení, což velmi prodlužuje čas pro odesílání a příjem zpráv a souborů.

Kapitola 7

Závěr

Cílem této práce byl návrh a vytvoření knihovny v jazyce Java, která by byla použitelná v práci "Distribuované úložiště dat - uživatelské rozhraní" [2] a splňovala její požadavky. Nejdřív byla provedena analýza již hotového distribuovaného systému, implementovaného v rámci tří diplomových prací "Distribuované úložiště dat - správa metadat" [3], "Distribuované úložiště dat - správa dat" [4] a "Distribuované úložiště dat - klientská část" [1]. Nejdůležitější částí této analýzy byla klientská část aplikací a metadata. Během vývoje bylo nalezeno několik chyb v kódu, které následně byly úspěšně opraveny. Protože nebyla všechna funkcionalita implementována v diplomových pracích, bylo třeba je částečně dodělat v Java knihovně.

Implementovaná JavaLib knihovna byla otestována a splňuje všechny požadavky GUI Klienta, implementovaného v bakalářské práci "Distribuované úložiště dat - uživatelské rozhraní" [2].

Zůstává mnoho možností pro rozšíření a zlepšení systému. Pro kompletní nezávislost klienta na platformě je možné přepsat celou knihovnu MetaManagement do jazyka Java. Také se dá zvýšit rychlost zpracování požadavků pomocí úpravy logiky komunikace mezi přístupovým a datovým servery.

Během této práce jsem prohloubil a rozšířil své znalosti programovacích jazyků C a C++. Nejvíce času a úsilí mi zabralo zkoumání a úpravy kódu knihovny MetaManagement, která byla implementována v rámci diplomové práci "Distribuované úložiště dat - správa metadat" [3].

Literatura

[1] J. Janura, "Distribuované úložiště dat - klientská část," 2014, diplomová práce. Praha. ČVUT FEL, Katedra počítačů.

[2] A. Grigorian, "Distributed data storage - desktop ui," 2019, bakalářská práce. Praha. ČVUT FEL, Katedra počítačů.

[3] D. T., "Distribuované úložiště dat - správa metadat," 2014, diplomová práce. Praha. ČVUT FEL, Katedra počítačů.

[4] M. Kudrnáč, "Distribuované úložiště dat - správa dat," 2014, diplomová práce. Praha. ČVUT FEL, Katedra počítačů.

[5] "About sqlite," https://www.sqlite.org/copyright.html, accessed : 12/03/2019.

[6] "Github - libfuse/libfuse: The reference implementation of the linux fuse (filesystem in userspace) interface," https://github.com/libfuse/libfuse, accessed : 28/02/2019.

[7] "Intellij idea: The java ide for professional developers by jetbrains," https://www.jetbrains.com/idea/, accessed : 01/05/2019.

[8] "Eclipse ide for c/c++ developers | eclipse packages," https://www.eclipse.org/downloads/packages/release/kepler/sr2/eclipse-ide-cc-developers, accessed : 01/05/2019.

[9] "Oracle vm virtualbox," https://www.virtualbox.org/, accessed : 10/05/2019.

[10] "Github - java-native-access/jna: Java native access," https://github.com/java-native-access/jna, accessed : 22/03/2019.

[11] "Gradle build tool," https://gradle.org/, accessed : 14/04/2019.

[12] "Maven repository: Top projects at maven repository," https://mvnrepository.com/popular, accessed : 13/04/2019.

[13] "Github - google/guava: Google core libraries for java," https://github.com/google/guava, accessed : 26/04/2019.

[14] “Apache commons – apache commons,” https://commons.apache.org/, accessed : 01/05/2019.

[15] Mvnrepository. (2019) Top projects. [Online]. Available: https://mvnrepository.com/popular

[16] A. PRODROMOU. (2015) Tls security 1: What is ssl/tls | acunetix. [Online]. Available: https://www.acunetix.com/blog/articles/tls-security-what-is-tls-ssl-part-1/

[17] “Junit – about,” https://junit.org/junit4/, accessed : 10/05/2019.

Příloha A

Uživatelská příručka

1. Spustit datový server
2. Spustit přístupový server
3. Zkompilovat knihovnu MetaManagement pomocí programu make
4. Vytvořit konfigurační soubor. Příklad obsahu:

```
username=MyName
privateKeyPath=/keys/myPrivateKey.pk
cachePath=/cache/
numberOfReplication=0
accessServerIp=127.0.0.1
accessServerPort=6000
storageSize=4000000
```

5. Vytvořit instance třídy Configuration následovně:

```
public static void main(String[] args) throws IOException
    ↪ {
            Configuration config = Configuration.
                ↪ fromInputStream(new FileInputStream("/
                ↪ config.cfg"));
    }
```

6. Vytvořit instance třídy JavaLib a zavolat funkci init():

```
JavaLib javaLib = new JavaLib.Builder(config).build();
javaLib.init();
```

7. Používat. Příklad použití metody write:

```
    javaLib.execute(() -> {
            return javaLib.write("/helloWorld", "/home/
                ↪ me/hello.cpp", false);
    });
```

Příloha B

Struktura priloženého CD

/
- Document
  - Latex ..... zdrojové kódy Latex
    - figures ..... obrázky dokumentu
- JavaLib ..... zdrojové kódy knihovny
  - jna ..... zkompilovaná knihovna MetaManagement