Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Server pro přístup k datovému úložišti

Plzeň, 2019

Filip Míka

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 2. května 2019

. . . .

Filip Míka

Abstract

The goal of this thesis is to create a server for access to the data repository. For metadata of saved data use the RDF, the OWL, and the SPARQL technologies. The theoretical part describes the technologies that will be used for data repository, comparison of selected data repositories and analysis of requirements on data repository . The practical part is the design and the implementation of the server. The conclusion of the thesis contains an evaluation of the results.

Abstrakt

Cílem této práce je vytvoření serveru pro přístup k datovému úložišti. Pro metadata uložených dat jsou využity technologie RDF, OWL a SPARQL. V teoretické části jsou popsané technologie, které budou použity pro datové úložiště, a porovnání vybraných úložišť včetně analýzy požadavků na datové úložiště. Praktická část sestává z návrhu a implementace serveru. Závěr práce obsahuje zhodnocení výsledků.

Poděkování

Rád bych poděkoval panu Ing. Petru Včelákovi zejména za trpělivost. Dále bych rád poděkoval za vedení bakalářské práce a za jeho cenné rady.

Obsah

1 Úvod 1

2 Teoretická část 2
2.1 HyperText Transfer Protocol . . . 2
2.1.1 Požadavek . . . 2
2.1.2 Odpověď . . . 3
2.2 Representational State Transfer . . . 4
2.3 Resource Description Framework . . . 5
2.3.1 Zdroj . . . 6
2.3.2 Prefix a namespace . . . 6
2.3.3 Literál . . . 7
2.3.4 RDF slovník . . . 7
2.4 RDF Schéma . . . 8
2.5 Web Ontology Language . . . 9
2.6 SPARQL . . . 9
2.7 TDB úložiště . . . 12
2.8 Fuseki . . . 12
2.9 Vlákna . . . 13

3 Existující řešení 14
3.1 Rozdělení podle poskytovaných služeb . . . 14
3.2 Rozdělení podle přístupu k datům . . . 15
3.3 Rozdělení podle vyhledávání obsahu . . . 16
3.4 Rozdělení podle použitých technologií . . . 16

4 Analýza požadavků na poskytování souborů klientům 17
4.1 Vlákna a vzájemné vyloučení vláken . . . 17
4.2 Rozdělení úloh řešení do prioritních tříd . . . 18
4.3 Chybové stavy . . . 19
4.4 Data serveru . . . 19
4.5 Data úložiště . . . 20
4.6 REST API serveru . . . 20

5 Návrh řešení 22
5.1 Názvosloví . . . 22
5.2 Výběr programovacího jazyka a frameworku . . . 22
5.3 Zpracování požadavku klienta . . . 23
5.4 Data . . . 23
5.5 Session . . . 24

5.6 Logy 25
5.7 Uživatelé 26
5.8 Omezení stahování 26
5.9 Výjimky 26
5.10 Vlákna 26

6 Implementace serveru **28**
6.1 Ontologie použitá pro vnitřní data serveru 28
6.2 Konfigurace 28
6.3 Zavaděč aplikace 29
6.4 Regulární výrazy 29
6.5 Vykonávání SPARQL dotazů 29
6.6 Relace 30
6.7 Příprava vykonání požadavku klienta 31
6.8 Logs 31
6.9 Omezení stahování 31
6.10 Servlety 32
6.11 Výjimky 34
6.12 Komprimace 35
6.13 Vyhledávání podle vlastností 35

7 Testování a měření **37**
7.1 Jednotkové testy 37
7.1.1 Data 37
7.1.2 Prostředí 37
7.1.3 Vyhodnocování testů 38
7.2 Testování API 38
7.3 Testování získávání dat z úložiště 39
7.3.1 Postup testu 39
7.3.2 Druhy testu 40
7.3.3 Vyhodnocení testu 40
7.4 Měření 41
7.5 Nasazení aplikace 42

8 Diskuze **43**
8.1 Zhodnocení výsledků 43
8.2 Porovnání výsledků s jinými autory 43
8.3 Výkon serveru 45
8.4 Testování 46
8.5 Možnosti rozšíření 47
8.6 Nedostatky a omezení aplikace 47

9 Závěr 49

Seznam zkratek 51

Literatura 52

Přílohy

A Tabulky HTTP protokolu 56

B Ukázky uložených dat v TDB úložišti aplikace 59

C Hledání podle vlastností - JSON data a SPARQL dotaz 61

D Implementované REST API 63

E Vybrané SPARQL dotazy 66

F Diagram zpracovaní požadavku na server 69

G Grafické zobrazení ontologie mikafil.owl 70

H Snímky obrazovky 71

I Grafy 75

J Konfigurační proměnné 80

K Struktura přiloženého CD-ROM 85

1 Úvod

S rozvojem počítačů, postupnou digitalizací lidského poznání a jeho života se neustále zvětšuje objem dat. S rostoucím objemem dat vzniká potřeba popsat je tak, aby jejich význam nebyl znám pouze jejich tvůrci, ale aby byl srozumitelný pro co největší počet lidí. Rozvoj přímo souvisí i s globální sítí internet, přičemž vzniká potřeba, aby data a jejich popis bylo možné sdílet po internetu a aby byl popis srozumitelný v globálním měřítku.

Díky zvýšenému používání mobilních telefonů a rozvoji mobilních sítí nemají lidé už potřebu ukládat data na osobní počítače. Proto mají stále častěji svá data uložená v datových úložištích, ke kterým mohou přistupovat z různých zařízení.

Cílem práce je vytvořit server pro přístup k datovému úložišti, přičemž data pro úložiště budou poskytnuta zadavatelem práce. Pro vypracování úlohy bude nezbytné najít vhodné řešení podle dodaných dat a vybranému řešení uzpůsobit výběr frameworku a ukládání dat. Dále bude nutné vybrat vhodný HTTP server pro obsluhu požadavků klientů.

Kromě přístupu k datům v úložišti bude muset server obsahovat správu uživatelů, administraci systému a uchovávání informací o přihlášených uživatelích. Dále bude vhodné implementovat vyhledávání dat úložiště podle metadat uložených v úložišti.

V diskuzi plánuji své řešení porovnat s vybranými datovými úložišti.

2 Teoretická část

2.1 HyperText Transfer Protocol

HyperText Transfer Protocol (HTTP) [1] je protokol aplikační vrstvy referenčního modelu ISO/OSI, který se používá pro přenos dat mezi klientem a serverem.

HTTP protokol vznikl v roce 1991 a první verze byla označena jako 0.9. Poslední verze HTTP 2.0 je z roku 2015 a je popsána v RFC 7540 [2]. Protokol verze 2.0 je zpětně kompatibilní s protokolem verze 1.1. Protokol verze 1.1 je popsán v RFC 2616 [3]. Dále se ve své práci budu zaměřovat pouze na protokol verze 1.1.

Komunikace mezi klientem a serverem je komunikace typu požadavek-odpověď. Klient posílá požadavek na server, server požadavek klienta zpracuje a výsledek požadavku odešle zpět klientovi. Součástí požadavků a odpovědí je hlavička. Iniciátorem komunikace je vždy klient.

Komunikace mezi klientem a serverem probíhá pomocí zpráv, které si navzájem klient a server posílají. Zpráva, kterou posílá klient serveru, se nazývá požadavek (request). Zpráva, kterou server posílá klientovi, se nazývá odpověď (response). Zprávy jsou formátované podle standardu RFC 822 [4].

Zpráva je vždy rozdělena do dvou částí. První částí je hlavička zprávy, která je povinná. Druhou částí zprávy jsou posílaná data mezi klientem a serverem. Pokud zpráva obsahuje data, musí být v hlavičce uveden typ a délka dat. Délka dat se udává v bytech a hodnota je udávána v parametru `Content-Length` hlavičky zprávy. Typ dat určuje parametr `Content-Type`.

Hlavička zprávy je textový řetězec, který obsahuje řádky. V prvním řádku jsou odesílány parametry HTTP protokolu (například metoda, návratový status a verze protokolu). Ostatní řádky představují jednotlivé parametry, které si klient se serverem posílají. Název parametru končí znakem „:“. Za tímto znakem následuje mezera a hodnota parametru. Hlavička je ukončena prázdným řádkem.

2.1.1 Požadavek

Požadavek (request) jsou data, která klient posílá serveru, přičemž klient očekává od serveru odpověď. První řádek hlavičky HTTP protokolu je tvořen ze tří částí: metody, URL a verze protokolu. Metody protokolu jsou uvedeny v tabulce 3 na straně 58.

Metoda může určovat, jak budou data serveru posílaná nebo co má server se zdrojem vykonat, popřípadě jaký typ odpovědi klient od serveru očekává.

GET Je nejpoužívanější metodou. Server na takový požadavek vrátí klientovi data podle URI v hlavičce požadavku. URI je zdroj, který může obsahovat data. Pokud zdroj neexistuje, vrátí server klientovi informaci o chybě.

HEAD Určuje, že server nevrátí klientovi žádná data. Odpovědí na požadavek klienta při použití metody HEAD je jen hlavička.

POST Při použití metody POST by měl server pohlížet na přijatá data od klienta jako na nový zdroj. To znamená, že zdroj, který je určený URI v požadavku, nemusí existovat před přijetím požadavku na server. POST metoda poskytuje jednotné rozhraní pro následující funkce:

- Odesílání bloku dat (například formuláře).
- Vytvoření zdrojů na straně serveru.

PUT Metoda slouží pro nahrazení zdroje. Předpokládá se, že na serveru již existují data popsaná pomocí URI v požadavku na server. Data budou nahrazena a URI se nezmění.

DELETE Tato metoda předpokládá, že server odstraní zdroj, který je identifikován URI v požadavku. Po zpracování požadavku serverem bude URI ukazovat na neexistující zdroj.

OPTIONS Metoda slouží pro určení komunikace mezi klientem a serverem. Klient se může dozvědět podrobnosti například o formátu dat nebo o komprimaci odpovědí podporované serverem.

Vybrané proměnné hlavičky HTTP požadavku jsou v tabulce 1 na straně 56.

2.1.2 Odpověď

Odpovědí (response) jsou data odeslaná serverem klientovi. Odpověď je tvořena ze tří částí stejně jako požadavek. Na rozdíl od požadavku je v hlavičce místo metody uváděn status zpracování požadavku na straně serveru. Ten určuje, jak server požadavek klienta zpracoval. Návratový status je vždy trojmístné přirozené číslo. První číslice určuje, jaký je druh návratového statusu odpovědi.

- **1xx** Odpověď je jen informační, pokračuje se ve zpracování požadavku.
- **2xx** Úspěch - požadavek byl správně doručen serveru. Server požadavek akceptoval a výsledek odešle v těle odpovědi.

- **3xx** Přesměrování - ještě další požadavek na server musí být vykonán, aby byl výsledek kompletní.
- **4xx** Chyba na straně klienta.
- **5xx** Chyba na straně serveru.

Tabulka 2 na straně 57 obsahuje vybrané proměnné, které se používají v hlavičce HTTP odpovědi.

2.2 Representational State Transfer

Representational State Transfer (dále jen REST) je architektura, kterou v roce 2000 popsal Roy Fielding ve své dizertační práci [5]. Jeho práce se věnuje obecně REST architektuře, která je realizovaná pomocí protokolu HTTP. Nejdůležitější částí je pátá kapitola, ve které jsou popsány vlastnosti REST. Pro REST jsou nejdůležitější tyto čtyři vlastnosti: Client-Server, Stateless, Cache a Layered System.

Client-Server Architektura klient-server je nejpoužívanější architekturou na internetu. Klientská část této architektury požaduje, aby služba byla provedena a odešle serveru požadavek prostřednictvím počítačové sítě. Server provede požadavek klienta a výsledek požadavku vrátí klientovi.

Statelessness (bezstavovost) Server si nepamatuje poslední požadavek klienta. Proto nemůže určit, jaký bude další stav, do kterého se klient může dostat, nebo jaké další zdroje by mohl klient požadovat.

Caching (ukládání dat v mezipaměti) Klient může uchovávat výsledky požadavků na server, protože se předpokládá, že požadavek na server bude proveden vždy se stejným výsledkem. Pokud zdroj představuje data, která se mění v souvislosti s provozem na serveru, měl by server informovat klienta, že se tento zdroj nemá ukládat do mezipaměti. Taková informace je součástí hlavičky odpovědi serveru.

Uniform Interface (jednotné rozhraní) zjednodušuje architekturu, což každému dílu umožňuje nezávisle se vyvíjet. Kompromisem je však to, že jednotné rozhraní odbourává efektivitu, neboť informace jsou předávány ve standardizované podobě a nikoliv takové, která je specifická potřebám aplikace.

Jednotné rozhraní má výhodu, že jej lze použít pro různá zařízení a je nezávislé na operačním systému. Stejné rozhraní lze použít například v aplikaci mobilního telefonu nebo osobního počítače.

Systém REST API [6] je tvořen nezávislými komponentami: servery, klienty, mezipamětí, proxy a tak dále. Tyto komponenty byly vytvořeny různými subjekty a komunikují mezi sebou předáváním dat prostřednictvím protokolu HTTP. Proto je nezbytné, aby se všichni dohodli na množině protokolové sémantiky tak, aby si jednotlivé komponenty rozuměly. Sémantika komunikace je realizovaná metodami protokolu HTTP.

Application Programming Interface (API) [7] je rozhraní pro programování aplikací, které umožňuje ostatním subjektům využívat data nebo služby pro vytváření vlastních aplikací. Příkladem API je GitHub API [35], které poskytuje vývojářům přístup k funkcím a službám na GitHub. Vývojáři mohou vyvíjet své vlastní aplikace a k tomu využít funkce a služeby GitHub, například spravování verzí svého software. Součástí API bývá podrobná dokumentace, která usnadňuje vývojářům pochopit funkce API.

REST API je architektura, která je implementovaná prostřednictvím HTTP protokolu. Platí pro ni, že zdroje jsou určeny URI, přičemž URI je stálé a nemění se, ani když se změní obsah zdroje. API poskytuje služby metodou klient/server. Vývojář prostřednictvím standardních metod protokolu HTTP využívá služeb serveru. Metody jsou popsané v tabulce 3 na straně 58.

Každý požadavek a odpověď musí obsahovat informaci o svém typu. Například pokud je odpověď ve formátu JSON, je nutné tuto informaci popsat v hlavičce HTTP protokolu parametrem `Content-type = application/json`.

2.3 Resource Description Framework

Resource Description Framework (dále jen RDF) [8] je doporučení, které umožňuje vědomosti o zdrojích reprezentovat grafem, kde jsou jednotlivé uzly grafu spojeny pojmenovanou orientovanou hranou. Tato hrana vyjadřuje vztah mezi dvěma uzly grafu. Listy grafu jsou zpravidla nejjednodušší datové typy reprezentované textovým řetězcem, který se nazývá literál.

První doporučení RDF bylo vytvořeno v roce 1999 [9]. Jeho hlavním cílem bylo umožnit jednoduché strojní zpracovávání obsahu webových stránek. Proto původní myšlenka využití RDF byla vkládání metadat o obsahu webových stránek přímo do kódu HTML. Prvním formátem pro RDF byl RDF/XML. Protože pro ukládání dat byl použit formát XML souborů, byly hned zpočátku použity jmenné prostory (namespace viz kapitola 2.3.2) z jazyka XML.

Datový model RDF je syntakticky neutrální způsob reprezentace výrazů RDF.

Základní datový model se skládá ze tří typů objektů:

zdroj Vše, co se popisuje pomocí RDF, se nazývá zdroj.

vlastnost Vlastnost je specifický aspekt, charakteristika, atribut nebo vztah používaný k popisu zdroje.

tvrzení Tvrzení (statement) je trojice zdroj (subjekt) + vlastnost + hodnota nebo zdroj (objekt).

$$\texttt{subjekt} \xrightarrow{\texttt{vlastnost}} \texttt{objekt}$$

RDF je doporučení, jak ukládat informace o zdrojích. Doporučení jsou obecná a nepopisují, jak konkrétně mají být data uložena. Obecnost specifikace umožňuje ukládat data v různých formátech, například RDF/XML, Turtle nebo N-Triples.

2.3.1 Zdroj

Resource[11] znamená zdroj, který obsahuje nějaká data. Zdrojem může být prakticky cokoliv, například stránka na internetu, emailová adresa nebo kniha v půjčovně.

Uniform Resource Identifier (URI) [12] je uniformní identifikátor zdroje. Tímto identifikátorem je textový řetězec, který je v ASCII kódování. RDF od verze 1.1 používá Internationalized Resource Identifier (IRI) [13], což je mezinárodní popis zdroje, který dovoluje používat znaky v UTF-8 kódování. URI je podmnožinou IRI, protože ASCII kódování je podmnožinou UTF-8 kódování.

V této práci se bude zdrojem myslet jen zdroj, jenž je dostupný prostřednictvím internetu. Takový zdroj je jednoznačně určený pomocí URI, které je používáno jako identifikátor zdroje.

2.3.2 Prefix a namespace

Namespace [16] (jmenný prostor) je část URI, která je společná pro určitou množinu URI. Tato část URI může být nahrazena jedinečnou zkratkou, která se nazývá prefix (předpona). Pro zápis prefixu a jmenného prostoru se používá QName (Qualified Name - kvalifikovaná jména).

Například pro URI : `http://www.w3.org/1999/02/22-rdf-syntax-ns#type` je jmenným prostorem: `http://www.w3.org/1999/02/22-rdf-syntax-ns#`. V daném případě je konvencí používat prefix „rdf“. Zápis příkladu je `rdf:type` a má stejný význam jako celé URI příkladu.

Takto zkrácená URI jsou dobře čitelná pro člověka a používáním prefixů lze zmenšit objem dat. Některé formáty pro ukládání dat používají prefixy (například RDF/XML a Turtle). Formát N-Triples ukládá data bez použití prefixů.

2.3.3 Literál

Literál [10] je přímý zápis určité hodnoty, například řetězce znaků, čísla nebo datumu. Literál má dvě hodnoty: textový řetězec a přesnou hodnotu (celočíselná hodnota, datum, logická hodnota atd.). Literál je vždy uložen jako textový řetězec.

Například pro celočíselnou hodnotu lze zapsat: "25"^^xsd:decimal. Tímto způsobem zapsaná hodnota jasně popisuje, jakého datového typu je textový řetězec.

Existují dva druhy literálu:

plain - Je výchozí literál, kterým je textový řetězec. Tento druh literálu lze určit různě pro libovolný jazyk.

```
"text"
"word"@en
"slovo"@cs
```

typed - Je literál, který je určen hodnotou a zároveň typem. Například pro celočíselnou hodnotu 25 lze zapsat :
`"25"^^xsd:decimal.`

Literály, které nemají určený typ a jsou zapsány pouze textovým řetězcem, lze převést na požadovaný typ přiřazovacím příkazem BIND ve SPARQL dotazu. Zmíněné přetypování ukazuji v příkladu 2 na straně 12.

2.3.4 RDF slovník

RDF je jazyk a stejně jako každý jazyk potřebuje slovník, aby měl vyjadřovací schopnost. V RDF jsou slova reprezentovaná URI. RDF slovník nemá velkou vyjadřovací schopnost, protože pouze popisuje, že něco existuje.

RDF poskytuje základní slovník [21], který umožňuje popsat zdroje. Pro RDF slovník se používá prefix `rdf`.

rdf:type Určuje, jakého typu je zdroj. Hodnota vlastnosti rdf:type zdroje musí být třídy rdfs:Class nebo její podtřídy. Takovým způsobem popsaný zdroj se nazývá individuál.

rdf:Property Třída je nadtřídou všech zdrojů, které jsou vlastnostmi. To znamená, že každá vlastnost musí být potomkem třídy rdf:Property, která je instancí třídy rdfs:Class [22].

2.4 RDF Schéma

RDF Schéma (dále jen RDFS) [22] rozšiřuje slovník a vlastnosti RDF. Zatímco RDF popisuje jen vlastnosti zdrojů, RDFS popisuje, jak jsou vlastnosti organizovány. Popisuje vztahy mezi zdroji a hierarchii vztahů.

rdfs:Resource Resource je nejdůležitějším objektem RDF. Všechny třídy v RDF jsou podtřídami rdf:Resource. Všechny zdroje jsou potomci třídy rdfs:Resource.

rdfs:Class Class je definice třídy. Vše v RDF je odvozeno od třídy rdf:Class a zároveň rdfs:Class je sama sobě typem.

rdfs:SubClassOf Definuje, jaké třídy je podtřída. Definuje vztah mezi třídami zdroje. Vlastnost rdfs:subClassOf je tranzitivní. To znamená, že pokud je třída B podtřídou A a pokud třída C je podtřídou B, musí platit, že třída C je podtřídou A.

rdfs:Literal je podtřídou rdfs:Resource a představuje objekt, který je v listech grafu RDF a je hodnotou určité vlastnosti zdroje. Příkladem rdfs:Literal je textový řetězec.

rdfs:subPropertyOf Vlastnost se používá pro zdroje, které jsou vlastnostmi a které jsou tedy typu `rdfs:Property`.Nový zdroj může rozšiřovat vlastnosti od nadřazeného zdroje. Například [15] vlastnost *hasSon* dědí od vlastnosti *hasChild*. Pokud má zdroj vlastnost *hasSon*, tak má zároveň i vlastnost *hasChild*, aniž by byla taková vlastnost *hasChild* přímo vyjádřena v popisu zdroje. Všechny vlastnosti `rdfs:range` a `rdfs:domain` jsou děděny.

rdfs:domain je instancí rdfs:Property a určuje, jaké třídy mohou být subjektem této vlastnosti.

rdfs:range je instancí rdfs:Property a určuje, jaké třídy mohou být objektem této vlastnosti.

rdfs:label Vlastnost popisuje název zdroje, aby byl srozumitelný pro člověka.

2.5 Web Ontology Language

Ontologie [17] je popis reálného objektu a vztahů mezi objekty. Vytvořený popis je velmi pragmatický: „Pro umělou inteligenci platí, že to, co existuje, může být reprezentováno.“ [17]

W. Borst [17] popsal v roce 1997 ontologii jako „Formální specifikace sdílené koncepce“. Zmíněná definice popisuje ontologii jako popis objektů. Tento popis je objektivní a nevyjadřuje individuální nebo subjektivní popis existujících objektů.

Ontologie [20] v kontextu počítačových systémů definuje množinu reprezentačních primitiv, pomocí kterých lze popsat reálné existující objekty. Definice reprezentačních primitiv zahrnují informace o jejich významu, omezeních a vztahů mezi objekty.

Web Ontology Language 2 (dále jen OWL) [18] je ontologickým jazykem sémantického webu s formálně definovaným významem. OWL poskytuje třídy, vlastnosti, jednotlivce a hodnoty dat, které jsou uloženy jako dokumenty sémantického webu. Ontologie OWL může být použita společně s informacemi napsanými v RDF a samotné ontologie OWL se primárně poskytují jako dokumenty RDF.

OWL [19] je jazyk sémantického webu navržený tak, aby reprezentoval různorodé a komplexní znalosti o věcech, skupinách věcí a vztazích mezi nimi. OWL je jazyk, který umožňuje pomocí výpočetní logiky ověřit konzistenci těchto znalostí.

OWL poskytuje tři profily [19]:

EL Je navržen tak, aby zohledňoval velké zdravotnické ontologie. Společné charakteristiky takových ontologií zahrnují složité strukturální popisy, velké množství tříd a velkého množství dat.

QL Profil QL je realizován za pomoci standardních relačních databází. Je vhodný pro reprezentaci databázových schémat a jejich integraci pomocí přepisovacích dotazů. Je zaměřen na zpracovávání velkého množství dat.

RL Vhodná implementace založená na pravidlech OWL RL v rámci sémantiky založené na RDF lze použít s libovolnými grafy RDF. V důsledku toho je OWL RL ideální pro obohacení dat RDF.

2.6 SPARQL

SPARQL [23] je dotazovací jazyk, který umožňuje vyhledávat data v RDF. Jazyk byl navržen konsorciem W3C a jeho aktuální verze je 1.1.

Ve SPARQL se dají definovat proměnné, které vždy začínají znakem „?“.

Dotaz SPARQL se dělí na tři části. V první části jsou definovány prefixy, které umožňují zpřehlednění zápisu SPARQL dotazu.

V druhé části se definuje, co se má hledat. Definicí můžou být příkazy SELECT, ASK, DESCRIBE nebo CONSTRUCT. V této části jsou určené proměnné, které budou součástí odpovědi na dotaz. Pokud chceme vypsat všechny nalezené hodnoty proměnných, používá se zástupný znak „*“.

SELECT Příkaz SELECT vrací hodnoty proměnných, které byly použity v dotazu. Za příkazem SELECT může následovat výčet proměnných nebo znak „*“. Za příkaz SELECT lze použít modifikátor DISTINCT[24], který z výsledku odstraní duplicitní záznamy.

Ukázka nejjednoduššího příkazu SELECT se nachází v příkladu 1.

ASK Dotaz ASK zjistí, zda zadaná vyhledávací kritéria mají alespoň jedno řešení. Pokud ano, návratová hodnota je `true`. V opačném případě návratovou hodnotou je `false`.

DESCRIBE Dotaz lze použít pro nalezení všech vlastností určitého subjektu, proto jsou výsledkem všechny RDF trojice pro daný subjekt a podgraf původního grafu.

CONSTRUCT Při hledání pomocí CONSTRUCT je návratovou hodnotou RDF graf. V grafu jsou jednotlivé zdroje popsány pouze vlastnostmi, které jsou definované podle proměnných ve vyhledávacím příkazu. Tento graf není stejný jako RDF graf, ve kterém bylo hledáno.

Ve třetí části se udávají vyhledávací kritéria, která jsou napsaná v příkazu WHERE. Příkaz FILTER nastavuje omezovací kritéria pro vyhledávání.

Pro vyhledávání se používá vzor (pattern), který je tvořen RDF trojicí. Jakoukoliv položku z trojice lze nahradit proměnnou. Tímto způsobem definovaná proměnná může nabývat všech hodnot, které budou vyhovovat ostatním položkám ve trojici vzoru. Nejjednodušší příkaz SPARQL, který vrátí všechny trojice v datasetu, je uveden v příkladu 1.

```
SELECT *
 WHERE {
   ?subjekt ?vlastnost ?objekt.
 }
```

Příklad 1: SPARQL příkaz SELECT, kterému vyhovují všechny trojice

Agregáty (agregates) [25] jsou výrazy, které umožňují aplikovat jednoduché funkce na nalezené hodnoty jedné proměnné. Výsledkem tohoto výrazu je hodnota, kterou lze přidělit k proměnné klíčovým slovem `AS`.

Agregáty se používají tam, kde se požaduje výpočet přes skupinu řešení. Příkladem může být nalezení maximální hodnoty proměnné.

Předdefinované agregáty jsou :

COUNT Agregát vrátí počet výskytu hodnot proměnné ve výsledku dotazu.

SUM Agregát vrátí součin všech výskytů hodnot proměnné.

MIN Agregát vrátí minimální hodnotu proměnné.

MAX Agregát vrátí maximální hodnotu proměnné.

AVG Agregát vrátí aritmetický průměr hodnot proměnné.

V příkladu 2 je ukázka použití funkcí `SUM` a `COUNT`. V daném případě je nutné použít i klauzuli BIND, která převede proměnnou ?fileSize, jejíž hodnotou je netypový literál, na celočíselnou hodnotu. Příkaz najde hodnoty vlastnosti `dcm:Exposure` menší než 100. Příkaz BIND nastaví hodnotu proměnných `?fSize` podle hodnoty vlastnosti `?fileSize`, přičemž provede přetypování z netypového literálu na literál typu `xsd:long`. Modifikátor DISTINCT odstraní duplicitní řádky z výsledku hledání. Agregát `COUNT` vrátí počet výskytu proměnné `?subj` a přiřadí výsledek proměnné `?count`. Agregát `SUM` sečte hodnoty proměnné `?fSize` a výsledek přiřadí k proměnné `?sum`.

```
PREFIX dcm: <http://mre.zcu.cz/ontology/dcm.owl# >

SELECT DISTINCT (SUM(?fSize) AS ?sum) (COUNT(?subj) AS ?count)

WHERE {
?subj <http://www.semanticdesktop.org/ontologies/2007/03/22/nfo#fileSize>?fileSize.
?subj dcm:Exposure ?exposure.
BIND( xsd:long(?fileSize) AS ?fSize)
FILTER( ( ?exposure < 100 ))
}
```

Příklad 2: SPARQL agregátní funkce a příkaz BIND

2.7 TDB úložiště

TDB [29] je komponenta Jena frameworku. TDB je datové úložiště pro ukládání dat ve trojicích subjekt, vlastnost, objekt.

Protože data uložená v TDB úložišti jsou na disku serveru, nazývá se způsob uložení perzistentní. Přístup k datům je možný prostřednictvím rozhraní Apache Jena API.

Přístup k datům uložených v TDB datovém úložišti musí být pouze z jedné JVM. Pokud by k datům přistupovalo více JVM, mohlo by dojít k poškození dat uložených v TDB úložišti.

2.8 Fuseki

`Fuseki` [30] je server, který vykonává dotazy SPARQL pro datová úložiště. Fuseki umožňuje jednotný přístup k datovému úložišti. Pro přístup k datům prostřednictvím Fuseki serveru používají klienti REST metody.

Fuseki může být spuštěn na pozadí aplikace jako vložený (embedded) [31] server. K takovým způsobem spuštěnému serveru může přistupovat pouze aplikace, která ho spustila. Tím je zabezpečena ochrana dat proti přístupům z ostatních aplikací.

2.9 Vlákna

Vlákna (threads) [33] se používají pro úlohy, které lze řešit souběžně. To znamená, že řešenou úlohu lze rozdělit do několika menších nezávislých úloh. Pro paralelní zpracování úloh se používají v programovacím jazyce Java vlákna.

Výhodou používání vláken je urychlení vykonávání úloh zejména na víceprocesorových počítačích.

Třída `java.util.concurrent.ThreadPoolExecutor` poskytuje funkce pro spouštění vláken. Thread pool si lze představit jako množinu vláken. Pokud chceme spustit úlohu v novém vlákně, `ThreadPoolExecutor` vybere vlákno, které nic nevykonavá a kterému přiřadí úlohu.

3 Existující řešení

V současné době existuje velké množství datových úložišť, které často obsahují multimediální obsah a jsou určeny pro širokou veřejnost. Hlavním cílem takových datových úložišť je poskytnout co nejvíce souborů pro co nejvíce uživatelů.

Pro srovnání jsem vybral následující datová úložiště:

OneDrive [37] je cloudové datové úložiště společnosti Microsoft. Jeho předností je integrace do prohlížeče souborů přímo v operačním systému Windows.

Uložto [38] je datové úložiště umožňující uživatelům serveru veřejně sdílet jakýkoliv obsah a zaměřuje se na poskytování dat libovolného typu a obsahu.

Google Drive [39] je cloudové řešení společnosti Google a umožňuje zdarma uložit až 15 GB dat. Lze jej používat z emailové služby Gmail.

Facebook [40] je sociální síť, která umožňuje svým uživatelům sdílet data s ostatními uživateli. Facebook umožňuje pro přístup k datům využít `Graph API`, což je REST API pro ukládání a prohlížení dat.

Stream [41] je datové úložiště pro šíření multimediálního obsahu v podobě videí a je provozováno společností Seznam a.s. Zaměřuje se na šíření editovaných videí vytvořených společností Seznam a.s., přičemž ostatní uživatelé nemají možnost přidávat vlastní videa.

YouTube [42] je datové úložiště, které umožňuje svým uživatelům zveřejňovat videa. Uživatelé zde publikují různorodý audiovizuální obsah.

GitHub [43] umožňuje vývojářům programů zveřejňovat své projekty. Součástí tohoto serveru je datové úložiště, kde jsou ukládány zdrojové soubory programů. Pro přístup k uloženým datům lze použít `REST API v3` [35].

Fedora [34] je open source datové úložiště, které používá pro ukládání metadat RDF trojice. Zkratka FEDORA znamená flexibilní rozšiřitelná architektura úložiště digitálních objektů (Flexible Extensible Digital Object Repository Architecture). Jejím hlavním cílem je propojení zdrojů pomocí Linked Data.

3.1 Rozdělení podle poskytovaných služeb

Hlavním kitériem, jak rozdělit datová úložiště je podle služeb, které server nabízí. Server může uživatelům úložiště nabízet pouze přístup k datům anebo může nabízet i další služby. V druhém případě není datové úložiště primárním zaměřením takového serveru.

primární Servery, které mají jako primární cíl datové úložiště, poskytují jen přístup k datům. Ostatní služby jsou pouze pro správu dat a uživatelů. Takové servery jsou například Uložto, Fedora nebo Google Drive. Do stejné kategorie lze řadit i servery, které poskytují pouze multimediální obsah jako například Stream nebo YouTube.

sekundární Jsou servery, které sice obsahují datové úložiště, ale úložiště není jejich hlavním zaměřením. Mezi datové úložiště se sekundárním zaměřením patří například GitHub, jehož hlavním cílem je poskytování služeb vývojářům programů, nebo Facebook, jehož hlavním cílem je sdílení multimediálního obsahu uživatelů.

3.2 Rozdělení podle přístupu k datům

Další způsob rozdělení je podle způsobu poskytování dat uživatelům, kteří nejsou vlastníci dat a data do datového úložiště neuložili. Datová úložiště nelze striktně rozdělit do následujících třech množin. Rozdělení je možné určit pouze podle hlavního či výchozího poskytování dat.

veřejná data Datové úložiště Uložto poskytuje data, která jsou nahraná uživateli serveru. Data jsou poskytovaná veřejně, takže každý k nim má přístup. Do stejné kategorie patří i většina multimediálních datových úložišť jako například Stream nebo YouTube.

Datovým úložištěm, kde jsou data považovaná za veřejná, lze nazývat i sociální sítě. Zde jsou data zpravidla poskytovaná podle jednotlivých uživatelů, nežli podle obsahu, jak je to u ostatních datových úložišť. Příkladem může být sociální síť Facebook.

soukromá data Na druhou stranu existují soukromá datová úložiště jako například Google Drive a OneDrive, kde jsou data dostupná jen uživateli, který je na server uložil. Uživatel může dát souhlas se zveřejněním obsahu, a pak může sdílet obsah s jinými uživateli.

Do stejné kategorie můžeme zařadit i všechna cloudová řešení, kde není žádoucí, aby data byla sdílená s ostatními uživateli, například OneDrive.

kombinace Existují datová úložiště, která nabízí obě výše uvedené možnosti. Příkladem může být GitHub, kde jsou data poskytovaná veřejně, ale uživatel má možnost vytvářet i soukromé repozitáře.

3.3 Rozdělení podle vyhledávání obsahu

Většina datových úložišť umožňuje vyhledávání dat, aby uživatelé mohli vyhledávat obsah podle zvolených kriterií.

jednoduché vyhledávání Většina datových úložišť řeší vyhledávání dat pomocí jednoho vstupního pole pro textový řetězec. Zpravidla probíhá vyhledávání jen podle názvu souboru jako například v úložišti OneDrive. GitHub vyhledává mimo jiné i podle názvu repozitářů. Stream a YouTube též umožňují vyhledávání jen podle názvu. Uložto navíc přidává možnost výběru typu dokumentu.

rozšířené vyhledávání Úložiště Google Drive nabízí nejpropracovanější vyhledávání. Vyhledávání je možné v konkrétním adresáři, například podle data, jména, nahraného souboru, obsahu anebo sdílení. Navíc se nabízí možnost vyhledávání v datech ostatních uživatelů, ale v takovém případě musí uživatel označit obsah jako veřejný.

bez vyhledávání Fedora nenabízí vyhledávání v metadatech. Vyhledávání obsahu neumožňuje také Facebook, protože umožňuje vyhledávání podle uživatelů serveru.

3.4 Rozdělení podle použitých technologií

Datová úložiště lze dělit i podle použitých technologií (například podle použitých frameworků nebo podle způsobu uchovávání metadat). Všechna datová úložiště nelze ale tímto způsobem porovnat, protože u komerčních řešení není známo, jaké technologie používají. Každé datové úložiště musí uchovávat metadata o uloženém obsahu. Pro ukládání se můžou používat relační databáze, popřípadě jiné databázové systémy.

Jediný projekt, který je OpenSource, je Fedora. Ta ukládá metadata v RDF trojicích.

4 Analýza požadavků na poskytování souborů klientům

Nejdůležitější funkcí serveru bude poskytování souborů z úložiště klientům. Komunikace mezi klientem a serverem bude pomocí HTTP protokolu.

Požadavky klientů na server jsou vstupními daty, které bude muset server zpracovat. Podle jednotlivých požadavků bude muset server vytvořit odpověď, kterou poté předá klientovi. Při požadavku klienta na zdroj bude odpovědí klientovi jeden komprimovaný soubor, který bude obsahovat jeden, nebo více souborů. Počet souborů bude záviset na typu zdroje. V příloze F na straně 69 je detailně uveden diagram zpracování požadavku klienta na data z úložiště.

Server musí vždy poslat odpověď klientovi, takže bude reagovat i na neplatné požadavky klientů. Více o chybových stavech se zmiňuji v kapitole 4.3.

Dalším požadavkem na server bude správa informací o uživatelích, kteří budou mít přístup k datům v úložišti. Protože je HTTP protokol bezstavový, vzniká požadavek na server uchovávat informace o přihlášených uživatelích.

Server bude řešit i omezení přístupu k souborům, které se určí podle poskytnutých dat uživateli nebo podle počtu poskytnutých soborů. Server bude ukládat informace o počtu souborů, které uživateli poskytl, ve vhodném datovém úložišti.

Zabezpečení serveru bude realizováno prostřednictvím přihlášení uživatele ke službám serveru. Všichni uživatelé nebudou mít přístup ke všem částem serveru, proto bude nutné vyřešit přístupová práva k jednotlivým částem serveru. Zadání neřeší zabezpečení komunikace mezi klientem a serverem.

Dalším požadavkem na server je možnost vzdálenému přístupu k metadatům. Metadata nemusí být nutně uložena na stejném počítači, na kterém je spuštěn server.

4.1 Vlákna a vzájemné vyloučení vláken

Důležitým požadavkem na server je nutnost zpracovávat požadavky klientů ve více vláknech a tím využít výkonu víceprocesorových počítačů.

Využitím více vláken při zpracovaní požadavků klientů vzniká potřeba vzájemného vyloučení přístupu ke společným datům objektů v jednotlivých vláknech, zejména při zápisu dat a odesílání dat klientovi v odpovědi serveru.

U komprimace souboru pro odpověď serveru se nepředpokládá nutnost vzájemného vyloučení vláken. V tomto případě bude docházet k přistupu k souborům pouze

pro čtení.

Dalším požadavkem využití vláken je zpracování většího objemu dat ve vlastním vlákně, které nebude totožné s vláknem serveru vykonávající požadavek klienta. I u těchto vláken bude nutnost zabezpečit společná data a objekty pro přístup z různých vláken.

4.2 Rozdělení úloh řešení do prioritních tříd

Zadání bakalářské práce je vhodné rozdělit do dílčích úloh, které lze rozdělit do prioritních tříd. K daným úloham lze přidat další, které přímo neřeší zadání bakalářské práce, ale přesto je vhodné tyto úlohy vyřešit, aby výsledný server správně pracoval a byl uživatelsky přívětivý.

Vysoká priorita představuje úlohy, bez kterých není server plně funkční a nesplňuje zadání bakalářské práce. Do této třídy lze zařadit následující úlohy:

- Poskytování dat z úložiště klientům
- Konfigurace serveru
- Zabezpečení přístupu k datům v úložišti a k jednotlivých částem serveru
- Testování serveru
- Vytváření a odstraňování uživatelů
- Správa dat, která bude server uchovávat o uživatelích a o přístupu k datům v úložišti

Ve středně prioritní třídě jsou úlohy, které umožní snažší práci se serverem, ale přímo neřeší zadání bakalářské práce. Jejich smyslem je usnadnění práce se serverem například při administraci serveru.

Patří sem:

- Vzdálený přístup k metadatům
- Udělování a odebírání práv uživatelům

V nízké prioritní třídě jsou úlohy, které nemají přímo vliv na práci serveru nebo jsou rozšířením některé z vyšších prioritních tříd.

Sem můžeme zařadit:

- Vyhledávání v datovém úložišti
- Konfigurace serveru bez nutnosti jeho restartování

- Formulář pro konfiguraci serveru

4.3 Chybové stavy

Chybové stavy budou odeslány klientům podle standardu HTTP protokolu.

Možné chybové stavy:

400 - Bad Request - špatný nebo neplatný požadavek. Chybový stav nastane, pokud klient odešle na server neplatný požadavek.

401 - Unauthorized - nepřihlášen. Vznikne, pokud není ověřena identita uživatele.

403 - Forbidden - zakázáno. Nastane, pokud uživatel nemá dostatečná práva pro přístup k danému zdroji.

404 - Not Found - nenalezeno. Požadovaný zdroj neexistuje.

413 - Request Entity Too Large - požadovaná entita je příliš velká. Tento chybový stav nastane, pokud by výsledek stahování překročil některé omezení stahování dat nebo pokud je již vyčerpáno některé omezení pro stahování dat. Výsledný požadavek může překročit omezení v počtu souborů nebo celkové velikosti dat.

429 - Too Many Requests - příliš mnoho požadavků. Nastane, pokud uživatel serveru zadá příliš mnoho požadavků.

500 - Internal Server Error - vnitřní chyba serveru, která bude klientovi vrácena v odpovědi při výjimkách, které narušily vykonání požadavku. Server by měl vracet tento chybový stav jen ve zvlášť výjimečných situacích.

503 - Service Unavailable - služba je nedostupná. Chybový stav vznikne, pokud bude zpracováváno velké množství úloh v odděleném vlákně.

4.4 Data serveru

Server bude uchovávat dva druhy dat:

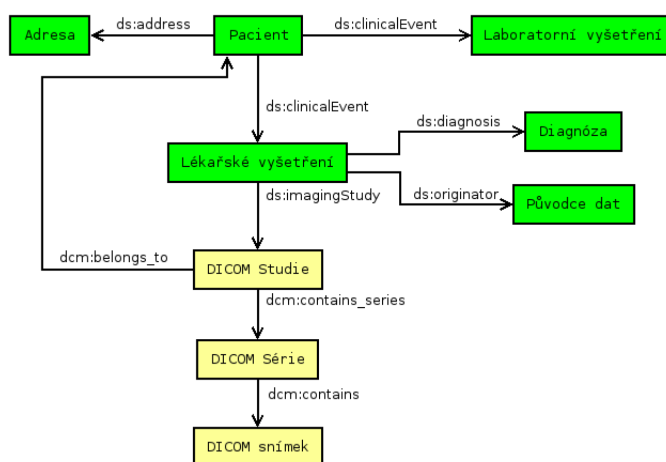
Stálá data o uživatelích serveru budou ukládána do vhodného datového úložiště a musí být dostupná i po restartu serveru. Patří sem i informace o přístupech uživatelů k datům v datovém úložišti. Tyto záznamy bude muset server uchovávat aktuální a bude je muset strojně zpracovávat, aby mohl rozhodnout o případném omezení stahování.

Dočasná data , kterými jsou informace o přihlášených uživatelích, mohou být ukládána do paměti počítače, protože při restartu serveru nejsou potřeba. Zároveň se tím docílí větší rychlosti při přístupu k datům.

4.5 Data úložiště

Metadata popisují jednotlivé snímky, které se nacházejí v datovém úložišti a které budou poskytovány klientům. Metadata dále popisují Série. Vztah mezi sérií a snímkem je 1:N. V metadatech může být Studie, vztah mezi Studií a Sérií je 1:N. Vztah mezi Studií, Sérií a snímkem je zobrazen na obrázku 1.

Snímky budou uloženy v adresáři, ze kterého je bude server poskytovat klientům.



Obrázek 1: Schéma vybraných tříd a vlastností v systému MRE [36]

4.6 REST API serveru

REST API bude realizováno pomocí HTTP protokolu.

Požadavky na server se musí rozdělit do několika části:

zdroje Nejdůležitější úkolem REST API je přístup k datům uložených v úložišti pomocí zdrojů. U těchto požadavků bude použita pouze metoda GET. Odpovědí na požadavky bude komprimovaný soubor obsahující jeden, nebo více snímků. Počet snímků bude záviset na typu zdroje.

uživatelé Další částí budou požadavky, pomocí kterých bude prováděno přihlášení uživatele prostřednictvím metody POST. K poskytování informací o přihláše-

ném uživateli a o omezení stahování pro daného uživatele se využije metody GET.

administrace Přístup k API administrace serveru bude umožněn pouze uživatelům s právy administrátora.

Dalším nárokem na administraci bude správa uživatelů. Ta bude muset umožnit vytvoření a odstranění uživatele nebo změnu práv jednotlivých uživatelů.

5 Návrh řešení

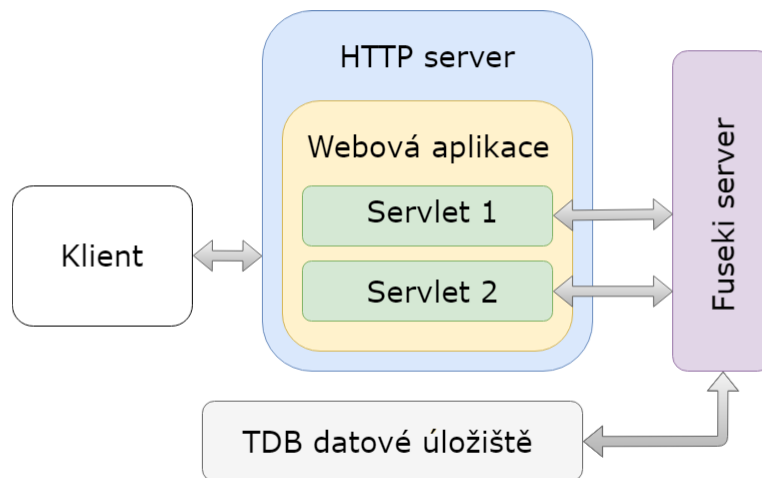
5.1 Názvosloví

Řešení zadaného úkolu bude rozděleno do několika komponent podle obrázku 2, a tak je vhodné ujasnit názvosloví, které bude dále používáno.

server - Pojmem `server` budeme dále rozumět HTTP server, který přijímá požadavky klientů.

aplikace - Pojem `aplikace` budu dále používat pro aplikaci, která běží v prostředí HTTP serveru a která je řešením zadané úlohy.

fuseki - Pojem `fuseki` určuje server, který vykonavá SPARQL příkazy v úložišti TDB.



Obrázek 2: Grafické zobrazení architektury.

5.2 Výběr programovacího jazyka a frameworku

Pro implementaci aplikace jsem zvolil programovací jazyk Java, protože se jedná o jeden z nejpoužívanějších programovacích jazyků [27] a protože mi ho doporučil vedoucí mé bakalářské práce. S volbou programovacího jazyka přímo souvisí i výběr frameworku Apache Jena [26].

Pro metadata datového úložiště jsem zvolil TDB datové úložiště, které je volitelnou částí Apache Jena frameworku. Moje volba přímo souvisela s požadavky na server a poskytnutými daty.

5.3 Zpracování požadavku klienta

Po přijetí požadavku od klienta musí server předat požadavek aplikaci, která ho zpracuje a prostřednictvím serveru vrátí výsledek klientovi. Takto navržené řešení umožní oddělit část obsluhy komunikace klient-server od vykonání samotného požadavku klienta aplikací.

Pro řešení úlohy jsem použil server Apache Tomcat [28], který umožňuje zpracování požadavků klientů pomocí servletů napsaných v programovacím jazyce Java. Apache Tomcat jsem použil pro jeho jednoduchou konfiguraci a způsob přidání nové aplikace do serveru.

Synchronizace společných objektů bude realizovaná pomocí standardního monitoru [47] jazyka Java, který je definován klíčovým slovem `synchronized`. Synchronizace bude použita i pro zabezpečení výstupu aplikace. Zabezpečení objektů použitých při zpracování požadavků klientů bude vykonávat třída `Preparation`. Každé vlákno bude vytvářet jedinečnou instanci třídy `Preparation`, přičemž ostatní vlákna o této instanci nebudou vědět. Tím se omezí používaní monitorů na co nejmenší počet.

5.4 Data

Data aplikace lze rozdělit do dvou částí. V první části jsou systémová data aplikace a ve druhé se nacházejí metadata o poskytovaných souborech a soubory, které budou poskytovány klientům.

Při implementaci bude nutné udržovat tyto dvě skupiny dat od sebe. Proto budou použity dvě TDB úložiště. Jedno úložiště bude obsahovat stálá systémová data a druhé úložiště bude obsahovat metadata o poskytovaných souborech klientům.

Systémová data Systémová data aplikace, například logy stahování a informace o uživatelích, budou ukládaná do TDB úložiště. Data popisuji v ontologii `./owl/mikafil.owl` a budou dostupná prostřednictvím fuseki. Více se o nich zmiňuji v kapitolách 5.6 a 5.7. Ukázka uložených dat je uvedena v příloze B na straně 59.

Dalšími daty aplikace jsou dočasná data, která se ukládají do paměti počítače jako například informace o přihlášených uživatelích a která jsou uložena v kolekcích Javy (například HashMap, ArrayList). Tento způsob jsem zvolil kvůli větší rychlosti při přístupu k datům, která se nacházejí v paměti počítače. Více o daném tématu zmiňuji v kapitole 6.6.

Data úložiště Daty uložiště jsou soubory, které budou poskytovány klientům. Aplikace musí mít k daným souborům přístup pro čtení. Metadata o uložených souborech jsou uložena v TDB úložišti.

5.5 Session

Session (relace) umožňuje bezstavovému HTTP serveru ukládat informace o přihlášených uživatelích, aniž by uživatelé museli posílat uživatelské jméno a heslo při každém požadavku na server.

Přihlášení uživatele k serveru bude realizováno podobně, jako je tomu u datového úložiště GitHub. Po odeslání požadavku na server s přihlašovacími údaji je generován autorizační token, což je zobrazeno na příkladu 3.

```
curl -u mikafilip -d '{"scopes": ["repo", "user"], "note": "getting-started"}' \
https://api.github.com/authorizations
Enter host password for user 'mikafilip':

{
"id": 195100251,
"url": "https://api.github.com/authorizations/195100251",
"app": {
"name": "getting-started",
"url": "https://developer.github.com/v3/oauth_authorizations/",
"client_id": "00000000000000000000"
},
"token": "ea1b853d358fdc94b152371fbf4c95dedd13c568",
"hashed_token": "c1352b56064dc2112e12bdb33ff0875aafc4867fede1c92a6e32038e88becf83",
"token_last_eight": "dd13c568",
"note": "getting-started",
"noteq_url": null,
"created_at": "2018-06-10T06:41:57Z",
"updated_at": "2018-06-10T06:41:57Z",
...
}
```

Příklad 3: Generování autorizačního tokenu v GitHub.

Pro další požadavky na server GitHub lze používat autorizační token, kterým je hodnota proměnné `token` v odpovědi serveru v příkladu 3. Ukázka dalšího požadavku pomocí tokenu je v příkladu 4.

```
curl -H "Authorization: token ea1b853d358fdc94b152371fbf4c95dedd13c568"\
https://api.github.com/user

{
"login": "MikaFilip",
"id": 26443865,
"node_id": "MDQ6VXNlcjI2NDQzODY1",
"avatar_url": "https://avatars3.githubusercontent.com/u/26443865?v=4",
"gravatar_id": "",
"url": "https://api.github.com/users/MikaFilip",
"html_url": "https://github.com/MikaFilip",
...
}
```

Příklad 4: GitHub - požadavek na informace o přihlášeném uživateli.

V mém řešení bude při komunikaci mezi klientem a serverem používaná proměnná `ses_id`. Hodnotu proměnné lze poslat serveru několika způsoby:

Cookies `ses_id` lze ukládat na straně klienta v Cookies, které posílají proměnnou `ses_id` v hlavičce požadavku.

URL `ses_id` lze poslat serveru přímo v URI požadavku. V takovém případě je možné poslat `ses_id` buď jako poslední hodnotu v cestě URI, nebo jako parametr URI s názvem `ses_id`.

JSON Hodnotu proměnné `ses_id` lze poslat serveru v datech JSON formátu, pak musí být `ses_id` v JsonObject v první úrovni vnoření. Ukázka odeslání `ses_id` v JSON datech je zobrazena na příkladu 5.

```
{"ses-id":"ba45610b74d54e92a583f6051c900cb7",
"command":[],
...
}
```

Příklad 5: Ukázka odeslání session ID v JSON datech

5.6 Logy

Aplikace bude ukládat informace o aktivitách jednotlivých uživatelů do takzvaných logů, které budou uloženy do TDB úložiště. Pro správnou funkci aplikace je nutné mít k dispozici logy v aktuální podobě.

Datovou strukturu logu popisuji v ontologii `./owl/mikafil.owl`. Příklad uloženého logu v TDB uvádím v příkladu 8 na straně 59.

5.7 Uživatelé

Datová struktura uživatele se nachází v ontologii `./owl/mikafil.owl`. Uživatelé jsou ukládáni do systémového TDB úložiště a budou rozlišováni podle emailové adresy. Ukázka uložení jednoho uživatele v TDB databázi je v příkladu číslo 9 na straně 60.

Uživatelé budou mít různá přístupová práva. Pro zabezpečení hesla bude použita hash funkce. Toto řešení představuje zabezpečení dat, díky kterému nemůže nikdo číst hesla ze záložních souborů.

Vytváření uživatelů bude povoleno pouze uživatelům s přistupovými právy administrátora.

5.8 Omezení stahování

Omezení stahování bude pracovat se systémovými daty v TDB úložišti. Třída `Limits` vybere prostřednictvím SPARQL dotazu konkrétní logy za poslední jednotku času (den, týden, měsíc) a výsledkem dotazu bude objem dat a počet souborů, které byly uživateli poskytnuty.

Třída `Limits` rozhoduje podle konfiguračního souboru a podle poskytnutých dat uživateli, zda bude uživateli umožněno stažení dalších dat z datového úložiště. Třída určí, podle které podmínky bylo zamítnuto další stahování dat.

Omezení stahování bude konfigurovatelné a bude určeno podle objemu dat nebo podle počtu souborů.

5.9 Výjimky

Vzniklé výjimky se postupně vyhazují (throw) až do `BaseServlet`, který jediný klientovi výjimku zobrazí. Všechny výjimky jsou potomci jedné třídy, a tak nemusí `BaseServlet` rozlišovat, jaká konkrétní výjimka vznikla. Zároveň `BaseServlet` ví, jaký návratový status má klientovi odeslat, protože daná hodnota je nastavena při vzniku výjimky a odvíjí se od jejího druhu.

5.10 Vlákna

Pro sestavení odpovědi s objemem dat větším než hodnota konfigurační vlastnosti `directly_download_limit` jsem zvolil vypracování požadavku klienta v ji-

ném vlákně, protože v jedné chvíli je nutné udělat dvě úlohy. První úlohou je informovat uživatele o sestavování odpovědi na pozadí aplikace. Druhou úlohou je samotné sestavování odpovědi.

Výhodou daného řešení je, že vlákna poběží, dokud neukončí vykonávání svého kódu. Vlákna dokončí svůj kód, i když se HTTP server zastaví.

Další výhodou je, že nebude nutné měnit v nastavení HTTP serveru čas potřebný na dokončení odpovědi na větší hodnotu než standardních 30 sekund.

Pro vlákna je vytvořena jedna instance třídy `ThreadPoolExecutor`, což umožňuje rychlejší spouštění vláken v operačním systému, protože odpadá režie při zavádění vlákna před jeho spuštěním.

6 Implementace serveru

6.1 Ontologie použitá pro vnitřní data serveru

V příloze G na straně 70 je grafické zobrazení ontologie, které jsem vytvořil nástrojem WebVOWL [49] , přičemž ontologii jsem vytvořil nástrojem Protégé [48].

Ontologie obsahuje následující třídy:

mikafilUser Popisuje uživatele a má jen datové vlastnosti. Například vlastnost `mbox` je emailová adresa uživatele, která je zároveň rozlišujícím identifikátorem jedntlivých uživatelů.

result Třída `result` popisuje uložené odpovědi serveru. Obsahuje objektovou vlastnost `belongs_to`, jenž popisuje, který uživatel vytvořil požadavek na server. Dále obsahuje objektovou vlastnost `contains_log`, jejíž hodnotou je URI `log`.

log Třída popisuje informace o poskytnutých datech z úložiště. Stejně jako třída `result` obsahuje objektovou vlastnost `belongs_to`. Mezi její další vlastnosti patří datové typy, například o začátku a konci vytváření odpovědi klientovi. Obsahuje i statistické informace o vypracovávání odpovědi, například jak dlouho probíhalo dotazování na server a jak dlouho probíhala komprimace odpovědi.

6.2 Konfigurace

Konfigurace je uložena ve dvou konfiguračních souborech, které se nacházejí v pracovním adresáři serveru.

aplikace Konfiguraci spravuje třída `ServerConf`, která dědí své vlastnosti od třídy `BaseConf` a je v balíčku `cz.zcu.students.mikafil.conf`. Instance třídy `ServerConf` je implementovaná jako jedináček a poskytuje metody pro přímý přístup k nejpoužívanějším vlastnostem v konfiguraci aplikace.

Konfiguraci lze měnit prostřednictvím HTML formuláře. Změny v konfiguraci se ihned projeví v aplikaci. Dané řešení jsem zvolil zejména kvůli možnosti testovaní funkčnosti aplikace bez nutnosti restartování.

server Pro správnou funkčnost aplikace je nutné nakonfigurovat i server. Zejména je důležité nastavení maximální velikosti paměti, kterou server poskytne aplikaci. Tato konfigurace je zmíněna v uživatelské příručce.

Maximální velikost paměti je nutné nastavit na 2 GB, jinak může nastat situace, kdy aplikaci dojde paměť. Aplikace nebude schopná například vkládat metadata do datového úložiště z dataset `urna.nt`.

Konfigurační proměnné uvádím v příloze J na straně 80.

6.3 Zavaděč aplikace

Třída `MikafilLoader` implementuje rozhraní `ServletContextListener` a provádí zavádění aplikace. Volání funkcí pro zavádění aplikace provádí server automaticky podle konfiguračního souboru `./WEB-INT/web.xml`. V třídě `MikafilLoader` se vytváří nová instance třídy `Session`, dále provede spuštění serveru Fuseki.

V následujícím kroku provede `MikafilLoader` kontrolu zámků v TDB úložištích. Pokud zámky existují, tak se je pokusí odstranit. Tento úkon je potřebný udělat, protože pokud by server ukončil svou práci nestandardně, zůstanou zámky v adresářích TDB úložiště. Pokud jsou datová úložiště zamčená jinou aplikací, spuštění aplikace proběhne bez Fuseki serveru, takže aplikace nemusí být funkční.

Posledním úkolem `MikafilLoader` je načtení konfiguračních souborů. Pokud konfigurační soubory neexistují v pracovním adresáři aplikace, `MikafilLoader` provede kopírování konfiguračních souborů s výchozím nastavením. Tímto je zaručeno, že i po prvním spuštění aplikace budou vytvořeny potřebné konfigurační soubory.

6.4 Regulární výrazy

Pro práci s textovými řetězci jsou použity regulární výrazy, které zjednodušují práci při vyhledávaní v textových řetězcích.

Třída `Patterns` obsahuje pouze statické objekty, kterými jsou regulární výrazy. V třídě jsou obsaženy pouze regulární výrazy, u kterých se předpokládá, že budou použity několikrát v aplikaci. Další regulární výrazy jsou používány v kódu aplikace.

6.5 Vykonávání SPARQL dotazů

Třída `Executor` zajišťuje rozhraní mezi aplikací a fuseki, což umožňuje jednotný přístup k datům aplikace. Třída obsahuje pouze statické metody, které mají návratové typy podle druhu vykonávaného SPARQL dotazu.

Řeší i synchronizaci zápisu dat do datového úložiště aplikace. Všechny příkazy, které mohou měnit obsah databáze, obsahují zabezpečení kritické sekce pomocí klíčového slova `synchronized`.

6.6 Relace

`Session` je implementováno uloženými daty v paměti počítače a používá základní datové struktury jazyka Java (například ArrayList, HashMap).

`Session` pro identifikaci přihlášeného uživatele používá textový řetězec, který je generován třídou UUID [32]. V `Session` se o uživateli ukládají následující informace: zdroj(URI), identifikátor (náhodný textový řetězec), email (přihlašovací jméno), čas posledního požadavku (celočíselná hodnota) a přístupová práva.

Na příkladu 6 je zobrazeno, jak bude generován identifikační textový řetězec.

Klient bude při další komunikaci se serverem používat identifikační řetězec, který je dán hodnotou proměnné `ses_id`. Odeslání hodnoty v požadavku na server je zobrazeno na příkladu 5 na straně 25.

```
curl -d '{"email":"mikafil@students.zcu.cz", "password":"1234"}' \
-H "Content-Type: application/json" -H "Accept: application/json"\
-X POST http://10.0.2.2:8084/mikafil/user/login

{"ses-id":"4e6c4dda8aed46dcbdfc707e2f8fc071"}
```

Příklad 6: Přihlášení do implementované aplikace.

`Session` umožňuje odstraňovat nepřihlášené uživatele bez nutnosti zásahu administrátora. Pro tyto účely obsahuje frontu používaných identifikátorů. Účelem fronty je pravidelně odstraňovat záznamy uživatelů, kteří již nejsou přihlášeni, a přitom nezpomalovat vykonávání požadavku procházením všech známých identifikátorů.

Při každém požadavku klienta na server je vybrán první záznam z fronty. Poté je porovnáno, zda poslední požadavek klienta na server nebyl přijat dříve než aktuální čas mínus hodnota proměnné `auth_timeout` v konfiguraci. Pokud byl požadavek proveden dříve, proběhne odstranění záznamu. V opačném případě je vybraný identifikátor zařazen na konec fronty. Tím je zaručeno, že kontrola přihlášených uživatelů probíhá rovnoměrně a zároveň při každém požadavku bude vyzkoušena pouze jedna položka v `Session`.

Třída `Session` řídí omezení počtů požadavků za určité časové období. Pokud je požadavků více, než kolik činí hodnota vlastnosti `too_many_request_count` v kon-

figuraci za časové období dané vlastností too_many_request_timeout, server vrátí klientovi chybový stav 429 Too Many Requests. Implementace je řešena spojitým seznamem, který má maximální velikost too_many_request_count + 1 a ve kterém jsou ukládány časy požadavků . Chybový stav nastane, pokud rozdíl mezi prvním a posledním prvkem v seznamu je menší než hodnota vlastnosti too_many_request_timeout.

6.7 Příprava vykonání požadavku klienta

Preparation vykoná přípravu proměnných pro obsloužení požadavků klientů a provádí zápis dat do odpovědi serveru. Servlety, které jsou potomky BaseServletu, využívají instanci třídy Preparation.

Pro každý požadavek klienta na server je vytvořena instance třídy Preparation, což umožňuje, aby byly jednotlivé požadavky na server zpracovávany tzv. „thread safety". To znamená, aby server mohl vykonávat požadavky klientů ve vláknech a nedocházelo k narušení dat, která budou vrácena klientovi v odpovědi serveru.

Preparation obsahuje i společná data, například konfiguraci aplikace. Zápis do konfigurace je možný pouze administrátorům, proto vzájemné vyloučení přístupu k těmto datům není řešeno. Dalšími společnými daty je instance třídy Session, která je v aplikaci pouze jedna. Proto musí být Session zabezpečena monitorem především při zápisu dat.

6.8 Logs

Při každém požadavku na server vzniká instance třídy Logs, do které lze během celého průběhu zpracování požadavku ukládat informace. Pokud byly přidány nějaké informace, proběhne uložení logu do úložiště aplikace.

Informace ukládané aplikací do Logs jsou zobrazeny v příkladu 8 na straně 59 a odpovídají ontologii mikafil.owl.

6.9 Omezení stahování

Třída Limits rozhoduje o podmínkách omezení stahování pro uživatele. Konfigurace počtu stahování lze nastavit v konfiguračním souboru server.properties v pracovním adresáři aplikace.

Třída nejdříve získá data z TDB úložiště aplikace o stahování uživatele za poslední období. SPARQL dotaz používaný pro zjištění předchozích stahování je uveden v příkladu 14 na straně 68.

Následně třída určí, kolik dat nebo souborů může být ještě uživateli poskytnuto. Zároveň stanoví, která podmínka je rozhodující pro omezení. Ta je daná minimální hodnotou ze skupiny možných poskytnutých dat, která mohou být poskytnuta podle stažení za měsíc, nebo týden či den.

6.10 Servlety

Konfigurace servletů se nachází v souboru `./WEB-INT/web.xml`. HTTP server načte soubor při zavádění aplikace a podle obsahu souboru vybírá, který ze servletů bude obsluhovat požadavek klienta. Výběr servletu je závislý na URI požadavku. Servlety jsou implementovány pro jednotlivé URI podle přílohy D na stránce 63. V příloze jsou uvedeny pouze relativní cesty.

V souboru `./WEB-INT/web.xml` je definováno, jaké akce mají být vykonány při zavádění aplikace. Zavádění aplikace proběhne při nasazení (deploy) nebo po restartu serveru. Pro implementaci zavádění systému jsem zvolil `ContextListener`.

InstallServlet umožňuje vykonat prvotní instalaci stránce `./install.jsp`. Pro nastavení je použit HTML formulář viz obrázek 3 na straně 71. Při instalaci je prováděno nastavení portu fuseki, nastavení adresáře s poskytovanými daty úložiště, převod metadat ze souboru do TDB úložiště a vytvoření jednoho uživatele, který bude mít práva administrátora.

InstallServlet není potomkem `BaseServletu`. Jeho úkolem je připravit aplikaci k prvotnímu spuštění, takže ještě nemůže využívat všech funkcí aplikace.

BaseServlet , který je potomkem třídy `HttpServlet`, je základním servletem aplikace. V servletu jsem implementoval metody, které jsou společné pro ostatní servlety používané v aplikaci. Tomuto servletu není přidělená žádná URI adresa a všechny dále zmiňované servlety jsou potomci `BaseServletu`.

Jeho hlavní úlohou je poskytování jednotného rozhraní mezi servlety a ostatními komponentami aplikace a tím i zjednodušení kódu ostatních servletů.

Všechny servlety se nacházejí v jednom balíčku a od `BaseServlet` se neočekává, že by mohl mít potomky z jiných balíčků, a proto jsou všechny metody definovány jako `protected`. Tím se zaručí, že ostatní komponenty nebudou mít přístup k metodám servletů.

AdminServlet obsluhuje požadavky na administraci aplikace a jeho hlavní využití je umožnit vzdálenou správu aplikace. Obsluhuje požadavky na konfiguraci, správu datového úložiště a správu uživatelů.

K přístupu k funkcím servletu musí mít uživatel nastavenou vlastnost `isAdmin` na `true`. Přístupová práva se kontrolují v metodě `doInit(Preparation)`. Pokud uživatel nebude přihlášen, vznikne výjimka `UnauthorizedException`. Další výjimkou je `ForbiddenException`, která nastane v případě, pokud je přihlášen uživatel, který má nastavenou vlastnost `isAdmin` na `false`.

`AdminServlet` je rozdělen do několika částí:

services Obsluhuje služby aplikace.

config Obsluhuje požadavky na konfiguraci aplikace.

users Zde se nachází administrace uživatelů.

sessions Zobrazuje otevřené relace uživatelů.

backup V této části se nacházejí požadavky na uložení zálohy ze serveru a obnova dat ze zálohy.

logs Výpis z Log4J souboru. Zde může administrátor sledovat logy aplikace.

expired Tady může administrátor smazat výsledky požadavků, které byly vykonávány nepřímo a jejichž platnost již vypršela.

ResourceServlet je servlet, který klientům poskytuje data z datového úložiště. Pro přístup musí být uživatel přihlášen. Pokud není uživatel přihlášen, vznikne výjimka `UnauthorizedException`.

Diagram zpracování požadavku pro `ResourceServlet` se nachází na obrázku v příloze F na straně 69.

Odpovědí klientovi na požadavek bude zkomprimovaný soubor, který bude mít název podle identifikátoru zdroje, jenž byl předán v URI požadavku klienta. Zkomprimovaný soubor bude obsahovat jeden nebo více souborů, přičemž počet souborů bude záviset na typu zdroje.

Postup zpracování požadavku klienta:

ověření uživatele V prvním kroku aplikace určí, zda je uživatel přihlášen.

omezení Aplikace zjistí, kolik souborů a dat uživatel již stáhl a poté podle konfiguračního souboru určí, zda uživatel může stahovat další data.

typ zdroje Servlet určí, jaký je typ zdroje, pro který má být vykonán požadavek. Podle typu zdroje bude volaná konkrétní funkce. Typy zdrojů jsou celkem tři: *DICOM snímek*, *DICOM Série* a *DICOM Studie* viz obrázek 1 na straně 20.

načtení zdroje Aplikace dotazem SPARQL zjistí počet souborů a celkovou velikost dat, které budou uživateli poskytnuty. Následujícím dotazu SPARQL aplikace zjistí seznam souborů a velikosti jednotlivých souborů.

omezení Dále aplikace určí, zda může uživatel stahovat data. Pokud by uživatel stahováním dat překročil některé omezení, nebudou data uživateli poskytnuta.

určení typu odpovědi Nakonec aplikace určí, jakým způsobem budou data klientovi předána. Data mohou být předána přímo jako odpověď serveru nebo nepřímo prostřednictvím výzvy k vyzvednutí dat. Konkrétní řešení popisuji v kapitole 6.12.

SearchServlet vykonává obsluhu požadavků klientů při vyhledávání podle vlastností. Servlet obsluhuje AJAX požadavky, které vytváří dynamická stránka JSP. Více popisuji v kapitole 6.13.

UserServlet obsluhuje požadavky běžných uživatelů. Pro přístup k těmto funkcím musí být uživatel přihlášen. Pokud není uživatel přihlášen, servlet propaguje výjimku `UnauthorizedException`. Výjimka nevznikne, pokud probíhá přihlášení uživatele do aplikace.

Servlet poskytuje uživatelům informace o posledních stahování. Uživatel může spravovat odpovědi serveru, které byly uloženy do souboru.

6.11 Výjimky

V balíčku `cz.zcu.students.mikafil.exception` jsou definovány všechny výjimky. Každá výjimka, která vznikne v průběhu zpracovávání požadavku, má jako parametr kód návratové hodnoty HTTP protokolu, který bude vrácen klientovi v HTTP hlavičce.

`MikafilException` je hlavní třídou výjimek, která je potomkem třídy `Exception`. Všechny dále popsané výjimky dědí vlastnosti od třídy `MikafilException`.

Níže uvádím výčet výjimek, které mohou nastat s větší pravděpodobností:

BadRequest 400 neplatný požadavek. Server nerozumí přijatému požadavku.

UnauthorizedException 401 je chybou požadavku. Nastane, pokud se uživatel, který není přihlášen, pokouší přistoupit ke zdroji, ke kterému může přistoupit jen přihlášený uživatel.

ForbiddenException 403 je chybou požadavku. Chyba nastane, pokud přihlášený uživatel nemá nastavenou proměnnou `isAdmin` na `true`.

UnknownResourceException 404 je chybou požadavku a nastane, pokud se klient pokusí přistoupit ke zdroji, který je neplatný anebo byl odstraněn.

LimitException 413 je chybou požadavku. Uživatel požaduje data, která překročí limitu pro stahování dat, nebo již vyčerpal povolené množství stahování dat.

FileReadException 500 je vnitřní chybou, která nastane, pokud nebude možné číst z existujícího souboru.

6.12 Komprimace

Třída `Zip` komprimuje soubory do jednoho archivu, který bude vrácen klientovi v odpovědi, popřípadě bude uložen na disk jako výsledek požadavku.

Metoda `append(String, String, Resource)` umožňuje komprimovat soubor do `ZipStream`. Prvním parametrem je cesta k souboru, který bude vkládán. Druhým parametrem této funkce je identifikátor, který je převzat z URI subjektu z datasetu a který je použit jako název souboru.

`ZipStream` používá k zápisu dat `OutputStream`, do kterého zapisuje zkomprimované soubory. Konkrétní `OutputStream` závisí na způsobu vykonávání požadavku. Pokud součet velikostí nalezených souborů je menší nebo roven než hodnota `directly_download_limit`, bude použit `ByteArrayOutputStream` jako vyrovnávací paměť. V opačném případě bude zápis dat probíhat do souboru a bude použit `FileOutputStream`.

6.13 Vyhledávání podle vlastností

Vyhledávání podle vlastností v metadatech je realizováno pomocí dynamické stránky JSP. Zde lze vyhledávat soubory podle vybraných vlastností. Výsledkem vyhledávání je množina odkazů na zdroje. Po kliknutí na konkrétní odkaz může uživatel stáhnout jeden soubor. Pro daný způsob vyhledávání jsou používány asynchronní

požadavky AJAX na server. Kvůli nezávislosti na prohlížeči je používaná standardní knihovna JQuery.

Na straně klienta dochází k vytváření dat pro vyhledávání zadáváním textového řetězce do formuláře. Data jsou odeslána na server v JSON formátu. Na straně serveru dochází k vygenerování SPARQL dotazu, jehož řešením je počet nalezených souborů a celková velikost dat zdrojů. Průběžně je uživatel informován o počtu nalezených souborů, které vyhovují zadaným kritériím vyhledávání.

Pro vyhledávání v metadatech jsem vybral tři vlastnosti, které mění svoje hodnoty: `exposure`, `slice_location` a `slice_thickness`. Odesílání dat a generování SPARQL dotazu je vytvářeno dynamicky. To znamená, že lze měnit vlastnosti v HTML formuláři. Nové vlastnosti budou použity při dalším vyhledávání, aniž by se musel měnit způsob odesílání a generování SPARQL dotazů.

Výsledkem vyhledávání v odpovědi serveru je vždy jen jeden soubor, protože vlastnosti vyhledávání jsou vybrané z metadat typu `dcm:CT_Image`.

Příklad odeslání dat klientem serveru a vygenerování SPARQL dotazu je uveden v příloze C na straně 61.

7 Testování a měření

Pro testování aplikace jsem vytvořil tři druhy automatizovaných testů. Dva druhy jsou testy funkčnosti aplikace: první testy jsou jednotkové testy, které se provedou před kompilací aplikace a druhými testy ověřuji funkčnost již nasazené aplikace. Třetím testem je datový test poskytování zdrojů, který lze spustit až po spuštění serveru.

7.1 Jednotkové testy

Pro jednotkové testy jsem zvolil framework `JUnit` verze 4. Volba verze frameworku `JUnit` byla provedena podle používané verze Javy.

Pro jednotkové testy je nutné mít volný port 3341 na počítači, kde je prováděna kompilace.

Jednotkové testy jsou navrženy tak, aby proběhly v prostředí, které je podobné `Apache Tomcat` serveru, a přístupu k datům prostřednictví `Fuseki` serveru.

7.1.1 Data

Data, která jsou určená pro testování aplikace pomocí `JUnit` se nacházejí v adresáři *./src/test/data/*. V tomto adresáři jsou systémová data aplikace uložena v souboru `system_data.nt`.

Dalšími testovacími daty jsou data úložiště, která obsahují jednu Studii s jednou Sérii obsahující 30 snímků. Na začátku testování budou testovací data uložena do TDB úložiště a přístup k nim bude prostřednictvím `Fuseki` serveru. Toto řešení poskytuje při testování stejný přístup k datům, jaký bude mít aplikace při normálním běhu.

7.1.2 Prostředí

Pro testování aplikace jsem vytvořil prostředí, které je podobné prostředí `Apache Tomcat` serveru, který bude poskytovat funkce pro aplikaci. Dané řešení jsem zvolil jako alternativu k vytváření mock objektů frameworku `Mockito`. Výhodou je znovupoužitelnost pomocných testovacích objektů, čímž je dosaženo lepší přehlednosti testovacího kódu.

Před spuštěním testů se odstraní data, která mohla zůstat v testovacím adresáři po předchozím spuštění testů.

Prostředí pro testování využívá rozhraní, která jsou používaná `Apache Tomcat` serverem. Nejdůležitější třídou je třída `Env`, která ve své statické části vytvoří prostředí pro testování. To znamená, že při novém spuštění testování vytvoří třída potřebné adresáře, zkopíruje konfigurační soubory, vytvoří databáze TDB a v posledním kroku provede zavedení aplikace (deploy). Zavedení proběhne stejným způsobem, jakým vykonává zavedení aplikace `Apache Tomcat`.

7.1.3 Vyhodnocování testů

`JUnit` testy jsou použity pro generování odpovědí klientovi. Zde můžou být volány konkrétní metody v servletech a následně vyhodnocovány odpovědi, které jsou pouze textové řetězce bez HTTP hlavičky.

Příklad použití testu nalezneme ve třídě `UserServletTest`, kde je generovaná odpověd serveru pro požadavek *./user/info*. Poté je odpověď převedena na `JsonObject` a testuje se, zda je ve výsledné odpovědi klíč *user* v JSON datech.

7.2 Testování API

Smyslem testu je vyzkoušet aplikaci s reálnými daty a v reálném prostředí HTTP serveru.

Pro testování API jsem vytvořil webovou stránku, která obsahuj skriptovací jazyk JavaScript, který se používá pro vytváření dynamických HTML stránek. Způsob testování jsem zvolil zejména z důvodu snadného vytváření skriptu a jeho použití na stránce HTML. Další výhodou je využití vývojářských nástrojů, které jsou součástí prohlížečů HTML stránek. Tím odpadá nutnost implementovat obsluhu spojení mezi klientem a serverem a zároveň prohlížeč umožňuje používat konzoli pro výstup skriptu. Velkou výhodou je i grafické rozhraní pro sledování jednotlivých požadavků, včetně možnosti prohlížení hlaviček požadavků na server a odpovědí serveru.

Stránka pro testování správné funkce API je `./validation.jsp`. Takto řešené testování vlastního API umožňuje provádět test kdykoliv po nasazení aplikace. To znamená, že je možné otestovat funkčnost aplikace při změně konfigurace. Test pouze ověřuje, zda aplikace vyhovuje stanoveným kritériím, takže účelem testu není odhalit, jaká chyba vznikla v aplikaci, ale pouze zda aplikace vykonává správně odpovědi.

Testování je prováděno tak, že postupně jsou klientem posílány na server požadavky. Poté proběhne vyhodnocení odpovědí serveru, kde jsou testovány návratové kódy HTTP protokolu nebo přijatá data odpovědi serveru.

Test obsahuje i testování vzdáleného přístupu k metadatům úložiště a vzdálený přístup k systémovým datům aplikace. Testy jsem zkoušel na jednom počítači, kde `Fuseki` server spouštěla stejná aplikace, ale na jiném operačním systému. Testování jsem prováděl pomocí `VirtualBox`.

Na obrázku 4 na stránce 72 je zobrazen snímek obrazovky při úspěšném provedení testu.

7.3 Testování získávání dat z úložiště

Primární funkcí serveru je poskytovat data z datového úložiště, proto jsou nezbytným testem serveru požadavky klientů na server pro získaní dat z úložiště. Smyslem testu je získávání dat ze serveru a ověření, zda jsou data úplná a zda je server schopen paralelně zpracovat více požadavků klientů.

Před spuštěním testování je vhodné nastavit `directly_download_limit` na hodnotu 2147483647 v konfiguraci, což je maximální velikost datového typu `Integer` v Javě. Testy jsem prováděl v prohlížeči `Chrome`.

7.3.1 Postup testu

Test se provádí tak, že stránka `tester.jsp` nejdříve odešle na server požadavek, jehož odpovědí je seznam všech dostupných zdrojů, ve kterém jsou zdroje roztříděny podle typu: snímky, série a studie. Dále uživatel vybere, jaké typy zdrojů budou použity v testování: zda požadavky na sérii nebo na snímek. Konkrétní snímek nebo série je pak vybírán náhodně. Požadavek na studii není předmětem testu, protože výsledek požadavku nebude server předávat přímo v odpovědi klientovi.

Odpovědi serveru jsou ukládány na straně klienta v datovém typu `Blob`[46], což je objekt, který je podobný souboru v klasickém souborovém systému. Výhodou daného řešení je, že uživatel, který provádí testování, bude mít možnost data uložit na svůj počítač.

Poté bude vytvořen požadavek na server pro získání informací o všech požadavcích, které server vykonal v časovém intervalu testu. Seznam obsahuje všechny zpracovávané požadavky klientů na server.

Nakonec proběhne vyhodnocení získaných dat. Na obrázku 5 na straně 73 a na obrázku 6 na následující stránce, jsou zobrazeny snímky obrazovky provedeného testu získaní dat z úložiště.

7.3.2 Druhy testu

Testovací skript může získat seznam zdrojů pro testování dvěma způsoby:

statický seznam zdrojů Seznam zdrojů je předem vybrán z datasetu `urna.nt` a je možné jej použít při porovnání výkonu serveru na různých počítačích. Zdroje jsou uloženy ve dvojrozměrném poli.

dynamický seznam zdrojů Seznam zdrojů je získán při načítání stránky, který se změní při změně datasetu. Výhodou tohoto přístupu je vždy aktuální seznam zdrojů.

Testy s dynamickým obsahem lze provádět ve třech režimech:

synchronní V testu je generován uživatelem volitelný počet požadavků, které jsou vykonávány postupně za sebou. Takže po obdržení odpovědi serveru na požadavek klienta bude vykonáván další požadavek na server.

synchronní ve více vláknech Test je prováděn stejným způsobem jako synchronní test s tím rozdílem, že požadavky na server jsou spouštěny ve více vláknech.

asynchronní V testu může uživatel spustit určitý počet požadavků na server za určitý čas, přičemž čas začátku vykonávání požadavku je stanoven náhodně. Cílem testu není vyvolat nejvyšší zátěž serveru, ale simulovat nepředvídatelný přístup klientů na server.

7.3.3 Vyhodnocení testu

Vyhodnocování testu se provádí po ukončení všech testovacích požadavků na server, aby nedocházelo k ovlivnění měření z hlediska času potřebného na získání všech odpovědí serveru.

Testuje se, zda server je schopen vykonávat souběžně více než jeden požadavek. Kladné vyhodnocení testu nastane v případě, že existují alespoň dva požadavky, které server vykonával souběžně.

Dále je testováno, zda výsledky odpovědí serveru jsou dekomprimovatelné. Pro testování výsledného zkomprimovaného souboru jsem použil Javascript knihovnu

`zip.js` [45]. Smyslem testu je, zda při zpracovávání požadavků klientů v paralelních vláknech nedochází k přístupům k proměnným objektů z různých vláken serveru. Pokud při pokusu o dekomprimaci přijatých dat na straně klienta dojde k chybě, výsledek testu je negativní.

7.4 Měření

Měření výkonu serveru se nachází na stránkách `./tester.jsp` a `./threads.jsp`. Měřenou veličinou je čas potřebný na vykonání požadavku, přičemž nižší hodnota znamená větší výkon serveru.

Na stránce `./threads.jsp` je měřeno vykonávání 100 požadavků na snímek v závislosti na počtu zpracovávaných požadavků klientů, které jsou vykonávané ve zvláštním vlákně. Pro požadavky, které budou zpracovávány v odděleném vlákně jsou použity 3 zdroje typu série. Zdroje musí obsahovat alespoň 300 snímků, aby nedošlo k ukončení vypracování požadavku na danou sérii před vykonáním měřených 100 požadavků na snímek. Výstupem měření jsou CVS data.

Na stránce `./tester.jsp` se měří požadavky na server, které získávají data z úložiště stejným způsobem jako testovací požadavky v kapitole 7.3.

celková doba měření Je čas naměřený od prvního odeslaného požadavku do posledního přijmutí odpovědi.

čekání na server - medián Je prostřední hodnotou všech časů, po které klient čekal na obsluhu serverem. Jedná se o rozdíl hodnot začátku požadavku na straně klienta a začátku požadavku na straně serveru.

čekání na server - max Vyjadřuje maximální čas rozdílu začátku požadavku na straně serveru a klienta.

příjem dat - medián Jedná se o prostřední hodnotu všech rozdílů mezi ukončením požadavku na straně klienta a na straně serveru.

příjem dat - max Představuje nejvyšší hodnotu všech rozdílů mezi ukončením požadavku na straně klienta a na straně serveru.

čas dotazu klient - medián Je prostřední hodnotou všech požadavků, která je určena rozdílem mezi ukončením požadavku na straně klienta a počátkem požadavku na straně klienta.

čas dotazu klient - max Jedná se o maximální hodnotu všech požadavků klienta na server, která je určena rozdílem mezi ukončením požadavku na straně klienta a počátkem požadavku na straně klienta.

čas dotazu server - medián Vyjadřuje prostřední hodnotu všech požadavků, která je určena rozdílem mezi ukončením požadavku na straně serveru a počátkem požadavku na straně serveru.

čas dotazu server - max Představuje maximální hodnotu všech požadavků, která je určena rozdílem mezi ukončením požadavku na straně serveru a počátkem požadavku na straně serveru.

7.5 Nasazení aplikace

Pro nasazení (deploy) aplikace jsem použil Apache Tomcat server verze 8.0.27 v systému Windows a Apache Tomcat server verze 8.0.32 v systému Linux, který byl nainstalován jako virtuální operační systém.

Nasazení proběhlo na počítači s procesorem Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz, 2904 Mhz, jádra: 2, logické procesory: 4 , 8GB RAM a SSD diskem 256 GB.

8 Diskuze

8.1 Zhodnocení výsledků

Vytvořil jsem server, který umožňuje klientům přístup k datovému úložišti. Implementoval omezení stahování uživatelů a omezení počtu požadavků na server od jednotlivých uživatelů.

Správa serveru je řešena dynamickou HTML stránkou, která využívá implementovaných REST API požadavků. Mým záměrem bylo vytvořit server, který by poskytoval veškeré funkce pouze pomocí REST API.

8.2 Porovnání výsledků s jinými autory

V sekci 3 uvádím existující řešení přístupu k datovému úložišti. Porovnávat mé řešení s těmito datovými úložišti není snadné, protože úložiště neposkytují stejné funkce jako mé řešení, například komprimaci všech požadavků klientů na data v úložišti do zip souboru. Komprimace požadavků klienta byla zadaná přímo vedoucím bakalářské práce. Dále mé řešení souvisí s dodanými metadaty. Jediné datové úložiště, které je podobné mému řešení je Fedora, která umožňuje k datům svého úložiště ukládat metadata v RDF trojicích.

Pro porovnání přihlášení uživatele na server jsem vybral API GitHub [35], které srovnávám při přihlašování příkazem `curl` s mým řešením. Na GitHubu je autorizace uživatele realizovaná pomocí autorizačního tokenu, který je jedinečný pro každého přihlášeného uživatele. Toto řešení jsem vybral, protože je jednodušší než přihlašování uživatelů v datovém úložišti `Google Drive`, které používá k autorizaci požadavky popsané v *Using OAuth 2.0 to Access Google APIs* [50].

Pouze datová uložiště Google Drive a GitHub umožňují stahovat více dat v komprimovaném souboru. Server GitHub podporuje stažení celeho repozitáře uživatele v jednom komprimovaném souboru. V datovém úložišti Google Drive může uživatel například stahovat celou složku v jednom komprimovaném souboru.

Dalším úhlem pohledu při poskytování dat datovým úložištěm je omezení stahování. Omezení je nejlépe vypracované u datového úložiště Uložto, ve kterém nepřihlášený uživatel může stahovat data pouze omezenou rychlostí a přihlášený uživatel si může za vyšší rychlost stahování dat připlatit. Datové úložiště, které jsem implementoval, takovou možnost neposkytuje. V zadání úlohy bylo jen omezení na stažená data.

Z datových úložišť, která jsem porovnával v kapitole 3, obsahují funkci pro vyhledávání v metadatech všechna úložiště s výjimkou Fedory. Tato funkce mě inspirovala, a proto jsem také vytvořil jednoduchý způsob vyhledávání v metadatech.

Všechna datová úložiště kromě Fedora umožňují vyhledávání minimálně podle názvu souboru. V mém řešení toto vyhledávání není implementováno, protože názvem souboru je pouze identifikátor. Pro vyhledávání jsem se zaměřil na metadata, která mění svoji hodnotu, proto je implementováno pro vlastnosti `Exposure`, `Slice_Location` a `Slice_Thickness` .

Protože dodaná data jsou anonymní a zdravotnického charakteru, bylo by pro server, který by pracoval přímo s metadaty vhodné implementovat vyhledávání, které by bylo možné konfigurovat podle uživatelů například podle profilů uživatele. Předpokládá se, že k datům budou přistupovat různí uživatelé, kteří budou vyhledávat odlišná data. Profily by umožnili usnadnit uživatelům vyhledávání a zároveň zabezpečily, aby uživatelé nepřistupovali k metadatům, pro které nemají oprávnění.

U komerčních řešení se mi bohužel nepodařilo zjistit, jak jsou metadata ukládaná. Pouze u úložiště Facebook mohu z názvu dokumentace k API „Graph API - Documentation“ předpokládat použití grafového ukládání dat [44]. Mému řešení ukládání dat se podobá jen projekt Fedora, který ukládá metadata v RDF trojicích. Fedora ale neřeší další funkce serveru jako například správu uživatelů či omezení přístupu k datům ve svém úložišti.

Datové úložiště Fedora jsem použil pro přímé srovnání s mým řešením. Srovnání bylo umožněno zejména z důvodu, že projekt Fedora je volně dostupný a mohl jsem ho spustit na svém počítači, takže při srovnávání mají obě datové úložiště stejné podmínky.

Abysem zajistil stejné podnímky pro srovnávání přístupu k datům u obou datových úložišť, uložil jsem do datového úložiště Fedora všechny soubory, které byly dodány k datasetu `urna.nt`. Dataset obsahuje 29 tisíc souborů s celkovou velikostí 14 GB a po uložení všech souborů do úložiště Fedora měl datový adresář Fedory 45 GB ve 105 tisících složkách. Počet složek v datovém adresáři Fedory je trojnásobný než počet uložených souborů, což je dáno způsobem ukládání dat na disk.

Pro porovnání přístupu k datům v obou datových úložištích jsem vybral jen požadavky na snímek, protože projekt Fedora řeší pouze ukládání a poskytování dat v datovém úložišti.

Na obrázku 10 na straně 78 je graf porovnávající získání dat z datového úložiště. Data pro graf byly získány pomocí JSP stránky `./fedora.jsp`.

Fedora byla při srovnání s mým datovým úložištěm 12krát rychlejší. Navýšení rychlosti je způsobeno zejména tím, že Fedora poskytuje soubory přímo bez komprimace dat v odpovědi serveru. Dalším faktorem nárůstu rychlosti u Fedory je absence řešení přístupu k datům jednotlivými uživateli serveru, takže Fedora nemusí ukládat žádná data při poskytování služeb datového úložiště.

Rychlost poskytnutí souborů Fedory je též způsobená ukládáním souborů podle názvu identifikátoru, což umožňuje poskytnutí dat bez dotazu SPARQL na umístnění souboru v adresářové struktuře úložiště.

8.3 Výkon serveru

Výkon serveru se nejlépe vyjádří podle toho, jak rychle dokáže vykonávat jednotlivé požadavky klientů. Na výkon serveru má největší vliv rychlost komprimace souborů. Na obrázku 7 na straně 75 se nachází graf, který zobrazuje poměr mezi dotazováním SPARQL a komprimací odpovědi klientovi. Z grafu je patrné, že s narůstající velikostí výsledného souboru narůstá čas potřebný na zkomprimování souboru pro odpověď serveru, přičemž velikost souboru nemá vliv na čas potřebný pro SPARQL dotaz.

Dalším porovnáním výkonu serveru je podle času potřebného na odpověď klientovi. V příloze I jsou dva grafy, které zobrazují čas a čekání při vytvoření 1000 požadavků na server při různém počtu vláken, která je vytváří. Na obrázku 8 na straně 76 se nachází graf, který zobrazuje čas potřebný pro vykonání všech požadavků. Na obrázku 9 na straně 77 je uveden graf, který ukazuje, jak dlouho čekal klient na obsluhu serverem. V druhém případě je použit medián ze všech generovaných požadavků na server.

Jak je z uvedených grafů patrné, k největšímu nárůstu výkonu (čas na obsluhu požadavků je menší) dochází při změně obsluhy požadavků generovanými jedním vláknem na obsluhu několika vlákny. Nárůst je dán použitým procesorem, který má v mém případě dvě fyzická jádra. Při použití více jak dvou vláken není patrný podstatný nárůst výkonu.

Na obrázku 9 na straně 77 se nachází grafické zobrazení čekání na obsluhu požadavku, kde je vidět postupný nárůst doby čekání podle počtu vláken. Do počtu čtyř generujících vláken je čas čekání na obsluhu klientů serverem zanedbatelný.

Výkon serveru je závislý na počtu zpracovávaných požadavků klientů v odděleném vlákně. Na obrázku 11 na straně 79 ukazuji graf, který znázorňuje čas potřebný na vykonání 100 požadavků na snímek z datového úložiště při různém počtu souběžného zpracovávání požadavků na sérii v odděleném vlákně.

Z grafu je patrný velký nárůst času potřebného na vykonání 100 požadavků v závislosti na počtu zpracovávaných požadavků v jiném vlákně. Čas vykonávání 100 požadavků při souběžném zpracovávání dvaceti požadavků je devětkrát vyšší než čas vykonávání 100 požadavků při souběžném zpracovávání jednoho požadavku.

Pro požadavky zpracovávané v odděleném vlákně by bylo vhodnějším řešením jejich zpracování na jiném počítači, než na kterém je spuštěn server. Počítač, který by převzal vypracování požadavku, by mohl mít navíc funkci poskytování výsledků těchto požadavků, které by byly uloženy na jeho pevném disku. Tím by se zadaná úloha vytvoření serveru pro přístup k datovému úložiště mohla rozdělit mezi dva počítače. Výhodou tohoto řešení je, že server by nemusel zpracovávat všechny požadavky, přičemž by druhý počítač vykonával časově náročnější požadavky na větší objem dat. Nevýhodou navrhovaného řešení je zpomalení zpracování požadavku v odděleném vlákně, protože navíc by přibyla režie za přenesení úlohy na jiný počítač.

8.4 Testování

V testech kvality software jsem se zaměřil na testy funkčnosti jednotlivých komponent aplikace. Aplikace je testovaná před kompilací pomocí JUnit testů. Navíc jsem vytvořil možnost testovat výslednou aplikaci pomocí požadavků klienta v reálném prostředí. Dané řešení jsem zvolil z důvodu možnosti ověření funkčnosti aplikace po změně v konfiguraci.

V JUnit testech jsem se zaměřil na přístup k systémovým datům úložiště, který je nejdůležitější pro správnou funkci aplikace. V testech jsou vykonávány SPARQL dotazy do databáze TDB pomocí Fuseki serveru. Pro tyto účely jsem vytvoři testovací data.

Integrační testy pro jednotlivé servlety nejsou kompletní, protože jsem v servletu `UserServlet` testoval pouze požadavek na informaci o přihlášeném uživateli. Tento test ukazuje možnosti testování webové aplikace pomocí mého řešení pseudo serveru.

Další prostor pro testování nabízí data úložiště. Bylo by vhodné testovat, zda jsou ke všem zdrojům data přiřazena, tedy zda jsou všechny soubory podle zdroje skutečně přítomné v adresáři úložiště. Protože v datasetu `urna.nt` jsou i hashe souborů, bylo by vhodným doplněním testů porovnání hashe souboru a jeho záznamu v datasetu `urna.nt`.

Při testování jsem narazil na některá omezení aplikace:

testování požadavku Pro testování požadavků je na straně klienta nastaveno omezení na testování pouze 500 požadavků pro dekomprimaci, protože na mém počítači nebylo možné dekomprimovat větší počet odpovědí serveru.

komprimace odpovědi Odpovědi na požadavek na data v datovém úložišti jsou ukládány do objektu `ByteArryOutputStream`. Maximální počet dat, která lze do objektu uložit, je dán maximální velikostí datového typu `Integer` v Javě. Proto maximální součet velikosti souborů, které můžou být poskytnuty klientům přímo v odpovědi serveru, je dán hodnotou `Integer.MAX_VALUE`.

8.5 Možnosti rozšíření

`ThreadPoolExecutor` lze nahradit za `ScheduledThreadPoolExecutor`, a tím by se dalo naplánovat zpracování požadavků, které budou zpracovávány v odděleném vlákně, na konkrétní čas, kdy není server příliš vytížen. Rozšíření by negativně ovlivnilo požadavek na získaní dat v co nejkratším čase.

Dalším rozšířením by mohl být veřejný přístup k metadatům, jako je u projektu Fedora [34]. Rozšíření by bylo vhodné zejména pro multimediální obsah datového úložiště, popřípadě pro sociální sítě. Pro dodaná data není tento přístup vhodný, protože data jsou anonymní a zdravotnického charakteru, takže by muselo být nastaveno omezení, která metadata mohou být zveřejňovaná.

8.6 Nedostatky a omezení aplikace

Absence ověření existence URI zdroje Aplikace neřeší, zda existuje již URI zdroje před ukládáním nového URI do TDB úložiště. Pravděpodobnost , že aplikace vytvoří dvě stejné URI je $1{:}16^{32}$, protože identifikator je tvořen 32 znaky z abecedy 16ti znaků.

Není implementováno stránkování V aplikaci by bylo vhodné implementovat stránkování při výpisech seznamu uživatelů, výsledků požadavků vypracovávaných v odděleném vlákne a výpisu relací přihlášených uživatelů. Stránkování by bylo vhodné implementovat i u výpisů `Log4J`.

URI prefix Instalační stránka poskytuje jednoduché řešení pro prvotní spuštění aplikace. Takto implementovaná instalace neřeší obsah metadat. Ve výchozím nastavení aplikace je nastaven URI prefix ke všem zdrojům, kterými jsou subjekty ve trojicích RDF na hodnotu `http://mre.zcu.cz/id/`. Pokud budou vkládána metadata, která budou mít jiný URL prefix, nebude aplikace pracovat správně.

Prefix URI musí vždy obsahovat hodnotu poslední položky v cestě před identifikátorem „`id`" například *http://mre.zcu.cz/id/*. Server nebude schopen poskytovat data z úložiště, pokud bude zadán prefix URI nesplňující toto omezení.

9 Závěr

Ve své práci se mi podařilo podle specifikovaných požadavků vytvořit server pro přístup k datovému úložišti.

V teoretické části jsem se seznámil s technologiemi potřebnými pro vytvoření serveru, zejména RDF, OWL, SPARQL, HTTP, REST API a technologii vláken v jazyku Java. Prostudoval a vyzkoušel jsem osm existujících řešení serverů pro přístup k datům v úložišti, přičemž jsem se nejvíce zaměřil na open source projekt Fedora. Servery jsem stručně charakterizoval, porovnal mezi sebou a rozdělil je do vhodných kategorií.

Zanalyzoval jsem požadavky na server, které jsem rozdělil do prioritních tříd, a možnosti využití technologií, se kterými jsem se seznámil v teoretické části. Popsal jsem nároky na data serveru včetně možnosti jejich ukládání a rozdělil jsem požadavky klientů na server do tří kategorií podle použití.

V navrhovaném řešení serveru jsem vybral programovací jazyk, framework a vhodné úložiště pro data serveru. Vytvořil jsem i vývojový diagram zpracování požadavku klienta pro přístup k datům v úložišti. Stanovil jsem způsob ukládání informací o přihlášených uživatelích a záznamů o přístupu k datům jednotlivými uživateli. Definoval jsem omezení přístupu k datům v úložišti.

Pro implementaci serveru jsem vytvořil ontologii popisující systémová data serveru. Popsal jsem konfiguraci aplikace a HTTP serveru, přičemž pro server jsem stanovil vhodný způsob zavádění aplikace. Dále jsem implementoval správu relací uživatelů a kontrolu jejich přístupu k serveru. Vytvořil jsem servlety, které zpracovávají jednotlivé požadavky klientů a navíc jsem vytvořil instalační stránku pro snadné prvotní nastavení a stránku pro vyhledávání v metadatech úložiště. Dále jsem vytvořil administraci serveru, kde se spravují uživatelé, zálohy dat a mj. relace uživatelů. Implementoval jsem odesílání informace o dokončení požadavku na objemná data emailem prostřednictvím SMTP protokolu.

Pro testování serveru jsem vytvořil automatizované testy, které jsou zaměřeny na jednotlivé komponenty serveru, na REST API a data serveru. Komponenty serveru testuji pomocí JUnit testů. Pro účely testování výsledného REST API serveru jsem vytvořil JSP stránku, která generuje požadavky na server. Testování přístupu k datům v úložišti jsem realizoval také pomocí JSP stránky, kde testuji přístup v jednom vlákně, v několika vláknech a náhodný přístup. Zároveň během testování měřím přístup k datům z hlediska času a objemu poskytnutých dat.

Na konci jsem porovnal výslednou práci s úložišti popsanými v teoretické části mimo jiné z hlediska přihlašování uživatelů nebo z hlediska poskytování dat, přičemž jsem se zaměřil na dvě vybraná úložiště. Zaměřil jsem se i na porovnání výkonu serveru z několika hledisek. Faktory ovlivňující výkon serveru jsem detailně popsal a zobrazil na několika grafech. Závěrem jsem navrhl možnosti rozšíření z hlediska zlepšení výkonu, které jsem odvodil na základě naměřených hodnot.

Seznam zkratek

AJAX — Asynchronous JavaScript and XML

API — Application Programming Interface

HTML — Hypertext Markup Language

HTTP — Hypertext Transfer Protocol

JSON — JavaScript Object Notation

JSP — JavaServer Pages

JVM — Java Virtual Machine

OWL — Web Ontology Language

RDF — Resource Description Framework

RDFS — Resource Description Framework Schema

REST — Representational state transfer

SPARQL — SPARQL Protocol and RDF Query Language

TDB — A component of Jena for RDF storage and query

URI — Uniform Resource Identifier

UUID — Universally Unique Identifier

XML — Extensible Markup Language

Literatura

[1] FIELDING, R. – GETTYS, J. – MOGUL, J. C. – FRYSTYK, H. – MASINTER, L. – LOACH P. – BERNERS-LEE, T. *Hypertext Transfer Protocol – HTTP/1.1* [online] 1999. [cit. 2.6.2018] dostupné z: `https://www.w3.org/Protocols/HTTP/1.1/rfc2616.pdf`

[2] BELSHE, M. – PEON, R. – THOMAS, M. *Hypertext Transfer Protocol Version 2 (HTTP/2)* [online] 1999 [citace 9.6.2018] dostupné z: `https://www.ietf.org/rfc/rfc7540.txt`

[3] BERNERS-LEE, T. – FIELDING, R. – MASINTER, L. *Hypertext Transfer Protocol – HTTP/1.1* [online] 1999 [citace 9.6.2018] dostupné z: `https://www.ietf.org/rfc/rfc2616.txt`

[4] CROCKER, D. H. *Standard for ARPA Internet Text Messages* [online] 1982 [citace 9.6.2018] dostupné z: `https://www.ietf.org/rfc/rfc822.txt`

[5] FIELDING, R. *Architectural Styles and the Design of Network-based Software Architectures* [online] 2000 Chapter 5. Representational State Transfer (REST) DISSERTATION `https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style`

[6] RICHARDSON, L. – AMUNDSEN, M. *RESTful Web APIs* Published by O'Reilly Media, Inc., 2013 ISBN: 978-1-449-35806-8

[7] JACOBSON, D. *APIs A Strategy Guider* Published by O'Reilly Media Inc. 2012 str. 4-5 ISBN: 978-1-449-30892-9

[8] RDF Working Group *Resource Description Framework (RDF)* [online] 2014 The World Wide Web Consortium [cit. 1.6.2018] `https://www.w3.org/RDF/`

[9] LASSILA, O. – SWICK, R. R. *Resource Description Framework (RDF) Model and Syntax Specification* [online] 1999 The World Wide Web Consortium `https://www.w3.org/TR/1999/REC-rdf-syntax-19990222/`

[10] *Apache Jena - Typed literals how-to* [online] 2018 The Apache Software Foundation [citace 9.6.2018] dostupné z: `https://jena.apache.org/documentation/notes/typed-literals.html`

[11] YO, L. *Introduction to the Semantic Web and Semantic Web Services* 2007 Taylor & Francis Group, LLC ISBN: 978-1-58488-933-5

[12] BERNERS-LEE, T. – FIELDING, R. – MASINTER, L. *Uniform Resource Identifier (URI): Generic Syntax* [online] 2005 [citace 9.6.2018] dostupné z: `https://tools.ietf.org/html/rfc3986`

[13] DUERST, M., T. – SUIGNARD, M. *Internationalized Resource Identifiers (IRIs)* [online] 2005 [citace 9.6.2018] dostupné z: https://tools.ietf.org/html/rfc3987

[14] *RDF Vocabulary Description Language 1.0* [online] 2002 The World Wide Web Consortium [cit. 2.6.2018] dostupné z: https://www.w3.org/2001/sw/RDFCore/Schema/200203/

[15] *PrimerExampleTurtle* [online] 2008 The World Wide Web Consortium [cit. 10.6.208] dostupné z: https://www.w3.org/2007/OWL/wiki/PrimerExampleTurtle

[16] *Namespaces in XML 1.0*[online]2009 The World Wide Web Consortium https://www.w3.org/TR/xml-names/#sec-namespaces

[17] STAAB, S – STUDER, R. *Handbook on Ontologies 1 Introduction* str. 1-4 Springer-Verlag Berlin Heidelberg 2009 ISBN 978-3-540-70999-2

[18] *OWL 2 Web Ontology Language* [online] 2012 The World Wide Web Consortium [cit. 10.6.2018] dostupné z: https://www.w3.org/TR/owl2-overview/

[19] HITZLEZ, P. – KROTZH, M. – PARSIA, B. – PATEL-SHNEIDER, F.P. – RUDOLPH, S. *OWL 2 Web Ontology Language Primer (Second Edition)* [online] 2012 The World Wide Web Consortium [cit. 10.6.2018] dostupné z: https://www.w3.org/TR/owl2-primer/\#OWL_2_Profiles

[20] GRUBER, T. *Ontology* [online] [cit. 11.6.2018] dostupné z: http://tomgruber.org/writing/ontology-definition-2007.htm

[21] HAYES, P – McBRIDE, B*RDF Semantics*[online] 2004 The World Wide Web Consortium dostuponé [cit. 10.6.2018] z: https://www.w3.org/TR/rdf-mt/

[22] BRICKEY, D – GUHA, R.V.*RDF Schema 1.1*[online] 2014 The World Wide Web Consortium [cit. 10.6.2018] dostuponé z: https://www.w3.org/TR/rdf-schema/

[23] PRUDHOMMEAUX, E. – SEABORNE, A. *SPARQL Query Language for RDF* [online] 2008 The World Wide Web Consortium [cit. 10.6.2018] dostupné z: https://www.w3.org/TR/rdf-sparql-query/

[24] FEIGENBUM, L *SPARQL By Example* [online] 2009 The World Wide Web Consortium [cit. 10.6.2018] dostupné z: https://www.w3.org/2009/Talks/0615-qbe/

[25] HARRIS, S. – SEABORNE, A. *SPARQL 1.1 Query Language* [online] 2013 the World Wide Web Consortium [cit. 10.6.2018] dostuponé z: `https://www.w3.org/TR/sparql11-query/#aggregates`

[26] *Apache Jena* [online] 2018 The Apache Software Foundation [cit. 11.6.2018] dostupné z: `https://jena.apache.org/`

[27] *PYPL PopularitY of Programming Language*[online] 2018 GitHub, Inc. [cit. 8.3.2018] dostupné z: `http://pypl.github.io/PYPL.html`

[28] *Apache Tomcat* [online] 2018 The Apache Software Foundation [cit. 11.6.2018] `http://tomcat.apache.org/`

[29] *Apache Jena - TDB* [online] 2018. The Apache Software Foundation [cit. 2.6.2018] dostupné z: `https://jena.apache.org/documentation/tdb/index.html`

[30] *Apache Jena Fuseki* [online] The Apache Software Foundation [cit. 10.6.2018] dostupné z: `https://jena.apache.org/documentation/fuseki2/`

[31] *Fuseki : Embedded Server* [online] 2018 The Apache Software Foundation [cit. 10.6.2018] dostupné z: `https://jena.apache.org/documentation/fuseki2/fuseki-embedded.html`

[32] *Java^{TM} Platform Standard Ed. 8* [online] 2018 Oracle Corporation [cit. 10.6.2018] dostupné z: `https://docs.oracle.com/javase/8/docs/api/java/util/UUID.html`

[33] OAKS, S. – WONG, H. *Java thread, third edition* Published by O'Reilly Media Inc. 2014 [cit. 10.6.2018] ISBN: 0-596-00782-5

[34] *Fedora* [online] 2018 The community DuraSpace [cit. 8.6.2018] dostupné z: `https://duraspace.org/fedora/about/`

[35] *REST API v3*[online] 2018 GitHub, Inc. [cit. 10.6.2018] dostupné z: `https://developer.github.com/v3/?`

[36] VCELAK, P. *Schéma vybraných tříd a vlastností v systému MRE* [online] [cite. 11.6.2018] `https://mre.zcu.cz/db2/rdf/html/mre-lite.png`

[37] *Microsoft OneDrive* [online] 2018 Microsoft, Inc. [cit. 12.6.2018] `https://onedrive.live.com/about/cs-cz/`

[38] *Ulož.to* [online] 2018 Uloz.to cloud, a.s. [cit. 12.6.2018] `https://www.ulozto.cz/kontakt`

[39] *My Drive - Google Drive* [online] 2018 Google, Inc. [cit. 12.6.2018] https://play.google.com/store/apps/details?id=com.google.android.apps.docs&hl=cs

[40] *Products — Facebook Newsroom* [online] 2018 Facebook, Inc. [cit. 12.6.2018] https://newsroom.fb.com/products/

[41] *Stream.cz Internetová televize, seriály online zdarma a videa* [online] 2018 Seznam.cz, a.s. [cit. 12.06.2018] https://onas.seznam.cz/en/stream-cz-en.html

[42] *O YouTube - YouTube* [online] 2018 Google, Inc. [cit. 12.6.2018] https://www.youtube.com/intl/cs/yt/about/

[43] *About* [online] 2018 GitHub, Inc. [cit. 10.6.2018] dostupné z: https://github.com/about

[44] *Overview - Graph API - Documentation - Facebook for Developers* [online] 2018 Facebook, Inc. [cit. 13.6.2018] https://developers.facebook.com/docs/graph-api/overview

[45] *A JavaScript library to zip and unzip files* [online] 2019 gildas.lormeau@gmail.com [cit. 15.2.2019] https://gildas-lormeau.github.io/zip.js/

[46] *Blob - Web APIs — MDN* [online] 2019 Mozilla and individual contributors [cit. 15.2.2019] https://developer.mozilla.org/en-US/docs/Web/API/Blob

[47] *The Java^{TM} Tutorials* [online] Oracle and/or its affiliates. [cit. 18.2.2019] dostupné z: https://docs.oracle.com/javase/tutorial/essential/concurrency/locksync.html

[48] *protégé* [online] 2016 Stanford Center for Biomedical Informatics Research [cit. 20.2.2019] http://www.visualdataweb.de/webvowl

[49] *VOWL: Visual Notation for OWL Ontologies* [online] contact@visualdataweb.org [cit. 20.2.2019] http://vowl.visualdataweb.org/

[50] *Google Identity Platform* [online] Google, Inc. [cit. 29.5.2019] https://developers.google.com/identity/protocols/OAuth2

Přílohy

Příloha A Tabulky HTTP protokolu

Název	Popis	Příklad
Accept	Určuje, jaké typy médií klient očekává v odpovědi od serveru.	Accept: text/*, text/html
Accept-Charset	Určuje, jakou znakovou sadu má použít server pro data obsažená v odpovědi.	Accept-Charset: UTF-8
Accept-Encoding	Určuje, jak má server data v odpovědi zakódovat. Používá se například ke komprimování odpovědi.	Accept-Encoding: gzip;q=1.0
Accept-Language	Určuje jazyk odpovědi serveru. Používá se pro vícejazyčné servery. Například google.com.	Accept-Language: cs
Host	Parametr určuje, pro jakou doménu klient požaduje odpověď, a je používán hlavně pro virtuální hosting.	Host : zcu.cz
Range	Parametr určuje, jakou část zdroje klient požaduje. Používá se pro přenos objemných multimediálních souborů.	Range: bytes=200-1000
Referer	URI předchozího zdroje nebo webové stránky.	Referer: https://portal.zcu.cz
User-Agent	Určuje typ prohlížeče, ze kterého byl odeslán požadavek na server	User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64)

Tabulka 1: Tabulka vybraných proměnných v hlavičce požadavku na server.

Název	Popis	Příklad
Accept-Ranges	Udává jednotku, podle které bude server poskytovat části zdroje.	Accept-Ranges: bytes
Age	Udává čas v sekundách, kdy byl zdroj v mezipaměti na straně serveru.	User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64)
ETag	Slouží k validaci cache zdroje na straně klienta. Pokud jsou cache zdroje a Etag totožné, prohlížeč použije obsah zdroje z vlastní cache paměti.	ETag: "686897696a7c876b7e"
Location	Absolutní adresa zdroje.	Location: https://www.w3.org/People.html
Retry-After	Udává, za jak dlouho či kdy by měl klient zkusit požadavek zopakovat.	Retry-After: 120
Server	Udává název a typ serveru, který vykonal daný požadavek.	Server : Apache

Tabulka 2: Tabulka vybraných proměnných v hlavičce odpovědi.

Metoda	Mění obsah zdroje?	Popis
GET	ne	Server vrátí zdroj podle URI.
POST	ano	Server vytvoří nový zdroj podle přijatého URI.
PUT	ano	Server zamění zdroj podle URI.
DELETE	ano	Server smaže zdroj podle URI. URI již nebude ukazovat na žádný zdroj.

Tabulka 3: Tabulka metod HTTP protokolu používaných v REST architektuře.

Příloha B Ukázky uložených dat v TDB úložišti aplikace

```
@prefix mik: <http://localhost:8084/mikafil/owl/mikafil.owl# >.
@prefix xsd: <http://www.w3.org/2001/XMLSchema# >.

<http://localhost:8084/mikafil/id/0eb75118aad64a1ab210d04ae33b9 >
a mik:result ;
mik:belongs_to <http://localhost:8084/user/94a0af7469843fb392692adbdf633d09d034bf >;
mik:expired ”2019-02-19T19:45:48.392Z”^^xsd:dateTime ;
mik:fileName ”0eb75118aad64a1ab210d04ae33b9bd7_0a549ad1bc293d1bb2833c519bb5bb3564a35d27.zip”;
mik:originalFileName ”0a549ad1bc293d1bb2833c519bb5bb3564a35d27.zip”;
mik:contains_log <http://localhost:8084/mikafil/id/afac2f579153463695b88eb79b125ab0 >;
mik:hasError false ;
mik:result_id ”0eb75118aad64a1ab210d04ae33b9bd7”;
mik:resource_id ”0a549ad1bc293d1bb2833c519bb5bb3564a35d27”;
mik:resource_uri ”http://mre.zcu.cz/id/0a549ad1bc293d1bb2833c519bb5bb3564a35d27”;
mik:files_count 29082 ;
mik:files_size 15640495492 ;
mik:optional_id ””.
```

Příklad 7: Ukázka uložení výsledku požadavku ve formátu `Notation3`

```
@prefix mik: <http://localhost:8084/mikafil/owl/mikafil.owl# >.
@prefix xsd: <http://www.w3.org/2001/XMLSchema# >.

<http://localhost:8084/mikafil/id/afac2f579153463695b88eb79b125ab0 >
a mik:log ;
mik:belongs_to <http://localhost:8084/user/94a0af7469843fb392692adbdf633d09d034bf>;
mik:start ”2019-02-19T19:02:36.385Z”^^xsd:dateTime ;
mik:finishQuery ”2019-02-19T19:02:36.689Z”^^xsd:dateTime ;
mik:originalSize 15640495492 ;
mik:countFiles 29082 ;
mik:compressedSize 6361490016 ;
mik:countLoggedUsers 1 ;
mik:finish ”2019-02-19T19:25:15.468Z”^^xsd:dateTime ;
mik:zippingRun 1358652 ;
mik:queryingRun 7566 ;
mik:preparingRun 24 ;
mik:run 1366368 .
```

Příklad 8: Ukázka uložení logu ve formátu `Notation3`

```
@prefix mik: <http://localhost:8084/mikafil/owl/mikafil.owl# >.
@prefix xsd: <http://www.w3.org/2001/XMLSchema# >.

<http://localhost:8084/user/94a0af7469843fb392692adbdf633d09d034bf>
a mik:mikafilUser ;
mik:uid 0 ;
mik:password ”b7367edaa7bb675764606b5a9343e09ee90242ce419eabfca14c7c7325e550f6”;
mik:isDeleted false ;
mik:isAdmin true ;
mik:mbox ”filip.mika@email.cz”.
```

Příklad 9: Ukázka uložení uživatele ve formátu `Notation3`

Příloha C Hledání podle vlastností - JSON data a SPARQL dotaz


```
{
"ses-id": "null",
"command": [ { "command": "SELECT DISTINCT (sum(?fSize) as ?sum) (count(?subj) as ?count)"}
],
"bind": [ { "value": "xsd:long(?fileSize) AS ?fSize"} ],
"prefix": [ {
"prefix": "xsd",
"uri": "<http://www.w3.org/2001/XMLSchema#>"
} ],
"where": [ {
"sub": "?subj",
"prd": "<http://www.semanticdesktop.org/ontologies/2007/03/22/nfo#fileSize>",
"obj": "?fileSize"
},
{
"sub": "?subj",
"prd": "<http://mre.zcu.cz/ontology/dcm.owl#Exposure>",
"obj": "?exposure"
},
{
"sub": "?subj",
"prd": "<http://mre.zcu.cz/ontology/dcm.ow#Slice_Location>",
"obj": "?slice_location"
} ],
filter": [ { "value": "( ?exposure < 12 )"},
{ "value": "( ?slice_location > 12 )"} ]
}
```


Příklad 10: JSON data v asynchronním požadavku AJAX

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT DISTINCT (sum(?fSize) as ?sum) (count(?subj) as ?count)

WHERE {
?subj <http://www.semanticdesktop.org/ontologies/2007/03/22/nfo#fileSize> ?fileSize.
?subj <http://mre.zcu.cz/ontology/dcm.owl#Exposure> ?exposure.
?subj <http://mre.zcu.cz/ontology/dcm.owl#Slice_Location> ?slice_location.
BIND( xsd:long(?fileSize) AS ?fSize)
FILTER( ( ?exposure < 12 ))
FILTER( ( ?slice_location > 12 ))
}
```

Příklad 11: Vygenerovaný SPARQL dotaz

Příloha D Implementované REST API

./admin/result/{id} `[DELETE]`
: Server odstraní odpověď která byla vykonavaná v odděleném vlákně, která je daná identifikátorem `id`.

./admin/backup `[GET]`
: Odpovědí serveru je soubor ve formátu N-Triples, který obsahuje systémová data serveru.

./admin/backup `[POST]`
: Server nahradí systémová data podle přijatého souboru od klienta.

./admin/config `[GET]`
: Server vypíše všechny konfigurační vlastnosti. Odpověď je ve formátu JSON. Odpověd obsahuje i metadata o všech konfigurovatelných vlastnostech.

./admin/config `[PUT]`
: Server změní konfiguraci podle přijatých dat v JSON formátu.

./admin/config `[GET]`
: Server vypíše všechny konfigurační vlastnosti. Odpověď je ve formátu JSON. Odpověd obsahuje i metadata o všech konfigurovatelných vlastnostech.

./admin/config/{val} `[GET]`
: Server vypíše hodnotu pouze vlastnosti *val*. Odpověď je ve formátu JSON.

./admin/cvs `[GET]`
: Server vygeneruje CVS soubor, který obsahuje informace o časech dotazování SPARQL a komprimaci odpovědí pro jednotlivé velikosti obsahu zdrojů.

./admin/exired_results `[GET]`
: Server vrátí seznam všech odpovědí, které jsou uloženy na serveru. Ve výpisu jsou pouze dotazy, jejihž platnost vypršela.

./admin/imgs/{cislo_poru} `[GET]`
: Server vypise vsechny zdroje. Vypis bude obdsahovat pouze obrazky.Tento vypis je urcen poouze pro porovnaní přístupu k datovému úložišti `Fedora` a `mikafil`.

./admin/log/{pocet} `[GET]`
: Server vypíše z logovacího souboru `Log4J` počet řádků daný hodnotou vlastnosti *pocet*. Řádky jsou řazeny v sestupném pořadí.

./admin/resources `[GET]`
: Server vypíše seznam všech dostupných zdrojů.

./admin/results/{od}/{do} `[GET]`
Seznam požadavků od hodnoty vlastnosti*od* do hodnoty vlastnosti *do*. Obě hodnoty jsou datum v ISO formátu.

./admin/services `[GET]`
Server vypíše služby, které využívá, a zda jsou spuštěny. Mezi služby patří Fuseki server a datová úložiště TDB.

./admin/sessions `[GET]`
Server vypíše seznam právě přihlášených uživatelů.

./admin/users `[GET]`
Server vypíše seznam vytvořených uživatelů.

./admin/user/{email} `[DELETE]`
Server odstraní uživatele daného parametrem *email*.

./admin/user `[POST]`
Server vytvoří uživatele podle vstupních dat v JSON formátu.

./admin/user/isAdmin/{uid}/{hodnota} `[PUT]`
Server nastaví vlastnost `isAdmin` u uživatele daného *uid* na hodnotu *hodnota*.

./admin/user/isDeleted/{uid}/{hodnota} `[PUT]`
Server nastaví vlastnost `isDeleted` u uživatele daného *uid* na hodnotu *hodnota*.

./install/one `[GET]`
Server Server vyzkouší, zda konfigurační proměnná `show_install` má hodnotu `true`.

./install/two `[POST]`
Server nastaví v konfiguračním souboru číslo portu Fuseki serveru podle přijatých dat a změní v konfiguraci všechny odkazy používající Fuseki server.

./install/three `[POST]`
Server nastaví v konfiguračním souboru adresář, kde jsou uložena data, která budou poskytovaná klientům.

./install/four `[POST]`
Server nastaví v konfiguračním souboru cestu k souboru s metadaty úložiště a nahraje obsah souboru do TDB úložiště.

./install/five `[POST]`
Server vytvoří uživatele a přidá mu práva administrátora.

./install/six [POST]
: Server nastaví konfigurační proměnnou show_install na hodnotu false.

./resource/info/{id} [GET]
: Server vrátí informace o obsahu zdroje podle hodnoty vlastnosti *id*.

./resource/download/{id} [GET]
: Server odešle obsah zdroje klientovi ve zkomprimovaném souboru podle *id*, nebo zpracuje obsah zdroje v odděleném vlákně.

./search [POST]
: Server vrátí HTML kód, který obsahuje množinu URI zdrojů, které vyhovují vyhledávacím kritériím podle přijatých JSON dat.

./search/count [POST]
: Server vrátí HTML kód, který obsahuje informace o počtu nalezených zdrojů a celkovou velikost všech snímků, které vyhovují vyhledávacím kriteriím.

./user [GET]
: Server vrátí JSON data o právě přihlášeném uživateli.

./user/info [GET]
: Server vypíše informace o přihlášeném uživateli.

./user/limits [GET]
: Server vrátí JSON data s informacemi o právě přihlášeném uživateli a data o limitách stahování pro právě přihlášeného uživatele.

./user/login [DELETE]
: Server ukončí relaci. Uživatel již nebude přihlášen.

./user/login [POST]
: Server přihlásí uživatele - vytvoří novou relaci pro daného uživatele.

./user/result/{id} [GET]
: Server vrátí výsledek požadavku, který byl generován serverem ve zvláštním vlákně.

./user/result/{id} [DELETE]
: Server odstraní výsledek požadavku, který byl vykonáván serverem ve zvláštním vlákně.

./user/results [GET]
: Server vrátí odkazy na všechny výsledky požadavků na zdroje, které byly zpracovávané v odděleném vlákně.

Příloha E Vybrané SPARQL dotazy

SPARQL dotaz v příkladu 12 najde všechny zdroje v datovém úložišti. Příkaz UNION spojí nalezené množiny řešení do jednoho řešení. Zároveň příkaz BIND každému nalezenému zdroji přiřadí druh daného zdroje.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema# >
PREFIX owl: <http://www.w3.org/2002/07/owl# >
PREFIX xsd: <http://www.w3.org/2001/XMLSchema# >
PREFIX dcm: <http://mre.zcu.cz/ontology/dcm.owl# >
PREFIX mre: <http://mre.zcu.cz/ontology/mre.owl# >
PREFIX nfo: <http://www.semanticdesktop.org/ontologies/2007/03/22/nfo# >

SELECT *
WHERE{
{
?sub rdf:type dcm:Series.
BIND("serie"AS ?type)
}
UNION
{
?sub rdf:type dcm:Study.
BIND("studie"AS ?type)
}
UNION
{
?sub rdf:type dcm:CT_Image.
BIND("image"AS ?type)
}
}
```

Příklad 12: SPARQL dotaz pro nalezení všech zdrojů datového úložiště

V příkladu 13: SPARQL dotaz najde všechny záznamy o přístupu do datového úložiště za určitý časový interval.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema# >
PREFIX owl: <http://www.w3.org/2002/07/owl# >
PREFIX xsd: <http://www.w3.org/2001/XMLSchema# >
PREFIX mik: <http://localhost:8084/mikafil/owl/mikafil.owl# >

SELECT *
WHERE{ ?log  rdf:type mik:log .
?log  mik:start  ?start .
?log  mik:finish  ?finish .
?log  mik:countFiles  ?count .
?log  mik:originalSize  ?size .
?log  mik:compressedSize  ?zipSize .
?log  mik:run  ?run .
?log  mik:zippingRun  ?zipRun .
?log  mik:queryingRun  ?qryRun .
?log  mik:preparingRun  ?prpRun .
?log  mik:finishQuery  ?finQry .
?log  mik:countLoggedUsers  ?cntUsers .
?log  mik:optional_id  ?optionalId .
FILTER (?start  <=  "2019-02-17T13:58:31.976Z" ^^xsd:dateTime
&&  "2019-02-17T13:58:29.100Z" ^^xsd:dateTime  <= ?finish ). }
ORDER BY DESC (?start)
```

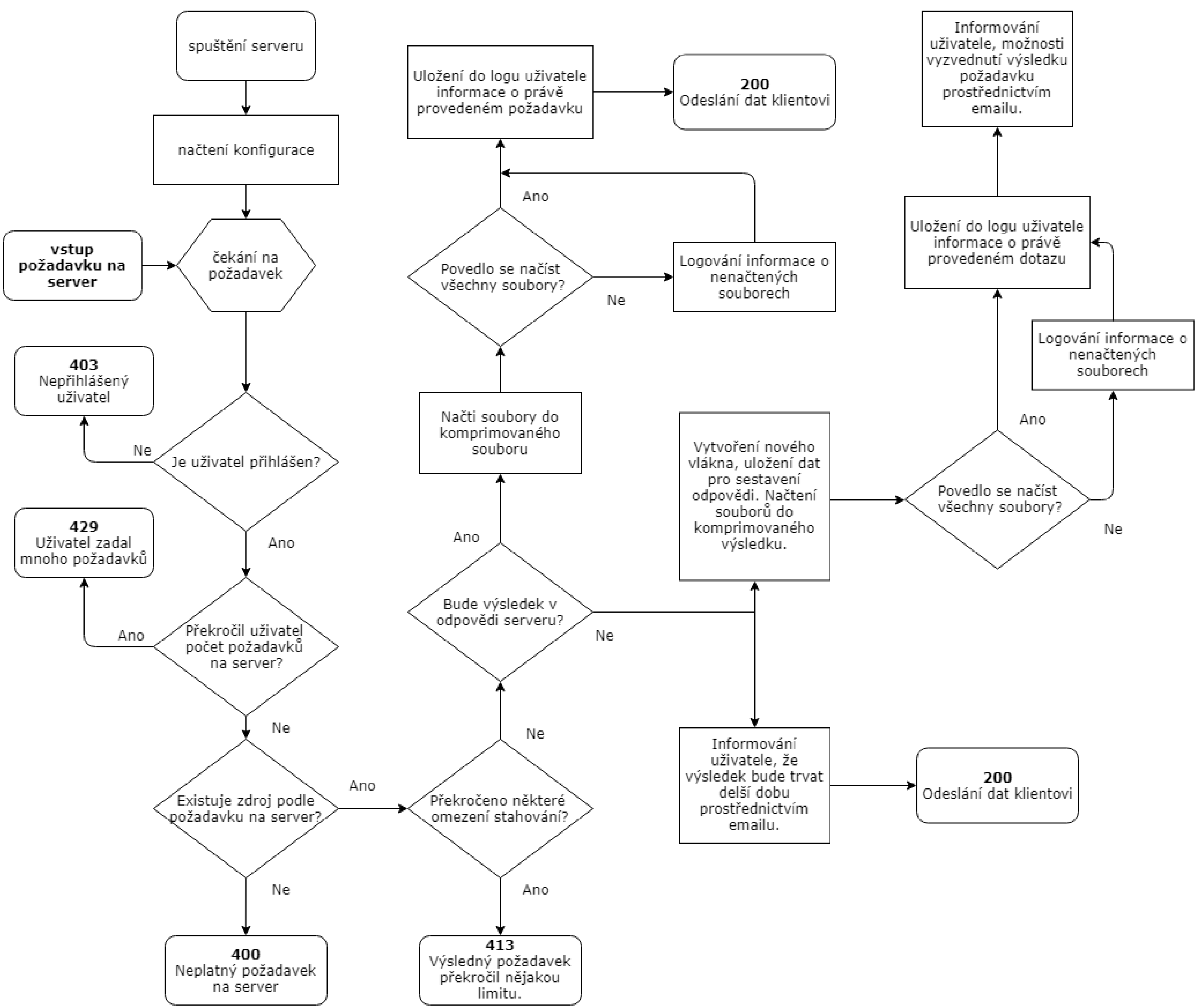
Příklad 13: SPARQL dotaz pro výpis přístupu do datového úložiště

Příkladem 14 je SPARQL dotaz, který hledá informace o přístupu uživatele k souborům v datovém úložišti. Hledá počet poskytnutých souborů a celkovou velikost souborů za den, týden a měsíc. Zde je použitá agregátní funkce MAX() pro určení typu počtu stahování, kde 1 = stahování za den, 2 = stahování za týden a 3 = stahování za měsíc. Příklad je zjednodušeným zápisem.

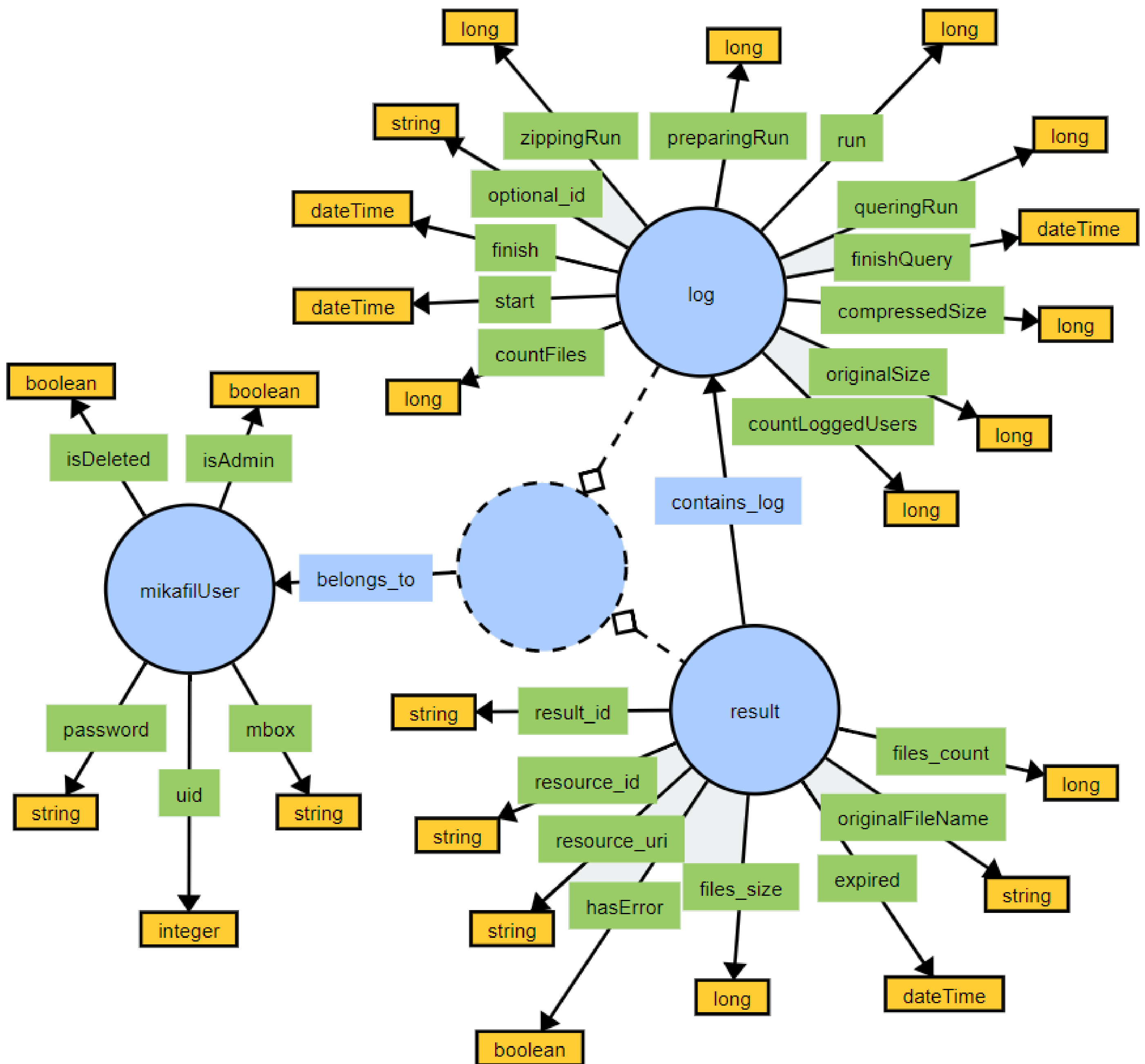
```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema# >
PREFIX owl: <http://www.w3.org/2002/07/owl# >
PREFIX xsd: <http://www.w3.org/2001/XMLSchema# >
PREFIX dcm: <http://mre.zcu.cz/ontology/dcm.owl# >
PREFIX mre: <http://mre.zcu.cz/ontology/mre.owl# >
PREFIX nfo: <http://www.semanticdesktop.org/ontologies/2007/03/22/nfo# >
PREFIX mik: <http://localhost:8084/mikafil/owl/mikafil.owl# >
SELECT *
WHERE{
{
SELECT (MAX(1) AS ?type) (SUM ( ?countFiles) AS ?count) (SUM(?sizeFiles) AS ?size)
WHERE {
?log   mik:belongs_to <http://localhost:8084/user/94a0af7469843fb392692adbdf633d09d034bfb24 >.
?log   rdf:type mik:log .
?log   mik:start   ?start
?log   mik:countFiles   ?countFiles.
?log   mik:originalSize    ?sizeFiles.
FILTER ( ?start >"2019-02-18T05:47:28.042Z" ^^xsd:dateTime )
}}

UNION

{ SELECT (MAX(2) AS ?type) (SUM ( ?countFiles) AS ?count) (SUM(?sizeFiles) AS ?size)
...

} } }
```

Příklad 14: SPARQL dotaz pro nalezení informací o stahování uživatele

Příloha F Diagram zpracovaní požadavku na server



Příloha G Grafické zobrazení ontologie mikafil.owl



Příloha H Snímky obrazovky

Obrázek 3: Snímek instalační stránky

localhost:8085/mikafil/validation.jsp

Mikafil - validace software

uspech ✓

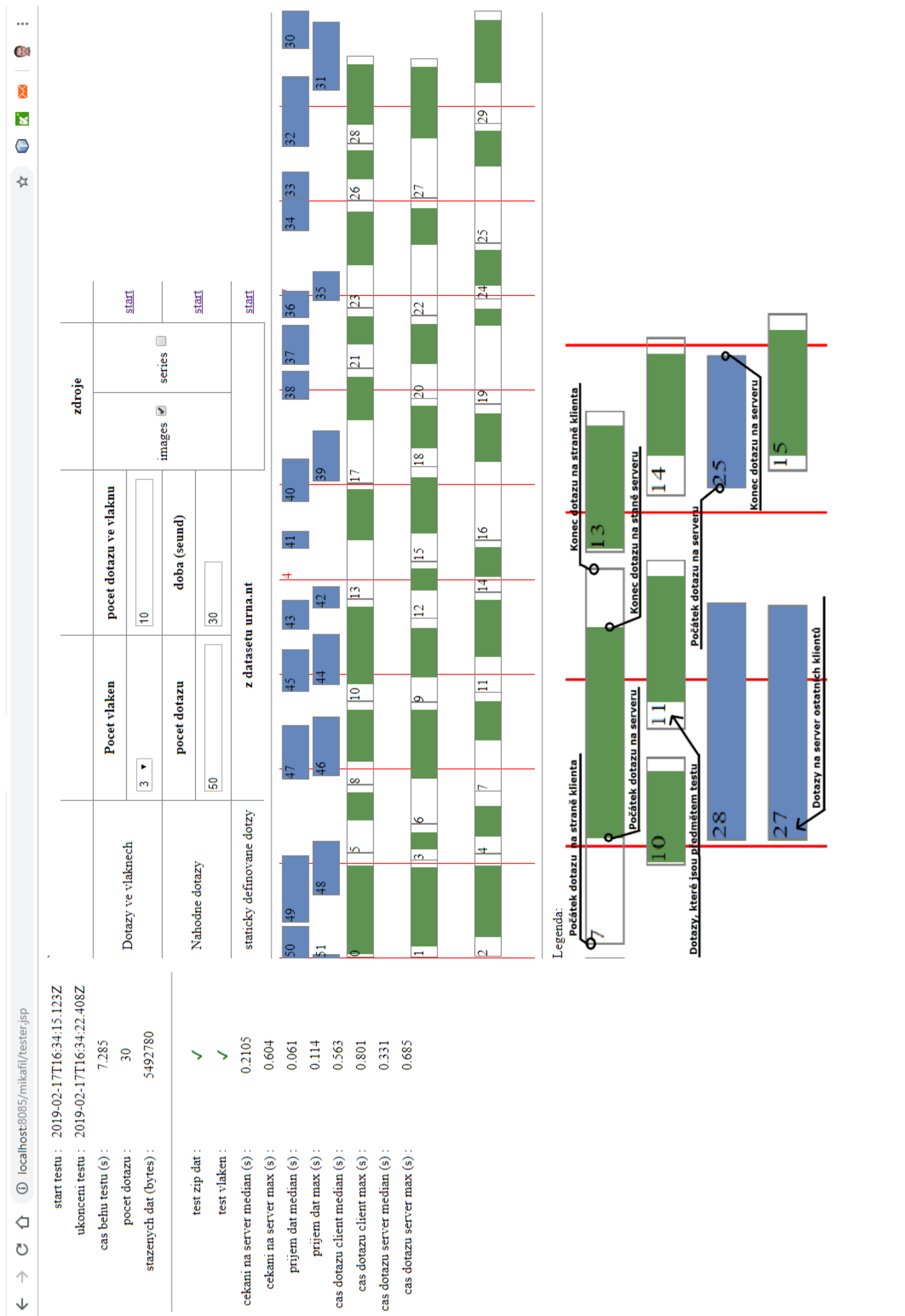
☑	Timeout pro cekani na odpovedi neprimo	int(sekunda)	30
	Pocet opakovani v cyklu pro cekani na odpoved neprimo.	int(pocet)	40
☑	URL serveru s daty pro SPARQL dotazy	URL:	http://192.168.56.1:3343/d
☑	URL serveru se systemovymi daty pro SPARQL dotazy	URL:	http://192.168.56.1:3343/d
	URL serveru se systemovymi daty pro SPARQL update	URL:	http://192.168.56.1:3343/d
	Prihlasovaci udaje pro novy dataset	emial:	filip.mika@email.cz
	Prihlasovaci udaje pro novy dataset	heslo:	
☑	Test vytvoreni uzivatele	emial:	filip@email.cz
		heslo:	

Spustit test | pozastavit | pokracovat

Vsechny testy probehly uspesne.

1 | pass | - Test ziskani parametru too_many_request_timeout z konfigurace
2 | pass | - test nastaveni vlastnosti 'data_endpoint_query' na hodnotu 'http://192.168.56.1:3343/data/query' v konfiguracnim souboru.
3 | pass | - test nastaveni vlastnosti 'system_endpoint_update' na hodnotu 'http://192.168.56.1:3343/dataset/update' v konfiguracnim souboru.
4 | pass | - test nastaveni vlastnosti 'system_endpoint_query' na hodnotu 'http://192.168.56.1:3343/dataset/query' v konfiguracnim souboru.
5 | pass | - Prihlaseni podle noveho datasetu.
6 | pass | - test sluzeb (fuseki, innerDataset, dataSet, config)
7 | pass | - ziskani zdroju
8 | pass | - zkouska existence infa
9 | pass | - test odeslani session id v Cookies
10 | pass | - test odeslani session id v URL
11 | pass | - test odeslani session id v JSON datech.
12 | pass | - test nastaveni hodnoty parametru konfigurace ocekavame hodnotu true
13 | pass | - test uzivatel - ocekavane informace - prave prihlaseny uzivatel
14 | pass | - test vytvoreni nevalidniho uzivatele ocekavame navratovy status 400 - bad request
15 | pass | - test vytvoreni uzivatele s prazdbyn heslem ocekavame navratovy status 400 - bad request
16 | pass | - test vytvoreni uzivatele ocekavame resut true v JSON datech
17 | pass | - test vytvoreni stejneho uzivatele, ocekavame navratovy status 409 - conflict
18 | pass | - test prihlaseni noveho uzivatele
19 | pass | - test zda je novy uzivatel skutecne prihlasen
20 | pass | - test limits.
21 | pass | - test stazeni zdroje, ktery obsahuje jeden soubor.
22 | pass | - Nacteni informace o stazeni.
23 | pass | - Test omezeni poctu stahovani poctu souboru za den . Ocekavame 413 - Request Entity Too Large.
24 | pass | - Test omezeni stahovani dat za den. Ocekavame 413 - Request Entity Too Large.
25 | pass | - test omezeni poctu stahovanych souboru za tyden. Ocekavame 413 - Request Entity Too Large.
26 | pass | - Test omezeni stahovani poctu dat za tyden. Ocekavame 413 - Request Entity Too Large.
27 | pass | - Test omezeni stahovani souboru za mesic. Ocekavame 413 - Request Entity Too Large.
28 | pass | - test omezeni stahovani dat za mesic. Ocekavame 413 - Request Entity Too Large.
29 | pass | - Test zda zdroj bude server poskytovat primo. Ocekavame stus 200 a v JSON datech ['serve'] = 'staight'
30 | pass | - Test zda bude server poskytovat data (jeden snimek) neprimo.
31 | pass | - Test stazeni dat (jeden snimek) zde provedeme jenom stazeni infa o ulozenem souboru.
32 | pass | - Test zda zdroj bude stahovat serii primo. Ocekavame stus 200 a v JSON datech ['serve'] = 'staight'
33 | pass | - Test zda bude server poskytovat data serie neprimo.
34 | pass | - Test stazeni dat serie zde provedeme jenom stazeni infa o ulozenem souboru.
35 | pass | - nastavime zpet v konfiguracnim souboru directly_download_limit
36 | pass | - zkouska zabezpeceni AdminServletu ocekavame navratovy status 403 - Forbidden
37 | pass | - zkouska zabezpeceni AdminServletu (neprihlaseny uzivatel) ocekavame navratovy status 401 - Unauthorized
38 | pass | - zkouska existence admin/infa
39 | pass | - test odstraneni uzivatele
40 | pass | - test vraceni vlastnosti 'system_endpoint_update' na hodnotu 'undefined' v konfiguracnim souboru.
41 | pass | - test vraceni vlastnosti 'system_endpoint_query' na hodnotu 'undefined' v konfiguracnim souboru.
42 | pass | - test vraceni vlastnosti 'data_endpoint_query' na hodnotu 'undefined' v konfiguracnim souboru.
43 | pass | - nastavime zpet v konfiguracnim souboru directly_download_limit

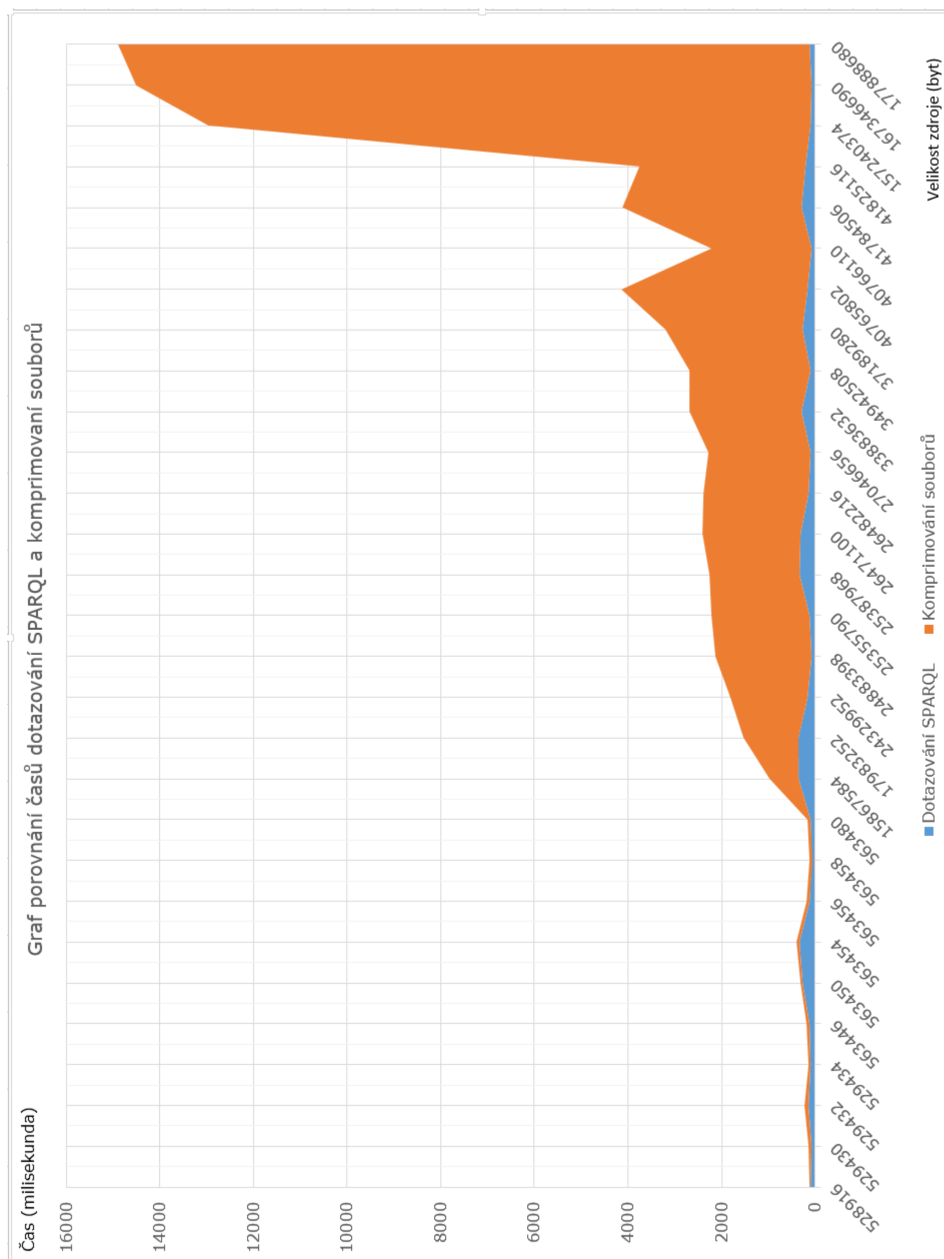
Obrázek 4: Snímek stránky testující REST API



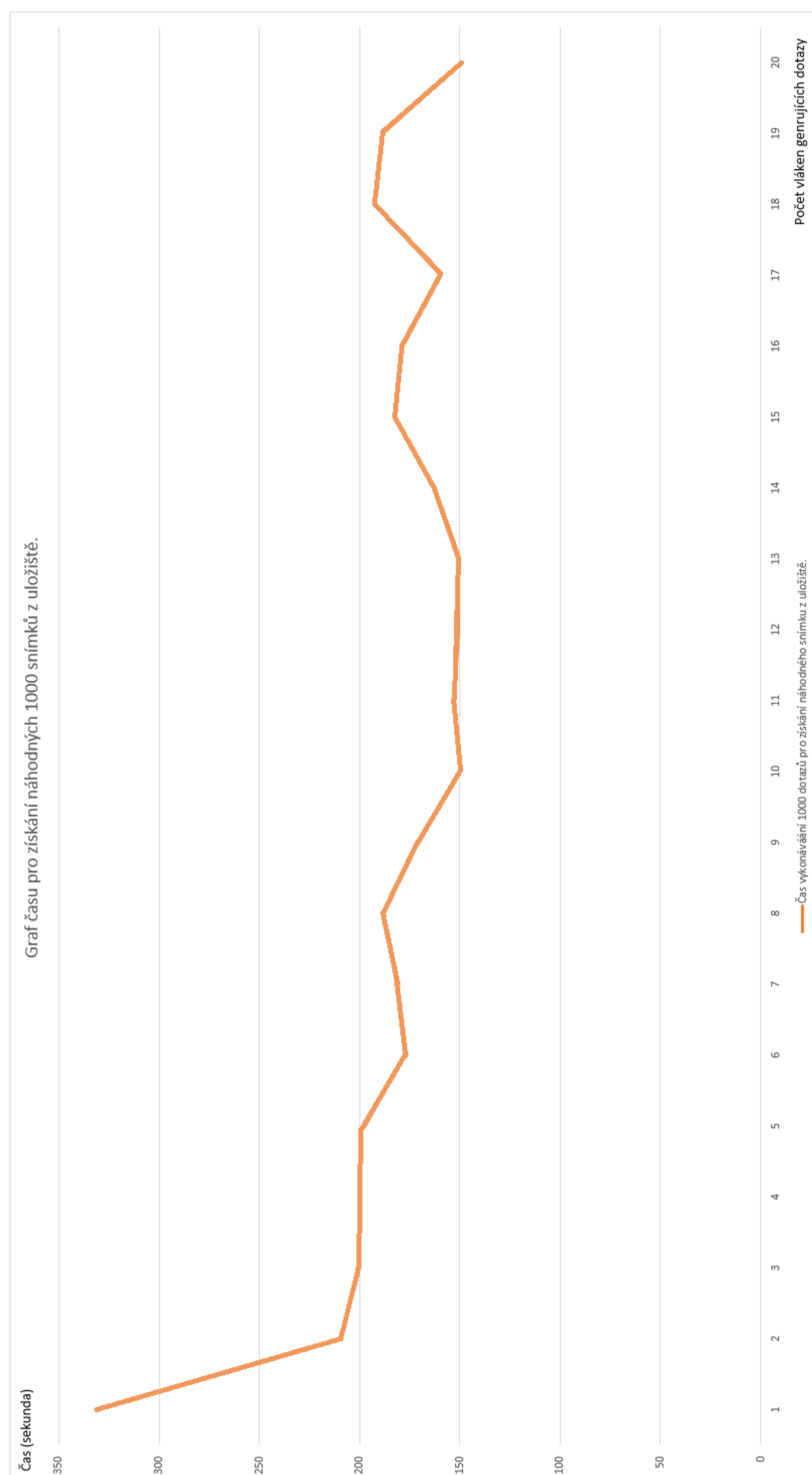
Obrázek 5: Snímek stránky testující přístup k datům v datovém úložišti

Obrázek 6: Snímek stránky testující přístup k datům s detailní informací o požadavku

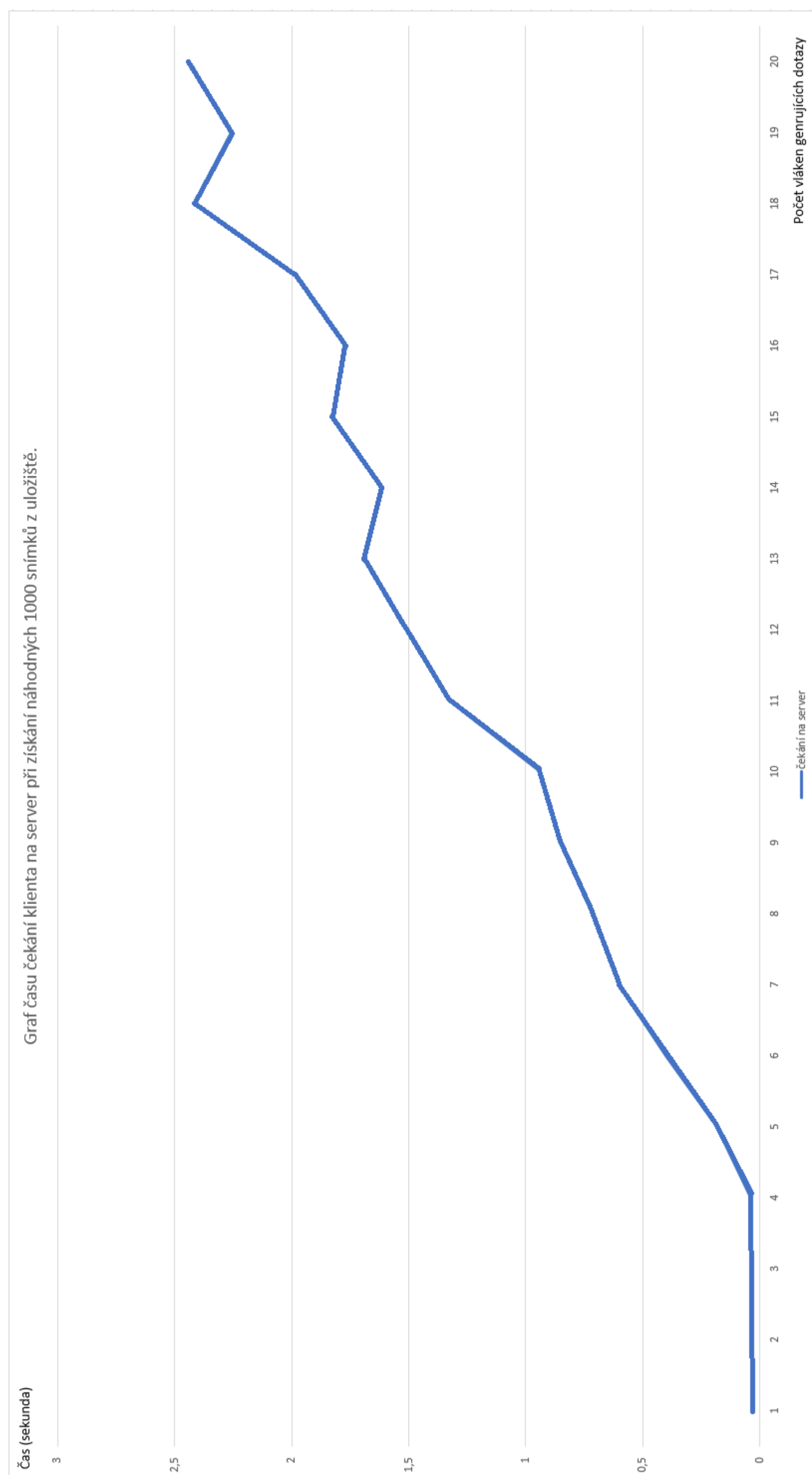
Příloha I Grafy



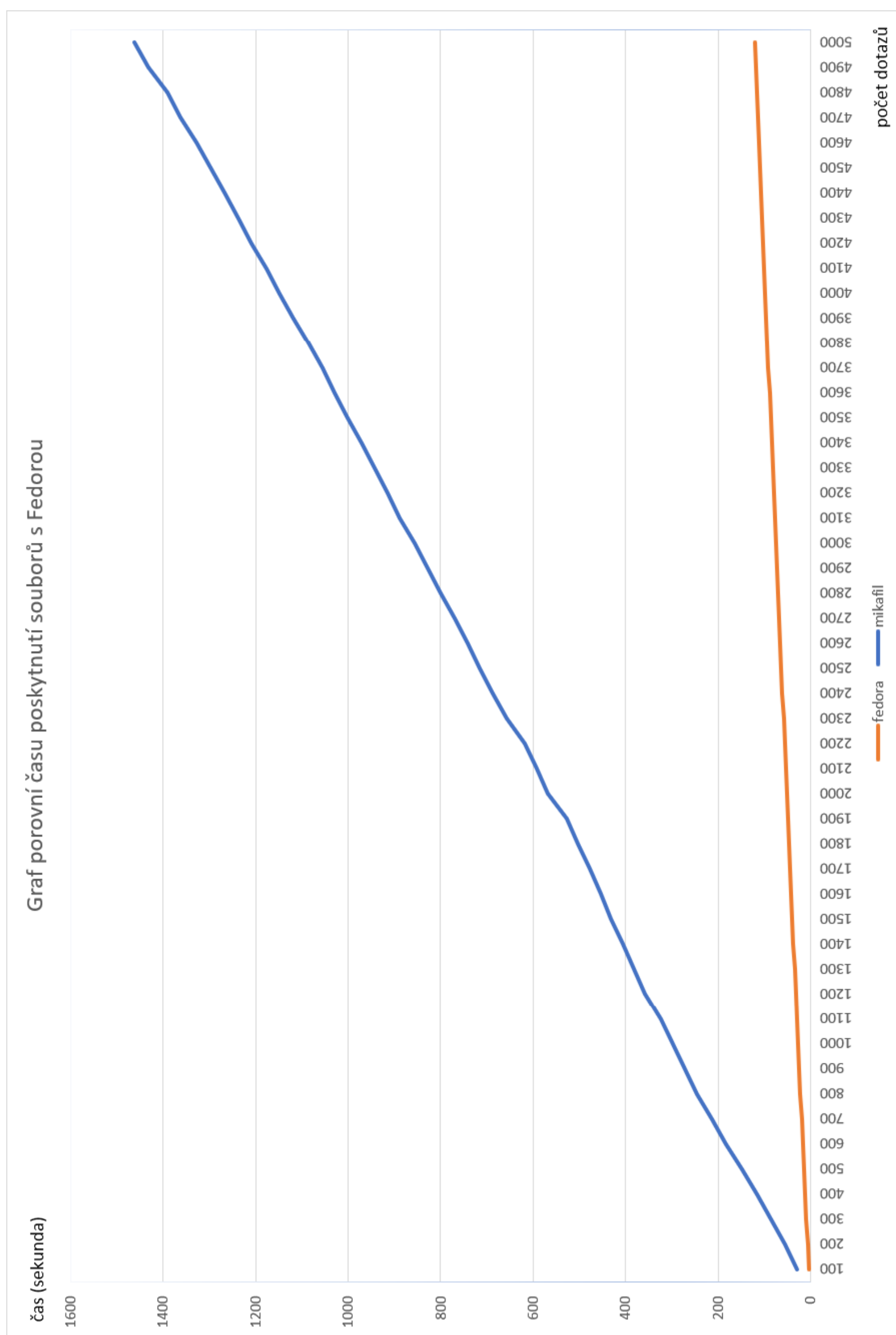
Obrázek 7: Graf porovnání času SPARQL dotazu a komprimace odpovědi



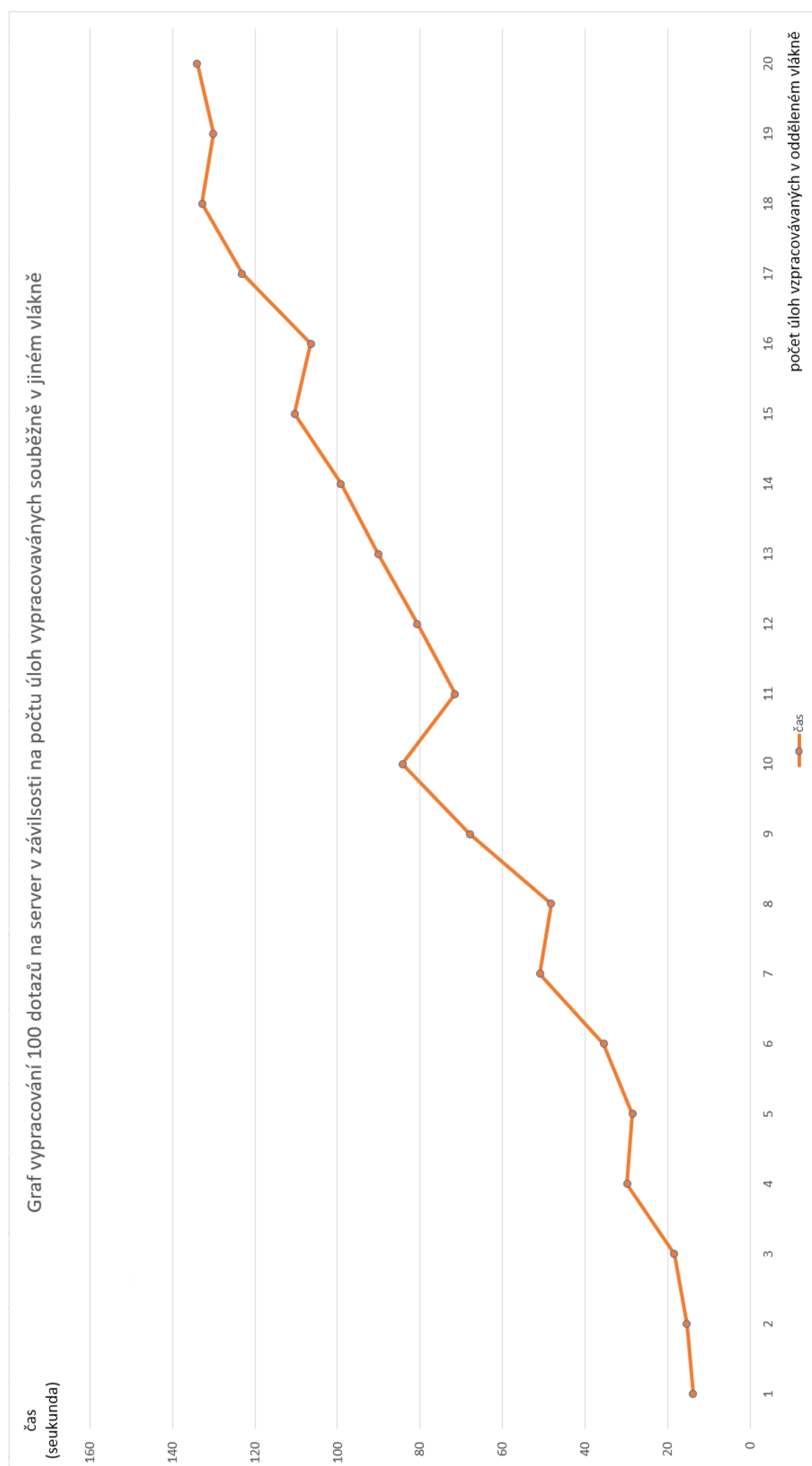
Obrázek 8: Graf 1000 náhodných přístupů (medián) v závislosti na počtu vláken



Obrázek 9: Graf času čekání na obsluhu serveru při náhodném získání 1000 snímků z úložiště (medián)



Obrázek 10: Porovnání času získání snímků s úložištěm Fedora



Obrázek 11: Závislost zpracování požadavků na počtu úloh vykonávaných na pozadí

Příloha J Konfigurační proměnné

auth_timeout

výchozí hodnota : `1800`
jednotka : `sekunda`
popis : `Čas, po který si bude Session pamatovat přihlášené uživatele.`

count_threads

výchozí hodnota : `20`
jednotka : `počet`
max : `500`
popis : `Počet vláken pro zpracovávání požadavků v odděleném vlákně.`

data_endpoint_query

výchozí hodnota : `http://localhost:3341/data/query`
jednotka : `URL`
popis : `URL na endpoint pro SPARQL dotazy na metadata. Tyto dotazy jsou pouze pro získaní metadat.`

data_endpoint_update

výchozí hodnota : `http://localhost:3341/data/update`
jednotka : `URL`
popis : `URL na endpoint pro SPARQL dotazy. Tyto dotazy slouží pro ukládání metadat při instalaci.`

data_store_dir

výchozí hodnota : `C\:\\data`
jednotka : `URL`
popis : `Cesta k adresáři, kde jsou uložena data, která budou poskytována klientům.`

download_limit_bytes_per_day

výchozí hodnota : -1
jednotka : počet
popis : Denní limita objemu dat. Hodnota -1 znamená bez limitu.

download_limit_bytes_per_month

výchozí hodnota : -1
jednotka : počet
popis : Měsíční limita objemu dat. Hodnota -1 znamená bez limitu.

download_limit_bytes_per_week

výchozí hodnota : -1
jednotka : počet
popis : Týdenní limita objemu dat . Hodnota -1 znamená bez limitu.

download_limit_files_per_day

výchozí hodnota : -1
jednotka : počet
popis : Denní limita počtu souborů. Hodnota -1 znamená bez limitu.

download_limit_files_per_month

výchozí hodnota : -1
jednotka : počet
popis : Měsíční limita počtu souborů. Hodnota -1 znamená bez limitu.

download_limit_files_per_week

výchozí hodnota : -1
jednotka : počet
popis : Týdenní limita počtu souborů. Hodnota -1 znamená bez limitu.

directly_download_limit

výchozí hodnota : 2000000
max : 2147483647
jednotka : pocet
popis : Součet velikostí souborů předávaných klientům přímo v odpovědi na požadavek klienta.

fuseki_port

výchozí hodnota : 3341
jednotka : cislo
popis : Číslo portu pro Fuseki server.

metadata_resource_prefix

výchozí hodnota : http://mre.zcu.cz/id/
jednotka : URL
popis : URL, které bude použito pro metadata.

mikafil_owl

výchozí hodnota : http://localhost:8084/mikafil/owl/mikafil.owl
jednotka : URL
popis : URL ontologie pro potřeby systémových dat serveru.

outcome_timeout

výchozí hodnota : 2592000
jednotka : sekunda
popis : Čas, po který budou ukládána odchozí data. Po tomto čase nebudou data uživatelům dostupná.

show_install

výchozí hodnota : true
jednotka : boolean
popis : Logická hodnota, zda se má zobrazovat instalační stránka.

smtp_address

výchozí hodnota : smtp.gmail.com
jednotka : text
popis : SMTP adresa serveru pro odeslání notifikace o dokončení požadavku.

smtp_email

výchozí hodnota : mikafil.students.zcu.cz@gmail.com
jednotka : text
popis : SMTP emailová adresa pro odeslání notifikace o dokončení požadavku.

smtp_password

výchozí hodnota : `tajn0 heslo`
jednotka : `text`
popis : `SMTP heslo pro odeslání notifikace o dokončení požadavku.`

smtp_port

výchozí hodnota : `587`
jednotka : `počet`
popis : `SMTP číslo portu pro odeslání notifikace o dokončení požadavku.`

system_endpoint_query

výchozí hodnota : `http://localhost:3341/dataset/query`
jednotka : `URL`
popis : `URL endpointu pro SPARQL dotazy na systémová data aplikace.`

system_endpoint_update

výchozí hodnota : `http://localhost:3341/dataset/update`
jednotka : `URL`
popis : `URL endpointu pro SPARQL dotazy pro ukládání systémových dat aplikace.`

system_resource_prefix

výchozí hodnota : `http://localhost:8084/mikafil/id`
jednotka : `URL`
popis : `URL, které bude použito pro ukládání dat do TDB úložitě.`

triples

výchozí hodnota : `C:\data\tcia\tcia-lung_phantom.nt`
jednotka : `cesta`
popis : `Cesty k souboru, kde jsou uložena metadata. Lze použít více souborů.`

too_many_request_count

výchozí hodnota : `5`
jednotka : `pocet`
popis : `Podle tohoto počtu posledních požadavků na server se bude rozhodovat o chybě TOO_MANY_REQUESTS.`

too_many_request_timeout

výchozí hodnota : `5`
jednotka : `sekunda`
popis : `Čas pro chybu TOO_MANY_REQUESTS.`

users_base_url

výchozí hodnota : `http://localhost:8084/user/`
jednotka : `URI`
popis : `Předpona URI pro ukládání zdrojů, kterými budou uživatelé.`

Příloha K Struktura přiloženého CD-ROM

```
+---apache-tomcat-8.5.39
+---bakalarska_prace
+---demo
+---dokumentace
+---income
+---scripts
+---src
    +---java
    +---main
    |   +---java/cz/zcu/students/mikafil
    |   |    +---exception
    |   |    |    +---data
    |   |    +---listener
    |   |    +---servlet
    |   |    +---util
    |   +---webapp
    |       +---META-INF
    |       +---owl
    |       +---WEB-INF
    +---test
        +---data
        |   +---tdb
        |   |    +---DATA_STORE_TDB
        |   |    +---OUTCOME
        |   |    +---SYSTEM_TDB
        |   +---resources
        |        +---urna/2014/08/07/22
        |   +---webapp/webapps/mikafil
        |        +---lib
        |        +---META-INF
        |        +---owl
        |        +---WEB-INF
      +---java/cz/zcu/students/mikafil
          +---data
          +---env
          +---servlet
          +---util
```