Bakalářská práce

České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra kybernetiky

Sémantický správce dokumentů

Marek Jaroš

Vedoucí: Ing. Martin Ledvinka
Studijní program: Otevřená informatika
Květen 2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Jaroš** Jméno: **Marek** Osobní číslo: **474734**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra kybernetiky**

Studijní program: **Otevřená informatika**

Studijní obor: **Informatika a počítačové vědy**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Sémantický správce dokumentů

Název bakalářské práce anglicky:

Semantic Document Manager

Pokyny pro vypracování:

1. Analyzujte stávající implementaci aplikace Document Manager a určete oblasti, které bude třeba upravit pro přidání podpory technologií Sémantického webu.
2. Analyzujte způsoby správy uživatelů a rolí v existujících dokument management systémech.
3. Navrhněte modul správy uživatelů a rolí pro Document Manager. Přihlédněte k faktu, že samotné uživatelské účty jsou spravovány externí službou.
4. Přidejte podporu pro ukládání metadat pomocí sémantických technologií do stávající implementace Document Manager.
5. Naimplementujte Vámi zvolený modul správy uživatelů a rolí do aplikace Document Manager.
6. Ověřte správnost implementace pomocí testovacích scénářů s reálnými soubory a externí autorizační službou.

Seznam doporučené literatury:

[1] I. N. Burtylev, K. V. Mokhun, Y. V. Bodnya, D. N. Yukhnevich, Development of Electronic Document Management Systems: Advantage and Efficiency, Science and Technology, Vol. 3 No. 2A, 2013, pp. 1-9. doi: 10.5923/s.scit.201301.01
[2] M. Shariff, Alfresco Enterprise Content Management Implementation, Packt Publishing, 2007
[3] D. Allemang, J. Hendler, Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL, Morgan Kaufmann, 2011

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Martin Ledvinka, skupina znalostních softwarových systémů FEL

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

Datum zadání bakalářské práce: **06.01.2020** Termín odevzdání bakalářské práce: **22.05.2020**

Platnost zadání bakalářské práce: **30.09.2021**

Ing. Martin Ledvinka	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

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

. Datum převzetí zadání

Podpis studenta

Poděkování

Rád bych poděkoval Ing. Martinu Ledvinkovi za vedení bakalářské práce, poskytnutý čas a zpětnou vazbu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 21. května 2020

Abstrakt

Tato bakalářská práce navazuje na vývoj správce dokumentů s názvem Document Manager, který byl vytvořený jako součást bakalářské práce Martina Malinova. Document Manager je jednoduchý správce dokumentů, který ukládá informace do SQL databáze.

Cílem této práce je aplikaci upravit tak, aby byla propojitelná s aplikací správy terminologií používanou skupinou znalostních softwarových systémů ČVUT FEL zvanou TermIt.

Výsledkem práce je systém pro správu dokumentů schopný ukládat informace jak do relační, tak do RDF databáze. Aplikace je schopná u dokumentů spravovat čtyři úrovně oprávnění. Tyto úrovně lze u dokumentů přiřadit uživateli nebo skupině uživatelů. Uživatele spravuje externí autorizační služba. Aplikace pracuje pouze s identifikátory uživatelů.

Aplikace byla otestována 159 automatickými testy JUnit a aplikací Postman. Napsané JUnit testy pokryly 78% tříd, 84% metod a 80% řádek kódu. Pro aplikaci Postman byly vytvořeny tři scénáře, každý obsahující pět až deset požadavků, které testují různé části aplikace.

Klíčová slova: Správa dokumentů, DMS, Sémantický web, Správa uživatelů

Vedoucí: Ing. Martin Ledvinka

Abstract

This bachelor thesis continues development of a document management system called Document Manager which was created as a part of bachelor thesis by Martin Malinov. Document Manager is a document manager that saves information to a SQL database.

The purpose of this thesis is to modify the application so that it can be linked with a terminology management tool used by Knowledge Based Software Systems Group of ČVUT FEL called TermIt.

The result of the thesis is a document management system capable of saving information into a relational database as well as an RDF database. The application is capable of managing four levels of permissions. These levels can be assigned to a document for certain users or to a user group. Users are managed by an external authorization service. The application uses only user's identifiers.

The application was tested with 159 automatic tests JUnit and application Postman. Written JUnit tests covered 78% classes, 84% methods and 80% lines of code. Three scenarios were written for apllication Postman, each containing five to ten requests, testing different parts of the application.

Keywords: Document management, DMS, Semantic web, User management

Title translation: Semantic document manager

Obsah

1 Úvod 1
1.1 Cíl práce 1

2 Náhled na použité technologie 3
2.1 Systémy pro správu dokumentů . . 3
2.2 Technologie sémantického webu . . 3
2.2.1 RDF 4
2.2.2 SPARQL 4
2.2.3 JSON-LD 5

3 Úprava aplikace a přidání podpory technologií sémantického webu 7
3.1 Analýza dosavadní aplikace 7
3.1.1 Architektura 7
3.2 Úprava aplikace 8
3.3 Implementace 10
3.3.1 Perzistentní vrstva 10
3.3.2 Úprava modelu 11
3.3.3 Kontrolní vrstva 12
3.3.4 Další úpravy 12
3.4 Ukázka funkcionality 13

4 Analýza řešení správy uživatelů v systémech pro správu dokumentů 15
4.1 Rozpis technologií 15
4.1.1 Seed DMS 15
4.1.2 Feng Office 16
4.1.3 LogicalDoc 17
4.1.4 Krystal DMS 18
4.1.5 OpenDocMan 18
4.1.6 OpenKM 19
4.2 Shrnutí 19
4.2.1 Porovnání typů oprávnění pro úrovňový styl 21
4.2.2 Další poznatky 21

5 Návrh vlastního modelu správy uživatelů 23
5.1 Udělování oprávnění 23
5.1.1 Možnosti nastavení oprávnění pro různé entity 23
5.1.2 Styl oprávnění 23
5.1.3 Typy oprávnění 23
5.2 Skupiny 24
5.3 Seznam řízení přístupu 24
5.4 Role 24
5.5 Podpora externí autentizace 25

6 Implementace modelu správy uživatelů 27
6.1 Úprava modelu 27
6.2 Autorizace požadavků 27
6.3 Úpravy na kontrolní a servisní vrstvě 28

7 Testování aplikace 31
7.1 JUnit testy 31
7.1.1 Funkce modelu 31
7.1.2 Perzistentní vrstva 31
7.1.3 Servisní vrstva 32
7.1.4 Kontrolní vrstva 32
7.1.5 Pokrytí kódu 32
7.2 Testy s pomocí externí aplikace . 33
7.2.1 Test vytváření dokumentu, souboru a jeho nových verzí 34
7.2.2 Test uživatelských oprávnění 34
7.2.3 Test skupinových oprávnění . 35
7.3 Propojení s aplikací TermIt 36

8 Závěr 37

A Literatura 39

B Seznam použitých zkratek 41

C Obsah přiloženého CD 43

D Návod ke spuštění 45
D.1 Kompilace a spuštění aplikace . 45

Obrázky

2.1 Příklad RDF grafu. Zelené ovály jsou referenty a modré obdélníky jsou literály. Šipky mezi nimi jsou predikáty a jsou označeny jejich příslušnými IRI. [7] 4

3.1 Doménový model aplikace Document Manager [12] 8
3.2 Příklad provozu vícevrstvé architektury. Černé šipky znázorňují požadavky a červené odpovědi na dané požadavky. [16] 9
3.3 Architektura aplikace Document Manager [12] 9
3.4 UML diagram pro DAO třídy Document. DocumentRepository je DAO pro relační databázi implemenové pomocí JpaRepository. DocumentSemanticDao je DAO pro RDF databázi. DocumentDao je nově vytvořené rozhraní, které je implementované těmito třídami a používá se na servisní vrstvě. 10
3.5 Porovnání změn ve výňatku implementace třídy Folder. 11

4.1 Nastavování oprávnění pro složku v Seed DMS 16
4.2 Nastavování oprávnění uživatele pro pracovní plochu ve Feng Office. Uživatel test user je pouze Collaborator User a nelze mu udělit vyšší oprávnění. 17
4.3 Nastavování oprávnění složky v LogicalDoc. Skupině admin nelze oprávnění měnit. U skupiny Guest jsou nastavená nejvyšší možná oprávnění. 18

5.1 Rozhodovací strom ukazující, jak se volí oprávnění u dokumentu pro daného uživatele. 25

6.1 Upravený model aplikace. Zelené entity jsou nově přidané entity. . . . 28

7.1 Statistiky získané pomocí knihovny Jacoco. Elementy představují různé balíčky aplikace. Element documentManager představuje celou aplikaci, tedy všechny balíčky dohromady. 33
7.2 Ilustrace scénáře Test skupinových oprávnění. 36

8.1 Výsledná struktura aplikace. Černě označené soubory nebyly upraveny nebo byly provedeny pouze jednoduché úpravy (méně než 5 upravených řádek). Modré soubory byly upraveny složitěji. Zelené soubory jsou nově přidané soubory. 38

Tabulky

3.1 Porovnání aplikace před a po úpravách. 13

4.1 Porovnání řešení v analyzovaných systémech pro správu dokumentů . 20
4.2 Porovnání úrovní v systémech, které používaly úrovňový styl 21

Kapitola 1
Úvod

Práce navazuje na vývoj aplikace s názvem Document Manager, což je systém pro správu dokumentů vytvořený jako součást bakalářské práce Martina Malinova. Jedná se o jednoduchou aplikaci, která umí strukturovat a ukládat soubory. Struktura a metadata o souborech se ukládá do relační databáze a soubory do souborového systému. [12]

1.1 Cíl práce

Cílem práce je vytvořit jednoduchý správce dokumentů, který lze propojit s aplikací správy terminologií používanou skupinou znalostních softwarových systémů ČVUT FEL zvanou TermIt[1], která používá technologie sémantického webu. K tomu bude použita aplikace Document Manager, která by měla být upravena tak, aby sémantické technologie podporovala a zároveň nebyla narušena její původní funkcionalita.

Dalším cílem je navrhnout vlastní model pro správu uživatelů, který se implementuje do aplikace Document Manager. K tomu bude provedena analýza řešení správy uživatelů v systémech pro správu dokumentů. Na závěr bude otestována korektnost implementace testovými scénáři.

[1] `https://github.com/kbss-cvut/termit` [navštíveno 2.1.2020]

Kapitola 2
Náhled na použité technologie

V této kapitule popisuji hlavní technologie relevantní k úpravě aplikace Document Manager.

2.1 Systémy pro správu dokumentů

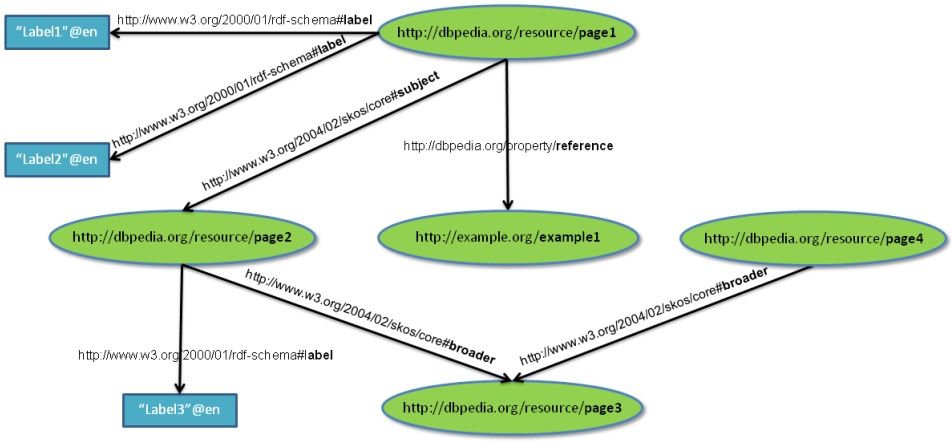
Systémy pro správu dokumentů jsou podtřídou enterprise systémů a jsou blízce spojeny se systémy pro správu obsahu (anglicky content management system, nebo CMS). DMS se liší tím, že berou dokumenty jako atomické prvky a neřeší jejich obsah. Každý dokument se obvykle skládá ze složek a souborů libovolného typu, nicméně struktura dokumentů v systémech pro správu dokumentů se může lišit. DMS se dále vyznačují verzováním, které uchovává předchozí verze souborů a spravuje je.

DMS mohou ukládat soubory do souborového systému nebo do databáze. Obě varianty mají své výhody a nevýhody. Hlavní výhody ukládání do souborového systému jsou lepší výkon, menší požadavky na paměť, jednodušší implementace a snadnější přesun souborů na jiné médium. Hlavní výhody ukládání do databáze jsou větší bezpečnost a konzistence. Navíc je zálohování snazší, jelikož zálohou databáze se zároveň zálohují i soubory. Na druhou stranu je výsledkem jeden velký soubor a data z databáze a soubory nejsou odděleny. [19][17]

2.2 Technologie sémantického webu

Cílem sémantického webu je strukturovat informace na webu tak, aby se s nimi dalo lépe pracovat a zároveň byly pochopitelné nejen člověkem, ale i strojem. Hlavní ideou je, aby mezi sebou nebyly propojené pouze webové stránky, ale i samostatné informace na nich. Tímto by se vytvořila globální databáze propojených dat zvaná Linked Data. Na vývoji standardů sémantického webu se podílí konsorcium W3C[1], které vyvíjí standardy a pokyny pro World Wide Web. K realizaci sémantického webu se používají technologie jako RDF, SPARQL nebo OWL. [5]

[1] `https://www.w3.org/` [navštíveno 2.1.2020]



Obrázek 2.1: Příklad RDF grafu. Zelené ovály jsou referenty a modré obdélníky jsou literály. Šipky mezi nimi jsou predikáty a jsou označeny jejich příslušnými IRI. [7]

2.2.1 RDF

Resource Description Framework (RDF) je framework, který je v technologiích sémantického webu standard pro modelování dat. Tento model obsahuje zdroje, které mohou být cokoli, například fyzické předměty, dokumenty, abstraktní koncepty, čísla či řetězce. Tyto zdroje dělíme na referenty a literály. Referenty jsou reprezentovány identifikátorem Internationalized Resource Identifier (IRI), zatímco literály reprezentují nějakou hodnotu. U literálů je potřeba definovat jejich datový typ a je možné jim připsat jazykový tag. Databáze, které ukládají data dle standardu RDF se nazývají triple store nebo také RDF store. [15]

Relaci mezi zdroji zapisujeme ve formě trojice subject, predicate a object (v češtině podmět, predikát a předmět). Podmět je referent a předmět je buď referent nebo literál. Predikát určuje, jaký je mezi zdroji vztah a je definován identifikátorem IRI. Těmito vztahy se také určuje, jaké bude mít podmět atributy. Příklad takové trojice je v listingu 2.1. Množinu trojic můžeme vizualizovat jako orientovaný graf, který nazýváme RDF graf (viz. 2.1). [15]

```
1 http://example.cz/name#Bob http://xmlns.com/foaf/0.1/knows http://
      example.cz/name#Alice
```

Listing 2.1: Příklad RDF trojice. Zde je http://example.cz/name#Bob podmět, http://xmlns.com/foaf/knows predikát a http://example.cz/name#Alice předmět.

2.2.2 SPARQL

Simple Protocol and RDF Query Language (SPARQL) je dotazovací jazyk pro databáze, které ukládají data dle standardu RDF. SPARQL stejně jako RDF pracuje s identifikátory IRI a trojicemi. SPARQL umí data vybírat,

vkládat, upravovat a mazat. Stejně jako v dotazovacím jazyce SQL existuje dotaz SELECT (viz. 2.2), jehož je odpověď tabulka dat splňující podmínky popsáné v daném dotazu. SPARQL poskytuje další typy dotazů, které v SQL nenajdeme. Příkladem je dotaz ASK, který na pravdivostní výrok vrací hodnotu true nebo false. [18]

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE
{
    ?person a          foaf:Person.
    ?person foaf:name ?name.
}
```

Listing 2.2: Příklad dotazu v SPARQL, který ptá na jména všech lidí v databázi.

2.2.3 JSON-LD

Jedním z používáných formátů k serializaci sémantických dat je formát JSON-LD, který je založený na formátu JSON. Tento formát je navržený tak, aby se systémy používající serializaci JSON daly snadno převést na serializaci JSON-LD. Tento formát poskytuje tzv. kontext, který blíže specifikuje význam atributů, aby se atributy nemusely psát jako IRI. Příklad JSON-LD s kontextem je v listigu 2.3. Kontext se následně odvodí z poskytnuté adresy, nicméně ho lze uvést i přímo v daném JSON-LD (viz. 2.4). Pokud JSON-LD obsahuje kontext, tak je možné ho přepsat do verze bez kontextu, kde používá jako atributy celé IRI. Příklady 2.3 a 2.4 se dají přepsat do verze bez kontextu 2.5. [10]

```
{
  "@context": "https://json-ld.org/contexts/person.jsonld",
  "name": "Manu Sporny",
  "homepage": "http://manu.sporny.org/",
  "image": "http://manu.sporny.org/images/manu.png"
}
```

Listing 2.3: Příklad informace serializované do formátu JSON-LD používající referovaný kontext [11]

```
{
  "@context": {
    "name": "http://schema.org/name",
    "image": {
      "@id": "http://schema.org/image",
      "@type": "@id"
    },
    "homepage": {
      "@id": "http://schema.org/url",
      "@type": "@id"
    }
  },
  "name": "Manu Sporny",
  "homepage": "http://manu.sporny.org/",
  "image": "http://manu.sporny.org/images/manu.png"
}
```

Listing 2.4: Příklad informace serializované do formátu JSON-LD používající in-line kontext [11]

```
[{
  "http://schema.org/name": [{"@value": "Manu Sporny"}],
  "http://schema.org/url": [{ "@id": "http://manu.sporny.org/" }],
  "http://schema.org/image": [{ "@id": "http://manu.sporny.org/images/
      manu.png" }]
}]
```

Listing 2.5: Přepis JSON-LD do verze bez kontextu [11]

Kapitola 3

Úprava aplikace a přidání podpory technologií sémantického webu

Aplikaci Document Manager nejdříve analyzuji. Zjistím, jak je aplikace sestavena a jak funguje. Následně určím, které části aplikace bude potřeba upravit a jakým způsobem.

3.1 Analýza dosavadní aplikace

Aplikace je postavena na frameworku Spring[1] s modulem Spring Boot. Spring je Java framework, který usnadňuje psaní enterprise aplikací. Spring boot je modul, jehož hlavní vlastností je, že zjednodušuje psaní stand-alone webových aplikací ve Spring frameworku.

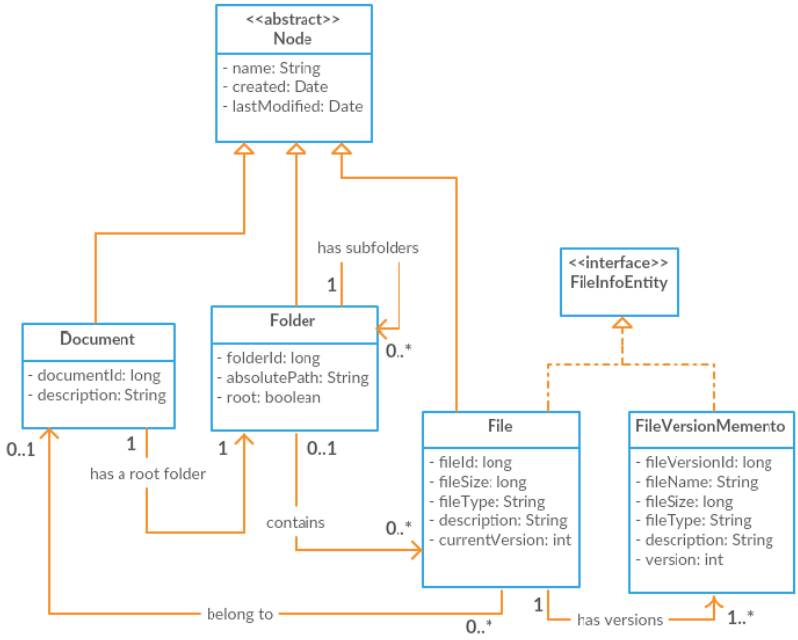
Document Manager umí vytvářet, upravovat, mazat a vracet informace o souborech, složkách a dokumentech. Tyto entity jsou v modelu aplikace pojmenovány File, Folder a Document (viz. 3.1). Entity lze vyhledat dle jejich identifikátoru nebo získat seznam všech entit jednoto typu. Soubory se ukládají do souborového systému a metadata o souborech společně s informacemi o složkách a dokumentech se ukládají do relační databáze.

Aplikace má ještě čtvrtou entitu FileMemento, která obsahuje metadata určité verze souboru a kterou na rozdíl od ostatních entit nelze přímo spravovat. Při vytvoření souboru se vytvoří instance File, která obsahuje metadata daného souboru a také se vytvoří instance FileMemento, která obsahuje totožné informace. Po nahrání nové verze souboru se pokaždé přepíší informace v korespondující instanci File za informace nově nahraného souboru a vytvoří se nová instance FileMemento s totožnými informacemi.

3.1.1 Architektura

Architektura aplikace Document Manager je vzorem vícevrstvé architektury (viz. 3.2), která je běžná pro Java webové aplikace. Vícevrstvé architektury se obvykle skládají ze čtyř vrstev a to prezentační (kontrolní), servisní, perzistentní a databázové. Prezentační vrstva komunikuje s uživateli a přijímá jejich požadavky. Ty jsou následně přeposlány servisní vrstvě, která je zpracovává.

[1] `https://spring.io/` [navštíveno 2.1.2020]



Obrázek 3.1: Doménový model aplikace Document Manager [12]

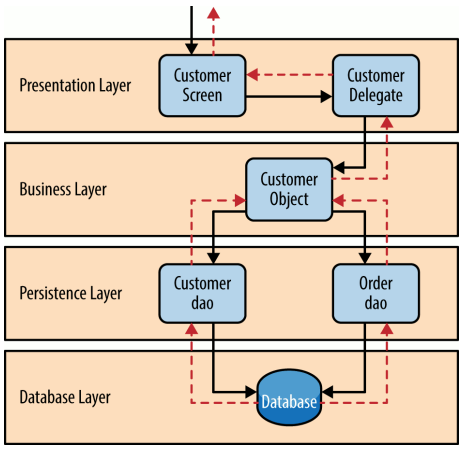
Využívá při tom perzistentní vrstvu, která zajišťuje komunikaci mezi aplikací a databázovou vrstvou. Databázová vrstva obvykle bývá nějaký databázový server, který běží nezávisle na aplikaci. Všechny vrstvy posílají požadavky pouze vrstvě přímo pod ní. [8][20][9]

Architektura aplikace Document Manager (viz. 3.3) se od tohoto standardu liší v tom, že servisní vrstva posílá požadavky jak perzistentní tak databázové vrstvě (databázová vrstva je zde jak relační databáze tak souborový systém).

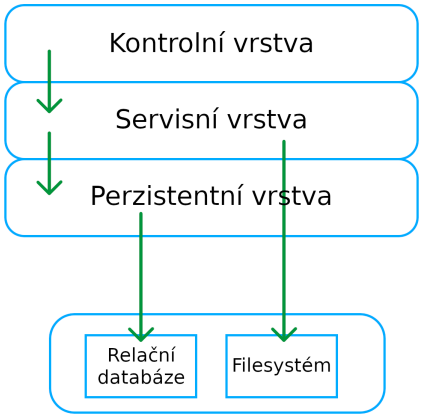
3.2 Úprava aplikace

Cílem je upravit dosavadní aplikaci Document Manager tak, aby byla schopna ukládat data do relační databáze nebo do RDF store. Jaká z těchto variant bude spuštěna, bude deklarováno v konfiguračním souboru aplikace. Aby Document Manager podporoval technologie sémantického webu, je potřeba udělat několik úprav na různých vrstvách aplikace.

Na databázové vrstvě se kromě relační databáze bude také používat databáze triple store. K ukládání dat je potřeba vytvořit OWL glosář, který bude obsahovat deklarace pojmů a vztahů z doposud používaného relačního modelu. Zároveň bude potřeba, aby entity měly IRI, který bude sloužit jako identifikátor pro jednotlivé entity v RDF store. Pro naše účely bude stačit identifikátor URI neboli Uniform Resource Identifier, který je stejný jako IRI, ale obsahuje znaky pouze z kódovací tabulky ASCII, zatímco IRI obvykle osabuje znaky ze znakové sady UTF-8. Dosavadní aplikace používá identifikátor typu Long, zatímco URI je typu String. Samotné soubory se budou stále



Obrázek 3.2: Příklad provozu vícevrstvé architektury. Černé šipky znázorňují požadavky a červené odpovědi na dané požadavky. [16]

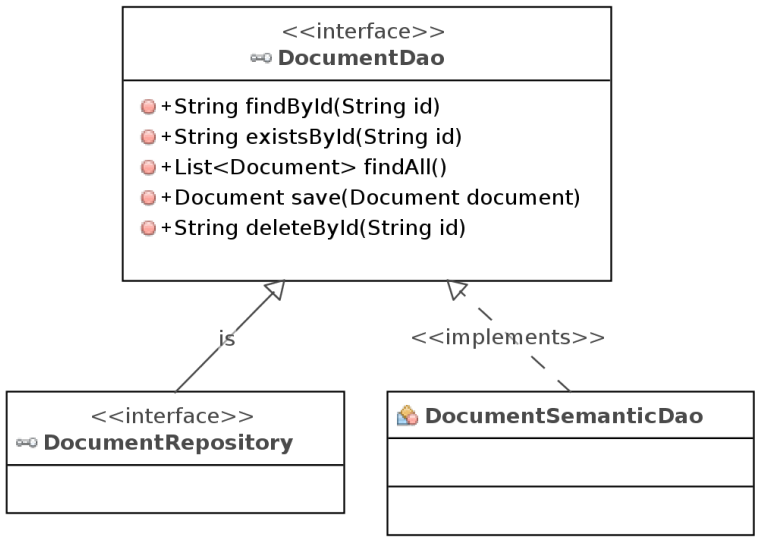


Obrázek 3.3: Architektura aplikace Document Manager [12]

ukládat do souborového systému.

Na perzistentní vrstvě bude potřeba přidat Data access object (zkráceně DAO), který bude komunikovat s RDF store. Servisní a kontrolní vrstvy bude potřeba upravit tak, aby byly schopny pracovat s novým identifikátorem. Dále bude potřeba upravit kontrolní vrstvu tak, aby podporovala kromě JSON i JSON-LD.

Dále bude aplikace upravena tak, aby servisní vrstva používala pouze perzistentní vrstvu, aby byl dodržen standard vícevrstvé architektury a také bude upraven model aplikace.



Obrázek 3.4: UML diagram pro DAO třídy Document. DocumentRepository je DAO pro relační databázi implemenové pomocí JpaRepository. DocumentSemanticDao je DAO pro RDF databázi. DocumentDao je nově vytvořené rozhraní, které je implementované těmito třídami a používá se na servisní vrstvě.

3.3 Implementace

K implementaci byla použita knihovna JB4JSON-LD Jackson[2], která slouží k serializaci a deserializaci Java objektů do a z JSON-LD, a knihovna JOPA[3], která slouží k perzistenci sémantických dat.

3.3.1 Perzistentní vrstva

Problematiku používání dvou různých databází jsem vyřešil pomocí funkcionality profilů poskytnutou frameworkem Spring. Při spuštění aplikace se odvodí název aktivního profilu z konfiguračního souboru aplikace. U tříd lze určit v jakém profilu bude vytvořena jejich instance, pokud si jí framework Spring vyžádá, pomocí anotace Profile.

Třídy perzistentní vrstvy jsou implementovány pomocí Spring Data JPA, která usnadňuje implementaci Data Access Objektů. K třídám perzistentní vrstvy jsem přidal rozhraní (interface) pro každou z nich. Tyto rozhraní byly již implementované existujícími třídami používající relační databázi. Rozhraní jsem následně implementoval i pro RDF databázi za pomocí knihovny JOPA. Na obrázku 3.4 je diagram DAO pro třídu Document.

Implementacím používající relační databázi byl přiřazen profil *jpa* a k implementacím používající RDF databázi byl přiřazen profil *sem*. Bude-li nějaká služba potřebovat DAO pro nějakou entitu, Spring vybere implementaci v závislosti na aktivním profilu. Třídy na servisní vrstvě jsem upravil tak, aby

[2] `https://github.com/kbss-cvut/jb4jsonld-jackson` [navštíveno 2.1.2020]
[3] `https://github.com/kbss-cvut/jopa` [navštíveno 2.1.2020]

```
-  @JsonIgnoreProperties({"folders", "files", "parentFolder", "root","document"})
+  @OWLClass(iri = Vocabulary.c_Folder)
+  @JsonIgnoreProperties({"persistenceContext", "folderPath", "id"})
   @Entity(name = "folder")
   public class Folder extends Node {
-      @Id
-      @GeneratedValue(generator = "folder_generator")
-      @SequenceGenerator(
-              name = "folder_generator",
-              sequenceName = "folder_sequence",
-              initialValue = 1000
-      )
-      protected long folderId;
-
+
+      @Column(name = "description")
+      @OWLAnnotationProperty(iri = Vocabulary.p_description)
+      private String description;
+
+      @Column(name = "isRoot")
+      @OWLAnnotationProperty(iri = Vocabulary.p_isRoot)
+      private Boolean isRoot;
+
+      @JsonIgnore
       @ManyToOne
       @JoinColumn(name = "subfolderId")
+      @OWLObjectProperty(iri = Vocabulary.p_parentFolder)
       private Folder parentFolder;
-
+
+      @JsonIgnore
       @OneToMany(mappedBy ="parentFolder", cascade = CascadeType.REMOVE, orphanRemoval = true)
-      private List<Folder> folders = new ArrayList<>();
-
+      @OWLObjectProperty(iri = Vocabulary.p_subfolders,
+              cascade = cz.cvut.kbss.jopa.model.annotations.CascadeType.REMOVE)
+      private Set<Folder> folders = new HashSet<>();
```

Obrázek 3.5: Porovnání změn ve výňatku implementace třídy Folder.

používaly nově vytvořené rozhraní. Vzhledem k tomu, že obě implementace rozhraní mají stejné funkce, servisní vrstva nepozná, že se perzistentní vrstva mění.

3.3.2 Úprava modelu

Byla vytvořena třída konstant Vocabulary, která obsahuje pojmy a vztahy z relačního modelu. Do tříd modelu byly přidány anotace z knihovny JOPA, které konstanty z třídy Vocabulary používají. Na obrázku 3.5 je změna implementace třídy Folder.

Aby entity neměly dva identifikátory, jeden pro relační databázi a jeden pro RDF store, byl původní identifikátor změněn z typu long na String a je do něj ukládán URI při běhu obou variant aplikace. Z toho je zřejmé, že bylo potřeba změnit původní DAO používané pro relační databázi.

U entity Folder byl odstraněn atribut folderPath a byla přidaná funkce, která cestu vypočítá. Hlavní důvod této změny je, aby se při přejmenování složky nemusel měnit atribut folderPath pro všechny její podsložky. Dalším důvodem je, že původní aplikace tuto problematiku ani neřešila a při změně názvu složky se atribut folderPath pro podsložky neměnil. Při serializaci složky se nově implementovaná funkce použije a vrátí se jako atribut. To platí pouze pokud se serializuje do formátu JSON, jelikož knihovna JB4JSON-LD Jackson tuto funkcionalitu nepodporuje, viz. sekce 3.3.3

3.3.3 Kontrolní vrstva

K tomu, aby aplikace podporovala serializaci a deserializaci JSON-LD, bylo přidáno několik konfiguračních tříd měnících chování aplikace. Jedna z těchto konfiguračních tříd obsahuje dvě beany typu ObjectMapper, jednu pro JSON a drouhou pro JSON-LD. Když aplikace potřebuje serializovat nebo deserializovat objekt, použije jednu z těchto tříd ObjectMapper. Která z těchto tříd se zvolí, závisí na poli Accept v hlavičce požadavku poslaného aplikaci. Při výchozím nastavení aplikace je použit ObjectMapper pro JSON-LD.

Kvůli změně typu identifikátoru se změnil způsob přijímání požadavků na kontrolní vrstvě. Dříve bylo možné snadno přistupovat k entitám podle jejich identifikátoru typu Long. Například cesta k dokumentům je */documents* a dříve se specifikoval dokument dle identifikátoru, tedy např. cesta */documents/13* přistupovala k dokumentu s identifikátorem 13. URI nicméně obsahuje lomítka, která by změnila cestu.

URI se skládá ze dvou částí a to namespace a fragment. Namespace je řetězec před poslední závorkou a fragment je řetězec za poslední závorkou. Tento problém je vyřešen tak, že se v cestě použije pouze fragment URI a namespace se poskytne jako parametr požadavku. Pro přístup k dokumentu s URI *http://example.cz/Document/Test* by cesta byla */documents/-Test?namespace=http://example.cz/Document/Test*.

3.3.4 Další úpravy

Jelikož se změnil typ identifikátoru, již není možné ukládat soubory do souborového systému stejným způsobem. Dříve aplikace ukládala soubory do složek, které byly pojmenovány stejně jako identifikátor daného souboru. To nyní není možné, jelikož URI obsahuje znaky, které se v názvu složek filesytému používat nesmí. Tento problém by se dal vyřešit například tak, že u názvu složky se zakázané znaky nahradí jinými nezakázanými znaky, nebo se smažou. Nicméně jelikož URI bývají někdy velmi dlouhé řetězce, tak by názvy složek byly dlouhé a nepřehledné.

Proto jsem zvolil metodu používající hashování. Název složky se vytvoří tak, že se ve fragmentu nahradí zakázané znaky za nezakázané a následně za fragment přidám hash namespace daného URI. Například pro soubor s URI *http://example.cz/File/Test?File* bude název jeho příslušné složky *Test_File_c6d5b42ezddv*. Tím zůstanou názvy složek relativně krátké a o něco více přehledné.

K ukládání souborů do souborového systému používala aplikace třídu na servnisní vrstvě s názvem FileStorageService. Tuto třídu jsem přejmenoval na FileStorageDao a přesunul jí na vrstvu perzistentní. Následně jsem doplnil do služby FileService funkce používající FileStorageDao pro ukládání souborů. Kontrolní vrstva byla upravena tak, aby používala pouze FileService, jelikož předtím používala i FileService i FileStorageService. Tím je již dodržen standard vícevrstvé architektury, jelikož kontrolní vrstva používá pouze vrstvu servisní a servisní vrstva používá pouze vrstvu perzistentní.

V aplikaci se nově pracuje s URI na několika vrstvách aplikace. Proto

	Před	Po
Formát serializace a deserializace objektů	JSON	JSON a JSON-LD
Podporované databáze	SQL	SQL a RDF
Typ identifikátoru entit	Long	String (URI)
Vícevrstvá architektura aplikace	narušena	dodržena
Způsob vytváření názvu složek pro soubory	použije se identifikátor Long	kombinace konce URI a hash zbytku URI
Způsob získaní cesty ke složce	proměnná	funkce

Tabulka 3.1: Porovnání aplikace před a po úpravách.

jsem vytvořil třídu UriCreationUtils v novém balíčku utils, která obsahuje statické funkce, které pomáhají s vytvářením URI. Je zde např. funkce, která z fragmentu a namespace vytvoří URI, hashovací funkce, která z datového typu String vytvoří String omezené délky nebo funkce, která kontroluje validitu URI. V tabulce 3.1 je shrnutí změn aplikace.

3.4 Ukázka funkcionality

V následujících příkladech 3.1 a 3.2 jsou těla požadavků poslaná aplikaci, která se týkají vytvoření nového dokumentu. V jednom případě je tělo ve formátu JSON a v druhém v JSON-LD, nicméně aplikace je schopna oba dva požadavky zpracovat.

Odpovědí aplikace je serializovaný dokument, který byl vytvořen dle informací v poslaném požadavku. Dokument je serializovaný do JSON (3.3) nebo do JSON-LD (3.4) dle informace v poslaného požadavku. Pokud tato informace není specifikována, dokument je serializovaný ve formátu JSON-LD.

```
1 {
2     "name": "Test document",
3     "description": "Document description"
4 }
```

Listing 3.1: Tělo požadavku ve formátu JSON pro vytvoření nového dokumentu

```
1 {
2     "@type": [
3         "http://example.cz/Document"
4     ],
5     "http://example.cz/name": "Test document",
6     "http://example.cz/description": "Document description"
7 }
```

Listing 3.2: Tělo požadavku ve formátu JSON-LD pro vytvoření nového dokumentu

```
{
    "uri": "http://example.cz/Document/Test_document-k55mcsyk",
    "name": "Test document",
    "created": "2020-01-08T18:11:13.436+0000",
    "lastModified": "2020-01-08T18:11:13.436+0000",
    "description": "Document description",
    "rootFolder": {
        "uri": "http://example.cz/Folder/Test_document_root-k55mct03",
        "name": "Test_document_root",
        "created": "2020-01-08T18:11:13.491+0000",
        "lastModified": "2020-01-08T18:11:13.491+0000",
        "description": null,
        "root": true
    }
}
```

Listing 3.3: Tělo odpovědi aplikace na požadavek ve formátu JSON

```
{
    "@id": "http://example.cz/Document/Test_document-k55mfopc",
    "@type": [
        "http://example.cz/Document",
        "http://example.cz/Node"
    ],
    "http://example.cz/description": "Document description",
    "http://example.cz/created": 1578507207888,
    "http://example.cz/lastModified": 1578507207888,
    "http://example.cz/name": "Test document",
    "http://example.cz/rootFolder": {
        "@id": "http://example.cz/Folder/Test_document_root-k55mfoqk",
        "@type": [
            "http://example.cz/Folder",
            "http://example.cz/Node"
        ],
        "http://example.cz/created": 1578507207932,
        "http://example.cz/lastModified": 1578507207932,
        "http://example.cz/isRoot": true,
        "http://example.cz/name": "Test_document_root"
    }
}
```

Listing 3.4: Tělo odpovědi aplikace na požadavek ve formátu JSON-LD

Kapitola 4

Analýza řešení správy uživatelů v systémech pro správu dokumentů

4.1 Rozpis technologií

Systémů pro správu dokumentů je velké množství, navíc ne vždy je tento systém samotná aplikace. Mnoho software pro správu informací nebo software na podporu spolupráce obsahuje nějaký systém pro správu dokumentů. Zaměřil jsem se hlavně na software, který je sám o sobě DMS, nicméně uvádím i několik výjimek.

Dále jsem se zaměřil převážně na open-source (nebo free) DMS. Při analýze bych se chtěl zaměřit hlavně na tyto aspekty:

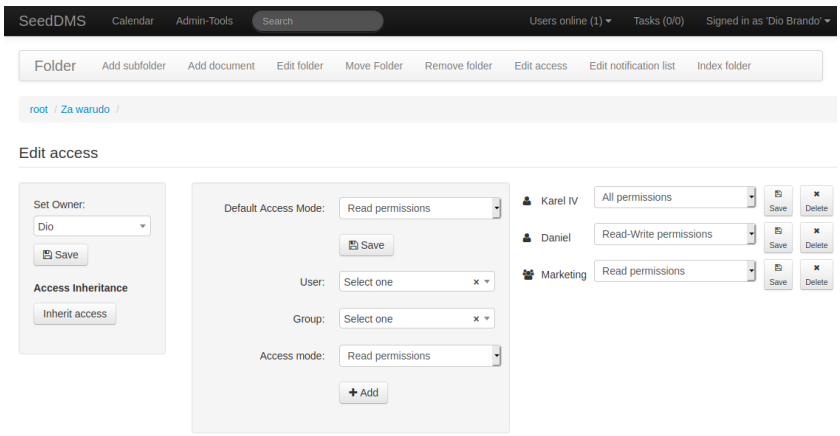
- Jaké globální role poskytuje systém po instalaci a jaké jsou jejich funkce
- Možnost vytváření vlastních rolí/skupin
- Zda-li je možné nastavit oprávnění individuálně u každého dokumentu, složky nebo souboru
- Pro jaké entity lze nastavovat oprávnění (uživatele, skupiny, role, atp.)
- Jaké typy oprávnění lze nastavovat (čtení, zápis, atp.) a jakým způsobem
- Podpora externí autentizace

4.1.1 Seed DMS

Seed DMS[1], dříve známý jako LetoDMS, je velmi jednoduchý document management system vyvíjený převážně jedním vývojářem. Zároveň je zcela zdarma a open-source.

V Seed DMS má každý uživatel přiřazenou roli a zároveň může být součástí několika skupin, které lze libovolně vytvářet uživateli v roli admin. Oprávnění lze nastavovat individuálně u každé složky a souboru. Zároveň po vytvoření nové podsložky se oprávnění automaticky dědí z nadsložky. U každé složky lze nastavit nejdříve základní oprávnění, které platí pro uživatele, jejichž

[1] `https://www.seeddms.org/` [navštíveno 2.1.2020]



Obrázek 4.1: Nastavování oprávnění pro složku v Seed DMS

oprávnění pro danou složku nebylo nijak specifikováno. Poté se specifikují oprávnění pro jednotlivé skupiny nebo uživatele. Je možné nastavit jedno ze čtyř dostupných oprávnění, a to sice No access, Read permissions, Read-Write permissions, All permissions (viz. 4.1). Úroveň All permissions má oprávnění jako úroveň Read-Write permissions, ale s možností upravovat oprávnění dané složky nebo souboru a jeho smazání. Pokud má uživatel oprávnění smazat složku, tak to může provést, i když se ve složce nachází složky nebo soubory, které smazat nemůže.

V Seed DMS najdeme tři role a to guest, user a admin. Uživatelé v roli guest mohou soubory pouze číst a nelze mu u složek nastavovat oprávnění. Ke složkám, kde je nastavené základní oprávnění No access, uživatelé v roli guest přístup nemají. Role admin má přístup ke všem souborům i složkám. Zároveň může vytvářet, měnit a mazat uživatele a skupiny. Prvního uživatele v sytému nelze smazat a je mu automaticky přiřazena role admin, nicméně jeho role jde změnit jiným uživatelem v roli admin. Další role nelze vytvářet. Seed DMS podporuje externí autentizaci přes protokol LDAP a službu Active Directory.

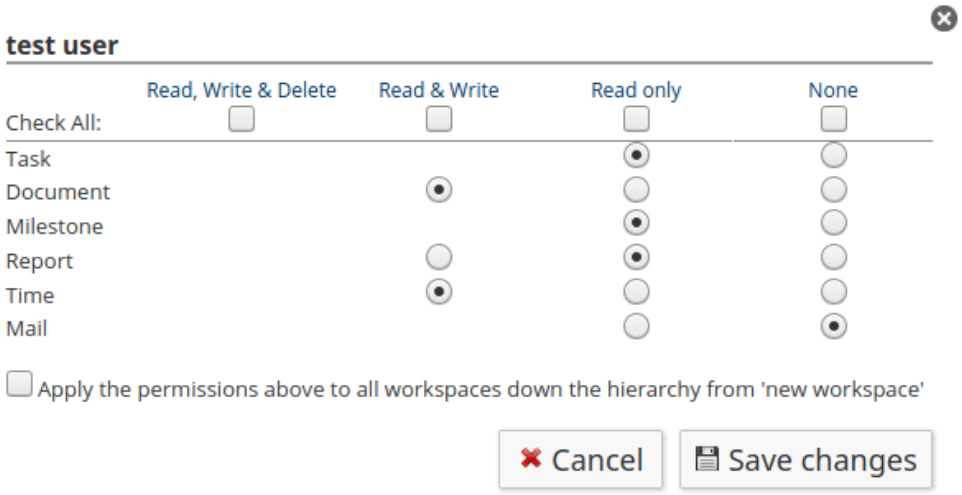
4.1.2 Feng Office

Feng Office[2] je společnost vyvíjející produkty pro zlepšení produktivity a týmové spolupráce. Jedním z jejich produktů je i aplikace pod stejným názvem, která obsahuje systém pro správu dokumentů.

Ve Feng Office jsou pracovní plochy (anglicky workspaces), ve kterých jsou moduly jako kalendář, úkoly a také modul dokumentů. U každého uživatele lze nastavit k jakým modulům má přístup a systémová oprávnění, kupříkladu zda-li může spravovat uživatele nebo oprávnění. Stejné nastavení je i u skupin, které lze libovolně vytvářet.

Co se týče rolí, tak zde najdeme tři hlavní globální role a to Full Access User, Collaborator User a Guest User. Každá role má pak své podrole, které

[2] `https://www.fengoffice.com` [navštíveno 2.1.2020]



Obrázek 4.2: Nastavování oprávnění uživatele pro pracovní plochu ve Feng Office. Uživatel test user je pouze Collaborator User a nelze mu udělit vyšší oprávnění.

dále specifikují, jaké má uživatel oprávnění. Full Access User má přístup k veškerému obsahu, ale nemusí mít například právo vytvářet, upravovat a mazat ostatní uživatele a skupiny. Collaborator User může přistupovat k obsahu a upravovat ho. Guest User může data jen prohlížet, podle podrole může i např. přidávat komentáře. Další role nelze vytvářet. [14]

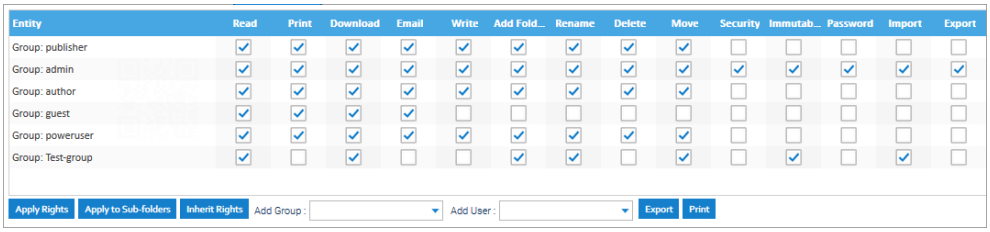
Modul dokumentů neobsahuje složky. Oprávnění se řeší na úrovni pracovních ploch, na které se nicméně dá pohlížet jako na složky, jelikož v pracovní ploše můžeme vytvářet další pracovní plochy. U každé pracovní plochy lze nastavovat různá oprávnění. Zde pro každého uživatele nebo skupinu lze nastavit jednu ze čtyř úrovní přístupu a to None, Read only, Read & Write, Read & Write & Delete s tím, že každou lze nastavit individuálně pro každý modul. Zároveň ale není možné nastavit oprávnění vyšší, než dovoluje role uživatele (viz. 4.2). Feng Office podporuje externí autentizaci přes LDAP. [13]

4.1.3 LogicalDoc

LogicalDoc[3] je proprietární DMS, nicméně má i Community Edition, která je zdarma a open-source. LogicalDoc je dostupný na Windows, Linux i macOS a má dokonce klientskou aplikaci pro Android a iOS. [2]

LogicalDoc Community Edition nemá role, nicméně má skupiny které jako role mohou fungovat. Po instalaci je v systému 5 skupin a to publisher, admin, author, guest a poweruser. Uživatelé ve skupině admin mají plná oprávnění ke všem souborům a složkám. Zároveň mohou vytvářet, upravovat a mazat ostatní uživatele. Výjimkou je uživatel v roli admin, který byl vytvořený při instalaci. Uživatelům ve skupině guest mohou být přiřazena pouze omezená oprávnění (viz. obrázek 3.3). U ostatních skupin lze u složek nastavovat libovolná oprávnění. U složek lze nastavit oprávnění i pro jednotlivé uživatele.

[3] `https://www.logicaldoc.com` [navštíveno 2.1.2020]



Obrázek 4.3: Nastavování oprávnění složky v LogicalDoc. Skupině admin nelze oprávnění měnit. U skupiny Guest jsou nastavená nejvyšší možná oprávnění.

Při vytvoření nové složky lze zvolit, aby oprávnění zdědila od nadsložky. Zároveň po vytvoření můžeme vybrat libovolnou složku, ze které zdědí její oprávnění.

Typů oprávnění je zde opravdu mnoho (viz. 4.3). kromě standardních oprávnění jako čtení a zápis má i kupříkladu právo na stažení dokumentu jako zip soubor, přejmenování dokumentu nebo jeho smazání. Jedno z nich je i oprávnění Security, které umožňuje uživateli nebo skupině nastavovat oprávnění dané složky.

Enterprise Edition přidává několik nových funkcí oproti Community edition. Mezi ně patří možnost integrace přes jiné aplikace jako Google Drive nebo Dropbox a možnost autentizace přes LDAP a Active Directory. [2]

4.1.4 Krystal DMS

Krystal DMS[4] je document management system vyvíjený společností PRIME-LEAF Consulting. Je dostupný ve čtyřech edicích včetně community edition. Enterprise Edition a Premium Edition podporuje LDAP i Active directory. [1]

Krystal DMS má globální role a skupiny, nicméně oprávnění se spravují přes tzv. ACL (Access-control list, česky seznam řízení přístupu) šablony, ve kterých se vybere množina rolí, skupin a uživatelů, které ovlivňuje. Následně se zvolí jaké oprávnění bude tato množina mít. Role, skupiny i ACL šablony se dají přidávat, upravovat a mazat. Tyto ACL šablony pak lze aplikovat na libovolný dokument k rychlému a snadnému nastavení práv. Krystal DMS nemá složky. Práva se nastavují pouze u jednotlivých dokumentů, kterým se říká Document Classes. Tyto šablony lze vytvářet administrátory. Role administrátora v systému není, tato vlastnost se nastavuje u jednotlivých uživatelů. Administrátoři zároveň mohou vytvářet, upravovat a mazat ostatní uživatele. [6]

4.1.5 OpenDocMan

OpenDocMan[5] je velmi jednoduchý open-source document management system, který je stejně jako SeedDMS vyvíjený převážně jedním vývojářem.

[4] `https://www.krystaldms.in` [navštíveno 2.1.2020]
[5] `https://www.opendocman.com/` [navštíveno 2.1.2020]

Vychází v několika edicích. Zaměřím se na jeho Community edici. [3]

V OpenDocMan nejsou role v tradičním smyslu. U každého uživatele lze nastavit, zda-li může upravovat a přidávat dokumenty. Dále lze nastavit, zda-li je admin, který může spravovat uživatele a skupiny, a zároveň má vždy plný přístup ke všem souborům. U každého souboru lze nastavit 5 úrovní oprávnění jak pro uživatele tak skupiny a to Forbidden, View, Read, Write a Admin.

U Forbidden se uživateli soubor nezobrazí v listu dokumentů, u View se mu pouze zobrazí, ale nelze s ním jakkoli interagovat. U oprávnění Read může uživatel zobrazit informace o dokumentu společně s jeho obsahem. Write umožňuje upravovat obsah souboru a Admin umožňuje upravovat informace o souboru společně s možností nastavovat jeho oprávnění. OpenDocMan nepodporuje externí autentizaci.

4.1.6 OpenKM

OpenKM[6] je správce dokumentů, který je zdarma a open source. Vychází ve dvou edicích a to Community Edition a Professional Edition. Obě edice podporují LDAP i Active directory. [4]

Community Edition po instalaci obsahuje 2 role admin a user (v systému ROLE_ADMIN a ROLE_USER). Další role lze vytvářet. Uživateli může být přiděleno více rolí najednou, nicméně vždy musí mít přiřazenou roli admin nebo user. Uživatelé, kteří jsou v roli admin mohou spravovat ostatní uživatele. V OpenKM lze nastavovat různá oprávnění pro individuální složky a dokonce i pro samotné soubory. Oprávnění se dají udělovat podle role a podle uživatele. Typy oprávnění jsou zde pouze čtyři a to read, write, delete a security. Stejně jako u LogicalDoc oprávnění security znamená možnost upravovat oprávnění dané složky nebo souboru.

4.2 Shrnutí

U všech vybraných DMS se vždy vyskytovaly tyto aspekty:

- Globální role, kde jedna z rolí je role admin, která má plná oprávnění u veškerého obsahu. Zároveň může vytvářet, upravovat a mazat uživatele systému
- Skupiny, které se daly spravovat uživateli v roli admin
- U entit, u kterých se dala nastavit oprávnění, se vždy mohla nastavovat oprávnění jak pro uživatele tak skupiny nebo role

V tabulce 4.1 porovnávám další vlastnosti jednotlivých systémů. Pokud lze nastavit každý typ oprávnění zvlášť (viz. 4.3), pak tento styl nazývám jednotlivý. Pokud se nastavuje jedna z možností, kde každá uděluje různá práva (viz. 4.2), pak tento styl nazývám úrovňový.

[6] `https://www.openkm.com/` [navštíveno 2.1.2020]

Název	Možnost tvořit globální role	U jakých entit lze nastavovat oprávnění	Styl oprávnění	Podpora LDAP a AC
Seed DMS	Ne	Složky a soubory	Úrovňový	Ano
Feng Office	Ne	Workspaces	Úrovňový	Jen LDAP
LogicalDoc	Ne	Složky	Jednotlivý	Ano
OpenKM	Ano	Složky a soubory	Jednotlivý	Ano
Krystal DMS	Ano	Dokumenty	Jednotlivý	Ano
OpenDocMan	Ne	Soubory	Úrovňový	Ne

Tabulka 4.1: Porovnání řešení v analyzovaných systémech pro správu dokumentů

Je zřejmé, že role nelze často vytvořit. Navíc i v případech, kdy se daly role vytvářet, role vytvořené uživatelem fungovaly spíše jako skupiny. V případě OpenKM musel mít uživatel přiřazenou jednu ze základních rolí, která určila jeho možnosti. Nicméně další přidané role nijak neovlivňovaly jaké má uživatel oprávnění. Podobně je tomu i u Krystal DMS, kde role šly vytvářet, ale fungovaly stejně jako skupiny.

Role, které se v systému nacházely po instalaci, se daly často kategorizovat do rolí guest, user a admin, kde guest mohl soubory maximálně prohlížet a user mohl soubory i upravovat, pokud k tomu měl oprávnění. Jediné výjimky byly systémy OpenKM a Krystal DMS. V OpenKM se každému uživateli přiřadila vždy role typu admin nebo user. To funguje podobně jako u Krystal DMS, kde někteří uživatelé měli roli typu user a někteří typu admin.

Dále bylo časté, že oprávnění šla nastavit u jednotlivých složek. Pro případy, kdy tomu tak nebylo, systém neměl možnost složky vytvářet a oprávnění se řešila jiným způsobem. U Feng Office se složky daly nahrazovat pracovními plochami, u kterých již oprávnění nastavit šlo. Pro Krystal DMS se oprávnění dala nastavovat pro dokumenty, které se dají představit jako složky v kořenovém adresáři bez podsložek. A u systému OpenDocMan byly všechny soubory v jednom dokumentu, nicméně bylo možné nastavit různá oprávnění pro každý soubor.

Oba styly oprávnění se vyskytovaly ve stejném počtu. Úrovňový styl se nicméně objevoval hlavně u systémů, které byly jednodušší. Naopak jednotlivý styl se objevoval u větších DMS s velkou škálou funkcionality.

Pokud systém používal jednotlivý styl oprávnění, tak měl vždy oprávnění typu read, write a delete. U systémů OpenKM a LogicalDoc se také vyskytovalo oprávnění security. V Krystal DMS byla oprávnění nastavovány pouze administrátory systému.

Podpora LDAP byla velmi častá. Jediná výjimka byla OpenDocMan, což je velmi jednoduchý systém. Ostatní systémy externí autorizaci podporovaly. Častá byla i podpora Active Directory. Je zřejmé, že se jedná o důležitou část systému.

Úroveň	Seed DMS	Feng Office	OpenDocMan
1	No Access	None	Forbidden
2	-	-	View
3	Read Permissions	Read Only	Read
4	Read-Write permissions	Read & Write	Write
5	-	Read & Write & Delete	-
6	Full Access	-	Admin

Tabulka 4.2: Porovnání úrovní v systémech, které používaly úrovňový styl

4.2.1 Porovnání typů oprávnění pro úrovňový styl

V tabulce 4.2 porovnávám, jaké typy oprávnění lze nastavit pro systémy, které používaly úrovňový styl oprávnění. Následně vysvětluji, co jednotlivé úrovně znamenají pro různé systémy.

Každá úroveň dědí vlastnosti z úrovně předchozí a přidává jí nová práva. V následujícím seznamu popisuji jaká nová práva každá vrstva přidává oproti té předchozí. Tuto skutečnost budu popisovat u souboru. Obdobná práva by platila i u složky nebo dokumentu. Práva tohoto souboru nastavujeme v Seed DMS a OpenDocMan přímo. U Feng Office se práva tohoto souboru nastaví z pracovní plochy, ve které se soubor nachází.

1. Soubor nelze zobrazit
2. Soubor jde vidět v seznamu mezi ostatními soubory
3. Obsah souboru lze číst
4. K souboru lze nahrávat nové verze
5. Soubor lze smazat
6. U souboru lze spravovat oprávnění

4.2.2 Další poznatky

V průběhu analýzy jsem zaznamenal několik dalších vlastností, které některé systémy sdílí. Pokud systém podporuje různá oprávnění pro jednotlivé složky, tak nabízí i možnost zdědit tyto vlastnosti z nadsložky a to buď trvale nebo jen při vytvoření.

Dále jsem zaznamenal, že pokud u nějaké entity bylo nastavené oprávnění pro nějakou skupinu a zároveň pro uživatele z této skupiny, tak bylo pro tohoto uživatele oprávnění skupiny ignorováno a aplikace brala v úvahu pouze jeho nastavení.

V každém systému je po instalaci vytvořený uživatel v roli typu admin. V některých systémech nebylo možné tohoto uživatele smazat a v některých případech ani změnit jeho roli. U systému Feng Office je role Super Administrator, který funguje jako admin a Administrator, který nemůže spravovat ostatní uživatele v roli Administrator ani Super Administrator.

Kapitola 5

Návrh vlastního modelu správy uživatelů

V této kapitole navrhnu systém pro správu uživatelů, který bude brát v úvahu model dosavadního systému a informace získané z analýzy.

5.1 Udělování oprávnění

5.1.1 Možnosti nastavení oprávnění pro různé entity

V aplikaci Document Manager jsou dokumenty a složky. Dokument obsahuje kořenovou složku, která obsahuje soubory a podsložky. Oprávnění by bylo možné spravovat u dokumentů, složek a souborů.

V analýze měly všechny systémy, které podporovaly vytváření složek, možnost nastavit různá oprávnění pro všechny složky. Na druhou stranu systému Krystal DMS stačily pouze dokumenty a udělování práv pro ně. Podobně systém pro správu dokumentů ve Feng Office měl úložiště pro každý workspace, u kterého se nastavovala práva.

V modelu tedy bude možné nastavovat oprávnění pouze u dokumentů. Dle mého názoru je užitečné nastavovat práva i u složek, avšak to není potřebné. Zároveň bude jednodušší určit, kdo má přidělená jaká oprávnění.

5.1.2 Styl oprávnění

Je zde na výběr z úrovňového a jednotlivého stylu. Případně bych mohl navrhnout ještě zcela jiný. Jednotlivý styl udělování oprávnění má oproti úrovňovému více možností, nicméně ne zdaleka všechny dávají smysl. Jedním z příkladů je uživatel, který může přepisovat soubor, ale nemůže ho číst.

Pro svůj model zvolím úrovňový styl pro jeho jednoduchost. Na rozdíl od jednotlivého stylu není tolik omezující a je používaný i v praxi. Zároveň je přehlednější a bude snadnější určit jaké má uživatel práva.

5.1.3 Typy oprávnění

Zde jsem zvolil stejné typy oprávnění jako u systému Seed DMS. Bylo by možné ještě přidat úroveň View, kdy budou dokumenty pouze vidět, ale nebude možné zobrazit jejich obsah. Nicméně tato varianta se objevovala

pouze u systému OpenDocMan a dle mého názoru není tolik užitečná. Dále bych umožnil, aby mohlo být uživatelům přiřazeno právo měnit oprávnění u daného souboru. Aplikace Feng office byla výjimkou, ale zároveň měla deset rolí namísto tří a tento problém se dal řešit na úrovni rolí.

Úrovně definuji:

- None - K dokumentu není přístup a nezobrazuje se v seznamu dokumentů
- Read - Dokument se zobrazuje v seznamu a lze číst jeho veškerý obsah
- Write - Práva z úrovně Read jsou zděděna a navíc se pro soubory v dokumentu mohou nahrávat nové verze
- Security - Práva z úrovně Write jsou zděděna a navíc lze soubory a složky v dokumentu mazat. U dokumentu lze nastavovat oprávnění pro ostatní uživatele. Zároveň bude možné měnit informace o složkách a dokumentu samotném.

5.2 Skupiny

Z analýzy je zřejmé, že funkce skupin je důležitá část systému. I ty nejjednodušší systémy měly podporu pro skupiny a udělování práv skupinám.

Do svého modelu zavedu stejnou funkcionalitu. Skupina bude určitá množina uživatelů. Každý uživatel může být ve více skupinách, nebo v žádné.

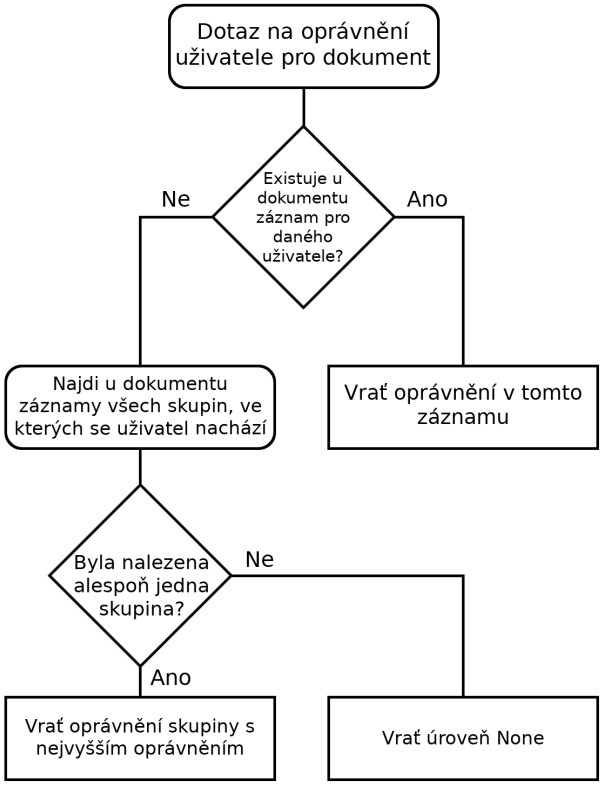
5.3 Seznam řízení přístupu

Každý dokument bude mít seznam řízení přístupu, ve kterém budou záznamy určující úroveň oprávnění uživatelů a skupin. Způsob volení oprávnění je popsán v rozhodovacím stromě na obrázku 5.1. Když uživatel přistupuje k dokumentu, nejdříve se bude hledat záznam pro tohoto uživatele. Pokud takový záznam existuje, tak se použije. Pokud takový záznam neexistuje, vyberou se záznamy všech skupin, ve kterých se uživatel nachází. Následně se zvolí ta, ve které má nejvyšší oprávnění. Pokud se žádná taková skupina nenajde, tak bude uživateli automaticky přidělena úroveň oprávnění None.

5.4 Role

Z analýzy je zřejmé, že je důležité, aby správce dokumentů obsahoval roli administrátora, který systém spravuje. Aby mohl tuto funkci plnit, má plná práva ke všem souborům v systému a zároveň může spravovat ostatní uživatele. Dále se v některých systémech objevovala role typu guest, kde uživatelé v této roli mohli maximálně zobrazovat dokumenty, složky nebo soubory. Avšak tuto roli lze nahradit pomocí funkcionality skupin.

Ve svém modelu zavedu dvě role a to user a admin. Každému uživateli bude přiřazena jedna z těchto rolí. Uživatelé v roli admin budou mít oprávnění



Obrázek 5.1: Rozhodovací strom ukazující, jak se volí oprávnění u dokumentu pro daného uživatele.

jako jediní vytvářet, upravovat a mazat uživatele. Zároveň mohou vytvářet skupiny a uživatele do skupin přidávat a odebírat je. Uživatelé budou moci vytvářet nové dokumenty. Když uživatel vytvoří dokument, tak se do seznamu řízení přístupu pro tento dokument přidá tomuto uživateli oprávnění Security. Uživatelelé v roli admin dostanou automaticky oprávnění Security nezávisle na záznamech u dokumentů.

5.5 Podpora externí autentizace

Jako způsob externí autentizace přidám podporu pro externí autentizační službu, kterou poskytuje skupina znalostních softwarových systémů ČVUT FEL. Tato služba k autorizaci používá JSON Web Tokeny [1], které slouží k přenosu informací ve formátu JSON. Tyto tokeny mohou být digitálně podepsány nebo dokonce zašifrovány. Kvůli těmto vlastnostem jsou převážně používány pro autorizaci. Tato autorizační služba používá pouze podepsané tokeny.

[1] `https://jwt.io/` [navštíveno 25.3.2020]

Kapitola 6

Implementace modelu správy uživatelů

Document Manager již obsahuje strukturu k autentizaci požadavků přes externí autorizační službu, nicméně pokud je uživatel autentizován, pak má automaticky přístup ke všem funkcím aplikace.

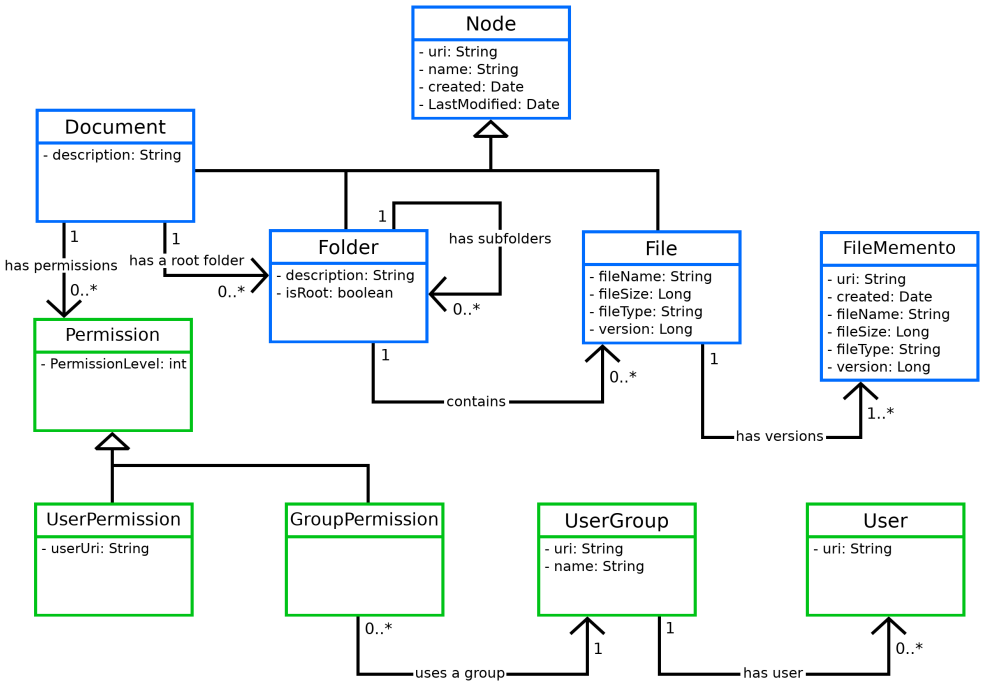
K přidání podpory správy uživatelů jsem upravil aplikaci na několika vrstvách. Na databázové vrstvě je potřeba přidat nové entity pro skupiny a oprávnění dokumentů. Na zbylých vrstvách bude potřeba přidat nové třídy a funkce, které s novými entitami pracují. Také je potřeba upravit konfiguraci aplikace tak, aby přeposílala autentizační token autorizační službě.

6.1 Úprava modelu

Jelikož se uživatelé spravují přes externí službu, v databázi budou uloženy pouze identifikátory daných uživatelů, v tomto případě jejich URI. Oprávnění souborů a složek se odvíjí od dokumentu, ve kterém se nachází. Je dostačující, aby informace o oprávnění byly v relaci pouze s entitou Document. Do modelu (viz. 6.1) jsem přidal entitu UserGroup, která bude fungovat jako skupina uživatelů. Dále jsem vytvořil entity GroupPermission a UserPermission, které slouží k udělování oprávnění dokumentům. Obě dědí vlastnosti od abstraktní entity Permission, která obsahuje úroveň oprávnění a relaci s entitou Document. UserPermission přidává atribut URI uživatele, kterému je daná úroveň přidělena. GroupPermission přidává relaci se skupinou, pro kterou je daná úroveň oprávnění přidělena. Úroveň oprávnění je reprezentovaná jako Enum, je nazvána PermissionLevel a obsahuje všechny čtyři úrovně oprávnění, tedy None, Read, Write a Security.

6.2 Autorizace požadavků

Pokud chce být uživatel autorizován, musí požadavek odeslaný aplikaci obsahovat token získaný od autorizační služby. Tento token obsahuje informace o uživateli, který ho vlastní a také dobu, kdy je token platný. Po přijmutí požadavku aplikace token přeposílá autorizační službě, aby se zjistilo, zda je validní. Pokud token není ve zprávě obsažen, nebo není validní, aplikace vrátí odpověď s kódem 401 Unauthorized. Pokud je token validní, autori-



Obrázek 6.1: Upravený model aplikace. Zelené entity jsou nově přidané entity.

zační služba vrátí informace o příslušném uživateli a aplikace je uloží do třídy SecurityContext, která je součástí frameworku Spring a je přístupná z libovolné třídy. Po zpracování požadavku se informace o uživateli z třídy SecurityContext vymažou. To znamená, že pokud chce uživatel provést další akci, tak musí i další požadavek obsahovat autorizační token.

6.3 Úpravy na kontrolní a servisní vrstvě

Na servisní vrstvě jsem upravil službu pro správu dokumentů tak, aby při vytvoření nového dokumentu bylo pro daný dokument vytvořeno nové uživatelské oprávnění. To bude obsahovat identifikátor uživatele, který požadavek poslal a bude mít nastavenou nejvyšší úroveň oprávnění. Také byly přidány nové služby pro správu oprávnění a uživatelských skupin.

Na kontrolní vrstvu bylo přidáno rozhraní pro správu oprávnění a uživatelských skupin. Autorizace požadavků je řešena pomocí anotace PreAuthorize poskytnutou frameworkem Spring. Ta má jako argument pravdivostní výrok v jazyce SpEL (Spring Expression Language). Pokud je výrok pravdivý, pak je funkce spuštěna. Pokud není výrok pravdivý, uživatel není autorizován ke spuštění dané funkce a aplikace vrátí odpověď s kódem 403 Forbidden. Tento výrok může obsahovat argumenty, které funkce obdržela a dokonce spouštět funkce definované v jiných třídách.

Cílem je, aby všechny požadavky, s výjimkou vytváření nového dokumentu a zobrazení seznamu dokumentů, byly před vykonáním autorizovány dle informace o uživateli ve třídě SecurityContext. Například pokud uživatel

chce zobrazit obsah souboru, tak aplikace ověří, že k dokumentu, ve kterém se soubor nachází, má uživatel oprávnění alespoň Read. Aby uživatel mohl spravovat uživatelské skupiny, tak musí být v roli admin. Zároveň pokud je uživatel v roli admin, je autorizován ke všem požadavkům.

K usnadnění psaní výroků pro anotaci PreAuthorize byla vytvořena služba AuthorizationService, která obsahuje funkce pro všechny entity, u kterých se určuje oprávnění podle jejich příslušného dokumentu. Oprávnění určuje podle uživatele, který je uložen ve třídě SecurityContext. Ve výroku se dále používá funkce hasRole definovaná frameworkem Spring, která zjišťuje, zda má uživatel uložený ve třídě SecurityContext danou roli.

Výsledný výrok je kombinací funkce hasRole a funkce z AuthorizationService (viz. 6.1). Jedinou výjimkou jsou požadavky spravující uživatelské skupiny, které používají pouze funkci hasRole.

Listing 6.1: Výňatek ze třídy DocumentController zobrazuje funkci getDocument, která vrací dokument dle URI. Před funkcí je anotace PreAuthorize, která ověřuje, zda je uživatel v roli admin, nebo má k danému dokumentu úroveň oprávnění alespoň Read.

```
1 @GetMapping(path = "/{fragment}")
2 @PreAuthorize("hasRole('ROLE_ADMIN') or
      @authorizationService.isAuthorizedForDocument(#fragment,
      #namespace, 'READ')")
3 public Document getDocument(@PathVariable String fragment,
      @RequestParam(name = "namespace", required = true) String
      namespace) {
4    return documentService.findByURI(ResolveUri(fragment, namespace));
5 }
```

Kapitola 7
Testování aplikace

Aplikace byla testována pomocí testovacích scénářů frameworku JUnit a jeho rozšíření Mockito. Framework JUnit umožňuje psát automatické testy v jazyce Java. Hlavní funkce knihovny Mockito je, že umožňuje vytvářet tzv. mock třídy. Tyto třídy se pro zbytek programu jeví jako jejich nemockované ekvivalenty, nicméně chování jejich funkcí je explicitně určeno vývojářem. Zároveň lze sledovat jaké funkce se volaly a kolikrát. Dále jsem aplikaci otestoval pomocí programu Postman[1], který slouží k testování webových služeb.

7.1 JUnit testy

Během testů se aplikace testuje pouze pro profil *sem*. DAO používané v profilu JPA je implementované frameworkem Spring a dá se předpokládat, že funguje správně. Používání pouze jednoho ze dvou Data Access Objektů by mělo být k otestování aplikace dostačující, jelikož v obou profilech perzistentní vrstva poskytuje stejnou funkcionalitu, pouze pro jiný typ databáze. Při běhu testů se používá databáze RDF, která je ukládána do paměti, což umožňuje knihovna JOPA.

7.1.1 Funkce modelu

Některé třídy modelu obsahují pomocné funkce, které slouží k usnadnění práce s danou třídou. Například třída Document obsahuje funkci, která pro dané URI uživatele vrátí jeho úroveň oprávnění. Na této vrstvě jsem napsal několik jednoduchých testů, které tyto pomocné funkce testují.

7.1.2 Perzistentní vrstva

Pro DAO, používané v profilu *sem*, jsem napsal několik jednoduchých testů. Testoval jsem zda správně funguje kaskádování při ukládání a mazání dokumentů. Testy jsem psal pouze pro entity Document a Folder. Pro zbylé entity jsem testy nepsal, jelikož implementace DAO se pro všechny entity téměř neliší.

[1] `https://www.postman.com/` [navštíveno 25.3.2020]

Dále jsem napsal několik testů pro FileStorageDao, které zajišťuje ukládání souborů do file systému.

7.1.3 Servisní vrstva

Pro servisní vrstvu bylo potřeba pracovat se třídou SecurityContext, aby například mohla služba pro správu dokumentů vytvořit dokument a přidat k němu oprávnění uživatele z této třídy. Mohl jsem zde vytvářet mock tříd z perzistentní vrstvy, což jsem neudělal z důvodu jednoduchosti jejich implementace. Testuje se zde nejen, že služby fungují pro správný vstup, ale i že házejí výjimky, pokud funkcím nejsou poskytnuty povinné informace, nebo pokud nejsou dané informace korektní.

7.1.4 Kontrolní vrstva

Testy na kontrolní vrstvě používají knihovnu Mockito k tomu, aby vytvořily mock třídy ze servisní vrstvy a izolovaly kontrolní vrstvu od zbytku aplikace. Spring navíc poskytuje framework MockMVC, který umožňuje posílat požadavky aplikaci, jako kdyby pocházely z externí aplikace.

Testovány byly všechny ovladače a téměř všechny testy v tomto případě dodržovaly stejnou strukturu. V každém testu se používala mock třída AuthorizationService a mock její příslušné služby, tedy například testy pro DocumentController používaly mock třídy DocumentService. Na začátku testu vytvoří informace o běžném uživateli a o uživateli v roli admin. Přes MockMVC byly v aplikaci poslány 3 požadavky, které se lišily pouze daným uživatelem a očekávaným výstupem.

První požadavek je poslán běžným uživatelem, nicméně třída AutorizationService je nastavena tak, aby každému uživateli nedávala žádná oprávnění. Kontroluje se, zda aplikace vrací kód 403 Forbidden, tedy že uživatel nemá k danému zdroji přístup. Bez změny chování AutorizationService je poslán stejný požadavek uživatelem v roli admin. Předpokládá se, že aplikace vrátí status 200 OK. Než se pošle poslední požadavek, změní se chování AutorizationService tak, aby libovolnému uživateli dala takovou úroveň oprávnění. Následně běžný uživatel pošle stejný požadavek a kontroluje se, že aplikace vrátí status 200 OK. V případech, kdy aplikace vrací nějaký obsah v JSON-LD, testuje se, zda-li obsahuje správné informace.

7.1.5 Pokrytí kódu

Napsal jsem celkem 159 JUnit testů, 11 z nich pro funkce modelu, 12 pro perzistentní vrstvu, 102 pro servisní vrstvu a 34 pro kontrolní vrstvu. K analýze pokrytí kódu byla použita knihovna Jacoco [2].

Pokrytí kódu může být měřeno různými způsoby. Class coverage (pokrytí tříd) popisuje, kolik tříd bylo testováno. Method coverage (pokrytí metod) popisuje, kolik funkcí bylo testováno. Line coverage (pokrytí řádek) popisuje,

[2] `https://github.com/jacoco/jacoco` [navštíveno 25.3.2020]

Element	Class, %	Method, %	Line, % ▲	Branch, %
service	100% (7/7)	98% (50/51)	98% (370/375)	92% (156/168)
util	100% (2/2)	100% (13/13)	97% (34/35)	80% (8/10)
model	92% (13/14)	88% (129/146)	84% (287/339)	66% (79/118)
controller	100% (6/6)	90% (40/44)	83% (62/74)	100% (0/0)
documentmanager	78% (41/52)	84% (280/333)	80% (901/1116)	81% (263/324)
repository	100% (8/8)	91% (42/46)	68% (137/199)	76% (20/26)
exception	42% (3/7)	27% (3/11)	27% (6/22)	100% (0/0)
conf	20% (1/5)	14% (2/14)	9% (4/43)	0% (0/2)
security	50% (1/2)	16% (1/6)	3% (1/26)	100% (0/0)

Obrázek 7.1: Statistiky získané pomocí knihovny Jacoco. Elementy představují různé balíčky aplikace. Element documentManager představuje celou aplikaci, tedy všechny balíčky dohromady.

kolik řádek kódu bylo testováno. Branch coverage (pokrytí větví) popisuje, kolik větví bylo testováno. Termínem Branch (větev) je míněno rozdělení běhu programu podmínkou (if, switch, atp.). Vysoký branch coverage znamená, že funkce byly testovány vícekrát s různými vstupy, aby došlo k různým vyhodnocením v podmínkách dané funkce.

Výsledky měření ukazují, že celá aplikace měla dohromady 78% class coverage, 84% method coverage, 80% line coverage a 81% branch coverage. Detailní výsledky jsou na obrázku 7.1.

Pravděpodobně nejdůležitější balíček je balíček service, ve kterém se nachází třídy servisní vrstvy aplikace. Tento balíček obsahuje nejen nejvíce řádek kódu a také nejvíce prostoru pro chyby, protože obsahuje buisness logiku aplikace. Balíček service má pokrytí řádků 98%, což je nejvíce ze všech balíčků. Další důležitý balíček je balíček controller, který obsahuje třídy kontrolní vrstvy a má pokrytí řádků 80%.

Balíček repository obsahuje třídy DAO pro RDF databázi. Jelikož byly třídy používány při testování servisní vrstvy, tak je Jacoco uznává z části jako pokryté a proto má balíček repository 68% pokrytí řádek. Podobně balíček util, obsahující pomocné třídy použitelné v různých vrstvách aplikace, nebyl přímo testován, ale třídy tohoto balíčku byly použity v jiných testech. V balíčku model je většina pokrytí řádků způsobena používáním těchto tříd v testech různých vrstev.

Balíček conf obsahující konfiguraci aplikace měl velmi nízké pokrytí řádek. Pro JUnit testy byl vytvořen vlastní konfigurační balíček, který nahrazoval většinu tříd z balíčku conf, v důsledku čehož nebyly při běhu testů třídy z tohoto balíčku používány. Výjimky obsažené v balíčku exception také nebyly testovány, kvůli trivialitě jejich implementace. Některé z nich byly v JUnit testech použity, což zapříčiňuje 27% pokrytí řádků.

7.2 Testy s pomocí externí aplikace

Na závěr jsem provedl několik testů pomocí programu Postman. JUnit testy se otestovaly různé části aplikace, ale přes Postman se otestuje aplikace jako

celek. Testy se skládají z několika požadavků, které jsou aplikaci poslané po sobě, čímž se také otestuje, zda jsou různé části aplikace správně propojeny.

7.2.1 Test vytváření dokumentu, souboru a jeho nových verzí

V každém požadavku se posílá token uživatele získaného od externí autorizační služby. Celý test probíhá takto:

1. Aplikaci se pošle požadavek na vytvoření nového dokumentu. V těle požadavku napsaném v JSON-LD je specifikován identifikátorem URI, název a popisek dokumentu společně s názvem, popiskem a URI jeho kořenové složky. Tělo odpovědi je stejné jako požadavku, ale obsahuje navíc informace vytvořené aplikací.
2. Další požadavek slouží k získání seznamu dokumentů, ke kterým má uživatel přístup. Jelikož se uživatel nezměnil, tak má uživatel k nově vytvořenému dokumentu přístup a zobrazí se v seznamu dokumentů.
3. Třetí požadavek vytvoří v kořenovém adresáři dokumentu nový soubor. V požadavku se nachází soubor, který se má nahrát, jeho identifikátor a název. V těle odpovědi je JSON-LD s metadaty o daném souboru, které se uložily do databáze.
4. Pro testovací účely se pošle požadavek na stáhnutí právě nahraného souboru.
5. V dalším požadavku se k souboru nahraje nová verze. Aplikace vrátí uložená metadata o nové verzi souboru.
6. Další požadavek slouží k potvrzení, že metadata o obou verzích souborů se uložily. Pošle se dotaz na všechny verze souboru. Aplikace vrátí JSON-LD, ve kterém jsou metadata o obou verzích.
7. Znovu se odešle požadavek na stáhnutí aktuální verze souboru. V odpovědi bude nově nahraná verze souboru.
8. V závěru testu dojde k potvrzení, že se v databázi stále vyskytuje původní verze souboru a pošle se i požadavek na stáhnutí této verze souboru.

7.2.2 Test uživatelských oprávnění

Požadavky v tomto testu posílají různý token. Používají se tokeny uživatelů User1 a User2. Poslané požadavky uživatelů obsahují i jejich příslušný token.

1. Uživatel User1 pošle požadavek o vytvoření nového dokumentu stejným způsobem jako v předchozím příkladu.
2. Stejný uživatel pošle požadavek na zobrazení všech dokumentů, ke kterým má přístup. Aplikace vrátí seznam dokumentů, ve kterém se nově vytvořený dokument nachází, jelikož User1 je jeho vlastníkem.

3. Stejný požadavek pošle i User2, který však v seznamu dokumentů tento dokument neuvidí. U nyní vytvořeného dokumentu je oprávnění definované pouze pro autora dokumentu a User2 má dle výchozího chování aplikace oprávnění None.

4. Uživatel User1 přidá k novému dokumentu uživatelské oprávnění Read pro uživatele User2. Pošle aplikaci JSON, ve kterém je URI uživatele User2 a úroveň oprávnění Read na adresu uživatelských oprávnění nově vytvořeného dokumentu.

5. User2 opět pošle požadavek na seznam dokumentů, ale nyní se zobrazí dokument vytvořený uživatelem User1.

6. Uživatel User2 pošle požadavek na vymazání nového dokumentu, nicméně má pouze oprávnění Read a k této akci nebude autorizovaný.

7. User1 změní oprávnění vytvořené pro uživatele User2 na úroveň Security.

8. User2 opět pošle požadavek na smazání dokumentu a nyní se dokument smaže, jelikož User2 má k tomuto dokumentu oprávnění Security.

7.2.3 Test skupinových oprávnění

Oproti předchozímu testu se zde vyskytuje další uživatel Admin, který, jak název napovídá, je v roli admin. Admin ve svých požadavcích opět posílá svůj autorizační token. Tento scénář je vizualizován na obrázku 7.2;

1. User1 opět vytvoří nový dokument.

2. User2 pošle požadavek na seznam dokumentů. Dokument vytvořený uživatelem User1 se nezobrazí.

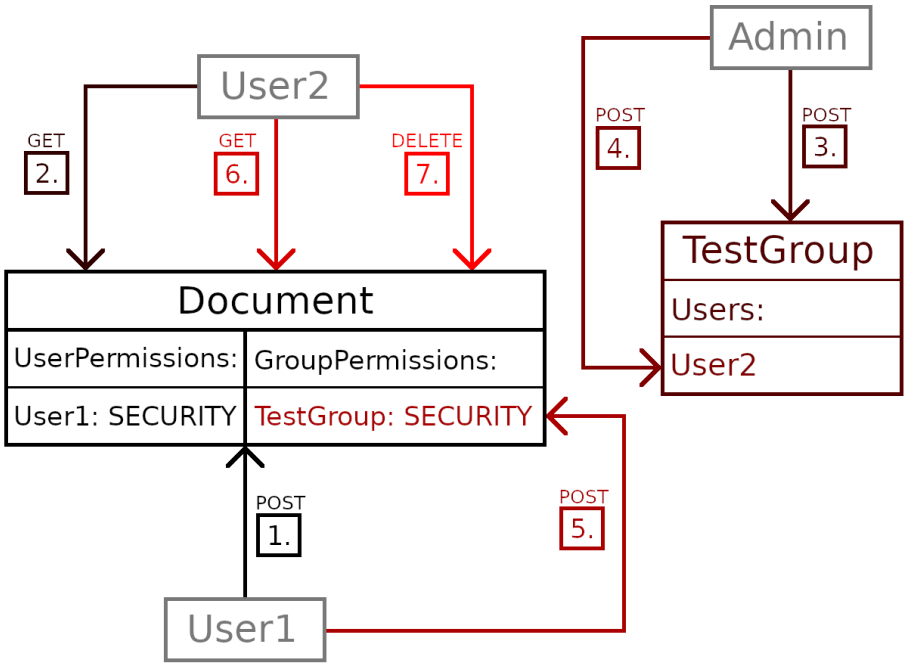
3. Admin vytvoří novou skupinu. V požadavku uvede URI a název skupiny. Pojmenuje jí TestGroup.

4. Admin následně do skupiny přidá URI uživatele User2. Aplikace potřebuje znát pouze URI uživatele a přijímá tělo ve formátu String a ne JSON.

5. User1 přidá k dokumentu skupinové oprávnění pro skupinu TestGroup a udělí jí oprávnění Security.

6. Uživatel User2 opět pošle požadavek na seznam dokumentů, avšak nyní se dokument vytvořený uživatelem User1 zobrazí v seznamu.

7. User2 pošle požadavek na smazání dokumentu a jelikož je ve skupině TestGroup, je autorizován a dokument se smaže.



Obrázek 7.2: Ilustrace scénáře Test skupinových oprávnění.

7.3 Propojení s aplikací TermIt

Aplikace Document Manager byla úspěšně spojena s aplikací TermIt, která je nyní schopna posílat požadavky aplikaci Document Manager. Požadavky, které posílá, využívá tyto funkce z kontrolní vrstvy aplikace Document Manager:

- createDocument - vytváří nový dokument
- uploadFileToDocument - nahrává do dokumentu s daným URI nový soubor
- getFileContent - vrací obsah souboru s daným URI
- getFile - vrací metadata souboru s daným URI
- deleteFile - smaže soubor s daným URI

TermIt na rozdíl od aplikace Document Manager nepoužívá složky, proto se požadavky týkají pouze souborů a dokumentů. Zbylá funkcionalita aplikace Document Manager se spravuje přes její aplikační rozhraní.

Kapitola 8

Závěr

Aplikaci Document Manager jsem přidal podporu sémantických technologií webu bez negativního ovlivnění původní implementace. Po analýze šesti různých systémů pro správu dokumentů jsem navrhl model pro správu uživatelů, který byl implementován do aplikace Document Manager. Pro aplikaci jsem napsal JUnit testy a provedl testy přes externí aplikaci Postman.

Na obrázku 8.1 je zobrazena výsledná strukturu aplikace v souborovém systému. Původní verze aplikace obsahovala celkem 1461 řádků kódů. Po všech úpravách a přidaných testech má aplikace celkem 8021 řádků, 4075 z toho jsou JUnit testy.

Samotné přidání nového DAO by aplikaci výrazně nezměnilo, avšak kvůli změně identifikátoru na URI byla aplikace významně upravena na všech vrstvách. Způsob přijímání požadavků, ukládání souborů, kontrola výjimek a další části aplikace musely být upraveny pouze kvůli této změně. Přidání správy uživatelů bylo časově náročné, nicméně nebylo potřeba měnit dosavadní implementaci. Při psaní JUnit testů se pouze výjimečně našly chyby v implementaci, jelikož při vývoji byla aplikace běžně testovaná přes Postman.

Díky této práci jsem se naučil programovat ve frameworku Spring a používat nové knihovny. Získal jsem vědomosti o sémantickém webu a enterprise aplikacích. Při úpravě aplikace Document Manager jsem získal cenné zkušenosti s vývojem větších aplikací a také s vývojem aplikací ve frameworku Spring.

Do aplikace může být dále přidáno uživatelské prostředí nebo externí autorizace přes LDAP. Také by bylo možné přidat roli guest, jelikož se v analyzovaných DMS často vyskytovala. Nemělo by být obtížné přidat možnost různých oprávnění pro dokumenty, složky a soubory.

Obrázek 8.1: Výsledná struktura aplikace. Černě označené soubory nebyly upraveny nebo byly provedeny pouze jednoduché úpravy (méně než 5 upravených řádek). Modré soubory byly upraveny složitěji. Zelené soubory jsou nově přidané soubory.

Příloha A

Literatura

[1] KRYSTAL DMS - Document Management System - Edition Comparisons. https://www.krystaldms.in/comparison. accessed 2020-1-2.

[2] LogicalDoc Features. https://www.logicaldoc.com/features. accessed 2020-1-2.

[3] OpenDocMan Document Management System Pricing. https://www.opendocman.com/document-management-system-pricing/. accessed 2020-12-30.

[4] OpenKM Knowledge Center. https://docs.openkm.com. accessed 2020-12-30.

[5] W3C semantic web standart. https://www.w3.org/standards/semanticweb. accessed 2020-1-2.

[6] KRYSTAL DMS - Enterprise Edition - 2019 - Administrators Guide. https://www.krystaldms.in/resources/documentation/enterprise/2019/admin, 2019. accessed 2020-12-30.

[7] Javier D. Fernández. Binary RDF Representation for Publication and Exchange, W3C Member Submission. https://www.w3.org/Submission/2011/SUBM-HDT-20110330/, March 2011. accessed 2020-1-2.

[8] Martin Fowler. *Patterns of Enterprise Application*. Addison-Wesley Professional, 1st edition, 2004.

[9] Hans Rohnert Peter Sommerlad Michael Stal Frank Buschmann, Regine Meunier. *Pattern-Oriented Software Architecture*. Wiley, 1996.

[10] Dave Longley Gregg Kellogg, Pierre-Antoine Champin. JSON-LD 1.1, A JSON-based Serialization for Linked Data, W3C Candidate Recommendation. https://www.w3.org/TR/json-ld/, December 2019. accessed 2020-1-2.

[11] Dave Longley Gregg Kellogg, Pierre-Antoine Champin. JSON-LD 1.1, A JSON-based Serialization for Linked Data, W3C Proposed Recommendation. https://www.w3.org/TR/json-ld11, December 2019. accessed 2020-1-2.

[12] Martin Malinov. Systém pro správu dokumentů, souborů a datových zdrojů. https://dspace.cvut.cz/handle/10467/82831, May 2019. accessed 2019-12-21.

[13] Natalia Martinez. Feng Office - LDAP Authentication. http://www.fengoffice.com/web/wiki/doku.php/ldap, January 2015. accessed 2020-1-2.

[14] Natalia Martinez. Feng Office - Permissions by User Type. http://www.fengoffice.com/web/wiki/doku.php/permissions:user_roles, September 2015. accessed 2020-1-2.

[15] Markus Lanthaler Richard Cyganiak, David Wood. RDF 1.1 Concepts and Abstract Syntax, W3C recommendation. https://www.w3.org/TR/rdf11-concepts, February 2014. accessed 2020-1-2.

[16] Mark Richards. Software architecture patterns. https://www.oreilly.com/ideas/software-architecture-patterns/page/2/layered-architecture. accessed 2020-1-2.

[17] Vaibhav Singh. Should I use DB to store file ? https://medium.com/@vaibhav0109/should-i-use-db-to-store-file-410ee22268c7, July 2019. accessed 2020-12-30.

[18] Andy Seaborne Steve Harris. SPARQL 1.1 Query Language, W3C recommendation. https://www.w3.org/TR/sparql11-query, March 2013. accessed 2020-1-2.

[19] Abuthahir Sulaiman. File System vs. Database. https://dzone.com/articles/which-is-better-saving-files-in-database-or-in-fil, April 2017. accessed 2020-12-30.

[20] Craig Walls. *Spring in Action*. Manning Publications, 4th edition, 2014.

Příloha B

Seznam použitých zkratek

- **DAO** - Data Access Object
- **DMS** - Document Management System
- **JPA** - Java Persistence API
- **JSON** - JavaScript Object Notation
- **LDAP** - Lightweight Directory Access Protocol
- **SQL** - Structured Query Language
- **URI** - Uniform Resource Identifier
- **IRI** - Internationalized Resource Identifier
- **RDF** - Resource Description Framework
- **SPARQL** - Simple Protocol and RDF Query Language
- **OWL** - Web Ontology Language
- **JSON-LD** - JavaScript Object Notation for Linked Data

Příloha C

Obsah přiloženého CD

CD přiložené k práci má následující obsah:

- `/`
  - `latex-source` ..... zdrojové kódy zprávy bakalářské práce
  - `semantic-document-manager` ..... hlavní adresář aplikace
    - `src`
      - `main`
        - `java` ..... adresář se zdrojovými soubory java
        - `resources`
          - `application.properties` ..... konfigurační soubor
      - `test` ..... adresář s automatickými testy JUnit
    - `SDM-postman-collection.json` ..... kolekce postman testů
    - `pom.xml`
  - `jarosm11-BP2020.pdf` ..... bakalářská práce ve formátu PDF

Příloha D

Návod ke spuštění

Ke spuštění je třeba mít nainstalované následující technologie:

- Java 8
- Apache Maven 3 (či novější)

D.1 Kompilace a spuštění aplikace

Před kompilací je potřeba nastavit konfigurační soubor aplikace, který se nachází v adresáři *application\semantic-document-manager\src\main\resources*. Pro účely testování výchozí konfigurace vytváří relační databázi v paměti a RDF databázi v souborovém systému.

- K připojení vlastní SQL databáze je potřeba odkomentovat položky pod sekcí *External database*, zakomentovat položky pod sekcí *Local file database* a nastavit adresu SQL databáze společně se jménem a heslem uživatele.
- K připojení vlastní RDF databáze je potřeba změnit položku *repositoryUrl* v sekci *Spring DATASOURCE - SEM*.
- Profil, ve kterém aplikace poběží, se definuje v položce *spring.profiles.active*. V profilu *sem* aplikace používá RDF databázi a v profilu *jpa* používá SQL databázi.

Poté se aplikace zkompiluje a spustí takto:

1. V hlavním adresáři projektu otevřete příkazový řádek a zadejte příkaz **mvn clean package**
2. V adresáři *target* se vytvoří nový soubor *document-manager-0.0.1-SNAPSHOT.jar*
3. Soubor se spouští příkazem **java -jar document-manager-0.0.1-SNAPSHOT.jar**