Bakalářská práce


F3 Fakulta elektrotechnická
Katedra počítačů

# Systém pro správu dokumentů, souborů a datových zdrojů

Martin Malinov

Vedoucí: Ing. Martin Ledvinka
Studijní program: Softwarové inženýrství a technologie
Květen 2019

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

## I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Malinov** Jméno: **Martin** Osobní číslo: **435576**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

## II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

**Systém pro správu dokumentů, souborů a datových zdrojů**

Název bakalářské práce anglicky:

**Document and file management system**

Pokyny pro vypracování:

1. Proveďte rešerši existujících systémů pro správu dokumentů a souborů a srovnejte strategie, které používají.
2. Vyberte vhodnou strategii a navrhněte vlastní systém pro správu dokumentů.
3. Naimplementujte systém dle vytvořeného návrhu.
4. Ověřte funkcionalitu výsledného řešení pomocí sady testovacích scénářů simulujících jeho použití správcem terminologií TermIt, vyvíjeným skupinou znalostních softwarových systémů.

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] K. Douglas, PostgreSQL, Sams Publishing; 2 edition, 2005

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: **24.01.2019** Termín odevzdání bakalářské práce: **24.05.2019**

Platnost zadání bakalářské práce: **20.09.2020**

| Ing. Martin Ledvinka<br>podpis vedoucí(ho) práce | podpis vedoucí(ho) ústavu/katedry | prof. Ing. Pavel Ripka, CSc.<br>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

Tímto bych chtěl poděkovat vedoucímu mé bakalářské práce, panu Ing. Martinu Ledvinkovi, a to za příležitost pracovat na reálném projektu, poskytnuté konzultace a rady, trpělivost a možnost získat cenné zkušenosti během práce na tomto projektu.

Obrovské díky patří mé rodině, která při mně vždy stojí, poskytuje mi nádherné rodinné zázemí, má mě ráda a já ji též.

V neposlední řadě děkuji i mé kamarádce KJ za naše hodiny strávené v kavárnách, které mi pomáhaly udržet si zdravý rozum.


# 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, 23. května 2019

# Abstrakt

Tato bakalářská práce se zabývá analýzou, návrhem, implementací a evaluací systému pro správu dokumentů.

V úvodní části projektu se zaměřuji na analýzu současné situace a existujících řešení v této oblasti. Dále pokračuji návrhem vlastního řešení, popisem implementace a užitých technologií a v závěru se věnuji evaluaci naprogramované aplikace.



**Klíčová slova:** správa dokumentů, DMS, správa souborů, datové zdroje



**Vedoucí:** Ing. Martin Ledvinka
Skupina znalostních softwarových systémů,
Resslova 307/9,
Praha 2



# Abstract

This bachelor thesis addresses analysis, design, implementation and evaluation of document management system.

The beginning of the project is focused on analysis of the current situation in this area and already existing solutions. Afterwards, I continue with a design of my solution, a description of the implementation process and used technologies and in the final part, I describe the evaluation of the application.



**Keywords:** document management, DMS, file management, data source

**Title translation:** Document and file management system

# Obsah


**1 Úvod** **1**

1.1 Motivace . . . . . 1

1.2 Cíl práce . . . . . 1

**2 Analýza** **3**

2.1 Definice . . . . . 3

2.2 Srovnání stávajících systémů . . . . 4

2.2.1 Alfresco DMS . . . . . 4

2.2.2 OpenKM . . . . . 4

2.2.3 LogicalDOC . . . . . 5

2.2.4 SeedDMS . . . . . 5

2.2.5 Kimios . . . . . 6

2.2.6 Feng Office . . . . . 6

2.2.7 Srovnání funkcionalit . . . . . 7

2.3 Architektura vybraných řešení . . . 7

2.3.1 AlfrescoDMS . . . . . 8

2.3.2 OpenKM . . . . . 8

2.3.3 LogicalDOC . . . . . 10

2.4 Funkční požadavky . . . . . 13

2.5 Technologie pro ukládání dokumentů . . . . . 14

2.5.1 Relační databáze . . . . . 14

2.5.2 NoSQL databáze . . . . . 15

2.5.3 File-system . . . . . 15

**3 Návrh vlastního řešení** **17**

3.1 Výběr technologie pro ukládání . 17

3.2 Návrh vlastní architektury . . . . . 18

3.2.1 Architektura aplikace . . . . . 18

3.2.2 Logická struktura databáze . . 19

3.3 Entity a vztahy mezi nimi . . . . . 21

3.3.1 Parametry entit . . . . . 21

3.3.2 Vztahy mezi entitami . . . . . 22

**4 Implementace** **25**

4.1 Popis použitých technologií . . . . 25

4.1.1 Java . . . . . 25

4.1.2 Spring . . . . . 26

4.2 Popis architektury systému . . . . 27

4.2.1 Kontrolní vrstva . . . . . 28

4.2.2 Servisní vrstva . . . . . 29

4.2.3 Perzistentní vrstva . . . . . 30

4.2.4 Datová vrstva . . . . . 31

4.3 Zabezpečení . . . . . 32

4.4 Konfigurace . . . . . 33

**5 Evaluace** **35**

5.1 Testování s Postmanem . . . . . 35

5.2 Testování se sadou testovacích scénářů . . . . . 39

**6 Závěr** **41**

**A Literatura** **43**

**B Seznam použitých zkratek** **47**

**C Obsah přiloženého CD** **49**

**D Návod ke spuštění** **51**

D.1 Spuštění příkazem . . . . . 51

D.2 Kompilace aplikace . . . . . 51

# Obrázky

2.1 architektura Alfresco DMS . . . . . 9

2.2 architektura OpenKM . . . . . 10

2.3 LogicalDOC - logická a fyzická reprezentace dokumentů . . . . . 12

2.4 architektura LogicalDOC . . . . . 13

3.1 High-level architektura systému . 18

3.2 Diagram ukládání souborů . . . . . 20

3.3 Doménový model . . . . . 21

4.1 Diagram architektury systému . . 27

# Ukázky zdrojových kódů

4.1 Controller: GET endpoint /files . . . . . 28

4.2 JSON response to GET /files 28

4.3 Service: vyhledání souboru dle ID . . . . . 30

4.4 Entity: Dokument . . . . 30

4.5 Konfigurace: soubor application.properties . . . . . 33

ctuthesis t1606152353

# Kapitola 1

# Úvod

## 1.1 Motivace

Důvodem pro vybrání tohoto tématu byla jeho příbuznost s mým zaměřením (softwarové inženýrství) a možnost vyzkoušet si v praxi znalosti, které jsem si osvojil během studia. Ty byly z velké části teoretické a týkaly se převážně návrhové části, zkušenost s vývojem webových aplikací jsem zatím neměl. Velkou motivací mi tedy byla i příležitost prozkoumat nové technologie a vyzkoušet si je v praxi implementací.

## 1.2 Cíl práce

Cílem práce je prozkoumat existující řešení v oblasti systémů pro správu dokumentů a navrhnout vlastní řešení, které následně implementuji. Výsledná aplikace by pak měla doplnit či nahradit současné řešení užívané skupinou znalostních softwarových systémů FEL ČVUT, ale měla by být připravena i pro napojení dalšími systémy.

# Kapitola 2

# Analýza

V této kapitole se zabývám analýzou již existujících systémů pro správu dokumentů z hlediska funkcionality, následně pak architekturou vybraných systémů.

## 2.1 Definice

Systém pro správu dokumentů (Document Management System, dále jen DMS) je počítačový systém, určený ke správě elektronických dokumentů. Narozdíl od systémů pro správu obsahu, které se zabývají také obsahem dokumentů a jeho tvorbou, DMS jejich obsah nijak neřeší - dokumenty bere jako atomické prvky. Většina DMS umožňuje nejen ukládání a získávání dokumentů ve formě souborů, ale také udržování metadat (dat, které obsahují informace o daných dokumetech), verzování dokumentů a možnost vrátit se v případě potřeby ke starším verzím, vyhledávaní mezi dokumenty podle specifických kritérií, zabezpečení přístupu atd.

Pojmem "dokument"se v našem případě rozumí logický celek, který je složen ze souborů s nějakou společnou vlastností. Rozhodnutí o tom, které soubory budou součástí toho celku (tzn. jaká vlastnost bude spojovat dané soubory), je již na správci či uživateli daného systému.

## 2.2 Srovnání stávajících systémů

V této sekci porovnávám existující systémy pro správu dokumentů. Takových systémů je obrovské množství, základní společnou funkcionalitou je možnost nahrávání souborů, jejich správa a katalogizace. Liší se mj. architekturou, doplňkovými funkcemi či formou licence - některé jsou placené, jiné zdarma, další zas open-source, tedy s možností průzkumu a úprav zdrojového kódu aplikace. V této práci jsem se zaměřil pouze na open-source řešení, jelikož nemám k dispozici žádný rozpočet a open-source umožňuje lépe prozkoumat funkce a chování systému.

Vybral jsem šest nejčastěji používaných systémů z této oblasti - Alfresco DMS, OpenKM, LogicalDOC, SeedDMS, Kimios a Feng Office. [1]

### 2.2.1 Alfresco DMS

Alfresco DMS je produkt společnosti Alfresco Software Inc., který se zaměřuje na správu enterprise obsahu. Umožňuje vysokou míru přizpůsobení, a to hlavně pomocí mnoha dostupných rozšíření. Lze si tak nainstalovat například rozšíření pro pokročilou úpravu kancelářských dokumentů z balíku LibreOffice, tvorbu formulářů, psaní a spouštění skriptů v programovém jazyce JavaScript atd. [2]

Systém je vysoce zaměřený na spolupráci v rámci týmů a obsahuje komplexní systém pro správu uživatelských rolí, lze tak jednoduše nastavit oprávnění pro určité skupiny.

Mezi další výhody se řadí jednoduchá integrace s programy jako Google Docs či Microsoft Office a následná možnost prohlížení a úprav souborů tohoto typu. Alfresco též nabízí kvalitně zpracovanou mobilní aplikaci a veřejně dostupný zdrojový kód. [3]

### 2.2.2 OpenKM

OpenKM je robustnější systém, který se zaměřuje na automatizaci úkolů a umožňuje dokonce nastavit vlastní pravidla automatizace. Správci tak mohou

například nastavit, ať se při přesouvání dokumentů do nové lokace ověří splnění určitých podmínek, spustí se nějaké úkoly na pozadí apod.

Podporuje většinu prohlížečů, formátů nahrávaných souborů a databází pro ukládání metadat. Pomocí metadat či klíčových slov lze také mezi dokumenty a soubory vyhledávat. Nabízí snadnou integraci s úložišti Google Drive a Dropbox, či věstavěny HTML editor pro online úpravy dokumentů.

Mezi pokročilé funkce tohoto systému patří možnost šifrování dokumentů, šablony a formuláře pro jednodušší tvorbu obsahu nebo funkce OCR (z anglického Optical Character Recognition), tedy optické rozpoznávání znaků, což se hodí pro digitalizaci tištěných dokumentů a případné vyhledávání v jejich obsahu. [4] [5]

### 2.2.3 LogicalDOC

LogicalDOC je oproti dvěma předchozím systémům jednodušší. Spíše než na velké množství pokročilých funkcí se snaží nalákat na jednoduchost instalace a následného používání uživatelského prostředí. [6]

Umožňuje fulltextové vyhledávání v různých jazycích, různé možnosti filtrace zobrazení obsahu (dle kategorie, tagů,...), plánovač úloh či např. import souborů ze ZIP archivů. Další užitečnou funkcí je uchovávání detailních logů chování systému, vytváření reportů za určitá období a tvorba různorodých statistik. [7]

### 2.2.4 SeedDMS

SeedDMS je další robustní systém, nabízející velké množství pokročilých funkcí. Je navržen tak, aby zvládal velkou zátěž a lze do něj uložit více než 32 tisíc dokumentů. Mezi nimi pak lze vyhledávat fulltextově, pomocí metadat a spousty dalších nastavení. [8]

Uživatelské prostředí umožňuje spolupráci vícera uživatelů na dokumentu v reálném čase. Dokumenty a soubory pak mohou podléhat schvalovacímu procesu dle nastavení oprávnění. S oprávněním též souvisí komplexní systém pro správu uživateských rolí a skupin.

Pro jednotlivé dokumenty a soubory lze nastavit funkci verzování, což umožní systému uchovávat jejich historii a v případě potřeby je obnovit. K dokumentům lze také přikládat přílohy či nastavit vazbu se souvisejícími dokumenty. [9]

### 2.2.5 Kimios

Kimios je nejjednodušší z vybraných systémů pro správu dokumentů. Umožňuje základní práci se soubory - nahrávání, aktualizaci a mazání. Nelze tedy zobrazovat obsah souborů ani jej žadným způsobem upravovat.

Nabízí jednoduchý verzovací systém, nastavení oprávnění jednotlivých uživatelů, vyhledávání pomocí metadat či funkci záložek. [10]

Nevýhodou tohoto systému může být absence pokročilých funkcí, popř. nutnost zakoupit komerční licenci v případě, že jej chcete provozovat na serveru s operačním systémem Microsoft Windows. [1]

### 2.2.6 Feng Office

Feng Office je naopak nejrobustnější ze zde představovaných systémů. Kromě správy souborů umožňuje i jejich online editaci, stejně tak zobrazení historie jejich obsahu. [11]

Nabízí pokročilé funkce pro spolupráci uživatelů - sledování času stráveného upravováním dokumentů, kalendář se zobrazením úkolů pro členy týmu, plánovač úloh, správu kontaktů, vytváření statistik a reportů apod.

Systém dokonce umožňuje integraci wiki stránky, diskuzního fóra či emailového klienta. [12]

### 2.2.7 Srovnání funkcionalit

Následuje stručný výčet funkcí dle jejich četnosti výskytu. Méně časté funkce pomíjím, nejsou-li užitečné pro tento projekt. Tento výčet sloužil pro zamyšlení se a diskuzi nad tím, které funkce bych mohl implementovat ve svém systému.

Velice časté:

- podpora metadat
- omezení přístupu
- komentáře k dokumentům
- vyhledávání
- verzování

Časté:

- uživatelské role
- podpora ukládání všech typů souborů
- kategorie pro soubory
- náhled dokumentů

Méně časté (ale užitečné):

- import souborů ze ZIP archivů
- podpora záložek

## 2.3 Architektura vybraných řešení

V této sekci porovnávám architekturu tří vybraných řešení - AlfrescoDMS, OpenKM a LogicalDOC.

### 2.3.1 AlfrescoDMS

Systém Alfresca se skládá ze tří hlavních částí - Platform, User Interface a Search. [13]

**Platform** obsahuje úložiště, kde jsou skladovány fyzické soubory, databázi obsahující metadata, a také všechny další služby, které se týkají obsahu a jeho zpracování. Je většinou propojený s SMTP serverem (úložiště e-mailů), LDAP serverem (uživatelé, skupiny) a dalšími službami.

**User Interface** poskytuje uživatelské prostředí pro platformu, pomocí kterého koncoví uživatelé spravují dokumenty, soubory a další obsah. Patří sem Alfresco Share, což je webová aplikace, mobilní aplikace připojené k platformě, desktopové aplikace... UI přistupuje k obsahu přes ReST API.

**Search** se stará o vyhledávání v uloženém obsahu. Je postaveno na Apache Solr 6 a indexuje veškerý obsah.

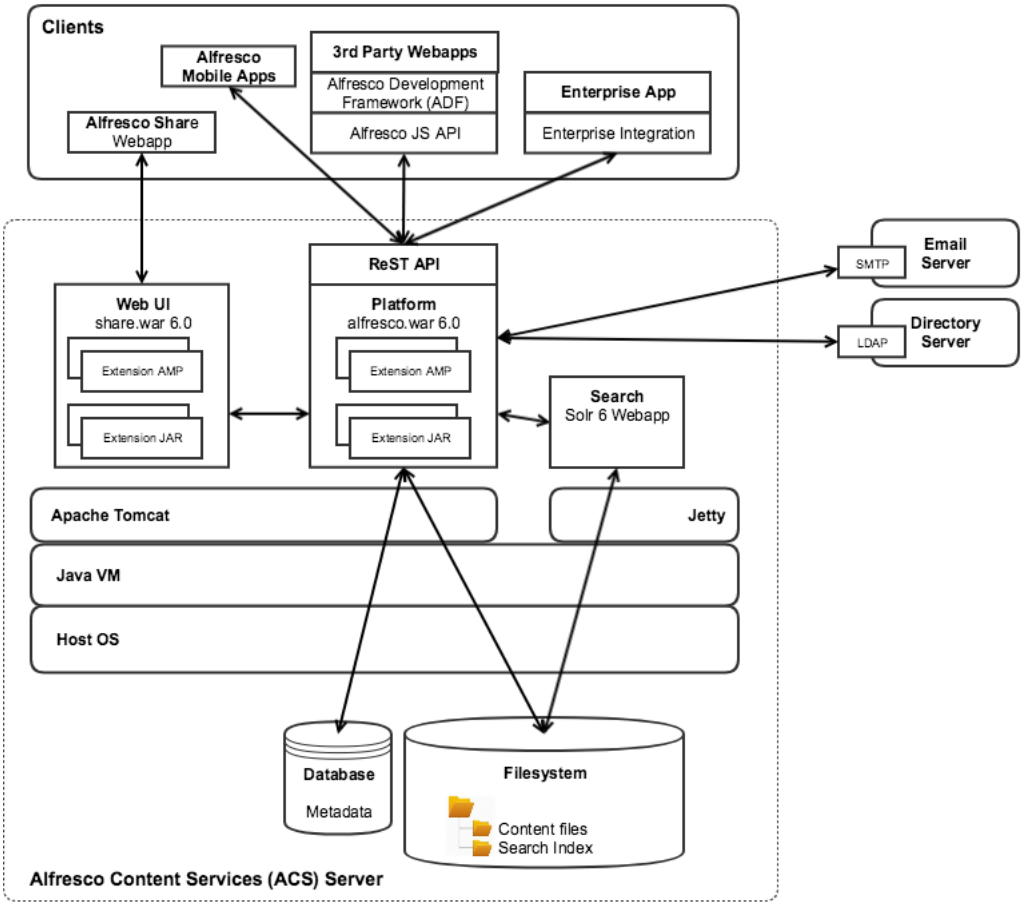
### 2.3.2 OpenKM

Architektura systému OpenKM je složena z mnoha částí, z nichž každá poskytuje odlišnou funkcionalitu. Je to aplikace JavaEE využívající Spring Framework. [14]

**API**, tj. programové rozhraní pro webové služby a další aplikace. OpenKM implementuje protokol CMIS (Content Management Interoperability Services), což je standard OASIS pro vrstvy zajišťující spojení a komunikaci mezi DMS a úložišti. Rozhraní pro ostatní aplikace je vytvořeno pomocí REST a protokolu pro webové služby SOAP.

**Security Layer** (česky bezpečnostní vrstva), jedna z nejdůležitějších částí systému, je vytvořena pomocí Spring Security. Ten umožňuje centralizaci kontroly oprávnění přístupu uživatelů v rámci aplikace a další bezpečnostní funkce. Autentizace probíhá probíhá přes službu CAS (centrální autentizační služba), LDAP, popř. databázi registrovaných uživatelů.

**Core** (jádro) implementuje logiku pro existenci a správu obsahu, který je reprezentován entitami typu dokument, adresář, email, záznam atd. K jádru




**Obrázek 2.1:** architektura Alfresco DMS [16]

je připojený antivirová služba, stejně tak vyhledávací a katalogová služba, databáze či samotné úložiště.

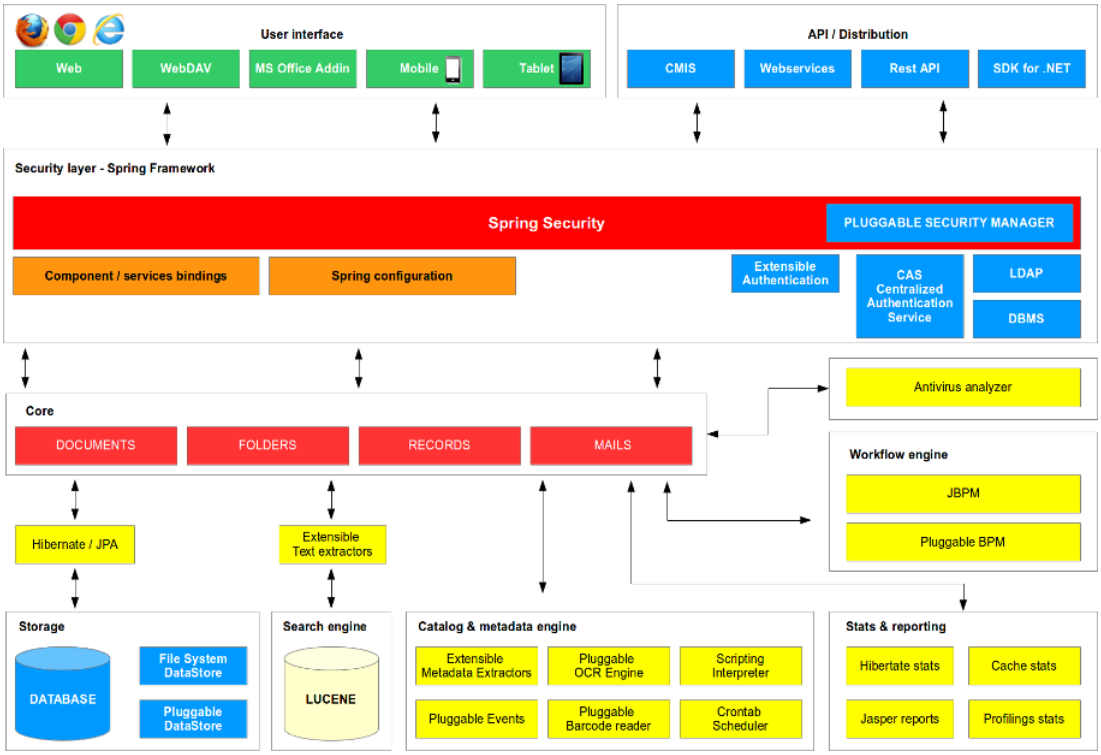
**Storage** (úložiště) využívá frameworku Hibernate pro objektově relační mapování mezi soubory a databází. OpenKM podporuje velké množství relačních databází, mezi nimi PostgreSQL, MySQL, Oracle či MS SQL. Veškeré informace a metadata k souborům jsou uloženy právě v relační databázi, zatímco samotné binární soubory mohou být uloženy v souborovém systému, databázi anebo cloudovém datovém úložišti.

**Search Engine** (vyhledávač) využívá knihovny Lucene pro vyhledávání mezi dokumenty. Veškeré soubory jsou indexovány. Než je však převezme Lucene, obsah souborů je analyzován pomocí textových extraktorů (u obrázků OCR) pro identifikaci textových řetězců, které by mohly být použity při

vyhledávání. Výsledky jsou pak zpracovány pomocí Security Manageru, který zajistí, že se uživatel dostane jen k informacím, ke kterým má povolený přístup.

**Catalog and Metadata Engine** tvoří několik menších aplikací, mezi nejvýznačnější patří Barcode Engine (identifikace a čtení čárových kódu v dokumentech), OCR Engine (rozpoznávání znaků v obrázcích), Shell Bean (skriptovací nástroj), Event system (správa událostí), plánovač úloh, atd. Kombinace těchto nástrojů podporuje automatizaci tvorby metadat.

**Statistics and reports** poskytuje nástroje pro analýzu dat. Ta může být použita pro predikci chování uživatelů, stavu systému, vývoje dat apod.




**Obrázek 2.2:** architektura OpenKM [17]

### 2.3.3 LogicalDOC

LogicalDOC se skládá z pěti hlavních částí - Data Layer, Business Process Layer, Business Entity Layer, GUI a Web Services. Vrstvy mezi sebou komunikují přes Spring application context. Kromě Springu systém využívá i

Hibernate (pro ORM) a JSF. Jádro systému funguje jako plugin a vystavuje core API, přes který lze aplikaci rozšířit pomocí dalších pluginů. [15]

### Data layer

Datová vrstva se zaměřuje na práci se zdroji. Jejím cílem je abstrahovat datové entity, čehož docílí staráním se o veškeré podrobnosti fyzického skladování dat.

Informace jsou ukládány v následujících třech úložištích:

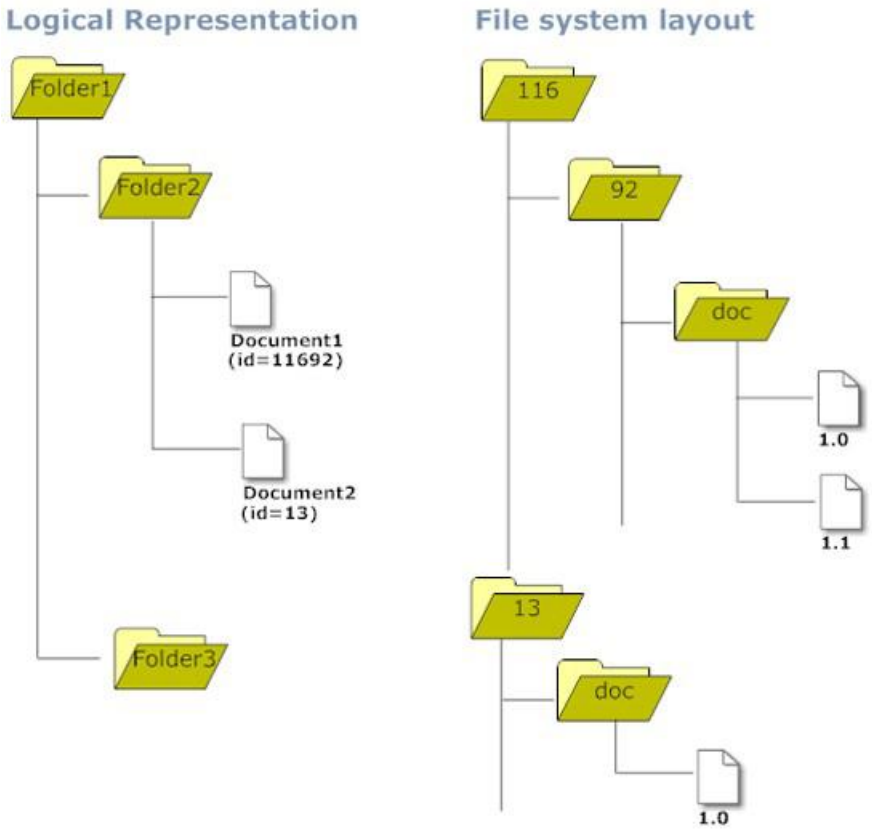
- **relační databáze** - obsahuje persistentní záznamy, informace o uživatelích, skupinách, dokumentech...
- **souborový systém** - soubory dokumentů na pevném disku, mají pevně dané rozložení
- **full-text index** - extrahovaný text z dokumentů, umožňuje rychlé vyhledávání v jejich obsahu

**Data access objects** (DAO) jsou objekty umožňující CRUD operace (vytváření, čtení, aktualizace a mazání) s entitami v databázi. Dělají entity persistentními a jejich metody podporují transakční zpracování. Pro každou entitu tedy existuje příslušný DAO (např. entitu dokument zde reprezentuje DocumentDAO).

**Storer**, neboli „skladník“, zajišťuje archivaci souborů. Organizuje soubory způsobem, který optimalizuje prostor na disku a přístupovou dobu. Každý dokument v LogicalDOC je jednoznačně identifikován svým ID. Toto ID je rozděleno na skupiny po třech číslicích a pro každou z nich je vytvořena podsložka (vizte obrázek 2.3). V nejhlubší podsložce je pak vytvořena další složka jménem „doc“ která obsahuje veškeré soubory daného dokumentu. Každá verze má svůj vlastní soubor se jménem dokumentu a příponou označující číslo verze.

### Business Process Layer

Nejdůležitější procesy jsou identifikovány v této vrstvě a jsou představeny business entitami. K procesům přistupují tzv. manažeři, což jsou komponenty




**Obrázek 2.3:** LogicalDOC - logická a fyzická reprezentace dokumentů [18]

pracující s vícerem business entit a provádějících na nich komplexní operace. Např. DocumentManager se stará o vytvoření dokumentu, indexaci obsahu, uložení souborů, atd.

BPL pracuje s datovou vrstvou, kde ukládá (popř. odkud načítá) informace do úložiště. Neví nic o implementaci nižší vrstvy. Účelem této vrstvy je vytvoření užitečného API pro vyšší vrstvy systému.

### Business Entity Layer

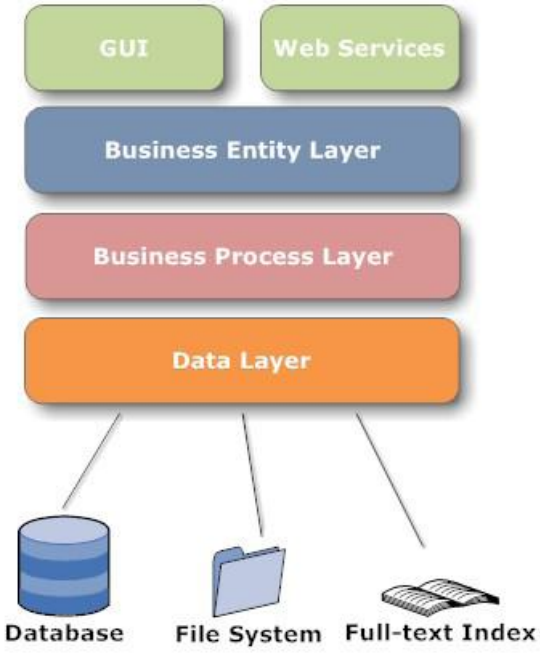
Každá business entita je reprezentována Java Beanou, např. Document, User, Menu,... Tyto entity neobsahují žádnou business logiku, ta je totiž implementována manažery. Spojení business entit a manažerů tvoří API, které umožňuje GUI a dalším službám interagovat s jádrem DMS.

### GUI

Grafické webové rozhraní umožňující interakci s aplikaci. Je implementováno pomocí JSF a AJAX.

### Web Services

Ostatní systémy mohou využít LogicalDOC jako prostředníka a integrovat jej do svého systému. Díky tomu získají přístup k funkcionalitám této aplikace a mohou nad ním vytvořit další vrstvu. Modul Web Services splňuje požadavky W3C specifikací SOAP a MTOM.




**Obrázek 2.4:** architektura LogicalDOC [19]

## 2.4 Funkční požadavky

Na základě předchozí analýzy vybraných řešení byly s vedoucím práce identifikovány a dohodnuty tyto funkční požadavky:

- **[FP1]** Vytvořit dokument

- **[FP2]** Nahrát nový soubor do dokumentu
- **[FP3]** Nahrát samostatný soubor
- **[FP4]** Nahrát novou revizi již existujícího souboru
- Získat metadata o souboru

  **[FP5]** defaultně o poslední verzi

  **[FP6]** získat metadata starší verze
- Stáhnout fyzickou kopii souboru

  **[FP7]** defaultně poslední verzi

  **[FP8]** stáhnout kopii starší verze
- **[FP9]** Odstranit soubor - včetně všech revizí
- **[FP10]** Odstranit dokument - včetně všech jeho složek, souborů a revizí

## 2.5 Technologie pro ukládání dokumentů

V této sekci porovnávám různé možnosti pro ukládání souborů a dokumentů. V zásadě existují tři možnosti - relační (SQL) databáze, NoSQL databáze a filesystem (souborový systém). Jsou také možné jejich kombinace pro různé účely.

### 2.5.1 Relační databáze

SQL databáze jsou strukturované. Kostru tvoří tabulky složené ze sloupců a řádek, kdy sloupec definuje typ informace a každá řádka pak reprezentuje jednotlivý záznam.

Vztah mezi tabulkami a poli se nazývá schéma. V relačních databázích musí být schéma jasně definováno ještě před přidáním záznamu, což snižuje redundanci dat a zajišťuje synchronicitu. Při změně pole je však potřeba editovat celou databázi. Daní za excelentní organizovatelnost je tedy kompromisní flexibilita. [20] [21] [22] [23]

Tento typ databáze je vhodné použít [23]:

- chceme-li zajistit ACID dat (předpis, jak mají transakce interagovat s databází) - redukujeme tak odchylky a anomálie v databázi
- máme-li strukturovaná data a struktura se moc nemění
- je-li pro nás kritická integrita dat

### 2.5.2 NoSQL databáze

NoSQL databáze jsou nestrukturované a dokumentově orientované. Existují různé typy - klíč-hodnota, sloupcová, dokumentová či grafová.

Odlišná nestrukturovaná data (fotky, videa, články,...) mohou být v jednom dokumentu, díky tomu lze informace o jedno dokumentu získat rychleji. Absence schématu (resp. jeho dynamičnost) umožňuje větší flexibilitu, nadruhou stranu snižuje integritu dat. NoSQL databáze též nepodporují transakční zpracování napříč dokumenty. [24]

Tento typ databáze je vhodné použít, když:

- skladujeme velké množství dat s malou či žádnou strukturou
- chceme ukládat data různého typu na jednom místě, i když dopředu nevíme, jakého typu data budou
- chceme zajistit horizontální škálovatelnost (např. rozvržení po cloudu)
- je pravděpodobné, že budeme databázi a její strukturu často měnit - s NoSQL to jde jednodušeji

### 2.5.3 File-system

File-system je kolekce nezpracovaných souborů na pevném disku. Díky tomu je ukládání a přístup k datům mnohem rychlejší než v případě databází. Práce se soubory je jednodušší, mohou být prakticky jakkoliv velké a jejich migrace je triviální. [26]

Na druhou stranu, file-system nepodporuje ACID, postrádá relační mapování, nezajišťuje integritu dat a nabízí jen velice nízkou úroveň bezpečnosti. [25] [27]

# Kapitola 3

# Návrh vlastního řešení

V této kapitole se zabývám výběrem technologie pro ukládání dokumentů, návrhem vlastní architektury, popisem entit z nichž se bude aplikace skládat a vztahy mezi nimi.

## 3.1 Výběr technologie pro ukládání

Jako řešení pro svůj projekt jsem zvolil **kombinaci relační databáze a file-systému**.

Samotné soubory budou uloženy ve file-systému, což umožní rychlý přístup k datům a neomezuje jejich velikost. V relační databázi pak budou uložena metadata příslušná daným dokumentům/souborům, a také k nim vedoucí cesta (adresářová struktura) ve file-systému. Díky spojení s relační databází budeme moci zajistit aciditu, synchronicitu, minimalizovanou redundanci dat a další výhody popsané v sekci 2.5.1.

Z databází jsem zvolil relační typ mimo jiné proto, že veškeré dokumenty (až na binární soubor) budou mít prakticky stejnou strukturu a pravděpodobně nebude potřeba ji nějak razantně měnit.

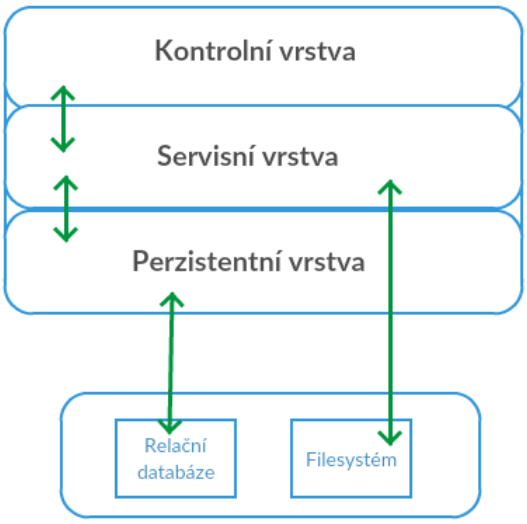
## 3.2 Návrh vlastní architektury

V této sekci popisuji, jakým způsobem budou reprezentovány dokumenty a soubory v databázi, jaká bude struktura databáze, způsob propojení logických reprezentací s fyzickými kopiemi souborů či vlastnosti databázových entit.

### 3.2.1 Architektura aplikace

Aplikace bude postavena na architektonickém vzoru **vícevrstvé architektury** (angl. layered architecture pattern). Ten umožňuje rozdělení odpovědností jednotlivých vrstev, a to prostřednictvím oddělení klientského rozhraní od business logiky a business logiky od té databázové. Díky rozdělení odpovědností zvyšuje znovupoužitelnost - chce-li vývojář vystavit nové rozhraní, ale nepotřebuje měnit business logiku aplikace, stačí upravit vrstvu s rozhraním. V neposlední řadě tento vzor zlepšuje přehlednost zdrojového kódu a umožňuje snažší orientaci nově příchozím vývojářům. [28].

Konkrétně jsem zvolil architekturu čtyřvrstvou. Bude se skládat z kontrolní vrstvy mající na starost zpracování HTTP požadavků klienta, servisní vrstvy obstarávající business logiku aplikace, perzistentní vrstvy zajišťující objektově-relační mapování a přístup k databázi, a nakonec vrstvy datové, která bude obsahovat samotnou databázi a filesystém [29].




**Obrázek 3.1:** High-level architektura systému

Do filesystému budou nahrávány fyzické kopie souborů, zatímco v databázi budou ukládána metadata o nahrávaných souborech. Dále bude databáze obsahovat asociace mezi dokumenty, jejich klasifikace a adresářovou strukturu. Ta se bude udržovat jen v relační databázi a nebude odrážet skutečnou adresářovou strukturu ve file-systému, která bude tvořena unikátním identifikátorem souboru z databáze a číslem označujícím verzi souboru.

### 3.2.2 Logická struktura databáze

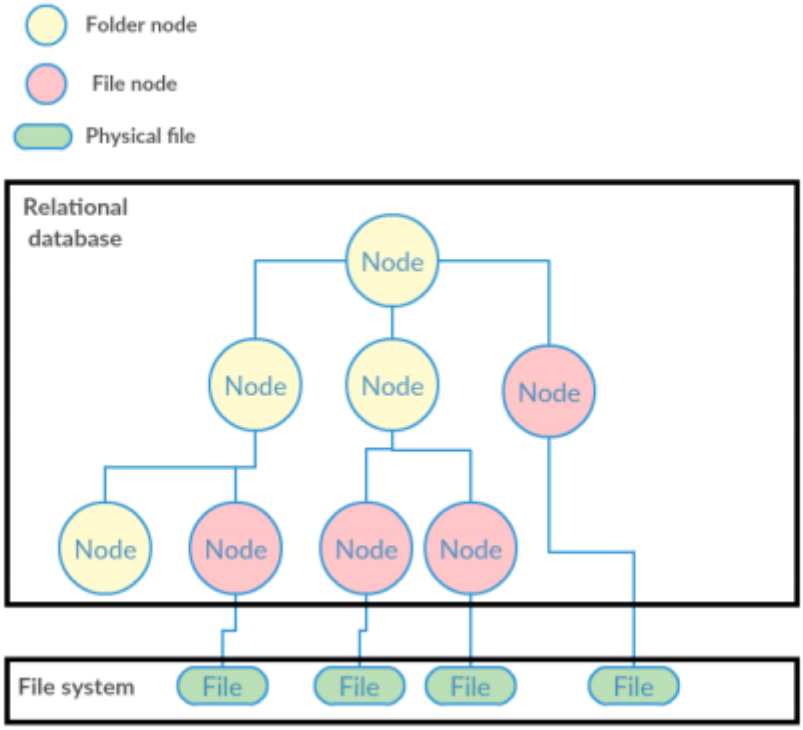
Většina entit bude typu Node, dle vlastností to pak bude složka, soubor, dokument apod. Jednotlivé Nody mezi sebou budou mít asociace, např. parent-child u složky a jejího obsahu.

**Node** bude cokoliv, co je uloženo v repozitáři (složky, soubory, XML fragmenty,...). Každý Node bude mít unikátní ID v rámci celého systému, podle kterého jej lze jednoznačně identifikovat. Může mít různé atributy o různých hodnotách, typ Nodu pak definuje atributy a možné vztahy k ostatním Nodům.

**Document** je logický celek a kontejner pro složky a soubory. Může spojovat i soubory zcela odlišného typu, pokud jsou podobné např. tématem.

**Folder** (složka) je kontejnerem, který může obsahovat soubory či další složky. Každá složka má svou rodičovskou složku (parent folder), nacházející se v hierarchii o úroveň výše. Výjimkou jsou pouze kořenové (angl. root) složky dokumentu.

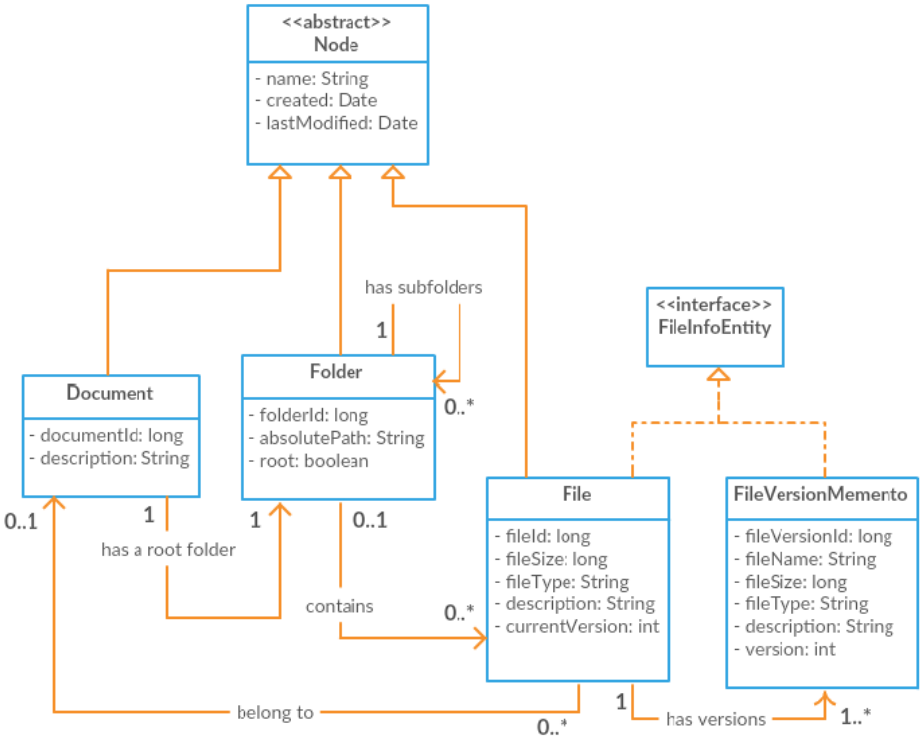
**File** (soubor) je nejnižší úrovní hierarchie. Jedná se o databázovou reprezentaci koncových souborů s metadaty, nikoliv o fyzické soubory - ty jsou umístěny ve file-systému a File na ně odkazuje pomocí svého identifikátoru určujícího cestu k souboru. Může mít také různé verze, kdy každá verze odkazuje na příslušný soubor ve file-systému.




**Obrázek 3.2:** Diagram ukládání souborů

## 3.3 Entity a vztahy mezi nimi

V této sekci se snažím navrhnout třídy, které pak budou tvořit entity v relační databázi. Na obrázku 3.3 lze vidět jednotlivé třídy, jejich parametry a vztahy mezi entitami.




**Obrázek 3.3:** Doménový model

### 3.3.1 Parametry entit

Následuje popis parametrů jednotlivých entitních tříd. Účel entit Node, Document, Folder a File vynechávám, neboť je již detailně popsán v sekci 3.2.2

- **Node**
  - **name** - název entity (dokumentu, složky, souboru)

- **created** - datum a čas vytvoření entity, budou doplněny automaticky
- **lastModified** - datum a čas poslední modifikace entity, budou doplněny automaticky

- **Document**
  - **documentId** - jednoznačný identifikátor dokumentu
  - **description** - popis dokumentu

- **Folder**
  - **folderId** - jednoznačný identifikátor složky
  - **absolutePath** - absolutní cesta ke složce v rámci adresářové struktury databáze
  - **root** - je-li kořenovou složkou dokumentu (tj. nemá rodičovskou složku)

- **File**
  - **fileId** - jednoznačný identifikátor souboru
  - **fileSize** - velikost odkazované fyzické kopie souboru v bytech
  - **fileType** - typ souboru (obrázek, textový soubor apod.)
  - **description** - popis souboru
  - **currentVersion** - číselné označení poslední verze souboru

- **FileVersionMemento** - představuje metadata konkrétní verze souboru, kvůli nutnosti zachovat jednotné fileId pro všechny verze souborů je ukládána jako odlišná entita

- **FileInfoEntity** - sdružuje File a FileVersionMemento pod jednu zastřešující třídu

### 3.3.2 Vztahy mezi entitami

Entity mezi sebou musí mít definované vztahy (neboli vazby), aby o sobě měly přehled a mohly mezi sebou komunikovat. V této podsekci uvádím jednotlivé vztahy, jejich kardinality a logické zdůvodnění. Triviální dědičnost a implementaci rozhraní dále nezmiňuji.

- **has a root folder (Document, Folder)**
  - **kardinalita Document** - 1
  - **kardinalita Folder** - 1
  - **důvod** - Dokument má vždy právě jednu kořenovou složku (a ta je kořenovou složkou právě jednoho dokumentu).

- **has subfolders (Folder, Folder)**
  - **kardinalita první Folder** - 1
  - **kardinalita druhé Folder** - 0..*
  - **důvod** - Složka může mít neomezené množství podsložek či nemít žádnou. Zároveň je jedinou rodičovskou složkou všech svých podsložek.

- **contains (Folder, File)**
  - **kardinalita Folder** - 0..1
  - **kardinalita File** - 0..*
  - **důvod** - Složka může obsahovat neomezené množství souborů (či žádné). Soubory se nachází nejvýše v jedné složce.

- **belongs to (File, Document)**
  - **kardinalita File** - 0..*
  - **kardinalita Document** - 0..1
  - **důvod** - Dokument může obsahovat neomezené množství souborů (či žádné). Soubory mohou být nahrávány do dokumentu, ale i samostatně.

- **has versions (File, FileVersionMemento)**
  - **kardinalita File** - 1
  - **kardinalita FileVersionMemento** - 1..*
  - **důvod** - Soubor má neomezené množství verzi, vždy však alespoň jednu (první verze vzniká nahráním souboru). Verze se vztahuje právě k jednomu souboru.

# Kapitola 4

# Implementace

V této kapitole se zaměřuji na popis použitých technologií, popis architektury systému a komunikaci mezi jejími jednotlivými vrstami, ale také způsobu řešení zabezpečení aplikace.

## 4.1 Popis použitých technologií

Již od začátku bylo jasné, že budu aplikaci vyvíjet pomocí programovacího jazyka Java. Mám s ním největší zkušenosti, a to převážně ze školy, neboť jej používáme ve víceru předmětech. Jakožto framework jsem si pak zvolil Spring, který v mnoha ohledech usnadňuje práci a eliminuje potřebu psát velké množství boilerplate kódu. Dále využívám PostgreSQL jako relační databázi pro ukládání metadat.

### 4.1.1 Java

Java je objektově orientovaný, silně typovaný programovací jazyk, při deklaraci proměnné je tedy vždy nutné uvést i její typ, aby věděla, jakých hodnot může nabývat. Zdrojový kód aplikací je zapisován do .java souborů, jež následně kompilátor zkompiluje do .class souborů, které již obsahují tzv. bytecode. **Java Virtual Machine (JVM)**, zodpovědný za spuštění hlavní

metody aplikace, je pak schopen tento bytecode přečíst a interpretovat jej pro zařízení, na kterém je spuštěn. Díky tomu stačí program v Javě napsat jen jednou a očekávat, že takto napsaný program bude spustitelný a funkční na každém zařízení podporujícím Javu [30]. JVM je součástí **JRE (Java Runtime Enviroment)**, což je prostředí poskytující minimální požadavky nutné pro spuštění java programů. Kromě JVM obsahuje základní třídy a další podpůrné soubory. Nadstavbou určenou pro programátory je pak **JDK (Java Development Kit)**, který obsahuje JRE, java interpreter, javac kompilátor, java archivátor (jar), generátor dokumentace (Javadoc) a další nástroje určené pro vývoj java aplikací [31].

Co se verze týče, používám **Java 8**, která je stále nejpopulárnější a nabízí nejrozšířenější podporu mezi vývojáři [32]. Oproti předchozím verzím přinesla mnoho novinek, mezi nejvýraznější se řadí podpora lambda výrazů, práce s datovými proudy, třída Optional či Date/Time API [33]. Vše zde zmíněné ve své aplikaci hojně využívám.

### 4.1.2 Spring

**Spring** (také **Spring Framework**) je aplikační rozhraní určené pro vývoj enterprise aplikací a důvodem jeho vzniku je usnadnění jejich vývoje. Je postaven na principu návrhového vzoru Inversion of Control (IoC), který přesunuje povinnost za vytváření a provázání objektů z aplikace na framework. Provázání objektů je možné pomocí tzv. dependency injection (vkládání závislostí) [34]. Spring je modulární framework a skládá se tedy z několika částí (modulů), které lze volně kombinovat [35].

**Spring Boot** umožňuje vytvořit enterprise aplikaci, kterou lze spustit jedním kliknutím. Nemusíme ani nastavovat konfiguraci v XML souborech, Spring Boot nastaví vše potřebné pro start aplikace automaticky. Přizpůsobit nastavení si pak lze s pomocí anotací [36].

**Spring Data JPA** velice usnadňuje práci s datovými repozitáři založenými na JPA (Java Persistence API), neboť opět snižuje množství boilerplate kódu. Není již třeba opakovaně psát tentýž kód a jednoduché dotazy s minimálními obměnami - stačí definovat daný repozitář a Spring již dodá svou vlastní implementaci [37].

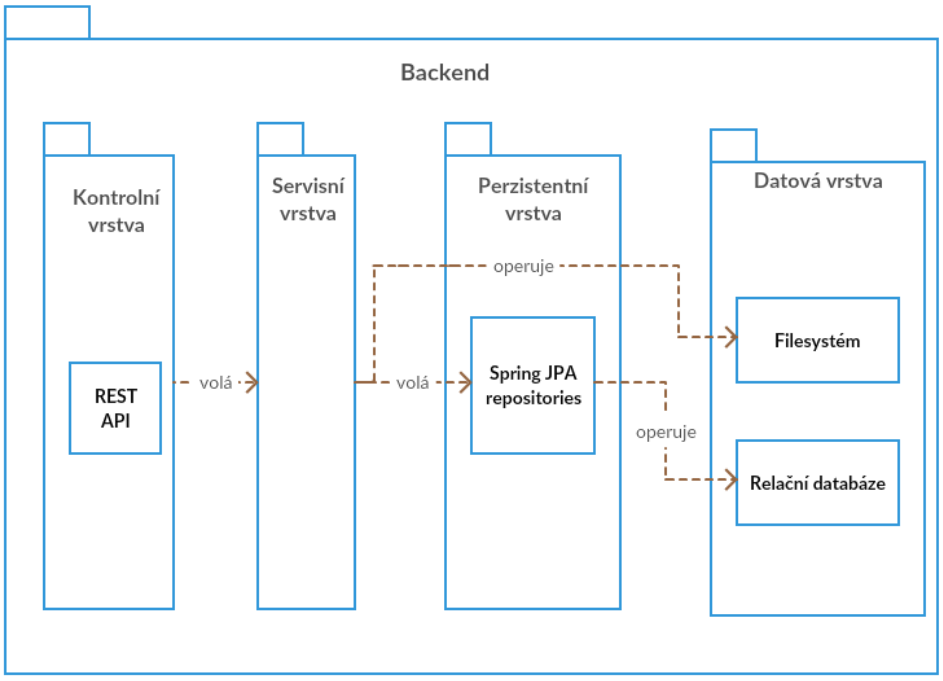
## 4.2 Popis architektury systému

V této sekci se zaměřuji na architekturu systému, popisuji jednotlivé části (vrstvy) aplikace a způsob, jakým mezi sebou komunikují.

Aplikace se skládá se čtyř hlavních vrstev, a to:

- řídící (controller layer)
- servisní (service layer)
- perzistentní (data access layer)
- datové (data layer)

Vrstvy nejsou zcela oddělené, ba naopak - vrstvy vyšších úrovní většinou provolávají nějaké metody z vrstev nižších úrovní, předávají jim parametry a očekávají výstup, který zas budou moci poslat výše. Diagram architektury systému je znázorněn na obrázku 4.1.




**Obrázek 4.1:** Diagram architektury systému

### 4.2.1 Kontrolní vrstva

Kontrolní vrstva představuje přístupový bod k aplikaci. Vystavuje REST rozhraní, ke kterému se může připojit jakýkoliv klient, který dokáže posílat a zpracovávat HTTP požadavky. Toto rozhraní je definováno v tzv. controllerech, což jsou java třídy označené anotací *@RestController*.

V těchto třídách se nachází metody představující jednotlivé přístupové body. Každý z nich musí mít definovanou povolenou HTTP metodu (GET, POST, PUT, PATCH, DELETE) a adresu relativní k URI adrese hostitele. Následuje klasická javovská deklarace a definice samotné metody. Níže je ukázka jednoduché GET metody pro vrácení seznamu všech souborů v databázi.

```
@GetMapping("/files")
    public List<File> getFileList() {
        return fileService.getAllFiles();
    }
```

**Ukázka zdrojového kódu 4.1:** Controller: GET endpoint /files

Jak je vidět v ukázce výše, kontrolní vrstva spolupracuje s vrstvou servisní. Je důležité zmínit, že spolupracuje vždy jen s vrstvou přímo pod sebou, nestává se tedy, že by kontrolní vrstva přímo operovala s metodami vrstvy perzistentní. Úkolem controllerů je pouze zpracovat příchozí požadavek a předat ho dále, kde se odehrává business logika.

Poté, co nižší vrstvy zpracují požadavek, vrátí controlleru typ dle definice příslušné metody. Ten jej pak převede na JSON (JavaScript Object Notation) a odpověď v tomto formátu pošle dotazujícímu se klientovi. Níže je příklad JSON odpovědi při poslání GET požadavku na přístupový bod */files*, obsahuje metadata všech souborů z databáze, tedy veškeré jejich parametry.

```
[
    {
        "name": "gentl.jpg",
        "created": "2019-05-18T20:29:28.264+0000",
        "lastModified": "2019-05-18T20:29:28.264+0000",
        "fileId": 1000,
        "fileSize": 10939,
        "fileType": "image/jpeg",
        "description": null,
        "version": 0
    },
    {
```

```
        "name": "smile.PNG",
        "created": "2019-05-18T20:29:28.334+0000",
        "lastModified": "2019-05-18T20:29:28.334+0000",
        "fileId": 1001,
        "fileSize": 30344,
        "fileType": "image/png",
        "description": null,
        "version": 0
    }
]
```

**Ukázka zdrojového kódu 4.2:** JSON response to GET /files

Controllery ale podporují i pokročilé nastavení pomocí anotací. Např. anotace *@RequestBody* s deklarací entity u parametru metody zajistí, že pro přístup k danému bodu je třeba zaslat v těle požadavku JSON objekt, jenž je strukturou shodný s danou entitou. Spring se pak již postará o automatickou deserializaci těla HTTP požadavku na příslušnou java entitu [40].

Vytvořil jsem celkem tři controllery - FileController pro operace se soubory, FolderController pro operace se složkami a DocumentController pro operace týkající se správy dokumentů. Kompletní seznam vytvořených přístupových bodů a jejich popis je v příloze todo.

### 4.2.2 Servisní vrstva

V servisní vrstvě se odehrává business logika aplikace. Má v sobě vložené závislosti na datových repozitářích (JPA), kombinuje je a provádí nad nimi složitější operace. V zásadě se dá říci, že controllery říkají servisní vrstvě co se od ní požaduje, ta definuje jak se to má udělat (jakou kombinací a pořadím operací) a nad jakými daty, od repozitářů si vyžádá příslušná data a provede dané operace.

Obsahuje několik servisních tříd (angl. services) pojmenovaných dle zaměření služeb, které mají poskytovat, v našem případě FileService (pro operace týkající se metadat souborů), DocumentService (dokumentů), FolderService (složek) a FileStorageService (kopií souborů ve filesystému).

DocumentService obsahuje jen metody pro základní správu dokumentů, FolderService ke svým základním metodám přidává další pro práci s podsložkami a přidruženými dokumenty, FileStorageService zas kromě základních

operací se soubory v filesystému umožňuje také vypsat seznam všech adresářů reprezentujících jednotlivé soubory z databáze, což se hodí pro automatické čištění. Nejbohatší nabídku metod má FileService, např. operace s verzemi souborů či automatické pročištění souborů z databáze, nemají-li svůj protějšek ve filesystému, a naopak.

Níže je ukázka jednoduché metody, která má za úkol najít soubor s určitým identifikátorem. Metoda pomocí repozitáře prohledá databázi a najde-li takový soubor, vrátí jej controlleru, ze kterého je volána. V opačném případě vyhodí výjimku a pošle ji dále kontrolní vrstvě.

```
@Override
    public File findFileById(long id) {
        Optional<File> file = fileRepository.findById(id);
        if (file.isPresent()) {
            return file.get();
        } else {
            throw new FileNotFoundException("File with id '" + id +
                "' could not been found.");
        }
    }
```

**Ukázka zdrojového kódu 4.3:** Service: vyhledání souboru dle ID

### 4.2.3 Perzistentní vrstva

V této vrstvě se nacházejí repozitáře, jejichž účelem je přístup k datům z databáze. Díky modulu Spring Data JPA není třeba manuálně psát metody pro jednoduché operace s entitami, stačí vytvořit rozhraní dědící z JpaRepository, označit jej anotací *@Repository* a Spring se pak již postará o dodání implementace základních operací. V případě požadavku na složitější metody je lze v tomto rozhraní nadefinovat.

Každý repozitář je spojen s příslušnou entitou, která je pomocí JPA anotací namapována na odpovídající tabulku v relační databázi. Aby mohl Spring provádět operace nad databází, potřebuje k tomu tzv. persistence providera, který poskytne konkrétní implementaci JPA. V případě, že není v aplikaci provider nakonfigurován, Spring Data JPA použije výchozího providera, kterým je Hibernate. Následuje ukázka entity využívající JPA.

```
@Entity(name = "document")
public class Document extends Node {
    @Id
    @GeneratedValue(generator = "document_generator")
```

```
@SequenceGenerator(
        name = "document_generator",
        sequenceName = "document_sequence",
        initialValue = 1000
)
protected long documentId;

@Column(name = "description")
private String description;

@OneToOne(cascade = CascadeType.REMOVE, orphanRemoval = true)
@JoinColumn(name = "fk_rootfolder")
private Folder rootFolder;
```

**Ukázka zdrojového kódu 4.4:** Entity: Dokument

Entita z ukázky 4.4 využívá jako primární klíč uměle vygenerovaný identifikátor. Každá entita má svůj vlastní generátor, neboť se jedná o samostatné logické celky. Dokument je spojen se složkou pomocí vztahu OneToOne, neboť každý dokument má právě jednu kořenovou složku. V tomto vztahu je použit kaskádní typ REMOVE, aby se při smazání dokumentu z databáze odstranila i příslušná kořenová složka dokumentu, a atribut orphanRemoval, který zajistí, že entita složky bude smazána, bude-li odstraněna ze vztahu k danému dokumentu. Atribut JoinColumn pak představuje cizí klíč vlastněný entitou Document a odkazující na danou složku.

### 4.2.4 Datová vrstva

Datová vrstva se skládá ze dvou částí - relační databáze (konkrétně PostgreSQL) a filesystému (souborový systém). V databázi jsou ukládána metadata o souborech, složkách a dokumentech. Příkladem metadat souboru je jeho název, datum vytvoření a poslední modifikace, jeho identifikátor, velikost, typ, popis apod...

Ve filesystému jsou pak ukládány fyzické kopie jednotlivých souborů, a to v adresářové struktuře odrážející identifikační číslo a verzi souboru. Hlavní adresář úložiště (defaultně nazývaný angl. "store") obsahuje podadresáře pojmenované dle identifikátoru (fileId) souborů z databáze. Ty pak mají další podadresáře reprezentující jednotlivé verze souboru, které již obsahují fyzické kopie daných souborů. Následuje ukázka možné adresářové struktury.

```
store
├── 1000
│   └── 0
│       └── smile.png
├── 1001
│   ├── 0
│   │   └── obrazek.jpg
│   └── 1
│       └── obrazeklepsi.jpg
├── 1002
│   └── 0
│       └── dokument.txt
└── ...
```

Pokud se stane, že soubor nacházející se v databázi nebude mít svůj protějšek ve filesystému, z databáze se automaticky smaže. Stejně tak naopak, bude-li se ve filesystému nacházet soubor, který již nemá svůj protějšek v databázi, bude z něj automaticky odstraněn. Automatická kontrola a čištění probíhá pravidelně, a to v intervalu, který lze nastavit v konfiguračním souboru application.properties, což popisuji v sekci 4.4.

## 4.3 Zabezpečení

Zabezpečení je řešeno formou kontroly platnosti Bearer tokenu při jakémkoliv požadavku klienta na controllery. Bearer token je bezpečnostní token umožňující jeho vlastníkovi (v angl. označovaný "Bearer") přístup ke stejným zdrojům, k jakým mají přístup ostatní vlastníci daného tokenu [41].

Při úspěšném přihlášení uživatele do autorizačního systému mu server odešle odpověď s platným tokenem v hlavičce "Authorization". Při požadavcích na controllery této aplikace se odešle také daná hlavička, aplikace si ji z požadavku extrahuje a odešle vlastní požadavek na autorizační endpoint definovaný v konfiguračním souboru. V případě platného tokenu dostane odpověď s informacemi o uživateli a stavem 200 OK a pošle jeho požadavek dále, pokud je však token neplatný, odpoví server stavem 401 Unauthorized, kontrolní metoda vyhodí výjimku, zabrání ve zpracování uživatelova požadavku a ten je o tom informován opět HTTP odpovědí se stavem 401.

## 4.4 Konfigurace

Hlavním konfiguračním souborem aplikace je application.properties. Zde jsou uložené přihlašovací údaje k databázi, nastavení limitů pro zpracování souborů, definice hlavního adresáře pro ukládání souborů, autorizační endpoint pro kontrolu platnosti bezpečnostního tokenu či nastavení intervalu pro čištění souborů.

```
## MULTIPART (MultipartProperties)
# Povoleni nahravani vicera souboru
spring.servlet.multipart.enabled=true
# Velikost souboru, po jake bude zapisovan na disk
spring.servlet.multipart.file-size-threshold=2KB
# Maximalni velikost souboru (-1 = neomezena)
spring.servlet.multipart.max-file-size=-1
# Maximalni velikost pozadavku (-1 = neomezena)
spring.servlet.multipart.max-request-size=-1

## Spring DATASOURCE
# URL adresa databaze
spring.datasource.url=jdbc:postgresql://localhost:5432/document-manager
# Uzivatelske jmeno v databazi
spring.datasource.username=postgres
# Heslo k databazi
spring.datasource.password=d4as5ds_ads45%das?hjgwkcvnv3

# Dialekt pouzity pro dotazy na databazi
spring.jpa.properties.hibernate.dialect =
    org.hibernate.dialect.PostgreSQLDialect
# Mod exportu entitnich schemat do databaze
spring.jpa.hibernate.ddl-auto = create

## Nastaveni uloziste (filesystemu)
# Adresar, kam budou ukladany soubory
file.upload-dir=./store

## Udrzba
# Interval pro kontrolu a cisteni zbytkovych souboru
maintenance.lonely-files-cleanup=0 0 4 * * *

## Bezpecnost
# Autorizacni endpoint pro kontrolu bearer tokenu
security.authorization-point=https://tokenchecker.com/checktoken
```

**Ukázka zdrojového kódu 4.5:** Konfigurace: soubor application.properties

# Kapitola 5

# Evaluace

Po implementaci aplikace bylo potřeba otestovat, zda-li splňuje požadavky na svou funkčnost a funguje-li vše tak, jak by mělo. Testování probíhalo dvojím způsobem, jednak pomocí programu zvaného Postman, druhak pomocí sady testovacích scénářů simulujících jeho použití správcem terminologií TermIt.

## 5.1 Testování s Postmanem

Postman je softwarový nástroj, který pomáhá vytvářet a testovat API rozhraní webových aplikací. Lze pomocí něj posílat různé typy HTTP požadavků (GET, POST, PUT, PATCH) a prohlížet si odpověď testovaného serveru např. ve formátu JSON, podporuje autorizační služby, proměnné prostředí (lze si tak např. uložit často používané textové řetězce), sdružování požadavků do kolekcí atd. [42]

Testování pomocí Postmanu probíhalo již během vývoje aplikace, neboť bylo třeba často otestovat, zda dílčí endpointy fungují správně a pokud ne, jaká je odpověď serveru. Požadavky byly vytvořeny pro každý endpoint vystavený vytvořeným REST API, a to i v různých obměnách URI adresy i těla požadavku, abych zkontroloval, jaká bude odpověď serveru v případě chybných argumentů či parametrů.

Celkem bylo vytvořeno a provedeno 45 testů - 12 pro DocumentController, 17 pro FileController a 16 pro FolderController. Struktura mnoha požadavků se opakuje a liší se entitou, které se týká (např. požadavek na zobrazení všech dokumentů vs. požadavek na zobrazení všech souborů). Kolekce všech postman požadavků je ve formátu JSON přidána na přiloženém CD. Následuje seznam vybraných testů s jejich stručným popisem.

- **Název:** createDocument

  **Požadavek:** POST /documents/

  **Odesláno:** JSON objekt s atributy "name"a "description"

  **Očekáváno:** vytvořená entita dokumentu z databáze se všemi atributy

  **Výsledek:** V pořádku

- **Název:** createDocumentWrongBody

  **Požadavek:** POST /documents/

  **Odesláno:** JSON objekt s atributem "description"

  **Očekáváno:** výjimka 400 BAD REQUEST

  **Komentář:** chybí povinný atribut "name"

  **Výsledek:** V pořádku

- **Název:** uploadFilesToDocument

  **Požadavek:** /documents/1000/files

  **Odesláno:** dva soubory (obrázky s příponou .png)

  **Očekáváno:** vytvořené entity souborů z databáze se všemi metadaty, připojené k dokumentu

  **Výsledek:** V pořádku

- **Název:** getDocumentByIdWrongId

  **Požadavek:** GET /documents/-1500

  **Očekáváno:** výjimka 404 NOT FOUND

  **Komentář:** dokument s daným id nemůže existovat, a tudíž ani být nalezen

  **Výsledek:** V pořádku

- **Název:** getFilesByDocument

**Požadavek:** GET /documents/1000/files

**Očekáváno:** metadata všech nahraných souborů přiřazených k danému dokumentu

**Výsledek:** V pořádku

- **Název:** uploadFiles

  **Požadavek:** POST /files/

  **Odesláno:** dva soubory (obrázky s příponou .png)

  **Očekáváno:** vytvoření entity souborů z databáze se všemi metadaty

  **Výsledek:** V pořádku

- **Název:** getFileById

  **Požadavek:** GET /files/1000

  **Očekáváno:** metadata o konkrétím souboru (jeho nejnovější verzi)

  **Výsledek:** V pořádku

- **Název:** downloadFile

  **Požadavek:** GET /files/1000/content

  **Očekáváno:** kopie souboru s daným id

  **Výsledek:** V pořádku

- **Název:** downloadFileVersion

  **Požadavek:** GET /files/1000/content?version=0

  **Očekáváno:** kopie nejstarší verze daného souboru

  **Výsledek:** V pořádku

- **Název:** getFiles

  **Požadavek:** GET /files

  **Očekáváno:** metadata všech souborů z databáze

  **Výsledek:** V pořádku

- **Název:** fileVersionsList

  **Požadavek:** GET /files/1000/versions

  **Očekáváno:** metadata všech verzí daného souboru

**Výsledek:** V pořádku

- **Název:** updateFile

  **Požadavek:** PUT /files/1000/content

  **Odesláno:** obrázek s příponou .jpg)

  **Očekáváno:** aktualizovaná metadata souboru, zvýšené číslo současné verze

  **Výsledek:** V pořádku

- **Název:** deleteFile

  **Požadavek:** DELETE /files/1000

  **Očekáváno:** HTTP odpověď 204 NO CONTENT

  **Výsledek:** V pořádku

- **Název:** createSubFolder

  **Požadavek:** POST /folders/1000/subfolders

  **Odesláno:** JSON objekt s atributem "name"

  **Očekáváno:** metadata nahrané složky, atribut cesty obsahující i rodičovskou složku

  **Výsledek:** V pořádku

- **Název:** getSubFolders

  **Požadavek:** GET /folders/1000/subfolders

  **Očekáváno:** metadata všech podsložek dané složky

  **Výsledek:** V pořádku

- **Název:** getFilesInfolder

  **Požadavek:** GET /folders/1000/files

  **Očekáváno:** metadata všech souborů nacházejících se v dané složce

  **Výsledek:** V pořádku

- **Název:** deleteRootFolder

  **Požadavek:** DELETE /folders/1000/files

  **Očekáváno:** výjimka 409 CONFLICT

  **Komentář:** Kořenová složka dokumentu nemůže být přímo smazána

(ani upravena), lze toho docílit jen smazáním dokumentu.

**Výsledek:** V pořádku

## 5.2 Testování se sadou testovacích scénářů

Cílem tohoto testování bylo ověřit funkcionalitu výsledné aplikace a zkontrolovat, jestli jsou splněny všechny funkční požadavky, které byly stanoveny na základě dohody s vedoucím práce po analýze existujících řešení.

Následuje seznam testovacích scénářů, jejich výsledek a případně způsob řešení vzniklých problémů.

- **Úkol:** Vytvořit dokument

  **Výsledek:** Splněno

- **Úkol:** Nahrát nový soubor do dokumentu

  **Chyba:** Nešlo nahrát soubor přímo do dokumentu, musel se nahrát do kořenové složky.

  **Oprava:** Byl vytvořen endpoint pro nahrání souborů přímo do dokumentu, tedy bez nutnosti znát identifikátor kořenové složky.

  **Výsledek:** Splněno

- **Úkol:** Nahrát samostatný soubor

  **Výsledek:** Splněno

- **Úkol:** Nahrát novou revizi již existujícího souboru

  **Výsledek:** Splněno

- **Úkol:** Získat metadata o souboru (defaultně o poslední verzi)

  **Výsledek:** Splněno

- **Úkol:** Získat metadata o určené verzi souboru

  **Výsledek:** Splněno

- **Úkol:** Stáhnout fyzickou kopii souboru (defaultně poslední verzi)

  **Výsledek:** Splněno

- **Úkol:** Stáhnout určenou verzi kopie souboru

  **Výsledek:** Splněno

- **Úkol:** Odstranit soubor, a to včetně všech revizí

  **Chyba:** Nebyl ošetřen chybový stav, kdy se uživatel pokoušel smazat neexistující soubor.

  **Oprava:** V případě pokusu o smazání neexistujícího souboru je vyhozena výjimka a uživateli vrácen HTTP stav 404 NOT FOUND společně s textovým popisem chyby.

  **Výsledek:** Splněno

- **Úkol:** Odstranit dokument, a to včetně všech jeho subnodů (souborů, revizí a složek)

  **Výsledek:** Splněno

# Kapitola 6

# Závěr

V rámci této bakalářské práce jsem se zabýval analýzou existujících řešení v oblasti systémů pro správu dokumentů, návrhem vlastního řešení, následně implementací a nakonec evaluací vytvořené aplikace.

V analýze jsem prozkoumal nejpoužívanější document management systémy, jejich vlastnosti, architekturu a podporované funkce. Následně jsem porovnal zastoupení jednotlivých funkcionalit v porovnávaných systémech a dle četnosti výskytu je roztřídil. Tento výstup mi pak sloužil jako zamyšlení se nad tím, které funkce bych měl implementovat v našem systému. Společně s vedoucím mé bakalářské práce, panem Ing. Ledvinkou, jsme se pak domluvili na funkčních požadavcích.

Během návrhu vlastního řešení jsem zvolil technologii pro ukládání dat, která nejlépe vyhovuje našim potřebám - kombinaci relační databáze, kam se ukládají metadata týkající se dokumentů, souborů a složek, a souborového systému, kam se ukládají samotné fyzické kopie nahrávaných souborů. Dále jsem popsal navrhovanou architekturu aplikace a logickou strukturu databáze, tj. jakým způsobem budou v ní budou dokumenty, soubory a složky reprezentovány, jaké mezi sebou budou mít asociace, jak propojím logické reprezentace z databáze se soubory v souborovém systému apod.

V implementační části jsem zmínil použité technologie, detailněji popsal implementovanou architekturu z pohledu jednotlivých vrstev, způsob zabezpečení aplikace a nakonec možnosti její konfigurace.

Evaluace mi umožnila ověřit, že identifikované funkční požadavky byly splněny, a že funkce aplikace dodávají korektní výsledek nejen při očekávaných vstupech, ale i těch nesprávných.

Díky práci na tomto projektu jsem získal povědomí o principech fungování document management systémů, výhodách a nevýhodách různých typů databází a řešení pro ukládání souborů, zkušenosti s typografickým systémem LaTeX a psaním technické zprávy většího rozsahu. Díky návrhové a implementační části jsem získal zkušenosti s tvorbou (z pohledu studenta) většího projektu a velice si zlepšil znalost programování webových aplikací a frameworku Spring, který je pro tyto účely vhodný

Co se budoucího vývoje aplikace týče, dala by se rozšířit o pokročilé funkcionality jako je vyhledávání souborů dle různých parametrů, řízení přístupu k souborům na úrovni dokumentů či na základě uživatelských rolí, popř. možnost stahování vícera souborů v jednom archivu.

# Příloha A

# Literatura


[1] Which Is The Open Source Document Management System That Best Meets Your Needs? [online] [vid. 19. 12. 2018].
http://drupalsun.com/bloggerserge/2018/02/17/which-open-source-document-management-system-best-meets-your-needs

[2] Alfresco DMS capabilities [online] [vid. 19. 12. 2018].
https://www.alfresco.com/capabilities/document-management-software

[3] 9 Things you must know about Alfresco [online] [vid. 19. 12. 2018].
https://opensourceforu.com/2016/04/9-things-you-must-know-about-alfresco-an-open-source-ecm/

[4] OpenKM features [online] [vid. 19. 12. 2018].
https://www.openkm.com/en/features.html

[5] OpenKM REVIEW [online] [vid. 19. 12. 2018].
https://reviews.financesonline.com/p/openkm/

[6] LogicalDOC REVIEW [online] [vid. 19. 12. 2018].
https://reviews.financesonline.com/p/logicaldoc/

[7] Product Features and Comparison Matrix [online] [vid. 19. 12. 2018].
https://www.logicaldoc.com/features

[8] SeedDMS features [online] [vid. 19. 12. 2018].
https://www.seeddms.org/index.php?id=3

[9] SeedDMS Open Source Document Management Software Review [online] [vid. 19. 12. 2018].
https://www.efilecabinet.com/seeddms-open-source-document-management-software-review/

[10] Kimios features [online] [vid. 19. 12. 2018].
http://open.kimios.com/features

[11] Feng Office software review [online] [vid. 19. 12. 2018].
https://project-management.com/feng-office-software-review/

[12] FengOffice features [online] [vid. 19. 12. 2018].
http://www.fengoffice.com/web/features.php

[13] Alfresco Content Services architecture [online] [vid. 21. 12. 2018].
http://docs.alfresco.com/6.0/concepts/dev-arch-overview.html

[14] OpenKM Architecture [online] [vid. 21. 12. 2018].
https://www.openkm.com/en/architecture.html

[15] LogicalDOC Architecture [online] [vid. 21. 12. 2018].
https://wiki.logicaldoc.com/wiki/Architecture

[16] ACS Architecture Overview. [cit. 21. 12. 2018], licencováno pod CC BY-SA 4.0. Dostupné z:
https://docs.alfresco.com/sites/docs.alfresco.com/files/public/images/docs/default6_0/acs_60_architecture_overview.png

[17] OpenKM Architecture Diagram [cit. 21. 12. 2018], licencováno pod CC BY-SA 4.0 Dostupné z:
https://www.openkm.com/resources/images/architecture_diagram.png

[18] LogicalDOC File system layout [cit. 21. 12. 2018], licencováno pod CC BY-SA 4.0 Dostupné z:
https://wiki.logicaldoc.com/wiki/File:App_layers.jpg

[19] LogicalDOC App layers [cit. 21. 12. 2018], licencováno pod CC BY-SA 4.0 Dostupné z:
https://wiki.logicaldoc.com/wiki/File:File_system_layout.jpg

[20] SQL vs. NoSQL Databases: What's the Difference? [online] [vid. 21. 12. 2018].
https://www.upwork.com/hiring/data/sql-vs-nosql-databases-whats-the-difference/

[21] SQL vs NoSQL: The Differences [online] [vid. 21. 12. 2018].
https://www.sitepoint.com/sql-vs-nosql-differences/

[22] The SQL vs NoSQL Difference: MySQL vs MongoDB [online] [vid. 21. 12. 2018].
https://medium.com/xplenty-blog/the-sql-vs-nosql-difference-mysql-vs-mongodb-32c9980e67b2

[23] Why use a database instead of just saving your data to disk? [online] [vid. 21. 12. 2018].
https://softwareengineering.stackexchange.com/questions/190482/why-use-a-database-instead-of-just-saving-your-data-to-disk

[24] NOSQL VS RELATIONAL DATABASE FILE STORING (MONGODB AND SQL SERVER COMPARISON) [online] [vid. 21. 12. 2018].
https://codingsans.com/blog/nosql-vs-relational-database

[25] Which is Better ? Saving Files in Database or in File System [online] [vid. 21. 12. 2018].
https://habiletechnologies.com/blog/better-saving-files-database-file-system/

[26] What is the advantage of storing on a file system rather than NoSQL database? [online] [vid. 21. 12. 2018].
https://www.quora.com/What-is-the-advantage-of-storing-on-a-file-system-rather-than-NoSQL-database

[27] Database vs File system storage [online] [vid. 21. 12. 2018].
https://stackoverflow.com/questions/38120895/database-vs-file-system-storage

[28] Layered Architecture Is Good [online] [vid. 22. 5. 2019].
https://dzone.com/articles/layered-architecture-is-good

[29] Software Architecture Patterns - Layered Architecture [online] [vid. 22. 5. 2019].
https://towardsdatascience.com/software-architecture-patterns-98043af8028

[30] How JVM Works – JVM Architecture? [online] [vid. 18. 5. 2019 ]
https://www.geeksforgeeks.org/jvm-works-jvm-architecture/

[31] Differences between JDK, JRE and JVM [online] [vid. 18. 5. 2019]
https://www.geeksforgeeks.org/differences-jdk-jre-jvm/

[32] Report: Java 8 remains the most dominant version of Java [online] [vid. 18. 5. 2019]
https://sdtimes.com/java/report-java-8-remains-the-most-dominant-version-of-java/

[33] Java 8 Overview [online] [vid. 18. 5. 2019]
https://www.oracle.com/technetwork/java/javase/overview/java8-2100321.html

[34] Inversion of Control Containers and the Dependency Injection pattern [online] [vid. 18. 5. 2019]
https://martinfowler.com/articles/injection.html

[35] Spring - Core Technologies [online] [vid. 18. 5. 2019]
https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html

[36] Spring Boot [online] [vid. 18. 5. 2019]
https://spring.io/projects/spring-boot

[37] Spring Data JPA [online] [vid. 18. 5. 2019]
https://spring.io/projects/spring-data-jpa

[38] What is maven? [online] [vid. 18. 5. 2019]
https://maven.apache.org/what-is-maven.html

[39] PostgreSQL: About [online] [vid. 18. 5. 2019]
https://www.postgresql.org/about/

[40] RequestBody and ResponseBody Annotations [online] [vid. 18. 5. 2019]
https://www.baeldung.com/spring-request-response-body

[41] The OAuth 2.0 Authorization Framework: Bearer Token Usage [online] [vid. 21. 5. 2019]
https://tools.ietf.org/html/rfc6750

[42] Introduction to Postman for API Development [online] [vid. 18. 5. 2019]
https://www.geeksforgeeks.org/introduction-postman-api-development/

# Příloha B

## Seznam použitých zkratek

- **ACID** - Atomicity, Consistency, Isolation, Durability - 4 základní vlastnosti databázových transakcí
- **API** - Application Programming Interface - rozhraní aplikace pro programátory
- **DAO** - Data Access Object - objekt poskytující abstraktní rozhraní pro databázi
- **DMS** - Document Management System - systém pro správu dokumentů
- **FP** - funkční požadavek
- **GUI** - Graphical User Interface - grafické uživatelské rozhraní
- **HTML** - Hypertext Markup Language - značkovací jazyk používaný pro tvorbu webových stránek
- **JPA** - Java Persistence API - Java standard pro objektivně-relační mapování
- **JSON** - JavaScript Object Notation - způsob objektového zápisu v jazyce Javascript
- **LDAP** - Lightweight Directory Access Protocol - protokol pro ukládání a přístup k datům na adresářovém serveru
- **OASIS** - Organization for the Advancement of Structured Information Standards - organizace spravující informatické standardy
- **REST** - Representational State Transfer - architektura rozhraní pro distribuované prostředí

- **SOAP** - Simple Object Access Protocol - síťový protokol na výměnu zpráv přes XML
- **SMTP** - Simple Mail Transfer Protocol – protokol pro přenos e-mailovů
- **SQL** - Structured Query Language - dotazovací jazyk užívaný v relačních databázích
- **UI** - User Interface - uživatelské rozhraní
- **URI** - Uniform Resource Identifier - jednotný identifikátor zdroje
- **XML** - eXtensible Markup Language - značkovací jazyk

# Příloha C

## Obsah přiloženého CD

Součástí práce je přiložené CD s následující strukturou a obsahem:

- `documentation`
  - `BP2019-malinovmartin-docmanagemer.pdf` .... bakalářská práce ve formátu PDF
  - `tests.postman-collection.json` ..... kolekce postman testů
- `application`
  - `document-manager.jar`
  - `document-manager-prototype` ..... hlavní adresář aplikace
    - `src`
      - `main`
        - `java` ..... adresář s java soubory
        - `resources` ..... adresář s konfigurací
          - `application.properties` ..... konfigurační soubor
    - `pom.xml`
- `latex` ..... adresář se zdrojovým kódem textu bakalářské práce

# Příloha D

# Návod ke spuštění

Ke spuštění je třeba mít nainstalované následující technologie:

- Java 8 (či novější)
- Apache Maven 3 (či novější)
- PostgreSQL 9.5 (či novější)

## D.1 Spuštění příkazem

Defaultně je konfigurace nastavena na mou školní databázi, stačí tedy aplikaci spustit. Otevřete příkazový řádek (/terminál) v adresáři se souborem document-manager.jar a zadejte následující příkaz:
**java -jar document-manager.jar**

## D.2 Kompilace aplikace

Můžete si ale také nastavit databázi vlastní. V takovém případě bude potřeba si aplikaci zkompilovat.

1. Otevřete soubor application.properties
   (adresář *document-manager-prototype\src\main\resources*).
2. Upravte si adresu databáze, uživatelské jméno a heslo dle svých potřeb.
3. Otevřete příkazový řádek v hlavním adresáři projektu a zadejte příkaz:
   **mvn clean package**
4. Přesuňte se do adresáře *target*, kde se vám vytvořil soubor s názvem
   file-manager-prototype-0.0.1-SNAPSHOT.jar
5. Spusťte příkazem: java -jar file-manager-prototype-0.0.1-SNAPSHOT.jar

Server se spustí na adrese http://localhost:8080.