FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Informační systém pro správu fyzických dokumentů
Student:	Matyáš Gallas
Vedoucí:	Ing. Marek Suchánek
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

Cílem práce je zanalyzovat problematiku správy fyzických dokumentů (například smluv a jejich příloh) a na základě toho navrhnout a implementovat informační systém podporující související činnosti. Tento systém musí být dobře udržitelný a snadno rozšiřitelný. Práce bude zpracována dle tradičních metod softwarového inženýrství.

- Analyzujte doménu a procesy práce s fyzickými dokumenty a sestavte požadavky na systém podporující tyto činnosti.
- Proveďte stručnou rešerši existujících řešení pro správu fyzických dokumentů.
- Navrhněte systém ve formě webové aplikace s využitím třívrstvé architektury. Při návrhu zohledněte potřebu rozšiřitelnosti systému s využitím modularizace.
- Implementujte systém dle návrhu a řádně jej otestujte a zdokumentujte. Volbu konkrétních technologií (např. webový framework) řádně zdůvodněte.
- Zhodnoťte přínosy využití systému v organizacích a popište možnosti rozvoje systému.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 6. prosince 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Informační systém pro správu fyzických dokumentů

Matyáš Gallas

Katedra softwarového inženýrství
Vedoucí práce: Ing. Marek Suchánek

31. května 2020

Poděkování

Rád bych poděkoval svému vedoucímu Ing. Marku Suchánkovi za pomoc a rady při zpracování této práce. Také mé poděkování patří Komerční bance za umožnění této práce. Velké poděkování náleží mé rodině a přátelům za podporu, cenné rady a trpělivost po dobu mého studia.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, avšak pouze k nevýdělečným účelům. Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 31. května 2020 ……

Odkaz na tuto práci

Gallas, Matyáš. *Informační systém pro správu fyzických dokumentů.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Tato bakalářská práce se zabývá vytvořením funkční aplikace pro správu fyzických dokumentů, které se týkají úvěrů v prostředí Komerční banky. Aplikace zjednodušuje a zefektivňuje jejich správu a přináší nové možnosti rozvoje oproti stávajícímu řešení v podobě webové aplikace. Práce obsahuje analýzu problematiky správy převážně v prostředí Komerční banky, ale i obecné správy dokumentů. Součástí analýzy je také sestavení požadavků, zhodnocení současného stavu a existujících řešení správy. Následuje část zaměřená na návrh systému společně s výběrem vhodné technologie pro implementaci. Dále práce obsahuje popis implementace a testování. Výsledkem je rozšiřitelný funkční informační systém, který je možné nasadit a využívat ke správě fyzických dokumentů.

Klíčová slova informační systém, webová aplikace, správa dokumentů, archivace, analýza, DMS, Spring framework, Angular, Java, Oracle

Abstract

This bachelor thesis deals with the development of a useable information system for the management of physical documents related to loans in the environment of Komerční banka. The information system implemented as a web application. It simplifies administration and allows for new future development possibilities compared to the existing solution. The thesis contains an analysis of domain issues mainly in the environment of Komerční banka, but also a general context. The analysis also includes system requirements specification, evaluation of the current application, and existing document management system solutions. The next part focuses on the system designing and selecting the technology for the application. The thesis also contains a description of the implementation and testing of the developed system. The result is an extensible information system that can be deployed and used for managing the physical documents.

Keywords information system, web application, document management, archive, analysis, DMS, Spring framework, Angular, Java, Oracle

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Společnosti (primárně banky) uchovávají ke svým klientům rozsáhlé množství dokumentů stále ještě v papírové formě, a to navzdory rozvíjejícímu se trendu uchovávání dat digitálně. Důvodů je často více. Za hlavní lze považovat finanční náročnost konverze. Jako druhý, který nelze bankou z velké části ovlivnit, jsou samotní klienti, kteří stále vyžadují dokumentaci ke své smlouvě v listinné podobě a nevystačí si pouze s elektronickým záznamem. Posledním důvodem, který považuji za podstatný, je elektronický podpis. K uzavření smlouvy tzv. on-line je nutné vlastnit elektronický podpis. Mezi českou populací není dostatečně rozšířen, proto by banka musela vystavit vlastní elektronický podpis. Banky v tomto ohledu vyčkávají, protože prvotní investice by byly značné. Případně by si klient musel na své náklady zařídit elektronický podpis, což by mohlo mít negativní vliv na uzavření smlouvy. Stále existují dokumenty, které není možné uchovávat elektronicky. A proto je stále ještě nutné pracovat s dokumenty v papírové podobě.

Fyzické dokumenty je třeba nejen skladovat, ale i uchovávat informaci o jejich umístění, pro potřeby přístupu, nebo vyloučení již expirovaných dokumentů k následné skartaci. Pokud společnost obecně řeší ukládání dokumentů, neorientují se na ukládání dle iniciálů na dokumentech. Vytvářejí specifické číselné řady, a to z toho důvodu, že není možné odhadnout rozložení úložné plochy dle těchto iniciálů. Tato skutečnost iniciuje zadání, jak vyřešit ukládání fyzických dokumentů tak, aby bylo efektivní a přehledné.

Předpoklady pro efektivní práci s dokumenty evidovanými ve vytvořené aplikaci, je v její možnosti využití nejen v rámci pracovníků oddělení, kteří s dokumenty přímo pracují, ale i napříč organizací pro případné vypůjčení daného dokumentu. Na druhou stranu při možnosti sdílení informace o dokumentech napříč společností, je obvykle vyžadováno dodržet vysoký stupeň bezpečnosti takových informací a zamezení nežádoucích úniků.

Informační systém pro správu fyzických dokumentů je proto nezbytnou součástí pro udržení pořádku a organizace. Mnoho firem s velkým objemem

dokumentů využívá již nějaký systém delší dobu, nejčastěji realizován pomocí desktopové aplikace, která se hůře rozšiřuje a špatně integruje do dalších systému. Navíc je nutná instalace na konkrétním zařízení, kde se mohou vyskytovat problémy s kompatibilitou.

Pro splnění výše uvedených předpokladů se nabízí využití informačního systému ve formě webové aplikace ke správě fyzických dokumentů.

Kapitola 1

Cíl práce

Cílem bakalářské práce je analyzovat, navrhnout a implementovat funkční informační systém pro správu fyzických dokumentů – konkrétně úvěrů – na základě metodik Komerční banky. Zpracována bude dle tradičních metod vývoje softwaru s možným budoucím rozšířením a integrací. Přístup bude zajištěn pomocí webového rozhraní dostupného primárně ze stolního počítače, notebooku nebo tabletu. Informační systém musí být snadno ovladatelný a zabezpečený před neoprávněným užitím nepovolaných osob.

Pro účely analýzy bude proveden průzkum již existujícího řešení z oblasti správy fyzických dokumentů za účelem poučit se z nedostatků a úspěchů těchto řešení. V neposlední řadě práce zohledňuje možné využití a rozšíření této aplikace.

Práce je zpracována dle tradičních metod softwarového inženýrství. V první části se zabývá analýzou dané problematiky, výběrem vhodných technologií a rozborem existujících řešení. Z této analýzy vyplývají funkční a nefunkční požadavky, dle kterých je následně navržena aplikace. Následuje popis implementace a testování společně s dokumentací. Poslední kapitola se zabývá potencionálním rozšířením a využitím navrženého systému.

Kapitola 2

Problematika správy fyzických dokumentů

Tato kapitola seznamuje s problematikou správy fyzických dokumentů, zaměřenou převážně na dokumenty týkající se úvěrů. Dále se zabývá jednotlivými metodami správy a procesy s fyzickými dokumenty spojené v prostředí Komerční banky. Veškeré informace o Komerční bance byly získány po osobním setkání a rozhovoru s jejím zaměstnancem. [10]

Banky potažmo jakékoli společnosti sjednávající úvěry jsou povinny se řídit platnými nařízeními ČNB (Česká národní banka) a respektovat platné zákony [6]. To v případě poskytování úvěru obnáší i uchovávání informací o údajích, které klient sdělil při své žádosti (např. žádost, dotazník nebo příjmy), případně doložil jako podklad pro poskytnutí úvěru (např. kupní smlouva nebo stavební dokumentace). Při sjednání úvěru s klientem vznikají další dokumenty (např. smlouva nebo zástavní smlouva), které je rovněž potřeba uchovat. V některých případech se k již sjednaným úvěrům přidávají další dokumenty. Všechny tyto dokumenty je zapotřebí skladovat a udržovat přehled o jejich umístění. Množství dokumentů u velkých firem není malé, a proto je zapotřebí využívat patřičný software pro jejich organizaci.

2.1 Úvěr

Úvěr je obecně formou poskytnutí finančních prostředků s určitou splatností vypůjčené částky společně s úroky. Úrokem se rozumí finanční odměna za vypůjčené prostředky [7].

Úvěrů je celá řada druhů dle typu subjektu, pro který jsou poskytovány, dle účelu úvěru a v neposlední řadě dle způsobu zajištění. U každého typu je určitý předpoklad množství dokumentů potřebných k jeho uzavření [8]. Například pro spotřebitelské úvěry je potřeba doložit informaci o příjmech klienta, pro podnikatelské úvěry je potřeba dokládat výkazy a daňová přiznání.

Již z tohoto srovnání je patrné, že doklad o příjmech klienta může být obvykle jednostránkový, naproti tomu výkazy klienta (např. roční konečné nebo čtvrtletní průběžné) a daňová přiznáni mohou obsahovat až stovky stránek. Není účelem této práce specifikovat úplné výčty dokumentů vázaných k jednotlivým typům úvěru a to i s ohledem na to, že některé případy uzavřených úvěrů si žádají individuální dokumenty.

Z pohledu procesů vázaných k ukládání dokumentů stojí za zmínku i skutečnost, že v průběhu trvání jednotlivých obchodů se i k těmto mohou přidávat další dokumenty, týkající se například fixace, předčasného splacení nebo dodatky či změny. [2]

2.2 Komerční banka

Komerční banka je akciovou a mateřskou společností Skupiny KB a součástí mezinárodní skupiny Société Générale. Banka byla založena v roce 1990 jako státní instituce a v roce 1992 byla transformována na akciovou společnost. KB je univerzální bankou se širokou nabídkou služeb v oblasti retailového, podnikového a investičního bankovnictví. Mezi její služby patří mimo jiné penzijní připojištění, stavební spoření, faktoring, pojištění a již zmíněné úvěry. [3, 4]

2.3 Způsoby uchovávání

Dokumenty lze uchovávat jak v elektronické, tak i fyzické podobě. Elektronická podoba má bezesporu mnoho výhod, ovšem zde je potřeba zohlednit, zda je pro banku důležité uchovávat originál či postačuje kopie. V případě, že je potřeba uchovat originální dokument, je potřeba se podívat, jak tento dokument vznikl. Pokud vznikl jako papírový dokument s fyzickým podpisem, nezbývá bance nic jiného nežli takový dokument uchovat v jeho původní podobě.

Uchovávání v elektronické podobě je efektivnější, dokumenty jsou okamžitě dostupné a odpadají problémy s fyzickou skartací, nebo uskladněním. Nepochybně v budoucnu zanikne fyzické ukládání dokumentů a bude se používat pouze elektronické. S touto metodou je spojeno vysoké zabezpečení dat a nutnost elektronického podpisu klienta u důležitých dokumentů.

Elektronický podpis nahrazuje fyzický podpis klienta u dokumentů v elektronické podobě. Problém nastane, pokud klient elektronický podpis nemá. V takovém případě může společnost vytvořit vlastní certifikáty pro každého klienta a umožnit mu tak elektronický podpis uskutečnit. Vydávání vlastních certifikátů je pro společnost velmi nákladné. Další možností je nechat klienta si elektronický podpis zařídit u certifikační společnosti. Tato služby je většinou paušálně zpoplatněna.

Rovněž je vhodné na tomto místě poukázat na skutečnost, že existují dokumenty, které se zatím ani legislativně nepodařilo upravit tak, aby mohly být elektronické již při svém vzniku. Příkladem takovéhoto typu dokumentu může být blankosměnka. Směnka obecně není přesně zákonem definovaná. Směnku si tedy můžeme představit jako cenný papír, na kterém je uveden bezpodmínečný závazek zaplatit majiteli směnky v určitý den a na určitém místě dlužnou peněžní částku. [5]

Neumožněním využívat papírové varianty dokumentů, může společnost přicházet o potencionální obchody. V současnosti je tedy stále žádoucí souběžně pracovat jak s elektronickými, tak papírovými dokumenty.

Zavádění elektronického uchovávání do již fungujícího fyzického uchovávání je velmi náročné z hlediska úpravy a přizpůsobení ostatních systémů, nemluvě o převodu již existujících dokumentů. I přesto se některé společnosti vydaly tímto směrem a zavádějí elektronickou evidenci.

Z výše uvedeného je patrné, že se v dohledné době bankám nepodaří plně zbavit papírových dokumentů (pokud by nedošlo k omezení některých produktů nebo služeb). To znamená, že přes snahu všech bank digitalizovat své procesy, nebude možné se plně zbavit papíru ještě po dlouhou dobu. Banky, i další instituce pracující s velkým množstvím dokumentů a vždy propočítávají návratnost svých investic. Pokud se investice do převodu stávajících papírových dokumentů do digitálních nebude vyplácet (nebo bude přinášet neadekvátní rizika), dělat je nebudou. Bude tedy existovat jak svět digitálních dokumentů, tak svět těch papírových vedle sebe souběžně. Je samozřejmě předpoklad, že svět papírový bude na ústupu. Po celou tuto dobu ovšem bude nutné mít přesnou evidenci jak o elektronických dokumentech, tak i o těch papírových.

2.4 Fyzické uchovávání

Fyzické dokumenty uchovávají společnosti převážně v klimatizovaných skladech. Sklady jsou vybaveny skladovacími jednotkami, nejčastěji ve formě regálů s krabicemi nebo složkami, kde fyzické dokumenty reálně leží.

V případě KB, jsou dokumenty úvěrových obchodů ukládány do složek, přičemž jedna složka odpovídá jednomu sjednanému produktu. V takové to složce jsou pak uchovávány všechny dokumenty s produktem spojené. Dokumenty musí v takových to úložištích (úložiště pracovní dokumentace) vydržet i dlouhá léta. Po uplynutí této doby se dokumenty přesunou do centrálního úložiště IRON (Iron Mountain) a po určité době bezpečně dle zákona zlikvidují tak, aby nebylo možné získat informace, které by mohly poškodit společnost nebo klienta. [9]

KB rozlišuje několik typů složek, a to primárně z důvodu formy dané složky. Například podnikatelské úvěry jsou ve složkách typu široký šanon, jelikož obsahují velké množství dokumentace. Naproti tomu kreditní karta je

vkládána do euro složky, poněvadž k jejímu vydání je zapotřebí pouze smlouva a minimum dalších dokumentů. Typ složky tedy odpovídá typu úvěru.

2.4.1 Manipulace se složkami

V průběhu skladování se ke složkám přistupuje. Mohou být vypůjčeny za účelem kontroly, zda obsahují požadované dokumenty, které mohou být i následně vypůjčené a vráceny zpět na předchozí místo vypůjčeni. Složky jsou v případě KB řazeny dle typu složky a označeny číselným pořadím v řadě.

2.4.2 Průběh skartace v KB

Každý produkt má určitou životnost. Po této době se může složka s daným produktem, případně její obsah, přesunout do archivační krabice. Pro dohledání v krabici je této skupině dokumentů (složce) přiděleno pořadové číslo v krabici. Archivační krabice je vytvořena na lokalitě úložiště pracovní dokumentace, kde jí pracovník přidělí skartační značku, viz tabulka 2.1. Po naplnění a uzavření se archivační krabice převeze z úložiště pracovní dokumentace do centrálního úložiště IRON (Iron Mountain), kde se po stanovené době dle skartační značky na pokyn pracovníků KB bezpečně skartují. [9]

Skartační znak	Význam
S4	Skartovat po 4 letech
S5	Skartovat po 5 letech
S10	Skartovat po 10 letech
V5	Po 5 letech ověřit skartaci
V10	Po 10 letech ověřit skartaci

Tabulka 2.1: Tabulka se skartačními znaky

2.4.3 Způsob řazení

V regálech jsou složky číslovány vzestupně. Při expiraci dokumentu a jeho následné archivaci se pozice složky uvolní. Tímto způsobem vznikají v souvislých řadách mezery, které je třeba zaplnit novými dokumenty. Po vyčerpání poslední volné složky se přidá další na konec řady s následujícím pořadovým číslem. Takto se pokračuje do vyčerpání místa v regálu. Poté se určí další regál a v číslování se pokračuje.

2.4.4 Úložiště pracovní dokumentace

Je úložiště dokumentů vedené společností KB na určitých lokalitách. Zde se skladují veškeré papírové dokumenty do jejich převezení do centrálního úložiště po uběhnutí expirační doby.

2.4.5 Centrální úložiště Iron Mountain

Společnost Iron Mountain je organizace zabývající se ukládáním, ochranou a správou dokumentů, dat a aktiv. Nezabývá se pouze fyzickým uchováváním dokumentů ale také elektronickým. Součástí správy je i zajištění bezpečné skartace dokumentů. [11]

KB tuto společnost využívá jako úložiště pro již expirované fyzické dokumenty, u kterých se nepředpokládá již další vypůjčování a čekají na skartaci.

Kapitola 3

Analýza

Tato kapitola se zabývá analýzou procesů souvisejících se správou v prostředí KB. Dále jsou sestaveny požadavky na samotnou aplikaci. Na závěr se zkoumá existující řešení společně s jeho výhodami a nevýhodami.

3.1 Procesy

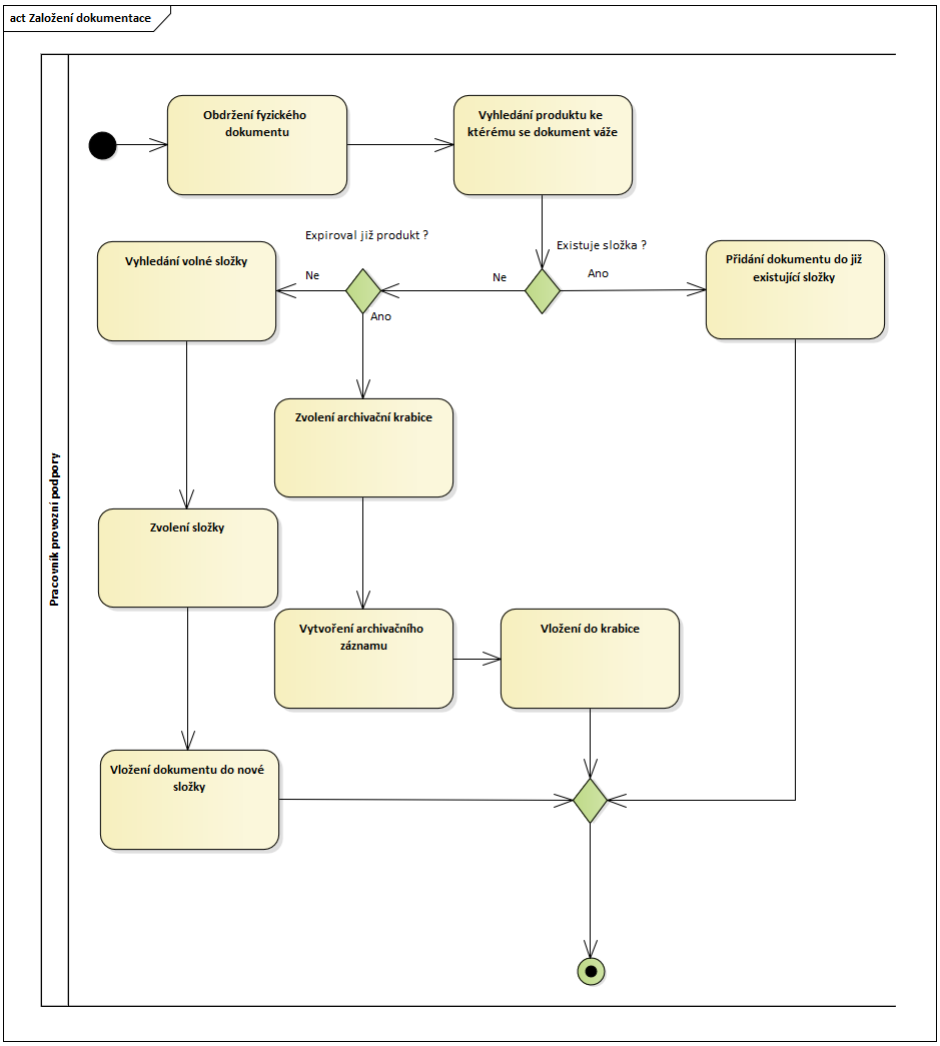
3.1.1 Založení dokumentu

Samotnému založení dokumentu do skladiště pracovní dokumentace předcházejí procesy v rámci kterých dokumenty vznikají. Jde o iteraci s klientem prostřednictvím dalších bankovních systémů, přičemž není účelem této práce tyto popisovat. Důležitá je skutečnost, že v rámci uvedené iterace vznikají záznamy o produktu, které lze následně využít v rámci aplikace pro správu dokumentů.

Proces založení fyzického dokumentu začíná v okamžiku, kdy pracovník provozní podpory obdrží fyzický dokument. Pracovník následně vyhledá, jestli již daný produkt, ke kterému je dokument vázán, již nemá na dané lokalitě svoji složku. Pokud složku má, pouze tento dokument určený k uskladnění do vyhledané složky přidá. V druhém případě musí nejprve rozhodnout, jestli dokumentu založí novou složku, nebo uloží do některé existující archivační krabice na dané lokalitě. Pracovník se rozhoduje o uložení na základě expirační doby produktu. Pokud již produkt expiroval a jeho dokumenty byly odeslány do centrálního skladiště Iron, umístí dokument do krabice.

Při založení nové složky pro produkt nejprve pracovník najde fyzicky volnou složku v regálu dle typu produktu. Složky se zaplňují vzestupně, což znamená, že po expiraci některého produktu vznikne mezera s prázdnou složkou, která se zaplní. Tímto způsobem nevznikají velké mezery a docílí se tak maximálního šetření místem. Pokud již žádná fyzická složka volná není, pokračuje se v řadě vytvořením další složky v pořadí a následným umístěním fyzického dokumentu do této složky. ¨

Obrázek 3.1: Proces založení dokumentu



3.1.2 Založení krabice

Krabice slouží pro fyzické dokumenty nebo složky k produktům, které již expirovaly. Činnost založení krabice dělá převážné pracovník provozní podpory.

Na dané lokalitě pracovník fyzicky označí krabici určenou pro archivaci dokumentů. Označením krabice se rozumí nalepení samolepky dodané společností IRON, která obsahuje jedinečné číslo (zapsané i v podobě čárového kódu). Toto jedinečné číslo se zaznamená do evidence a dále se může uvést dodatečná poznámka (specifika dané krabice). Krabice je dále opatřena skartačním kódem, který informuje o tom, co a po jaké době se s dokumenty má stát. Příkladem je skartační kód, viz tabulka 2.1.

3.1.3 Uzavření krabice

Tento proces je také prováděn převážně pracovníkem provozní podpory. Po naplnění archivační krabice dokumenty se tato uzavře a odešle do centrálního skladiště Iron (v dávkách). V tomto skladišti je následně uskladněna po stanovenou dobu dle skartačního kódu na krabici.

3.1.4 Archivace dokumentu

Pracovník provozní podpory vyhledá expirovanou složku s fyzickými dokumenty určenou k odeslání a umístí její dokumenty do archivační krabice. Dokumenty takto umístěné do krabice dostanou své pořadové číslo, které je unikátní pro danou archivační krabici. Všechny dokumenty konkrétního produktu sdílí jedno pořadové číslo.

3.1.5 Vypůjčení dokumentu

Proces vypůjčeni se dělí na několik subprocesů. Názorná ukázka vypůjčení, viz diagram na obrázku 3.2

3.1.5.1 Rezervace

Rezervaci dokumentu může provést jak žadatel dokumentace, tak pracovník provozní podpory. Tato osoba vyhledá dokument, který chce zarezervovat. Pokud je již rezervován, nebo vypůjčen, musí vyčkat do doby, než bude navrácen. Případně může kontaktovat člověka, který má dokumentaci vypůjčenou a vyzvat jej k vrácení. Rezervace v tomto kontextu znamená označení složky, že s touto bude aktuálně manipulováno. Je-li tedy složka volná, může se vytvořit záznam o rezervaci. Záznam o rezervaci obvykle obnáší údaje o pracovníkovi, který rezervaci učinil, o uživateli, který žádá dokumentaci, útvar žadatele dokumentace a místo pro doručení dokumentace.

3.1.5.2 Výpůjčka

Výpůjčkou se v kontextu uvedených procesů rozumí vytvoření záznamu na úrovni složky, že tato nebo její část se nachází mimo úložiště (tedy u konkrétního pracovníka, který o dokumentaci žádal), a dále fyzické předání nebo zaslání uvedené dokumentace. Vypůjčit dokument smí pouze člověk se stejnou lokalitou jako má tento dokument. Pokud o vypůjčení dokumentu žádá pracovník z jiné lokality, pracovník provozní podpory lokality dokumentu rozhodne, zda je vhodnější dokument jen naskenovat a zaslat žadateli, nebo se bude dokumentace opravdu zasílat poštou na jinou lokalitu. K tomuto rozhodnutí obvykle pracovník dospěje na základě objemu vypůjčené dokumentace. V případě rozhodnutí o skenování dokumentace tedy nemusí nutně dojít k výpůjčce, ale pracovník provozní podpory blokaci zruší, jelikož nedochází k fyzickému vypůjčení dokumentů.

3.1.5.3 Vrácení

Pokud u pracovníka, který si dokumentaci vypůjčil pomine potřeba s dokumentací dále pracovat, vrací tuto pracovníkovi provozní podpory. Tento vyhledá odpovídající složku a vrátí složku nebo dokumenty zpátky do úložiště pracovní dokumentace. Současně tento pracovník změní záznam na složce tak, že zruší záznam o výpůjčce a složka zůstává do další změny kompletní.

3.1.6 Vyhledání složky s dokumenty

Dokumenty se vyhledávají jednoznačně dle čísla učtu produktu. Také lze vyhledávat pomocí klienta, ke kterému byl dokument evidován. Klient může mít více produktů, a proto je třeba znát i jiná specifika. Na základě tohoto vyhledání evidence identifikuje odpovídající řadu a pořadové číslo. Pracovník následně vyzvedne dané dokumenty ve skladu dle uvedené řady (dle typu produktu) a pořadového čísla v řadě.

3.2 Požadavky

3.2.1 Uživatelské požadavky

Hlavním požadavkem uživatelů budoucí aplikace je stabilita, bezpečnost dat, snadná ovladatelnost a webové rozhraní. Dále také možnost přistupovat i k informacím o dokumentech z jiných úložišť pracovní dokumentace. Klíčovým faktorem je srozumitelný výstup a přehledné zobrazení požadovaných informací, aby mohli uživatelé rychle pracovat s daty a dosahovali lepší produktivity práce.

3.2.2 Funkční požadavky

F1. Evidence dokumentů

- F1.1. Existence několika úložišť
- F1.2. Evidence složek v několika souvislých řadách v každém úložišti
- F1.3. Číslování složek v řadách vzestupně (zaplňují se prázdné)
- F1.4. Typové pokrytí řady složek jsou jednotné pro všechny úložiště
- F1.5. Možnost evidence samostatných dokumentů
- F1.6. Umístění dokumentu do prázdné složky

F2. Evidence archivovaných dokumentů

- F2.1. Evidence beden s dokumentací
- F2.2. Evidence lokality bedny
- F2.3. Evidence pořadového čísla dokumentu v bedně

F3. Vytváření logu

- F3.1. Záznam při vytvoření dokumentu
- F3.2. Záznam při blokaci, vypůjčeni a vrácení dokumentu
- F3.3. Záznam při archivaci

F4. Poznámkování

- F4.1. Uvést poznámku na složce, dokumentu a krabici
- F4.2. Poznámka při blokaci, vypůjčení, vrácení

F5. Uživatelé

- F5.1. Přihlášení a odhlášení
- F5.2. Evidence uživatelů

F6. Archivace

- F6.1. Umístění dokumentu do krabice
- F6.2. Vytvoření archivační bedny

F7. Vyhledání

- F7.1. Hledání podle klienta nebo produktu
- F7.2. Zobrazení krabic na lokalitě
- F7.3. Zobrazení blokovaných dokumentů na lokalitě

F8. Manipulace

- F8.1. Blokace dokumentu
- F8.2. Výpůjčka dokumentu

3.2.3 Nefunkční požadavky

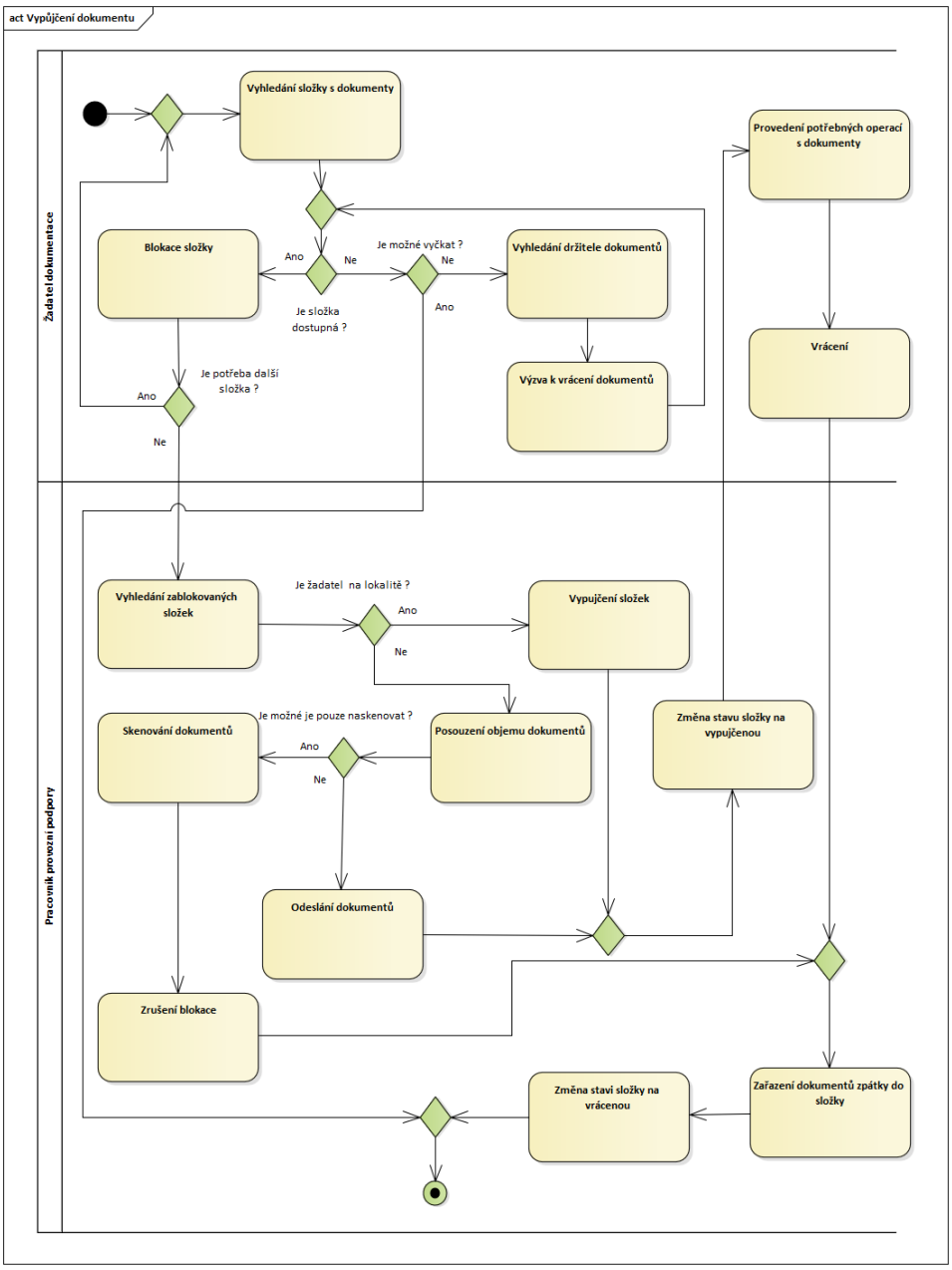
N1. *Webové rozhraní* — Aplikaci bude možné obsluhovat pomocí grafického rozhraní z webového prohlížeče. Bude podporovat prohlížeče Google Chrome a Edge. Minimální rozlišení je Full HD (1920 × 1080). V případě menšího rozlišení, může nastat situace nepřehlednosti zobrazených informací.

N2. *Rozšiřitelnost* — Návrh systému bude proveden s ohledem na budoucí rozšiřitelnost nebo úpravu existujících funkcí.

N3. *Různé uživatelské role* — Uživatelé budou mít omezen přístup pro vykonání některých operací dle jejich role.

N4. *Bezpečnost* — Do aplikace se nebude možné přihlásit bez správného uživatelského jména a hesla. Bezpečnost a použité protokoly jsou vybrány na základě bezpečné interní sítě, kde se předpokládá provozovaní tohoto softwaru.

Obrázek 3.2: Proces vypůjčení



3.3 Účastníci

Pro přístup do aplikace potřebuje uživatel oprávnění. Existuje několik typů uživatelů, kteří mají různá oprávnění.

- **Pracovník provozní podpory** — Zabývá se manipulací s dokumenty. Má veškerá oprávnění manipulovat s dokumenty na lokalitě pracovníka, až na právo mazací. Smazání jakéhokoliv vytvořeného záznamu, ať už dokumentu, logu nebo krabice není tomuto uživateli umožněno. V případě chyby a následné nutnosti mazání, musí kontaktovat uživatele s vyšším oprávněním. Pracovník provozní podpory se stará o zakládání fyzických dokumentů do složek, archivací expirovaných dokumentů. Dále řeší výpůjčky na základě požadavku blokace od žadatele dokumentace. Tyto dokumenty může buď doručit osobně, odeslat interní poštou nebo naskenovat a odeslat v elektronické podobě.

- **Žadatel dokumentace** — Pracovník, který požaduje určitý druh dokumentace. Může požadovat dokumentaci i z jiné lokality. U tohoto typu uživatele se předpokládá vyhledání dokumentace prostřednictvím klienta nebo konkrétního produktu. Typickým představitel těchto uživatelů jsou pracovníci zabývající se auditem a správou klientských účtů. V případě pracovníků auditu, vyžadují konkrétní typy dokumentace pro provedení auditu v určitém časovém úseku. Pracovníci správy klientských účtů provádějí kontrolu a opravy produktů, u kterých byla identifikována chyba při automatickém ukládání do systému.

- **Administrátor** — Jedná se o vedoucího provozní podpory, který má veškeré oprávnění společně s možností smazat dokumenty nebo krabice, za účelem opravení chyby způsobené pracovníkem provozní podpory.

- **Auditor** — Tento uživatel má možnost pouze vyhledávat a zobrazit dokumenty. Většinou se jedná o vedoucí pracovníky.

3.4 Rozsah produktu

Software bude splňovat základní funkčnosti nezbytné k jeho užívání. Bude podporovat zakládání složek, vyhledání, archivaci, logování a vypůjčování dokumentů. Pro plné nasazení do provozu, je ale třeba ještě do-implementovat některé funkčnosti. Jedná se například o rozhraní pro správu práv nebo statistiky či atypické manipulace nebo výpůjčky s dokumenty.

Software je navržen tak, aby tyto funkčnosti mohly být v budoucnu snadno implementovány.

3.5 Stávající řešení

Pokud budeme analyzovat stávající řešení je nutno uvést, že KB toto řešení považovala za dočasné. Vzniklo jako lokální řešení pro potřeby organizace fyzických dokumentů nesoucí název EPRDoc. Toto řešení z důvodu dočasnosti, bylo realizována na platformě Microsoft Access, a to jako desktopová aplikace uložená na sdíleném úložišti. Soubor obsahoval jak aplikaci, tak i samotná zdrojová data (záznamy o dokumentech). Vlastní aplikace spočívala v definici formulářových oken a maker nad vlastními daty viz uživatelské rozhraní 3.3.

Součástí vlastních dat aplikace byla uložena i informace o právech uživatelů. Pro získání přístupu do sdílených disků musel být uživatel přihlášen. Aplikace následně provedla ověření uživatele prostřednictvím čtení přihlášeného uživatele a vyhodnotila jeho oprávnění pro přístup do aplikace.

Díky otevřenosti dat uloženého na sdíleném disku se může aplikace zdát málo zabezpečená, ale v kombinaci se všemi dalšími oprávněními (např. sdílené disky nebo access listy) je toto riziko redukováno. Pro samotný přístup na disk musí být uživatel přihlášen na firemním počítači a mít oprávnění čtení a zápisu na tuto část disku. Toto oprávnění mají pouze povolané osoby. Pokud by tedy chtěla osoba s uvedeným oprávněním jakkoli manipulovat s daty, potřebovala by k tomuto odpovídající SW (např. MS Access), který má instalována pouze vybraná skupina uživatelů převážně zabývající se analýzou dat. Běžný uživatel má přístup pouze do rozhraní formulářových oken.

Každá lokalita s úložišti pracovní dokumentace má vlastní sdílené úložiště (disk), na kterém je umístěna aplikace EPRDoc společně s lokálními daty o umístěných dokumentech. Úložiště nejsou nijak propojena, a proto je aplikace vázaná právě k jedné lokalitě. Aplikace mezi sebou nekomunikují ani nesdílí svá lokální data s ostatními.

Aplikace díky její dočasnosti nenabízí velkou možnost spolupráce, nebo začleněni do dalších aplikací v KB. Díky použité platformě MS Access existují teoretická a praktická omezení, a to zejména v počtu uživatelů pracujících současně nad danou aplikací. Tato hodnota se odhaduje hypoteticky na dvaceti aktivně pracujících uživatelích. Reálná hodnota je menší. Takové to omezení nebylo aplikaci v době jejího vytvoření na přítěž, jelikož lokálnost znamenala aktivní přistup pouze několika desítek pracovníků v jednu chvíli.

Aplikace plně uspokojovala uživatelské potřeby a to jak z pohledu nabízených funkčností, tak z pohledu komfortu obsluhy.

Důvodem nahrazení fungující aplikace EPRDoc je postupná centralizace. Kdy v případě kumulace jak dat, tak počtu uživatelů by již nevyhovovala požadavkům a stala by se nestabilní. Banka již pracuje na dalším řešení nesoucí název CEP (Centrální evidence pracovní dokumentace), paralelně s touto se vývojem zabývá i tato bakalářská práce.

3.5.1 Výhody EPRDoc

1. Snadné používání.
2. Snadná rozšiřitelnost funkčností.
3. Rozdílné funkčnosti dle potřeb lokalit.

3.5.2 Nevýhody EPRDoc

1. Omezený počet uživatelů.
2. Nemožnost sdílení informací mezi úložišti.
3. Nemožnost integrace na další prostředí KB.
4. Aktuálně nesplňuje bezpečnostní standarty.

3.6 Existující řešení a porovnání s DMS

Mnoho řešení zabývající se touto problematikou správy přímo nenalezneme. Jelikož každá firma má svou metodiku uskladnění a formu řazení či uskladnění jak je zmíněno v kapitole 2. Proto je obtížné vyvinout software, který by pokryl požadavky většího množství firem. Typickým scénářem je přizpůsobeni softwaru na přání zákazníka. Takové softwary jsou ale obtížně dostupné a nebo dokonce neveřejné. Proto je těžké analyzovat jiná řešení.

Většina společností nabízejí software na bázi DMS (Document management system) nebo EDMS (Electronic document management system). Ty fungují na jiném principu a nezaměřují se reálným uskladněním dokumentu v papírové formě.

3.6.1 Rozdíl oproti DMS systémů

Základním rozdílem DMS je skenování dokumentu. V první řadě se dokumenty převedou ze strojově psaného textu do elektronické podoby. Tento převod je možný například pomocí OCR (Optical character recognition) a ICR (Intelligent Character Recognition). Kdy jsou jednotlivé dokumenty mnohonásobně rychleji převedeny než v případě ručního opisu. Tyto elektronické dokumenty se následně uloží s možnou informací (v případě některých softwarů), kde je uložen originál. Následně se k dokumentu přistupuje pouze jako k elektronickému a odpadají tak procesy spojené například s výpůjčkou. Informace o DMS byly čerpány především ze systémů zabývajících se touto problematikou mimo jiné zmíněných v sekci 3.6.2.

Výhodu má toto řešení v jeho rychlosti vyhledávání, kdy si uživatel nemusí žádat dokumentaci a má jí rovnou přístupnou. Navíc díky úplnému textu,

a ne pouze metadatům je software schopen vyhledávat i podle textu uvnitř dokumentu. Tato varianta také redukuje množství potřebných papírových dokumentů jejichž archivaci a skartaci můžeme delegovat na jiné firmy.

3.6.2 DMS systémy

3.6.2.1 eFileCabinet

Je americká společnost nabízející software (DNS) pro správu dokumentů v elektronické podobě. Software je vhodný jak pro malé, tak velké společnosti. Pomáhá organizovat dokumenty a soubory online. Do správy lze přistupovat skrze webovou, mobilní i desktopovou aplikaci. Podporuje funkce převodu dokumentu do elektronické podoby skenem nebo vyfocením dokumentu v mobilní aplikaci. Umožňuje nastavení různého stupně bezpečnosti pomocí vícefázového ověření, uživatelských rolí nebo IP autentifikace. Software lze integrovat se službami třetích stran např. DocuSign, Salesforce a Microsoft Office. Cena za využívání softwaru je placena měsíčně s ohledem na velikost úložiště a počet uživatelů. [38]

3.6.2.2 M-Files

Je také Americká společnost nabízející software pro správu dokumentů v elektronické podobě. Software automaticky organizuje data dle typu nebo zvoleného řazení a detekuje duplicity. Nabízí uložení široké škály typů dokumentů včetně emailů. Lze také propojit s již existujícím systémem. Výhodou je nabídka z již před-připravených řešení pro odvětví jako finanční služby, inženýrství, ropa a plyn, nemovitosti, výroba nebo služby v oblasti věd a umění. Cena za software je individuální, lze jej vyzkoušet zdarma po dobu třiceti dní. [37]

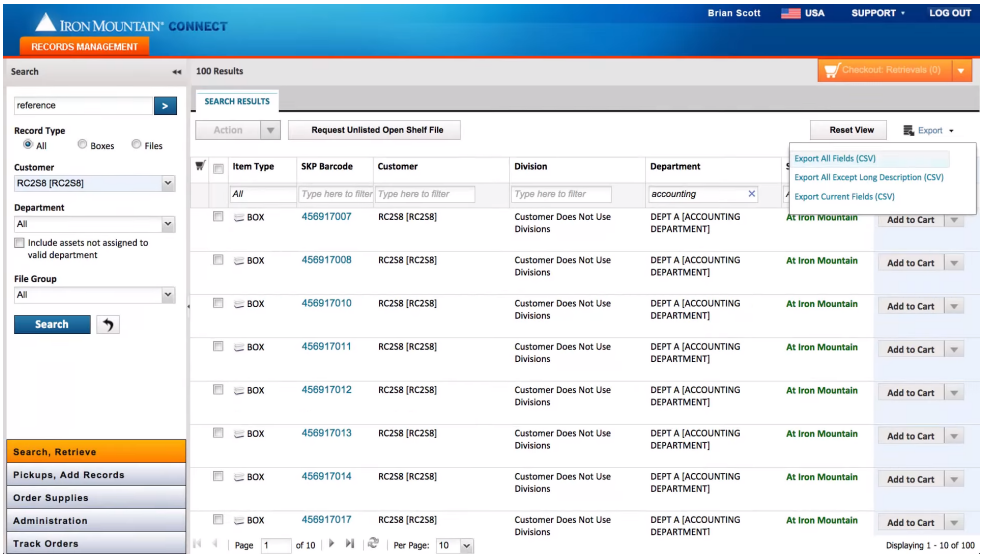
3.6.3 Podobné řešení

Zmíněné úložiště IRON Mountain používá podobný způsob skladování s menšími rozdíly. Místo složek využívá krabice a palety pro uskladnění dokumentů. K informacím o uskladněných dokumentech se dá dostat za pomocí webové aplikace Iron Mountain Connect. Aplikace slibuje kontrolu nad uskladněními dokumenty společně s podporou v podobě rychlého skenování nebo zaslání dokumentů. Funkce softwaru jsou:

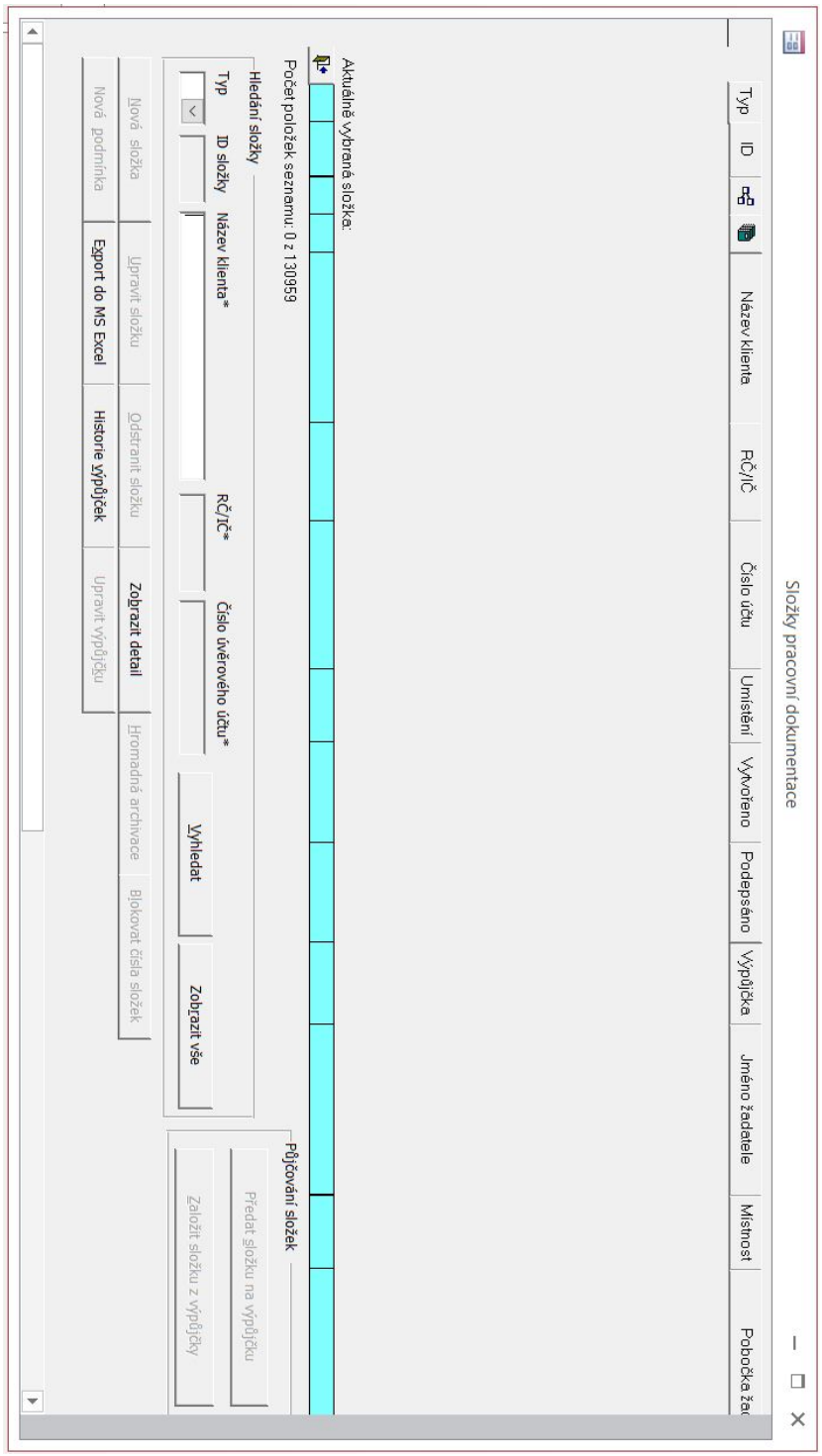
- Správa uživatelů Iron Mountain Connect
- Správa záznamů
- Správa úschovy
- Skartace
- Přehledy

Tento software je ale neveřejný a slouží pouze pro zákazníky společnosti IRON Mountain, kteří využívají služeb uskladnění v jejich skladech. Není ho proto možné využít jinde. [36] Náhled uživatelského rozhraní IM Connect na obrázku 3.4

Obrázek 3.4: Aplikace IM Connect



Obrázek 3.3: Aplikace EPRDoc



Kapitola 4

Analýza technologií

Tato kapitola se zabývá důvodem použití webové aplikace a výběrem vhodného frameworku pro vývoj front-end, back-end a databáze. Výběr frameworku je nezbytnou součástí pro návrh celého softwaru.

4.1 Webová proti desktopové aplikaci

Webová aplikace je taková aplikace, která běží na vzdáleném serveru a uživatel k ní přistupuje za pomocí webového prohlížeče. Po zadání specifické adresy v prohlížeči se stáhne stránka v podobě HTML kódu společně se styly v CSS.

Pokud stránka slouží pouze jako prezentační vrstva, stará se pouze o zobrazení dat a validaci, nazývá se tenkým klientem. Pro její správné fungování potřebuje server, na který se dotazuje pro získání požadovaných informací. Prezentační vrstva je často spojována s vícevrstvou architekturou. [12]

Desktopová na rozdíl od webové aplikace běží na konkrétním zařízení. Hlavní výhodou je možnost využití i v režimu off-line. Uživatel tuto aplikaci musí nejprve nainstalovat a poté spustit pro její užívání. Má mnohem větší nároky na provoz než webová aplikace. Další nevýhodou je udržování aktuálnosti verze aplikace.

Hlavní výhodou užívání webové aplikace z hlediska uživatelů je přístupnost téměř odkudkoli bez nutnosti instalace, což je i nevýhodou vzhledem k nemožnosti aplikaci používat bez přístupu k serveru prostřednictvím internetu nebo lokální sítě v závislosti na jeho pozici. Požadavky se provádí na serveru. Nejsou tak kladeny vysoké nároky na výkonost zařízení uživatele. Další výhodou je snadná možnost aktualizovat aplikace bez nutnosti provedení aktualizace na každém počítači zvlášť. Stačí pouze aktualizovat verzi, která se nachází na serveru.

V případě této práce se mnohem více hodí využít webovou aplikaci jak pro zmíněné výhody které poskytuje, tak pro splnění požadavku N1.

4.2 Srovnání front-end frameworků

4.2.1 Angular

Angular je framework od společnosti Google pro vývoj single-page client aplikací za použití HTML a TypeScript jazyka a v neposlední řadě také CSS. Základní stavební blok je NgModules, který dává dohromady komponenty a servisy do funkčního celku. Komponenty i servisy jsou třídy s dekorátorem, které poskytují metadata frameworku Angular pro jejich použití. Samotné servisy nemají vizuální prvky, neprezentují data uživateli. Zajišťují pouze funkce, které se dají předat pomocí dependency injection komponentě nebo další servise. Komponenty již vizuální prvky mají. [13, 15]

4.2.2 React

React open-source knihovna od společnosti Facebook pro tvorbu uživatelského rozhraní (vizuální stránky) na rozdíl od Angularu. Proto aplikace potřebuje další knihovny například Redux, React Router a další. Pro vývoj v frameworku React se používá JavaScript, HTML a CSS. Základním stavební kámen zde tvoří tzv. komponenty, což jsou různé znovupoužitelné HTML elementy se zapouzdřenou funkcionalitou, jejichž skládáním vzniká komplexní uživatelské rozhraní aplikace. [14, 15]

4.2.3 Porovnání

Angular i React jsou poměrné konkurenční produkty na celkem podobné úrovni podle článku [16] a [17]. Donedávna měl React převahu ve výkonu v případě složitějších dynamických webových aplikací. Tento defekt již opravili v předešlé aktualizaci a srovnali tak výkonnostní nepoměr. Angular je sice složitější z hlediska výukové křivky, ale má dobře propracovanou dokumentaci a informace o aktualizacích. Díky složitosti se často stává terčem kritiky. Naproti tomu je React mnohem snazší na porozumění díky jednoduchému designu. Dokumentace je oproti frameworku Angular horší a neaktuální z důvodu častých aktualizací. Tento defekt ale redukuje komunita React na diskuzních fórech, kde je často spousta návodů a rad. [15]

V konečném důsledku výběr mezi těmito frameworky je čistě na preferenci programátora. [16] Pro tuto práci byl vybrán Angular díky jeho ucelenosti a v neposlední řadě dobré dokumentaci. Velkou výhodou je Angular Material, který poskytuje sadu opakovaně použitelných komponent založených na materiálovém designu. [18]

4.3 Srovnání back-end frameworků

4.3.1 Spring

Spring je model-View-Controller framework používající stále ještě populární platformu Java. Dle statistik uvedených v článku [21]. Spring implementuje vzor IoC (Inversion of Control) a ID (Dependency injection) . IoC je princip přesunutí zodpovědnosti nebo její části za vytváření a provozování z objektu do frameworku nebo kontejneru. ID je návrhový vzor, který je použit při implementaci IoC. Spring je rozdělen do několika modulů: Core, Data Access, Web, Integration, Testing.

Hlavní výhodou Springu je pružnost a široká podpora rozšiřujících knihoven. Tento framework zjednodušuje psaní webových aplikací na platformě Java.

Spring boot je dále rozšíření samotného frameworku Spring. Zjednodušuje obtížné nastavení mimo jiné Hibernate Datasource, Entity Manager, Session Factory a Transaction Management. Nastavení v samotném frameworku Spring je velice zdlouhavé a i jednoduchá aplikace zabere spoustu času. Spring boot tento problém řeší pomocí zavedení anotací a autokonfigurace. [19, 20, 24]

4.3.2 Django

Django je Model-View-Template frakmework, který používá Python pro vývoj aplikací. Django implementuje návrhový vzor MTV (Model template view). Tento frakmework používají velké společnosti jako Google, Youtube a Instagram. Hodí se spíše na vývoj větších aplikaci. U menších je zbytečná nafouklost programu o funkce, které nevyužijeme. Django využívá tzv. batteries included (vše potřebné pro vývoj je již obsaženo a není tak třeba dalších softwaru třetích stran). [20]

4.3.3 Rails

Rails je model-View-Controller framework, který používá Ruby. Rails využívají například Airbnb, GitHub a Hulu. Tento framework je považován za vhodný pro začátečníky, díky množství komunitních tutoriálů a návodů. Podporuje velké množství rozšiřujících pluginů. Framework je jednoduchý v začátcích, později se ale začne komplikovat. [20]

4.3.4 Shrnutí

Výběr vhodného back-end softwaru závisí převážně na případu užití a na preferenci jazyka, který je programátorovi bližší. V dnešní době je poměrně populární dle statistik [21] v porovnání s předchozím rokem Python, proto spousta vývojářů mimo jiné využívá Django. Na tomto projektu je použit Spring díky

předchozím zkušenostem s tímto frameworkem a také široké škále možností díky velkému počtu rozšiřujících knihoven.

4.4 Srovnání databází

4.4.1 Oracle

Oracle je relační databáze od firmy Oracle Corporation. Určena převážné pro větší projekty. Obsahuje spoustu dodatečných funkcí, které ostatní databáze nemají. Tyto výhody jsou ale kompenzovány vysokými náklady jak na pořízení databáze, tak na samotné požadavky na hardware. Oracle databáze je nejstarší z jmenovaných, používá odlišné standarty než ostatní databáze a striktně se nedrží SQL normy. [22, 23]

4.4.2 MySql

MySQL je jednou z populárních open-source relačních databází pro webové aplikace. Dnes je již vlastněna firmou Oracle. MySql klade důraz spíše na spolehlivost než vysoký počet funkcí. Databáze není příliš automatizovaná, jak to bývá zvykem například u Oracle databáze. Hlavní výhodou je bezplatná licence pro nekomerční využití. [23]

4.4.3 Microsoft SQL Server

Je relační databáze od firmy Microsoft. Microsoft zajišťuje online podporu společně s obsáhlou dokumentací k databázi. Oproti Oracle databázi je mnohem jednodušší a přívětivější pro uživatele. Hlavní nevýhoda je běh pouze na operačním systému od Windows. Starší verze roku 2017 lze nasadit i na Linux. Oproti Oracle databázi má omezenější počet funkcí. [23]

4.4.4 Shrnutí

V potaz byly brány pouze některé relační databáze. V tomto projektu byla využita databáze od společnosti Oracle, převážně kvůli jejímu užívání ve společnosti KB. Na tomto rozhodnutí v konečném důsledku příliš nezáleží, jelikož je aplikace navržena tak, aby nebyla závislá na zvolené databázové technologii.

Kapitola 5

Návrh a implementace

Tato kapitola obsahuje samotný návrh aplikace. Struktura je znázorněna pomocí *Diagramu tříd* a předchází jí sekce s výběrem vhodné architektury. Dále se zabývá použitými technologiemi při implementaci. Velkou pozornost má v této kapitole bezpečnost a důvod výběru protokolu. V poslední části představuje samotné uživatelské prostředí front-end aplikace.

5.1 Architektura back-end aplikace

Pro účely této práce byla zvolena třívrstvá architektura jako nejvhodnější počet vrstev pro back-end aplikaci společně s návrhovým vzorem MVC pro tvorbu uživatelského rozhraní. Z důvodu využití MVC již v samotném Spring bootu byla tato architektura použita i v této práci.

5.1.1 Třívrstvá architektura

Třívrstvá architektura je složena ze tří takzvaných vrstev. Vrstva je v tomto případě část aplikace, která komunikuje přes dopředu definované rozhraní a může běžet odděleně od ostatních vrstev pro případ rozdělení výkonu mezi více zařízení. Při rozdělení vrstev je třeba brát v potaz i rychlost komunikace mezi vrstvami samotnými. Při zanedbání tohoto faktu, může naopak rozdělení zpomalit celou aplikaci.

Vrstvy v třívrstvé architektuře závisí pouze směrem shora dolům, viz diagram na obrázku 5.1. Vrstvy třívrstvé architektury [25]:

- **Prezentační vrstva** — Tato vrstva zprostředkovává komunikaci aplikace s uživatelem. Dotazuje se z uživatelskými požadavky obvykle na aplikační vrstvu. Tyto požadavky mohou být před odesláním další vrstvě validovány. Dále se stará o zobrazení následného výsledku dotazu.

- **Aplikační vrstva** — Vrstva implementuje základní funkčnost systému a zapouzdřuje obchodní logiku. Odpovídá na dotazy z prezentační vrstvy a dotazuje se na vrstvu datovou.

- **Datová vrstva** — Jedná se o nejníže položenou vrstvu. Zajišťuje přístup a práci s daty. Základní operace jsou ukládání dat do souboru, nebo databáze, výběr dat, agregace dat, předzpracování dat, integrita dat.

Zvolit méně vrstev, například dvě, se v tomto případě nehodí. U dvouvrstvé architektury máme pouze prezentační vrstvu (klientskou) a datovou. Důsledkem je spojení obchodní logiky společně s uživatelským rozhraním. Tento typ komunikace se také nazývá klient-server. Pro vývoj webové aplikace se ale příliš nehodí. Spojení logiky a UI nedává smysl a je výhodnější je držet ve dvou vrstvách.

Navýšením počtu vrstev nad zmíněné tři již nezískáme v tomto případě velkou výhodu. Tento krok má smysl v případě optimalizace výkonu nebo nutnosti budoucí výměny části komponent za jinou. S navýšením je ale spojena i větší míra komplexnosti. Proto byla zvolena právě třívrstvá architektura jako nejvhodnější. [26]

5.1.2 MVC

Navrhovaný vzor Model-View-Controller řeší rozdělení v rámci prezentační vrstvy. Rozděluje logiku programu a uživatelské rozhraní do tří vzájemně propojených částí. Tyto části stejně jako v třívrstvé architektuře mají závislost shora dolů, viz diagram na obrázku 5.1. Závislost se nemá porušit ani v případě nutnosti aktivně informovat o vzniklých změnách (Model informuje View). Tento problém se řeší pomocí rozhraní společně s návrhovým vzorem observer. V případě této aplikace postačí pouze pasivní model MVC [27]. Jednotlivé části MVC:

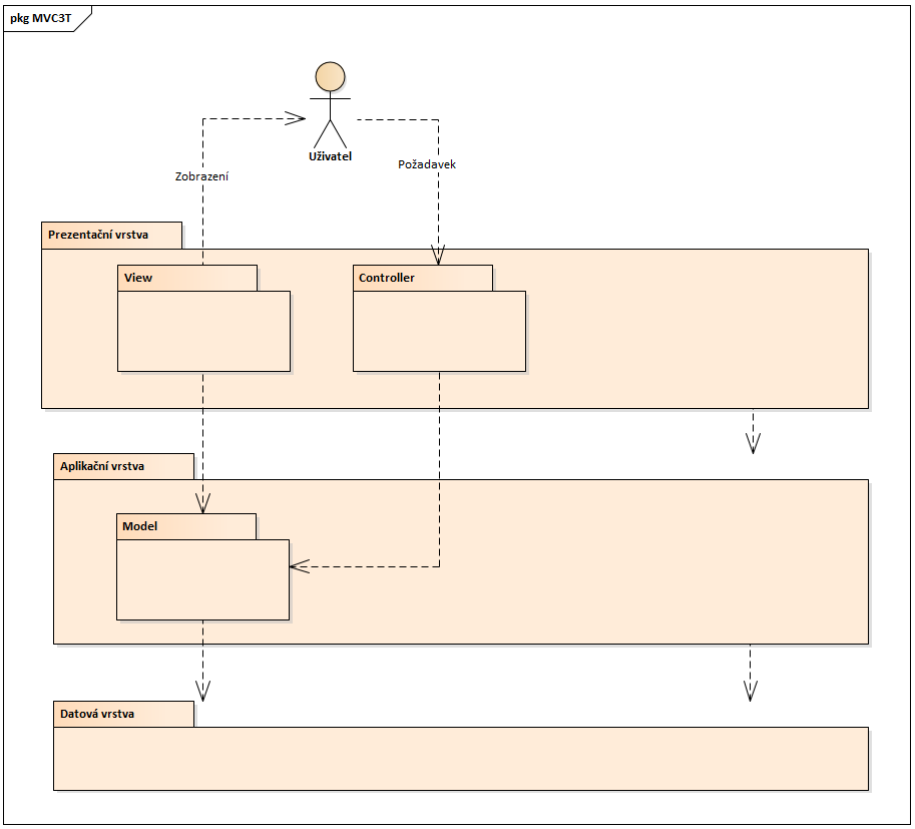
- **Model** — zpracovává (zapouzdřuje) byznys logiku celé aplikace. Obsahuje objekty a data specifické pro danou doménu s niž aplikace pracuje.

- **View** — je vrstva, která zajišťuje prezentaci dat. Na žádost o zobrazení dat od Controlleru se View dotáže Modelu pro aktuální informace, které následně zobrazí uživateli.

- **Controller** — zpracovává uživatelské požadavky. Controller uživatelský požadavek zpracuje a zajistí požadovanou akci, kterou následně pošle dál do Modelu. Následně požádá View o vykreslení informací.

Propojení třívrstvé architektury společně s využitím MVC můžeme vidět na diagramu na obrázku 5.1.

Obrázek 5.1: Propojení třívrstvé architektury a MVC



5.1.3 Moduly

Back-end aplikace bude rozdělena do několika modulů, za účelem jednoduché rozšiřitelnosti a znuvupoužitelnosti. Každý z modulů přinese do aplikace funkcionality, které mohou být podmíněny jiným modulem.

5.1.3.1 Dokument modul

O pokrytí požadavků F1. a F7.1. se bude starat modul *Dokumentu*. Jeho hlavní činnosti je založení a vyhledání dokumentu dle údajů. Založit dokument lze pouze pokud již na dané lokalitě složku nemá. Uživateli bude umožněno vybrat si volnou pozici v regálu kam umístí nový dokument. Problém volné složky bude řešen za pomocí speciální tabulky s volnými pozicemi. V případě již expirovaného produktu nebude třeba přiřazovat dokumentu složku, ale bude přímo přidělen do archivační krabice vybrané uživatelem.

Vyhledání dokumentu bude možné na základě jak úplné tak i částečné informace v podobě jména, čísla účtu nebo rodného čísla klienta, ke kterému se dokument váže.

5.1.3.2 Archivační modul

Po expiraci produktu je nutné dokumenty umístit do archivační krabice. Archivační modul bude poskytovat funkce na založení archivační krabice, uzavření krabice a v neposlední řadě archivaci dokumentu. Tento modul pokrývá požadavky F2., F6. a F7.2.

Pro založení krabice bude po uživateli vyžadováno specifikovat skartační znak společně s dodatečnou poznámkou. Následně se do ní již mohou umístit dokumenty, které se nacházejí na stejné lokalitě jako vytvořena krabice. Dokumentům bude systémem přiděleno jejich pořadové číslo na základě posledního přidaného dokumentu do krabice. V neposlední řadě se uvolní složka, kterou dokument obsadil a vytvoří se záznam do speciální tabulky s volnou pozicí.

Tento modul bude poskytovat uživateli také informace nutné pro vykonání archivace. S tím je spojeno vyhledání otevřených a uzavřených archivačních krabic nebo poskytnutí informace o jejím obsahu.

5.1.3.3 Logovací modul

Každá manipulace s dokumentem se bude zaznamenávat pomocí logovacího modulu. Při založení, archivaci nebo manipulaci s dokumentem bude vytvořen záznam v rámci požadavku F3. Součástí tohoto záznamu bude datum akce společně s provedenou akcí a vykonavatelem. Tento log dokumentu bude zpřístupněn na žádost uživatele.

5.1.3.4 Modul akcí

Modul akcí bude primárně sloužit k manipulaci s dokumentem na základě požadavků F8. a F4.2. Nad dokumentem se provádějí akce blokace, výpůjčka a vrácení. V neposlední řadě je třeba vyhledat a zobrazit tyto blokované a vypůjčené dokumenty uživateli F7.3.

Uživatel zažádá o blokaci konkrétního dokumentu nebo skupiny dokumentu. Společně s informacemi komu a kam mají být vypůjčeny společně s volitelnou poznámkou. Pokud je dokument možné zablokovat a uživatel má dostatečnou pravomoc, bude jeho blokace provedena. Systém označí dokument za zablokovaný. V případě blokace více dokumentů budou vráceny ty, které nebyly zablokovány z důvodu chyby.

Při vypůjčeni dokumentu uživatel může také přidat poznámku. Vypůjčeny mohou být pouze zablokované dokumenty na lokalitě žadatele. Stejně jako v případě blokace bude možné odeslat požadavek na vypůjčeni více dokumentů najednou. Systém při vypůjčeni označí dokument za vypůjčený.

Vrácení dokumentu lze tehdy, pokud byl v předchozí fázi zablokován, nebo vypůjčen. Stejně jako v předešlých případech bude možné při vrácení uvést poznámku, a také provádět výpůjčku, nad více dokumenty najednou.

5.1.3.5 Uživatelský modul

Uživatelský modul se bude využívat téměř v každém požadavku na rozhraní aplikace. Jeho hlavní činností je najít uživatele a zkontrolovat jeho práva pro vykonání požadované operace. Bude určen převážně pro jiné moduly, které vystavují uživatelské rozhraní a potřebují ověřit pravomoc.

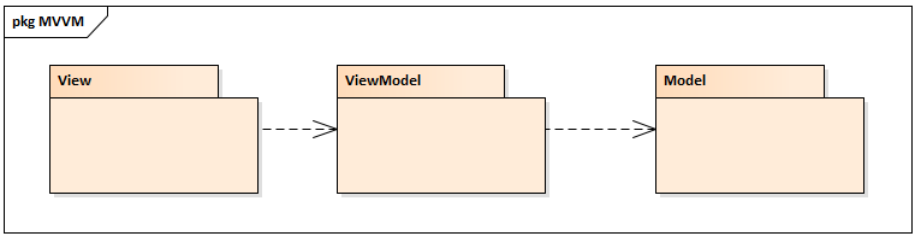
5.2 Architektura front-end aplikace

Pro tvorbu uživatelského rozhraní v Angularu byla zvolena architektura typu MVVM (Model-View-ViewModel). Tato architektura stejně jako MVC rozděluje aplikaci na tři části:

- **DataModel** — je vrstva zabývající se strukturou dat. Reprezentuje data, se kterými pracujeme. Neobsahuje žádné chování nebo servisy.
- **View** — je vrstva zodpovědná za vizuální stránku aplikace, obsahuje veškeré prvky, které uživatel vidí v podobě HTML, CSS nebo také Angular element.
- **ViewModel** — obsahuje veškerou logiku stránky. ViewModel funguje jako prostředník mezi View a Model vrstvou. V Angularu si pod ViewModel můžeme představit typescriptovou část komponenty.

MVVM vychází z již zmiňovaného MVC, pouze nahrazuje controller z MVC za ViewModel. Hlavním rozdílem je směr závislosti mezi View a ViewModel (Controller). Komunikace mezi vrstvami je znázorněna na diagramu na obrázku 5.2. Angular sice následuje komponentově orientovanou architekturu, ale ta je založena na velice podobném principu jako MVVM a to rozdělení uživatelského rozhraní do několika částí. [13, 28]. Proto se dá uvažovat i o MVVM.

Obrázek 5.2: Návrhový vzor MVVM



5.3 Diagram tříd

Diagram tříd 5.3 znázorňuje strukturu navrženého systému pomocí jazyka UML. Z diagramu jsou vidět třídy společně s jejich nezbytnými atributy a stavy, ve kterých se mohou vyskytovat.

5.3.1 Diskuze smyček

- Product — Document — Clinet

  S klientem byl sjednán produkt (Client — Product), ale ten již mohl převzít někdo jiný, například dědickým právem (Document — Client). Proto dokument může mít vazbu i na jiného klienta než na klienta, se kterým byl produkt sjednán. Vazba (Product — Document) nám udává k jakému produktu se dokument váže.

- Product — Document — Locality — Client

  Tyto vazby pouze určují lokalitu klientovy pobočky, dokumentu či místo sjednání produktu. Vazby mezi sebou nemají žádnou spojitost.

- Storage — Folder — Document — Box

  Folder i Document mají oba vazbu vždy na stejný Storage. Vazba udává lokalitu, kde se nacházejí.

5.4 Použitá technologie

5.4.1 Spring Data JPA

Tento modul se zabývá podporou pro vrstvy přístupu k datům založené na JPA. Usnadňuje vytváření Spring aplikací, které využívají technologie přístupu k datům, redukováním nutného kódu k tomuto přístupu. Spring Data JPA není implementací ani poskytovatelem JPA. Ke svému fungování potřebuje některého poskytovatele JPA. Tím muže být například Hibernate, zmíněný v sekci 5.4.2.

Spring Data JPA implementuje vrstvu s přístupem k datům. Součástí jsou mimo jiné repozitáře zajišťující CRUD operace nebo také querys. Query je obdoba SQL dotazu do databáze s rozdílem, že je veden nad Objektem, a ne nad tabulkami relační databáze. Z query se následně sestaví samotný SQL dotaz. Tohle neplatí pokud se jedná o Native Query, které je SQL dotaz vážící se k databázové technologii. [29]

5.4.2 Hibernate

Je ORM (Objektové relační mapování) framework pro Javu. Hibernate byl navržen jako alternativa pro opravu problémů s Entity Beans pro persistenci dat. Problémů nastává hned několik při mapování objektů a relační databáze:

- **Granularita** — Objektový model je granulárnější, než relační model.
- **Dědičnost** — RDBMS nepodporuje dědění, objekty však ano.

- **Identita** — RDBMS podporuje pouze porovnání na základě ID, naproti tomu Java objekt jak porovnání shodnosti, tak identity.
- **Asociace** — Cizí klíče v RDBMS naproti tomu odkazy na objekt.
- **Navigace dat** — Problém minimalizace SQL dotazů.

Hibernate je jednou z implementací samotného JPA, pro persistenci dat. Implementuje i další sadu nástrojů, jako Hibernate Search, Hibernate Validator a Hibernate OGM, které podporují mimo jiné i perzistenci pro NoSQL [30, 31].

5.4.3 Spring security

Je framework zjednodušující autentifikaci a autorizaci společně s ochranou proti standardním útokům jako control attacks, content sniffing, click jacking, cross-site scripting a dalším.

Funguje na bázi filtrů (`FilterChains`), které aplikuje na HTTP/HTTPS REQUEST ještě před kontrolerem. Tato sada filtrů se nastavuje pomocí třídy `WebSecurityConfigure`, která implementuje `configure`. V této metodě konfigurujeme samotné filtry. Ty si můžeme napsat také sami implemkentací funkce `doFilter`.

Autentifikaci lze provést několika způsoby dle dostupnosti dat. Nejběžnější varianta je dostupnost identifikačních údajů o uživateli s možností jejich porovnaní. Někdy z hlediska bezpečnosti tyto informace nejsou programem přímo dostupné. Je potřeba autentifikaci delegovat třetí straně a ověřit tak pravomoc přístupu uživatele. Populární implementací této varianty je například OAuth2.

Při možnosti přístupu k identifikačním údajům můžeme implementovat vlastní autentizaci, nebo využit několik možných implementací, jako například `JDBcUSerDetailManager`, který předpokládá uložení informací v databázi. Další možností je mít identifikační údaje přímo lokálně s využitím `InMemoryUserDetailManager`. Každá tato varianta potřebuje konfiguraci přístupu k identifikačním údajům uživatele (přihlašovací jméno, heslo, role). [35]

5.4.4 ModelMapper

Knihovna ModelMapper slouží pro inteligentní konverzi objektu na objekt podobný. Proces mapování je založen na transformaci názvů a typů atributů daného objektu na tzv. tokeny. Pomocí nich následně rozhodne o podobnosti. Pracuje s konvencemi, která se snaží použít k transformaci a následnému mapování. Například objekt obsahuje metodu `getPerson` a objekt, na který chceme mapovat, má metodu `setPerson`. Po transformaci vznikne person, který ModelMapper namapoval. Výsledkem je nový namapovaný objekt.

ModelMapper se hodí při konverzi entity objektů na DTO objekty. DTO se často používají při komunikaci napříč procesy. [34]

5.4.5 Redis

Redis slouží podobně jako databáze k ukládání dat. Oproti relační databázi je ale velice jednoduchý. Funguje in-memory, data ukládá do operační paměti, kde jsou rychleji přístupná než z běžného diskového úložiště. Tento typ úložiště ale není trvalý, po pádu nebo skončení programu se veškerá uložená data vymažou. Uložená data lze uchovat i po delší dobu než do ukončení serveru nastavením módu persistence. Je možné vybrat mód RDB, AOF nebo bez persistence.

Mód RDB je spíše zálohovací, podporuje například zálohování každou hodinu po dobu 24 hodin. Vhodný je při pádech, kdy je potřeba rychlého vzpamatování. Tento mód není příliš náročný na výkon. Nevýhodou je ztráta některých dat, jelikož nezálohuje každou změnu. Pro minimalizaci ztráty dat je zde mód AOF, který muže ukládat každou změnu. Často je tento mód pomalejší a paměťově náročnější než RDB. Mnohdy se vyplácí použít kombinaci těchto módů. Poslední možností, pokud nepotřebujeme dlouhodobě ukládat data, je persistenci úplně vypnout.

Redis nepoužívá koncept tabulkového schématu, ale řadí se do třídy NoSQL. Jako schéma pro ukládání a vyhledání používá tzv. klíč-hodnota. Podporuje širokou řadu datových struktur jako string, list, set, bitmap atd.

V případě této práce se používá Spring Session Data Redis jako cach pro ukládání sessionId jednotlivých přihlášených uživatelů. Při přihlášení se vytvoří sessionId, které se s odpovědí pošle uživateli a zároveň se uloží do Redis úložiště. Následně klient s každým dotazem posílá sessionId, kterým se autentifikuje. Persistence těchto záznamů není třeba řešit. Při případném pádu se bude muset uživatel pouze znovu přihlásit. Redis dále podporuje nastavení expirační doby, po které jsou data odstraněny. Takto se může omezit doba, po které bude po uživateli vyžadováno opakované přihlášení. [33]

5.4.6 Maven

Maven je software pro automatické sestavení a správu nejen Java projektů ale také C#, Ruby, Scala atd. Zjednodušuje konfiguraci projektu založené na Project Object Model definovaném v XML souboru s názvem *pom.xml*. Soubor obsahuje závislosti na externích modulech a komponentách, proces sestavení a celkový popis. V případě rozdělení projektu do menších, se samostatnými konfiguračními soubory pom. Maven, umožňuje jejich propojení za pomoci kořenového pom a spojení tak do jednoho celku. Maven také zajišťuje stahování potřebných pluginů definovaných v pom, díky spojení s externími repozitáři. [32]

5.5 Bezpečnost

Zabezpečení je otázkou každého většího softwaru. Nezabezpečený software je potencionální hrozbou, která může vést k nedostupnosti služby, nebo dokonce úniku informací. Jedním ze způsobů je vytvoření skupiny uživatelů s patřičným oprávněním. Následující informace o autentifikaci a autorizaci vycházejí převážně z knihy [39].

Jedním z požadavků, viz požadavek N4., je zajistit přistup více uživatelů s různými rolemi, viz požadavek N3., pro přihlášení do aplikace. Zároveň je vhodné využít automatického přihlášení uživatelů, kteří aplikaci pouze na chvíli zavřeli.

5.5.1 Autentifikace

Autentifikace je proces ověřování, identifikace pravosti osoby, co se snaží přihlásit do aplikace. Zjednodušeně řečeno, zda osoba je ta, za koho se vydává.

Vzhledem k uzavřenosti a zabezpečenosti sítě, ve které bude software nasazen je zbytečné uvažovat o vícefázovém ověření. V našem případě bude stačit pouze jednofázové ověření tzn. jméno společně s heslem. V případě nasazení do jiného prostředí se dá uvažovat o rozšíření na dvoufázové ověření například s využitím mobilního telefonu.

Je zde několik možných protokolů, jak vyřešit přihlášení uživatele skrze webovou aplikaci:

- HTTP Basic Auth — je nejzákladnější druh autentifikace. Jméno a heslo ve formátu Jméno:Heslo se zašifruje pomocí Base64. Tento získaný řetězec se následně přidá do hedrů samotného HTTP požadavku. Výsledný header odeslaný serveru může vypadat následovně: „Authorization= Basic YWRtaW46aHVudGVyMg==“. Server příjme požadavek HTTP, dekoduje hedr a porovná jméno a heslo se záznamy o uživatelích.

  Tento protokol má svá úskalí v jeho bezpečnosti. Počítá totiž s tím, že spojení mezi klientem a serverem je bezpečné. V opačném případě může útočník zachytit request a následně ho snadno dekódovat a získat tak přístupová práva. Další nevýhodou je zapamatování těchto údajů prohlížečem, kdy není možné nařídit uživateli jeho odhlášení. To se provede, až po fyzickém zavření stránky.

- API Keys — při prvním pokusu uživatele o přihlášení se vygeneruje takzvaný klíč. Tento klíč může být v podobě například IP adresy a hardwarevé kombinace nebo jedinečným vygenerovaným tokenem. Pokud se následně uživatel pokusí přistoupit do aplikace, použije se jeho tajný klíč k prokázání, že se jedná o stejného uživatele jako dříve.

  Protokol má stejná úskalí jako protokol Basic. Pokud je síť nezabezpečena, je možné tento klíč odposlechnout a následně zneužít. Výhodou je schopnost expirace těchto klíčů a s tím spojené odepření přístupu.

- OAuth — Pokud aplikace využívá OAuth, deleguje autentifikaci uživatele třetí straně. Při pokusu o přihlášení uživatele nejprve aplikace požádá o jednorázový token (request token) od aplikace třetí strany. Následně přesměruje uživatele na stránku třetí strany, kde se autentifikuje a do požadavku přidá tento toklen. Aplikace třetí strany autentifikuje uživatele, spojí si ho s vygenerovaným tokenem a následně přesměruje zpátky do aplikace. Aplikace následně zažádá o přístupový klíč (acces token) zasláním autorizovaného request tokenu. Pokud by třetí strana neautorizovala uživatele, reguest token by nebyl autorizován a aplikaci se nevydá ani acces token. Tento klíč (acces token) následně slouží pro komunikaci.

  Jedná se o nejbezpečnější způsob autentifikace, který je dnes již velice rozšířený. Pro účely této aplikace se ale příliš nehodí. Je předpoklad nasazení v interní zabezpečené síti a není nutné žádat o autentifikaci uživatele další stranu. Tento protokol nabývá smyslu pokud samotná společnost vystaví onu třetí stranu pro veškeré své aplikace.

- Session nebo token — Při použití session nebo token založené autentifikaci se uživatel prokazuje klíčem. Není proto nutné pokaždé posílat uživatelské údaje a tím prokazovat totožnost uživatele. Princip je založen na vygenerování unikátního klíče serverem, v případě session je to session_id v případě token je to JWT_Token. Tento klíč je následně odeslán v hedru uživateli, který pří následné další komunikaci se serverem přidá do hedru tento klíč.

Rozdílem Session a token založené autentifikaci je místo, kde je uložen stav uživatele. Session používá ukládání dat na serverové straně na rozdíl od token, kdy jsou data uložena na straně klienta, nejčastěji v lokálním úložišti.

Takových to způsobů je celá řada. V úvahu byly brány pouze nejznámější používané protokoly. Díky uzavřenosti a bezpečnosti sítě byl použit protokol basic. Ten však neumožňuje uživatele odhlásit do doby, než uzavře tab s aplikací. Společně s tímto protokolem byla proto také použita session autentifikace. Uživatel se tak poprvé přihlásí pod svým uživatelským jménem a heslem. Následně mu server vygeneruje session_id, které si uloží do Redis úložiště a odešle uživateli. Uživatel pak s každým dalším dotazem již neposílá hedr Authorization, ale pouze v cookie nastaví obdržené session_id. Díky tomuto způsobu se dá nastavit povolená doba, po které bude po uživateli vyžadováno znovu přihlášení.

5.5.2 Autorizace

Autorizace je proces, při němž se ověřují (server či jiná entita) dostatečná práva pro přístup do určité oblasti pro vykonání akce. Jelikož má uživatel různá práva na určité operace s dokumentem, nestačí pouze zjisti jestli je přihlášen nebo ne.

Uživatelé mají v systému přidělené role. Každá role má určité pravomocem v aplikaci. Při každém dotazu na controller se tyto pravomoci prověřují. Pokud má uživatel dostatečnou pravomoc, jeho dotaz je vykonán. Pokud uživatel danou pravomoc nemá, přístup je odepřen se status kódem 401 Unauthorized v odpovědi.

5.6 Nastavení datasource

Aplikace má několik datových úložišť. Primární úložiště je určeno pro ukládání dat aplikace jako jsou informace o dokumentech, produktech a podobně. Sekundární úložiště slouží pro přístup k informacím o uživateli pro potřeby autentifikace. Posledním úložištěm je Redis, kde se ukládají sessions s datem expirace a uživatelem, kterému byla vydána. Pro nastavení je třeba vyplnit v application.property určité informace, viz kód pro nastavení datasource 1.

```
#Oracle settings for data
spring.datasource.jdbc-url=jdbc:oracle:thin:@192.168.1.11:1539:oracle
spring.datasource.username=username
spring.datasource.password=password
spring.datasource.driver.class=oracle.jdbc.driver.OracleDriver
spring.datasource.dialect=org.hibernate.dialect.Oracle12cDialect

#Oracle settings for user
spring.users-datasource.jdbc-url= jdbc:oracle:thin:@192.168.1.11:1539:oracle
spring.users-datasource.username = username
spring.users-datasource.password = password
spring.users-datasource.driverClassName = oracle.jdbc.driver.OracleDriver

#Redis settings for session
spring.session.store-type=redis
spring.redis.host=127.0.0.1
spring.redis.port=6379
```

Výpis kódu 1: Nastavení datasource

5.7 GUI

Pro vstup do aplikace se musí uživatel nejprve přihlásit pomocí svých údajů, viz obrázek 5.11. Následně je přesunut do rozhraní, skrze které může provádět veškeré akce.

Vyhledání dokumentu pouze pomocí části informace lze v záložce *Práce s dokumenty* viditelné na obrázku 5.9. Při zadání nesprávného formátu viditelné na obrázku 5.8 je na něj uživatel hned upozorněn. Na základě uživatelem poskytnutých informací aplikace zobrazí shodující se produkty, viz obrázek (5.9). Pro zjištění místa a pozice v úložišti, znázorněné na obrázku 5.7, lze

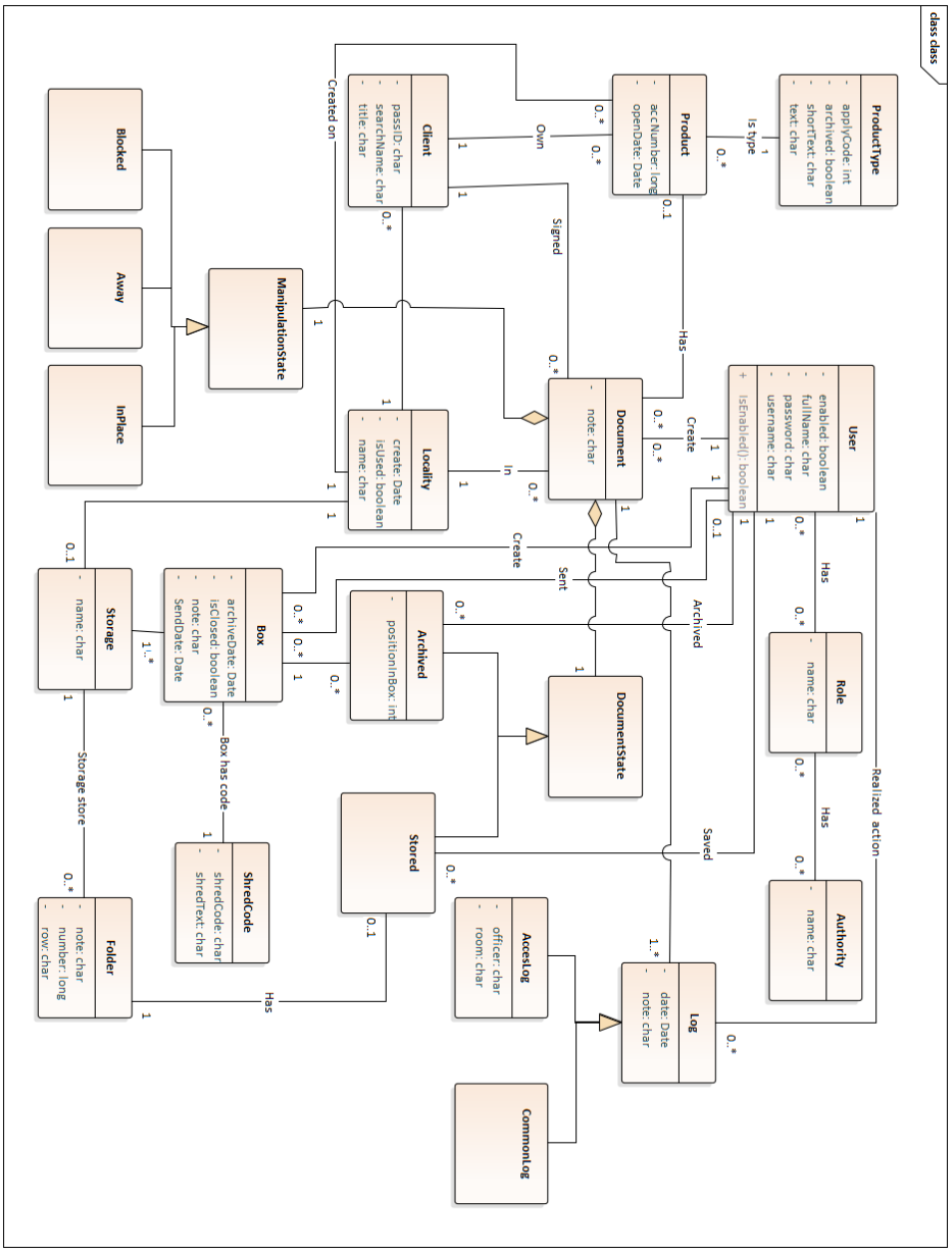
vyhledaný produkt rozkliknout. Následně se nabízejí manipulační akce s dokumentem. V případě rozkliknutí lze získat veškeré vedené informace v systému o tomto dokumentu, viz obrázek (5.6).

V záložce *Archivační krabice* lze vyhledávat, vytvářet a zavírat archivační krabice. Uživatel má umožněno manipulovat pouze s archivačními krabicemi na své lokalitě. Znázornění lze vidět na obrázku 5.4.

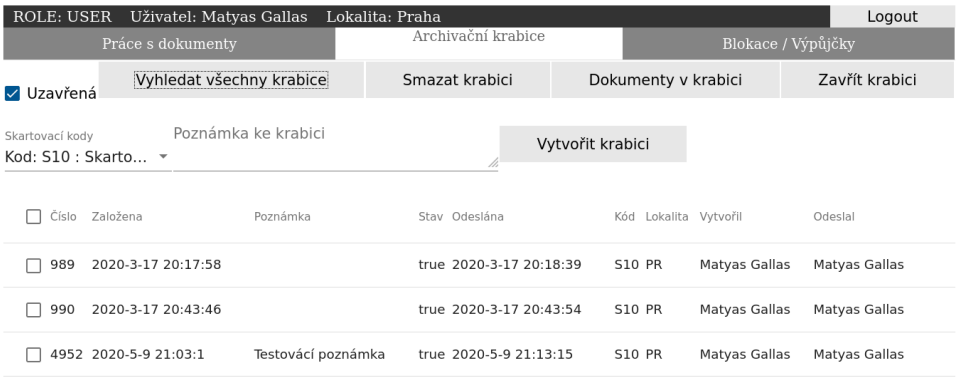
V poslední záložce s názvem *Blokace/výpůjčky* viz 5.5 má uživatel k dispozici své vlastní blokace a výpůjčky společně s blokacemi a výpůjčkami pro celou jeho lokalitu. Po zobrazení dokumentů je uživateli nabídnuta také již zmiňovaná nabídka manipulačních akcí s dokumentem.

Každá správně i špatně provedená akce je uživateli oznámena v podobě notifikace, viz obrázek (5.10). Při dlouhodobém užívání po expiraci poskytnutého klíče, nebo při neoprávněné operaci je uživateli zobrazena hláška s nabídkou pro opětovné přihlášení, viz obrázek (5.12). Tuto hlášku lze odmítnout v případě nedostatečného oprávnění uživatele. V případě provádění neoprávněné akce se tato hláška bude opakovat.

Obrázek 5.3: Diagram tříd

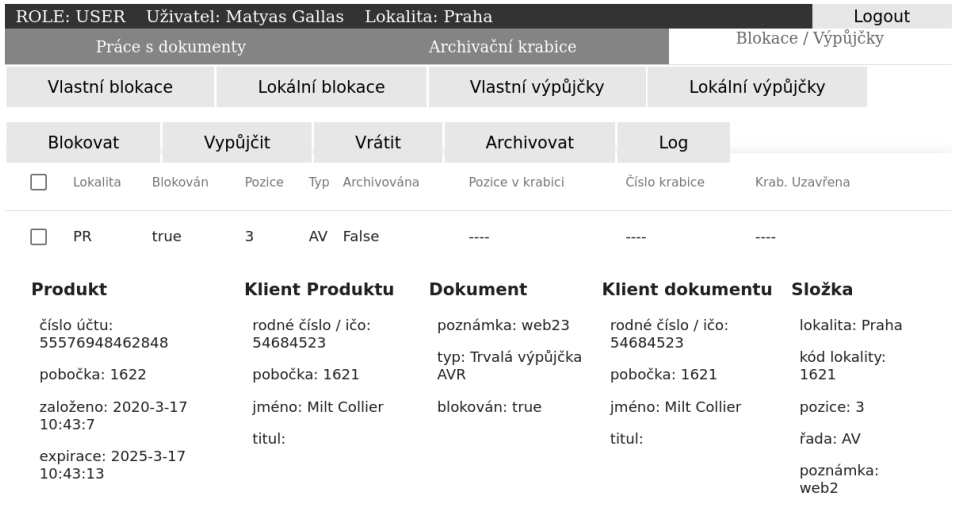


Obrázek 5.4: Archivace

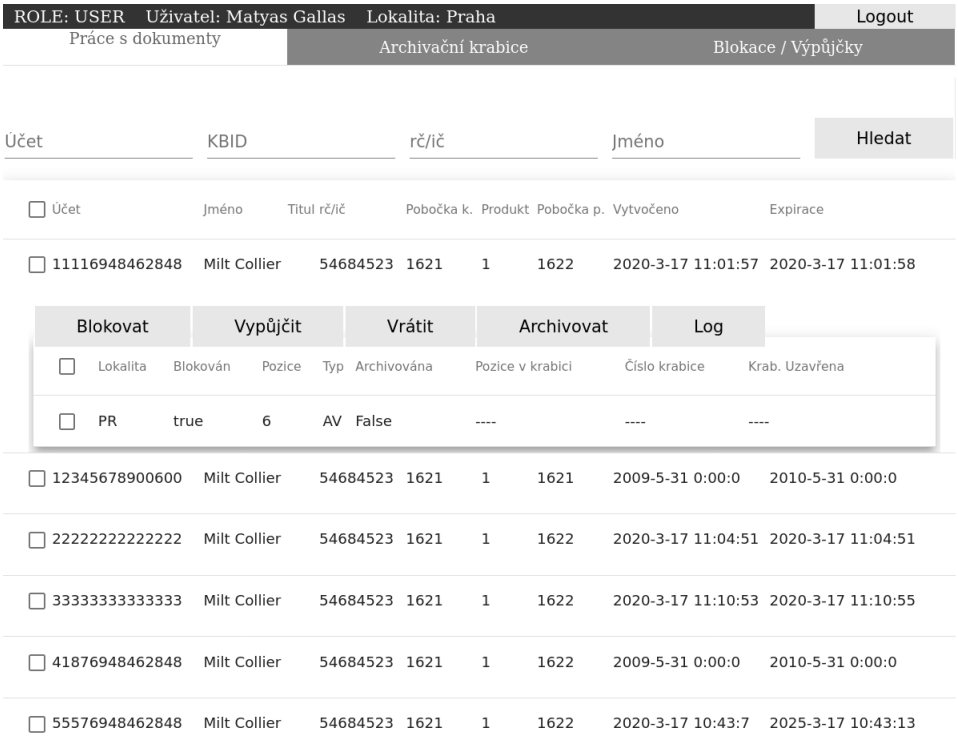


Obrázek 5.5: Přehledy blokací a výpůjček

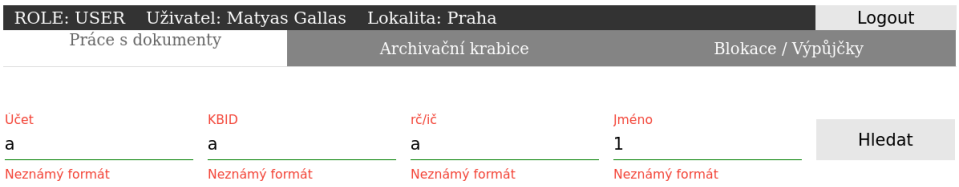
Obrázek 5.6: Detailní informace o dokumentu



Obrázek 5.7: Zobrazení dokumentů k produktu



Obrázek 5.8: Kontrola parametrů



Obrázek 5.9: Vyhledání dle parametrů

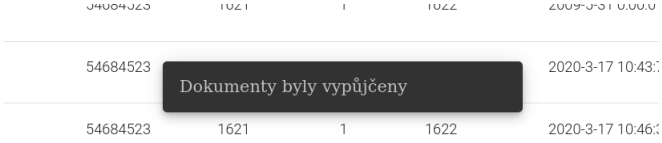
ROLE: USER Uživatel: Matyas Gallas Lokalita: Praha Logout

Práce s dokumenty | Archivační krabice | Blokace / Výpůjčky

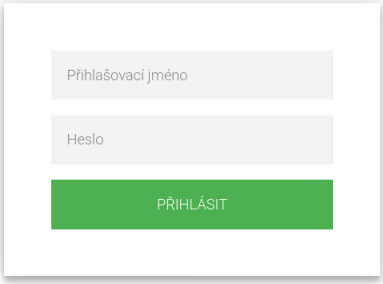
Účet KBID rč/ič Jméno Mil Hledat Vytvořit novou složku Rovnou archivovat

	Účet	Jméno	Titul	rč/ič	Pobočka k.	Produkt	Pobočka p.	Vytvořeno	Expirace
☐	111169484628 48	Milt Collier		54684523	1621	1	1622	2020-3-17 11:01:57	2020-3-17 11:01:58
☐	123456789006 00	Milt Collier		54684523	1621	1	1621	2009-5-31 0:00:0	2010-5-31 0:00:0
☐	222222222222 2	Milt Collier		54684523	1621	1	1622	2020-3-17 11:04:51	2020-3-17 11:04:51
☐	333333333333 3	Milt Collier		54684523	1621	1	1622	2020-3-17 11:10:53	2020-3-17 11:10:55
☐	418769484628 48	Milt Collier		54684523	1621	1	1622	2009-5-31 0:00:0	2010-5-31 0:00:0
☐	555769484628 48	Milt Collier		54684523	1621	1	1622	2020-3-17 10:43:7	2025-3-17 10:43:13
☐	666669484628 48	Milt Collier		54684523	1621	1	1622	2020-3-17 10:46:37	2020-3-17 10:46:39
☐	999769484628 48	Milt Collier		54684523	1621	1	1622	2020-3-17 11:00:12	2020-3-17 11:00:13

Obrázek 5.10: Notifikace



Obrázek 5.11: Přihlašovací formulář



Obrázek 5.12: Neoprávněný pokus o operaci

Kapitola 6

Testování

K zajištění kvality výsledného softwaru je nezbytnou součástí jeho testování. Cílem je otestovat jak určité funkcionality, tak aplikaci jako celek. Implementace testů je vhodná i pro budoucí rozšiřovaní vyvíjeného softwaru [41].

6.1 Unit testy

Pro účely testování byl využit framework JUnit 4. Tento framework zjednodušuje samotné testování. Stačí pouze vytvořit třídu a přidat k ní anotaci. `@RunWith(SpringRunner.class)`, do které již můžeme psát jednotlivé testy jako metody s anotací `@Test`. Ty lze spouštět jednotlivě nebo dohromady. Informace pro testování Springu čerpány z [40].

6.1.1 Repository testy

Jelikož aplikace obsahuje *Repository* ke každé entitní třídě a využívají je téměř všechny servisy. Je zapotřebí před dalším testováním nejprve otestovat právě tuto část aplikace. Není nutné široce testovat základní funkce CRUD, které *JPARepository* poskytuje, pouze ověřit jejich dostupnost. Hlavní pozornost je věnována nově implementovaným querys (`@Quary`).

Pro účely repository testů je zapotřebí přístup k databázi. Není vhodné využívat finální databázi za účelem testování. Je lepší ji nahradit za testovací. Pro tyto případy se dá využit in-memory databáze H2. Nejprve je nutné změnit datasource na H2 databázi, viz výpis kódu s anotací pro změnu datasource 2. Pro persistenci dat je použit *TestEntityManager*. Následně je možné začít testovat, viz ukázka kódu s testy 3.

6.1.2 Testování logiky

Testování správného fungování logiky celé aplikace je nedílnou součásti testů samotných. Většina logiky je umístěná v objektech *Service* a závisí na *Repo-*

```
@RunWith(SpringRunner.class)
@DataJpaTest
@AutoConfigureTestDatabase(replace = AutoConfigureTestDatabase.Replace.NONE)
@TestPropertySource(properties = {
    "spring.jpa.hibernate.ddl-auto=create",
    "spring.jpa.properties.hibernate.dialect =
                    org.hibernate.dialect.H2Dialect",
    "spring.datasource.url=
                    jdbc:h2:mem:;INIT=create schema IF NOT EXISTS GAMATY"
})
```

Výpis kódu 2: Anotace pro změnu datasource při testování

```
@Test
public void lastInBoxEmptyTest() {
    ArchBox box = new ArchBox();
    entityManager.persist(box);
    entityManager.flush();

    Optional<Integer> last = archBoxRepository.lastInBox((long)1);
    assertThat(last.isPresent()).isEqualTo(false);
}
@Test
public void lastInBoxTest() {
    ArchBox box = new ArchBox();
    DocArchRecord record1 = new DocArchRecord();
    record1.setNumberInBox(2);
    record1.setArchBox(box);
    record1.setDocument(new Document());
    entityManager.persist(record1);

    DocArchRecord record2 = new DocArchRecord();
    record2.setNumberInBox(5);
    record2.setArchBox(box);
    record2.setDocument(new Document());
    entityManager.persist(record2);

    box.setArchRecordsSet(new HashSet<>(Arrays.asList(record1, record2)));
    entityManager.persist(box);
    entityManager.flush();
    Optional<Integer> lastNumber = archBoxRepository.lastInBox(box.getId());

    assertThat(lastNumber.isPresent()).isEqualTo(true);
    assertThat(lastNumber.get()).isEqualTo(5);
}
```

Výpis kódu 3: Ukázka repository testů

sitory. Tuto závislost je vhodné pro testování pouze *Service* oddělit. To lze docílit pomocí tzv. *Mock* objektů, které simulují objekty v našem případě *Repository*. Pomocí *Mock* objektů jsme schopni nasimulovat chování, které očekáváme od *Repository*.

Pro testování tříd *Service*, musíme mít její instanci vytvořenou a dostupnou jako `@Bean`, abychom ji mohli `@Autowire` v naší testovací třídě. Této konfigurace je dosaženo pomocí anotace `@TestConfiguration`, viz kód se znázorněnou anotací 4. Následně musíme vytvořit `@MockBean` pro všechny *Repository* a *Service*, na která se testovaný *Service* odkazuje. Ukázka testování, viz kód s testy 5.

```
@TestConfiguration
static class ArchServiceTestContextConfiguration {
    @Bean
    public ArchService archService() {
        return new ArchServiceImp();
    }
}
@Autowired
private ArchService archService;
```

Výpis kódu 4: Konfigurace Bean pro testování

6.2 Integrační testy

Integrační testy slouží k ověření správnosti komunikace mezi všemi komponenty vyvíjeného softwaru. Tyto testy se dají provádět automatizovaně nebo manuálně. V této práci se využilo manuálního testování. [42]

Prověřeny byly veškeré požadované funkčnosti. Nejprve byl testován pouze back-end vystavené rozhraní pro komunikaci a následně celá aplikace ve webovém grafickém prostředí. Postupovalo se v obou případech podle testcase (testovací případ). Příklad samotného testcase:

- Přihlášení pod neplatným účtem.
- Přihlášení pod správným účtem.
- Vyhledání dokumentů bez specifikace.
- Vyhledání dokumentů s číslem účtu a následně se jménem.

V poslední řadě byly provedeny tzv *Exploratory tests*. Jsou podobné jako manuální s rozdílem v nepoužití testovacího scénáře. Dávají tak volnost a možnost najít skryté závady například neobvyklým užíváním.

```
@Test
public void inBoxTest(){
    ArchBox arch = new ArchBox();
    arch.setId(1);

    Set<Document> resDoc = new HashSet<>();
    Set<DocArchRecord> input = new HashSet<>();
    for(int i = 0; i < 5; i++){
        Document doc = new Document();
        DocArchRecord rec = new DocArchRecord();
        doc.setArchRecord(rec);
        rec.setDocument(doc);
        resDoc.add(doc);
        input.add(rec);
    }
    arch.setArchRecordsSet(input);
    Mockito.when(archBoxRepository.findById(arch.getId()))
            .thenReturn(java.util.Optional.of(arch));

    assertThat(archService.inBox(arch.getId())
            .get().collect(Collectors.toSet())).isEqualTo(resDoc);
}
@Test
public void inBoxEmptyTest(){
    ArchBox arch = new ArchBox();
    arch.setId(1);
    arch.setArchRecordsSet(new HashSet<>());
    Mockito.when(archBoxRepository.findById(arch.getId()))
            .thenReturn(java.util.Optional.of(arch));

    assertThat(archService.inBox(arch.getId()).get().count()).isEqualTo(0);
}
```

Výpis kódu 5: Ukázka testování logiky

Kapitola 7

Rozšíření, přínosy a nasazení informačního systému

Tato kapitola se zabývá možným dalším vývojem této aplikace a budoucím rozšířením, které by dále mohly zjednodušit práci s fyzickými dokumenty. Dále se také zabývá využitelností systému společně s jeho nasazením.

Díky využití třívrstvé architektury a návrhových vzorů je systém snadno rozšiřitelný. Návrhové vzory pomáhají k čitelnosti, zaměnitelnosti a znovupoužitelnosti kódu. Rozdělení systému do několika modulů usnadňuje přidání dalších nebo obměnu již existujících modulů.

7.1 Rozšíření funkcionalit

Možné rozšíření funkcionalit v budoucnu:

- **Podpora tisku** — Přidání možnosti tisku blokovaných dokumentů k vyzvednutí.
- **GUI pro správu uživatelů** — Rozšíření správy a možnosti přidělení uživateli oprávnění skrze grafické prostředí aplikace. Možnost přidání další role se specifickými pravomocemi nebo odebrání či zablokování konkrétního uživatele.
- **Statistiky** — Přidání statistik vztažených ke konkrétní lokalitě například o zaplněnosti nebo počtu vypůjčených dokumentů, které ještě nebyly navráceny.
- **Nestandardní operace** — Úplné smazání některého dokumentu či jeho logu. Obnovení již archivovaného dokumentu zpátky do jeho složky, pokud je stále volná. Úpravy poznámek nebo určitých vazeb.

7.2 Rozšíření platforem

Stále se rozšiřuje trend přidělení firemních telefonů zaměstnancům. Dalším možným usnadněním je proto rozšíření aplikace i na mobilní platformy.

Nabízí se zde možnost vytvořit mobilní aplikaci a podpořit tak webové rozhraní. Přímo nahradit celou webovou aplikaci za mobilní není příliš reálné z důvodu malé velikosti a nepřehlednosti mobilních telefonů. Mobilní aplikace může obsahovat například přehledy blokovaných dokumentů, které uživatel vyzvedává z úložiště. Nemusí si tak pamatovat či zapsat tyto informace o umístění. Což je při vyzvedávání většího počtu dokumentů obtížné. Zároveň rovnou po fyzickém vypůjčeni označí v mobilní aplikaci dokument za vyzvednutý nebo v opačném případě vrácený. Mobilní aplikace je myšlena pouze jako doplněk k webové aplikaci.

Vzhledem k použité architektuře je rozšíření na mobilní platformy možné pouhým do-implementováním mobilní aplikace s využitím již stávající back-end aplikace. Mobilní, stejně jako webová, aplikace následně pošle dotazy skrze protokol HTTP.

7.3 Přínosy pro společnost KB

Největším přínosem pro Komerční banku bude centralizace všech informací o dokumentech. Bude možné vyhledávat umístění dokumentů přes všechny lokality. Tím se zrychlí a zpřehlední samotný proces hledání a následného blokování dokumentace k úvěrům oproti původnímu řešení.

Dalším velkým přínosem je přístup prostřednictvím webového rozhraní, které ulehčí rozšíření této aplikace mezi všechny uživatele. Oproti stávajícímu řešení, díky využití technologií, je možný přistup většího množství uživatelů v závislosti na výkonosti serveru, na kterém bude aplikace nasazena.

Řešení dále umožňuje integraci do již existujících systémů a je snadno rozšiřitelné o další funkcionalitu. V budoucnu je možné informační systém také rozšířit na mobilní platformy.

7.4 Využitelnost i v jiných organizacích

Využitelnost systému zabývajícím se tato práce v jiných organizacích je zcela závislá na funkcionálních požadavcích a doménovém modelu dané organizace. Systém je sice navržen a implementován s možností budoucího rozšířeni stávajících funkcionalit, ale v případě zcela odlišného přístupu je úprava systému ekonomicky nevýhodná. Z tohoto ohledu je výhodnější využít již existující jiné řešení, které lépe vyhoví požadavkům organizace.

S určitými změnami dle firemní politiky lze systém nasadit do jiné společnosti. Podmínkou je přístup k dokumentům pouze ve formě metadat. Ne-

evidovaní každého dokumentu samostatně, ale vážícího se k subjektu nebo produktu. Ostatní funkcionality se dají přizpůsobit.

Software ocení společnosti s podobným přístupem ke správě fyzických dokumentů jako KB, kdy je třeba evidovat více úložišť současně. Zároveň řešit vypůjčení, archivaci a skartaci po expiraci dokumentu. V takovém případě je systém vhodný využít.

7.5 Nasazení

Celá aplikace se sestavuje ze tří oddělitelných částí, které mohou běžet samostatně nebo společně na určeném zařízení. Pro správný chod celé aplikace je nutná správná funkce a nasazení všech částí.

7.5.1 Nasazení Oracle databáze

Prvním krokem nasazení je založení a nakonfigurování databázového server Oracle, pokud společnost žádný nemá. Po založení je nutné vytvořit uživatele, pod kterým se bude aplikace následně hlásit. Tento uživatel musí mít přidělená následující práva:

- `CREATE DROP ALTER UPDATE -> PROCEDURE, SEQUENCE, TABLE, VIEW`
- `INSERT -> TABLE`
- `EXECUTE -> PROCEDURE`
- `SELECT -> SEQUENCE, TABLE`
- `CREATE -> SESSION`

Následně je nutné propojit back-end aplikaci s databázovým serverem. Konfigurace je provedena, viz ukázka konfiguračního souboru 1. Pokud má společnost již databázi uživatelů, je možné jí využít nakonfigurováním sekundárního *datasource*.

Minimální požadavky na hardware serveru pro nasazení databáze jsou:

- Procesor: AMD64 nebo Intel EM64T
- RAM: 4 GB
- Swap: 2 - 16 GB
- Disk: 15 GB + prostor pro samotná data (50 GB prostoru vystačí odhadem na 1,5 milionů záznamů o dokumentech)

7.5.2 Nasazení aplikácí Angular a Spring boot

Pro nasazení Back-end Spring boot aplikace je hned několik možností. Například nasazení pomocí JAR, WAR nebo kontejneru. Vhodným řešením může být využití Docker kontejneru. Docker přináší spoustu výhod jako izolovanost, jednoduché nasazení a kompatibilitu v rámci různých systému. Funguje jako virtuální prostředí pro naši aplikaci.

Pro nasazení v podobě Docker kontejneru je potřeba vytvořit JAR aplikace a následně *Dockerfile* v domovské složce projektu s konfigurací. JAR vygenerujeme pomocí nástroje maven (`mvn package`). Tím se vytvoří spustitelný soubor s veškerými závislostmi. Následně vytvoříme Docker image, který bude sloužit jako prostředí pro spouštění aplikace. [43]

Pro nasazení Angular aplikace je potřeba nakonfigurovat základní adresu pro přístup k Spring boot aplikaci v souboru `http.service`. Stejně jako Spring boot tak i Angular má několik možností jak aplikaci nasadit. Jednou z nich je pouhý přesun zkompilovaných souborů na server. Touto možností mohou vznikat problémy. Proto je výhodnější využít stejně jako v předchozím případě kontejner Docker.

Nejprve se Angular aplikace zkompiluje pomocí příkazu `ng build --prod` a tím se vytvoří složka `dist` se zkompilovanými soubory. Následující postup je stejný jako v předchozím případě při nasazení Spring boot aplikace.

Minimální hardwarové požadavky pro Back-end a Front-end aplikaci je těžké stanovit. Aplikace obecně neprovádí obtížné operace a nevyžaduje vysoký výkon. Proto postačují běžné servery s parametry (Core i5, 1.8 GHz, 4 GB RAM) při využívání v rámci omezeného počtu uživatelů.

Závěr

Cílem této práce bylo analyzovat, navrhnout a implementovat funkční informační systém podporující základní funkcionality správy fyzických dokumentů na základě metodik Komerční banky.

Součástí analýzy bylo seznámení se s problematikou správy, sestavení požadavků, výběr vhodných technologií a analýza stávajícího řešení, které KB používá. Mimo jiné byl proveden výzkum, nad již existujícími podobnými řešeními, které přineslo zjištění o nedostupnosti takovéhoto druhu správy. Tato nedostupnost pramení převážné z velké specifičnosti skladování, přístupu a řazení dokumentů různých společností. Častým kompromisem jsou softwary typu DMS, které z určité části umožňují pracovat a zaznamenávat informace o dokumentech v papírové podobě. Jejich zaměření je ale převážně na elektronickou podobu dokumentů.

Systém byl navrhnut jako webová aplikace na základě provedené analýzy a požadavků s využitím třívrstvé architektury a návrhových vzorů MVC a MVVM. Při návrhu se také bralo v úvahu budoucí rozšíření. Back-end aplikace byla implementována v jazyce Java s pomocí mimo jiné Spring frameworku a front-end aplikace v jazyce HTML a Typescript s pomocí frameworku Angular. Grafické uživatelské prostředí bylo navrženo a implementováno s ohledem na snadnou orientaci a přehlednost. Jednotlivé části systému jsou řádně otestovány pomocí automatických Unit testů a také manuálních testů. Podstatné části zdrojového kódu jsou zdokumentovány.

Veškeré vytyčené cíle této práce byly splněny společně s požadavky na aplikaci vzniklými při analýze. Aplikaci je možné v omezené míře nasadit do provozu. Pro úplné nasazení do prostředí KB a nahrazení stávající aplikace je třeba přidat některé moduly, které nejsou nezbytné pro základní užívání. V budoucnu je množné tyto chybějící moduly doplnit a rozšířit. Systém lze dále obohatit například o mobilní platformu, která také podpoří zjednodušení správy.

Příloha A

Seznam použitých zkratek

CEP Centrální evidence pracovní dokumentace

CRUD Create read update delete

CSS Cascading Style Sheets

ČNB Česká národní banka

DI Dependency injection

DMS Document management system

DTO Data transfer object

EDMS Electronic document management system

EPRDoc Aplikace evidence pracovní dokumentace

GUI Graphical user interface

HD High-Definition

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

ICR Intelligent Character Recognition

IoC Inversion of Control

IRON Společnost IRON Mountain

JPA Java Persistence API

JSON JavaScript Object Notation

JWT JSON Web Token

KB Komerční banka

MS Microsoft

MTV Model template view

MVC Model view controller

MVVM Model view viewModel

OCR Optical character recognition

ORM Object relational mapping

RDBMS Relational Database Management System

SQL Structured Query Language

UI User interface

UML Unified Modeling Language

XML Extensible markup language

Literatura

[1] ČESKO: Zákon č. 190/2004 Sb. ze dne 1. dubna 2004, o dluhopisech. In *Sbírka zákonů České republiky* 2004, částka 63, ISSN 1211-1244. Dostupné z: https://www.zakonyprolidi.cz/cs/2004-190

[2] KOMERČNÍ BANKA: *Seznam dokladů: Doklady předkládané klientem při podání žádosti o úvěr* [online]. [cit. 2020-03-08]. Dostupné z: https://www.kb.cz/getmedia/dc5bf373-e24f-4553-8255-642be773ebaa/kb-seznam-dokladu-predkladanych-klientem-konsolidace.pdf.aspx

[3] KOMERČNÍ BANKA: *Základní informace* [online]. [cit. 2020-03-09]. Dostupné z: https://web.archive.org/web/20111007123434/http://kb.cz/cs/o-bance/o-nas/zakladni-informace.shtml

[4] KOMERČNÍ BANKA: *Fakta a výsledky* [online]. [cit. 2020-03-09]. Dostupné z: https://www.kb.cz/cs/o-bance/vse-o-kb/fakta-a-vysledky

[5] ČESKO: Zákon č. 191/1950 Sb. ze dne 20. prosince 1950, Zákon směnečný a šekový In *Sbírka zákonů České republiky* 1950, částka 73, ISSN 1211-1244. Dostupné z: https://www.zakonyprolidi.cz/cs/1950-191

[6] ČESKO: Zákon č. 21/1992 Sb. ze dne 20. prosince 1991, Zákon o bankách. In *Sbírka zákonů České republiky* 1992, částka 5, ISSN 1211-1244. Dostupné z: https://www.zakonyprolidi.cz/cs/1992-21

[7] ČESKO: Zákon č. 21/1992 Sb. ze dne 3. února 2012, občanský zákoník. In *Sbírka zákonů České republiky* 2012, částka 33, ISSN 1211-1244. Dostupné z: https://www.mesec.cz/zakony/obcansky-zakonik-2014/f4584637/

[8] MINISTERSTVO FINANCÍ ČESKÉ REPUBLIKY: *Základní informace: Úvěr* [online]. 2014 [cit. 2020-03-20]. Dostupné z: https://financnigramotnost.mfcr.cz/cs/uvery/zakladni-informace

[9] ČESKO: Zákon č. 499/2004 Sb. ze dne 30. června 2004, o archivnictví a spisové službě a o změně některých zákonů. In *Sbírka zákonů České republiky* 2004, částka 173, Dostupné z: https://www.zakonyprolidi.cz/cs/2004-499

[10] RADOMÍR GALLAS [osobní rozhovor]. 2019. 08. 10.

[11] IRON MOUNTAIN: *O nás* [online]. [cit. 2020-04-10]. Dostupné z: https://www.ironmountain.cz/about-us

[12] MIROSLAV ČERMÁK: Vícevrstvá architektura: tenký, tlustý a chytrý klient. *Vývoj SW* [online]. 2010 [cit. 2020-04-10]. Dostupné z: https://www.cleverandsmart.cz/vicevrstva-architektura-tenky-tlusty-a-chytry-klient/

[13] ANGULAR: *Angular Concepts* [online]. [cit. 2020-04-12]. Dostupné z: https://angular.io/guide/architecture

[14] REACT: *Tutorial: Intro to React* [online]. [cit. 2020-04-12]. Dostupné z: https://reactjs.org/tutorial/tutorial.html

[15] OLEG ROMANYUK: Angular vs React: What to Choose for Your App? *KeenEthics* [online]. 2019 [cit. 2020-04-12]. Dostupné z: https://keenethics.com/blog/angular-vs-react-what-to-choose-for-your-app

[16] SHRIKANT SRIVASTAVA: *React vs Angular: What to Choose for Your App?* [online]. 2019 [cit. 2020-04-15]. Dosupné z: https://appinventiv.com/blog/react-vs-angular/

[17] PAVELS JELISEJEVS: *React vs Angular: An In-depth Comparison* [online]. 2019 [cit. 2020-04-15]. Dostupné z: https://www.sitepoint.com/react-vs-angular/

[18] ANGULAR: Introduction *AngularJS Material* [online]. [cit. 2020-04-12]. Dostupné z: https://material.angularjs.org/latest/

[19] SPRING: *Spring Boot* [online]. [cit. 2020-04-13]. Dostupné z: https://spring.io/projects/spring-boot

[20] AMAN GOEL: 10 Best Web Development Frameworks. *hackr.io* [online]. 2020 [cit. 2020-04-20]. Dostupné z: https://hackr.io/blog/top-10-web-development-frameworks-in-2020

[21] PIERRE CARBONNELLE: *PYPL PopularitY of Programming Language* [online]. 2020 [cit. 2020-05-06]. Dostupné z: http://pypl.github.io/PYPL.html

[22] ORACLE: Introduction to the Oracle Database *Database Concepts* [online]. [cit. 2020-04-22]. Dostupné z: https://docs.oracle.com/cd/B19306_01/server.102/b14220/intro.htm

[23] CODY ARSENAULT: The Pros and Cons of 8 Popular Databases *keycdn* [online]. 2017 [cit. 2020-04-22]. Dostupné z: https://www.keycdn.com/blog/popular-databases

[24] KUMAR CHANDRAKANT: Why Choose Spring as Your Java Framework? *baeldung* [online]. 2019 [cit. 2020-04-24]. Dostupné z: https://www.baeldung.com/spring-why-to-choose

[25] MICROSOFT: Chapter 5: Layered Application Guidelines *Microsoft Application Architecture Guide* [online]. 2010 [cit. 2020-04-24]. Dostupné z: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ee658109(v=pandp.10)?redirectedfrom=MSDN

[26] MIROSLAV ČERMÁK: Vícevrstvá architektura: popis vrstev *Vývoj SW* [online]. 2010 [cit. 2020-04-24]. Dostupné z https://www.cleverandsmart.cz/vicevrstva-architektura-popis-vrstev/

[27] JIŘÍ MLEJNEK: Architektonické vzory. duben LS 2018/2019, ČVUT, Praha, Přednáška BI-SI1.

[28] MICROSOFT: The Model-View-ViewModel Pattern *Enterprise Application Patterns eBook* [online]. 2017 [cit. 2020-04-28]. Dostupné z: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm

[29] SPRING: Spring Data JPA *Spring Data* [online]. [cit. 2020-04-28]. Dostupné z: https://spring.io/projects/spring-data-jpa

[30] MATTHEW TYSON: What is JPA? Introduction to the Java Persistence API *JavaWorld* [online]. 2019 [cit. 2020-04-28]. Dostupné z: https://www.javaworld.com/article/3379043/what-is-jpa-introduction-to-the-java-persistence-api.html

[31] HIBERNATE: *About* [online]. [cit. 2020-04-30]. Dostupné z: https://hibernate.org/orm/

[32] MAVEN: *What is Maven?* [online]. [cit. 2020-04-30]. Dostupné z: http://maven.apache.org/what-is-maven.html

[33] REDIS: *Introduction to Redis* [online]. [cit. 2020-04-30]. Dostupné z: https://redis.io/topics/introduction

[34] MODELMAPPER: *Simple, Intelligent, Object Mapping* [online]. [cit. 2020-04-30]. Dostupné z: http://modelmapper.org/

[35] SPRING: *Spring Security Reference* [online]. [cit. 2020-04-30]. Dostupné z: https://docs.spring.io/spring-security/site/docs/current/reference/html5/#features

[36] IRON MOUNTAIN: *Iron Mountain Connect* [online]. [cit. 2020-04-30]. Dostupné z: https://www.ironmountain.com/information-management/reporting-and-inventory-management/imconnect

[37] M-FILE: *Genius and easy Document Management System* [online]. [cit. 2020-05-05]. Dostupné z: https://www.m-files.com/en/document-management

[38] EFILECABINET: *Document Management Software* [online]. [cit. 2020-05-05]. Dostupné z: https://www.efilecabinet.com/

[39] NAHARI HADI a RONALD L. KRUTZ: *Web Commerce Security: Design and Development* [online]. 1. John Wiley Sons, Incorporated, 2011 [cit. 2020-05-06]. ISBN 9781118098899. Dostupné z: https://ebookcentral.proquest.com/lib/cvut/detail.action?docID=706729&query=%22API+Authentication%22#

[40] MARCOS LOPEZ GONZALES: *JUnit5 @RunWith* [online]. [cit. 2020-05-10]. Dostupné z: https://www.baeldung.com/junit-5-runwith

[41] MILI ALI a FAIROUZ TCHIER: *Software Testing: Concepts and Operations* [online]. 1. John Wiley Sons, Incorporated, 2015 [cit. 2020-05-05]. ISBN 9781119065593. Dostupné z: https://ebookcentral.proquest.com/lib/cvut/detail.action?docID=4040909#

[42] RADEK KITNER: *Typy testování software* [online]. [cit. 2020-05-10]. Dostupné z: https://kitner.cz/testovani_softwaru/typy-testovani-software-trideni-testu/

[43] MURAT ARTIM: *Deploying Spring Boot Applications* [online]. 2019 [cit. 2020-05-20]. Dostupné z: https://medium.com/swlh/deploying-spring-boot-applications-15e14db25ff0

Příloha **B**

Obsah přiložené SD karty

- `readme.txt` ..... stručný popis obsahu SD karty
- `application`
  - `src` ..... ukázka aplikace
    - `server` ..... back-end aplikace
    - `web` ..... front-end aplikace
  - `doc` ..... dokumentace zdrojových kódů
- `thesis`
  - `src` ..... zdrojová forma práce ve formátu LaTeX
  - `text` ..... zadání práce ve formátu PDF
  - `thesis.pdf` ..... text práce ve formátu PDF