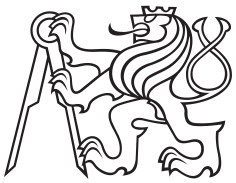


České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Aplikace pro správu dokumentů s využitím OCR

Phuong Dong Cu

Vedoucí: Ing. Kyrylo Bulat
Obor: Softwarové inženýrství a technologie
Květen 2023

Poděkování

Rád bych poděkoval společnosti Wflow, zejména Filipu Dolanskému a Petrovi Machovi, za to, že jakožto nepotenciálnímu zákazníkovi byli ochotni věnovat čas na představení jejich produktu a poskytnutí hlubšího porozumění problematice. Tato zkušenost mi velmi pomohla při analýze existujících řešení.

Velké díky patří i Ing. Kyrylu Bulatovovi za odborné vedení a konzultace při vypracování mé bakalářské práce. Byl ochotný investovat svůj čas a poskytnout cenné rady, díky kterým jsem byl schopen dosáhnout lepších výsledků.

Prohlášení

I declare that this work is all my own work and I have cited all sources I have used in the bibliography.

Prague, května 26, 2023

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

V Praze, 26. May 2023

Abstrakt

Tato bakalářská práce se zaměřuje na analýzu, návrh a implementaci webové aplikace pro správu dokumentů. Hlavními cíli práce jsou provést analýzu stávajících řešení, identifikovat požadavky a use case scénáře. Dále je v práci zkoumáno využití technologií a architektonických vzorů. Součástí práce je také zpracování doménového modelu a návrh uživatelského rozhraní z hlediska designu. Práce obashuje i implementaci webové aplikace, zahrnující také end to end testování a testování použitelnosti.

Klíčová slova: OCR, Kotlin, Spring Boot, React, Docker

Vedoucí: Ing. Kyrylo Bulat

Abstract

This bachelor thesis focuses on the analysis, design and implementation of a web-based document management application. The main objectives of the thesis are to analyze existing solutions, identify requirements and use case scenarios. Furthermore, the thesis explores the use of technologies and architectural patterns. The thesis also includes the development of a domain model and the design of the user interface from a design perspective. The thesis also covers the implementation of the web application, including end to end testing and usability testing.

Keywords: OCR, Kotlin, Spring Boot, React, Docker

Title translation: Document management application using OCR

Obsah

Obrázky

Acronyms

AES Advanced Encryption Standard. 10

AI Artificial intelligece. 4

API Application Programming Interface. 40, 44, 45, 51, 54, 69, 77

ARES Administrativní registr ekonomických subjektů. 9

B2B Business to business. 3, 8, 11

BSON binární JSON. 42

CRUD Create, Read, Update, Delete. 74

CSS Cascading Style Sheets. 43

DAO Data Access Object. 62

DIČ Daňové identifikační číslo. 8

ERP Enterprise Resource Planning. 8, 9

FIFO First in first out. 40

GB Gigabyte. 4, 5, 47

HTML Hyper Text Markup Language. 43

HTTP Hypertext Transfer Protocol. 45

HTTPS Hypertext Transfer Protocol Secure. 17

IČO Identifikační číslo osoby. 8

JSON JavaScript Object Notation. 42

JVM Java Virtual Machine. 36

JWT JSON Web Token. v, 45

MVC Model-View-Controller. 38, 39, 43

OCR Optical Character Recognition. v, 3–5, 8–11, 49–51, 59

ORM Objektově relační mapování. 74

PHP Hypertext Preprocessor. 38, 39

SEO Search engine optimization. 43, 44

SQL Structured Query Language. 41

SSO Single sign-on. 45

TB Terabyte. 4, 5

URL Uniform Resource Locator. 75

XML Extensible Markup Language. 42

Kapitola 1
Úvod

Před zahájením vývoje systému je nutné si udělat představu o tom, co by daný systém měl poskytovat. K tomu slouží analýza. Specifikace a analýza požadavků je důležitá fáze projektu. Cílem této kapitoly je analýza požadavků a technologií. V rámci analýzy je důležité provedení rešerše již existujícího řešení, která pomůže k definování požadavků nebo k identifikaci rizik. Pokud je systém vyvíjen pro klienta, je také nutné analyzovat potřeby u zadavatele. Analýza poskytuje i odpovědi na otázky jako časový odhad práce, rozpočet za daný projekt atd.

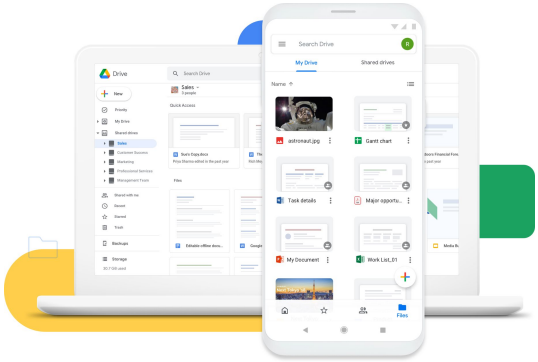
Současná řešení

Existuje mnoho systémů pro správu dokumentů na trhu, jako jsou Google Drive, Dropbox a OneDrive, avšak žádný z nich neposkytuje funkci Optical Character Recognition (OCR) pro automatické přečtení a zpracování textu z dokumentů. Takové řešení je vhodné i pro systémy pro digitalizaci účetnictví, kde se často ukládají faktury a další dokumenty, jejichž data lze vyčíst pomocí OCR, což usnadňuje práci, například Digitoo, Wflow nebo Rossum. Tyto systémy jsou většinou placené a nabízejí pouze základní funkce, neboť jsou určené pro Business to business (B2B).

V rámci analýzy byl proveden rozhovor s firmou Wflow, který zahrnoval setkání s Petrem Machem, vedoucím technikem a Filipem Dolanským, produktovým manažerem, za účelem hloubkové analýzy toho, co systém nabízí a technického řešení systému. Více informací o Wflow bude uvedeno níže v kapitole. Pro potřeby analýzy bylo vybráno pět systémů k porovnání: Google Drive, Dropbox, Nanonets, Digitoo a Wflow.

1.0.1 Google Drive

Google Drive je pravděpodobně jedním z nejznámějších cloudových úložišť pro ukládání různých souborů, včetně videí, .exe souborů atd. Vyznačuje se dostupností na všech platformách a přívětivým uživatelským rozhraním. Umožňuje uživatelům nahrávat soubory, kategorizovat je a označovat je různými tagy. Dále umožňuje uživatelům sdílet soubory nebo složky mezi různými uživateli. Google Drive podporuje integraci s nástroji od společnosti Microsoft Office, což umožňuje bezproblémové spolupráci bez nutnosti převodu formátu souborů. Součástí Google Drive je i technologie vyhledávání pomocí umělé inteligence (AI search), která umožňuje rychlé a spolehlivé hledání souborů.



Obrázek 1.1: Ukázka aplikace Google Drive
Zdroj: `www.google.com/intl/cs_CZ/drive/`

- Kapacita úložiště v rámci Google Drive je omezená na 15 GB, ale uživatel si může za měsíční předplatné objednat balíček "Business Standard", který poskytuje 2 TB úložiště.

Výhody:

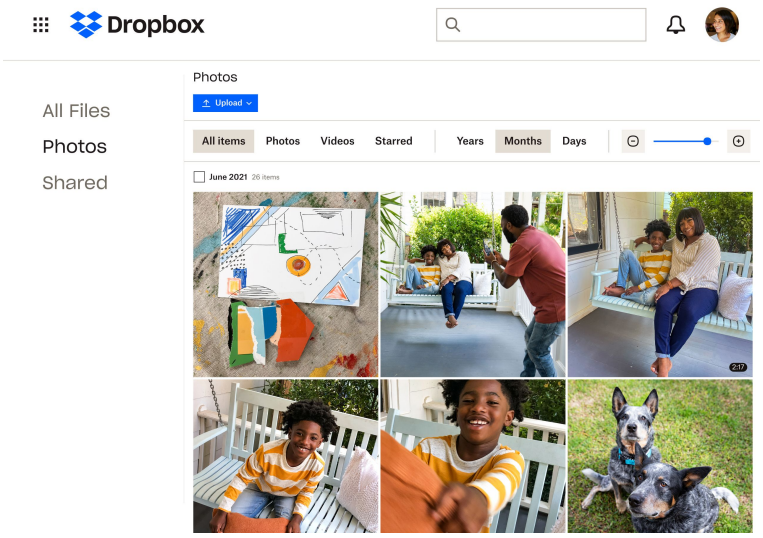
- Přehledná, intuitivní a snadno použitelná aplikace.
- Dostupný na více platformách, což umožňuje jeho snadné použití na různých zařízeních.
- Zajištěna stabilita a spolehlivost.
- Podporuje integraci s řadou různých aplikací.

Nevýhody:

- Úložiště je omezené.
- Předplatné je dražší než u konkurence.
- Neobsahuje OCR pro vyčítání textu.

1.0.2 Dropbox

Dropbox je podobně jako Google Drive cloudové úložiště, které umožňuje uživatelům nahrávat a sdílet dokumenty s ostatními uživateli. Kapacita úložiště je omezená na 2 GB, avšak lze si zakoupit měsíční nebo roční předplatné v rámci balíčku Plus, který poskytuje 2 TB úložiště, Family 2 TB pro 6 uživatelů nebo Professional 3 TB. Dropbox podporuje integraci s řadou aplikací, jako je Microsoft Office, Slack, Adobe a je dostupný na všech platformách od Linuxu až po mobilní telefony.



Obrázek 1.2: Ukázka aplikace Dropbox
Zdroj: `www.dropbox.com/`

Výhody:

- Přehledná, intuitivní a jednoduchá aplikace.
- Podporuje až 100 formátů dokumentu.
- Dostupnost na více platformách.
- Levnější předplatné než u konkurence.

Nevýhody:

- Úložiště je omezeno, pouze 2GB.
- Neobsahuje OCR pro vyčítání textu.

1.0.3 Nanonets

Nanonets je aplikace, která využívá umělou inteligenci k automatizaci ručního zadávání dat. Aplikace nabízí předpřipravené modely pro vyčítání textů z dokumentů, jako jsou faktury, účtenky, cestovní pasy a tabulky. Uživatel může v aplikaci vytvořit také vlastní model, kdy je nutné nahrát alespoň deset podobných souborů a označit na každém jednotlivé části, které chce v budoucnu automaticky vyčítat. Nahrané dokumenty lze upravovat, sdílet, označovat a v případě faktur i schvalovat. Aplikace umožňuje také nahrávání přes e-mail, stačí poslat například v příloze e-mailu fakturu na určitou e-mailovou adresu, faktura se automaticky nahraje do aplikace a vytěží se z ní automaticky.

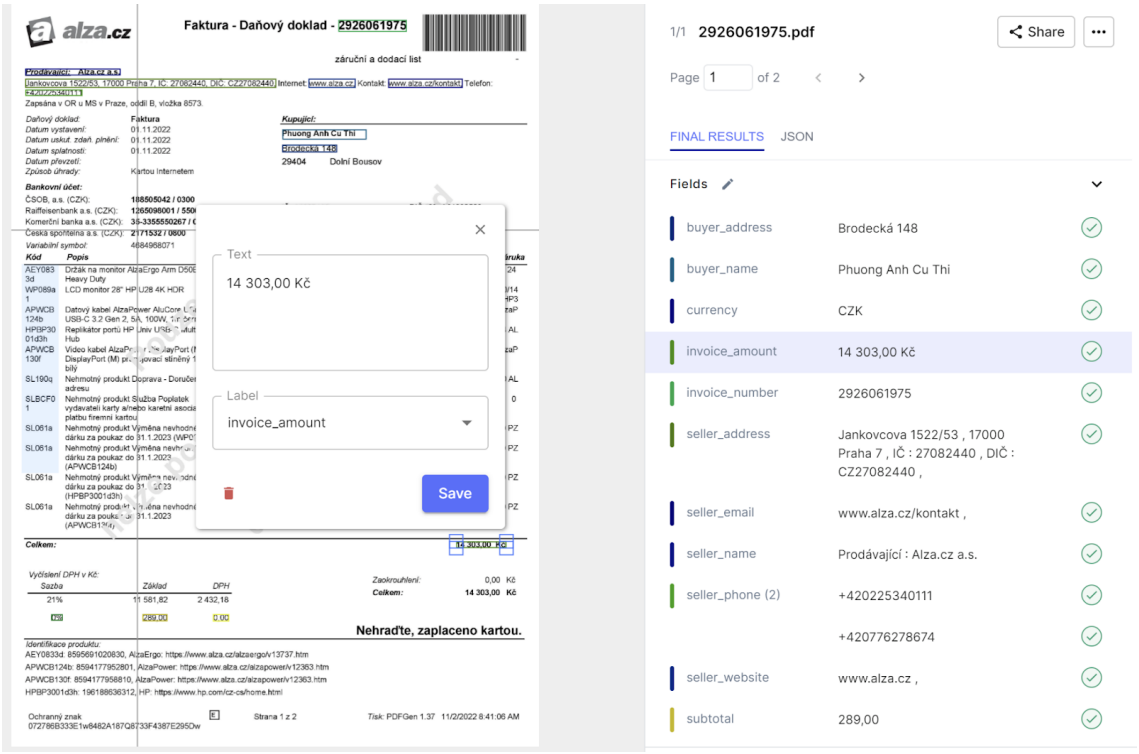
- Aplikace Nanonets je rozdělena do tří balíčků: STARTER, PRO a ENTERPRISE. Balíček STARTER je zdarma, ale má omezený limit pro nahrávání dokumentů. Balíček PRO za 499 dolarů měsíčně zahrnuje vytěžování položek z faktur, možnost sdílení projektů mezi uživateli a schvalovací práva pro uživatele. Balíček ENTERPRISE navíc obsahuje API pro integraci, možnost vytvoření onboarding stránky pro klienta, SSO login a SLAs (Služby úrovně smluv).

Výhody:

- Předpřipravené modely pro vyčítání.
- Aplikace umožňuje si vytvořit vlastní model pro vytěžování.
- Integrace s Gmail, Dropbox, Google Drive.
- Umožňuje uživatelům nahrávat dokumenty pomocí e-mailu.

Nevýhody:

- Předplatné balíčky jsou drahé.
- Vyčítání dokumentu trvá déle než například u Wflow.
- Nemusí být pro všechny uživatele intuitivní.

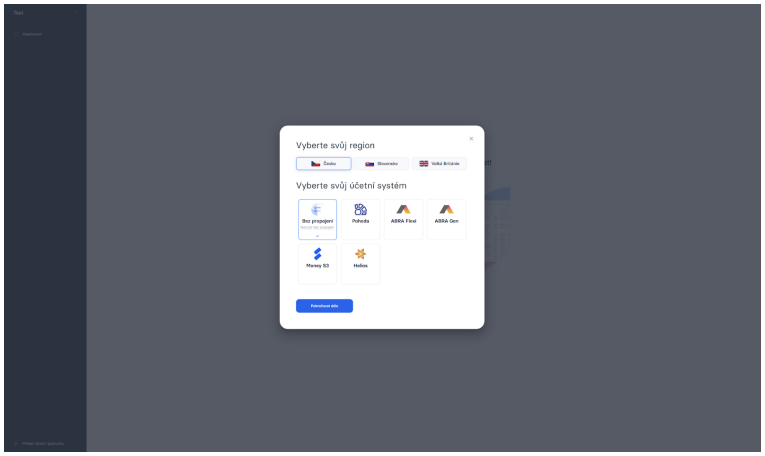


Obrázek 1.3: Ukázka využití OCR pro vyčítání textu
Zdroj: `www.nanonets.com/`

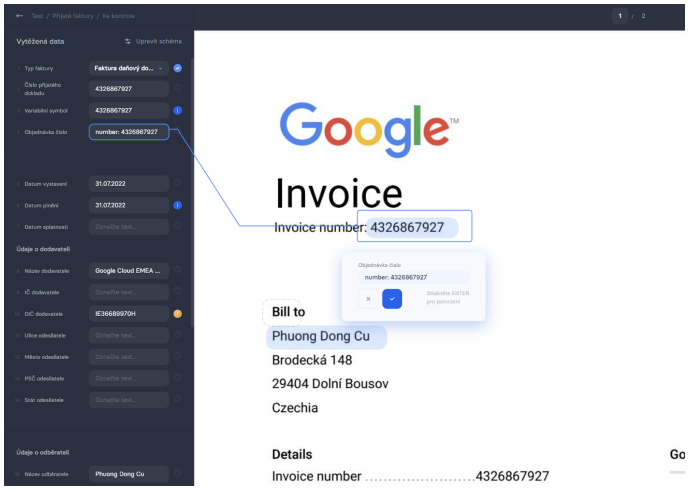
1.0.4 Digitoo

Digitoo je B2B produkt, který pomocí Optical Character Recognition (OCR) digitalizuje účetnictví. Aplikace umožňuje firmám nahrávat, schvalovat a správat faktury, vytvářet organizace a účetní fronty. Podporuje také integraci s různými ERP systémy, jako je Pohoda, Money S3 a Abra 1.4.

Systém Digitoo funguje tak, že firmy mohou nahrávat faktury, které jsou následně zpracovány pomocí OCR pro automatické vyčítání informací, jako je název dodavatele, IČO, DIČ, datum splacení, bankovní účet, jednotlivé položky a celková cena. Účetní mohou následně zkontrolovat, zda byly informace správně vytěženy, a pokud ne, mohou je manuálně upravit 1.5. Po schválení faktury je možné ji zaplatit a pokud je systém napojený na ERP, mohou se data automaticky přenést do tohoto systému.



Obrázek 1.4: Ukázka systémů, které Digitoo podporuje



Obrázek 1.5: Ukázka využití OCR pro vyčítání textu

- Systém Digitoo je vyvíjen s využitím technologií Node.JS pro backend a React pro frontend. V rámci své architektury využívá také GraphQL, ElasticSearch a PostgreSQL. Pro DevOps se spoléhají na Microsoft Azure a Firebase. Co se týče OCR, má k dispozici vlastní neuronovou síť, kterou neustále trénuje.

Výhody:

- Jednoduchá a intuitivní aplikace.
- Široká škála integrací s různými ERP systémy.
- Načítání dat pomocí DIČ z Administrativní registr ekonomických subjektů (ARES) databáze.
- Vlastní neuronový model pro OCR.

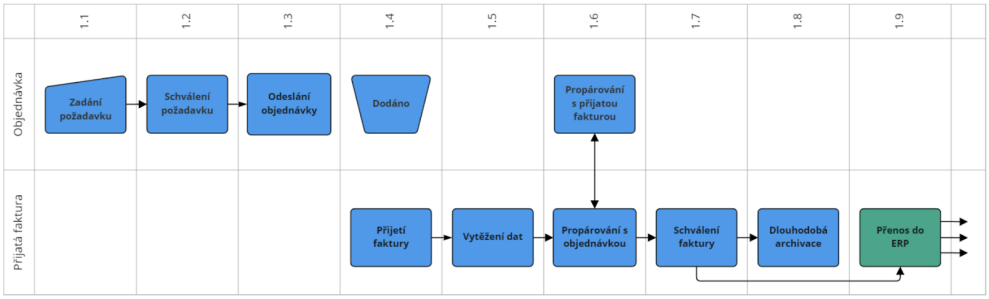
Nevýhody:

- Při srovnání s konkurencí se zdá, že aplikace Digitoo má pomalejší chod než její konkurenti.

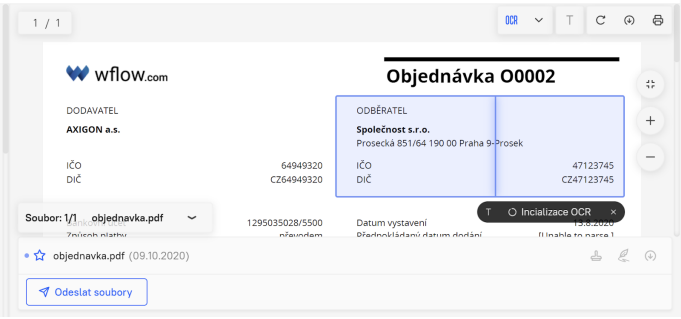
1.0.5 Wflow

Wflow se pyšní automatizací účetnictví pomocí OCR a umělé inteligence, což umožňuje šetřit čas a zefektivnit práci účetních oddělení. Aplikace je navíc přístupná z více platform a podporuje integraci s účetními systémy jako Pohoda nebo Money S3. Součástí systému je také online archivace dokumentů, což zjednodušuje jejich následné hledání a správu. V neposlední řadě Wflow nabízí možnost přiřazovat doklady k různým projektům a střediskům, což usnadňuje sledování výdajů a přehlednost účetnictví.

Je možné nahrát do systému účtenky, faktury a objednávky. Systém zajišťuje integraci s programy Pohoda, Money S3, Helios a Abra.



Obrázek 1.6: Průchod v aplikaci Wflow



Obrázek 1.7: Ukázka využití OCR pro vyčítání textu

Jedná se o monoliticky vyvíjený systém s komunikací prostřednictvím RESTových rozhraní se skládá z back-endu vyvíjeného v .NET Core 6 a front-endu vyvíjeného v Angularu. Pro vyhledávání textů v dokumentech je použita umělá inteligence Rossum s přesností až 95%. Dokumenty jsou ukládány jako BLOBy a data jsou ukládána v databázi Microsoft SQL Server. Systém je nasazen pomocí Azure Pipeline na Microsoft Azure. Co se týče bezpečnosti, pro autorizaci je použit OAuth2 Identity Server a pro šifrování dokumentů je použita technologie 256bitového Advanced Encryption Standard (AES), která je šifrovacím standardem.

Výhody:

- Aplikace je rychlá.
- OCR s vysokou přesností.
- Možnost vytváření objednávek v aplikaci.

Nevýhody:

- Nemusí být pro všechny uživatele intuitivní.
- Určeno pro B2B využití.

Kapitola 2

Analýza požadavků

- Analýza požadavků je jedním z klíčových kroků při analýze softwarového řešení. Jejím cílem je definovat požadavky a funkčnost aplikace, což může být děláno na základě předem stanovených požadavků zadavatele nebo na základě analýzy existujícího řešení. V případě této práce se bude jednat o analýzu existujícího řešení.
- Požadavek by měl popisovat potřebu, nikoli konkrétní řešení. Měl by odpovědět na otázky "Proč?"a "Co?", ale ne na otázku "Jak?". Požadavky lze rozdělit na funkční a nefunkční. Funkční požadavky popisují chování systému, zatímco nefunkční požadavky definují očekávané vlastnosti softwaru, aniž by popisovaly konkrétní business funkcionalitu.
- Výsledkem analýzy požadavků by měla být definice požadavků, use case diagram a prototyp aplikace.

Funkční požadavky

FR1 - **Autorizace a autentizace**

- Webová aplikace bude poskytovat uživatelům možnost registrace, přihlášení a odhlášení.

FR2 - **Správa složek**

- Aplikace bude umožňovat uživatelům následující operace:
  - Vytvářet složky.
  - Mazat složky.
  - Pojmenovávat složky.
  - Zobrazení s kým je složka sdílena.
  - Zobrazení datum vytvoření.
  - Zobrazení kým byla složka vytvořena.

FR3 - **Sdílení složek**

- Aplikace bude umožňovat uživatelům sdílet složky, včetně všech dokumentů obsažených v dané složce, s vybranými uživateli. Uživatel, který vlastní sdílenou složku, bude mít právo odebírat ostatní uživatele ze sdílené složky. Uživatelům, kterým byla složka sdílena, se zobrazí v pracovišti sekce "Sdílené složky". Ostatní uživatelé přiřazení ke složce budou moci editovat dokumenty obsažené v této složce.

FR4 - **Správa labelů**

- Labely jsou nástrojem pro označování dat. Každá složka bude mít vlastní skupinu labelů, které ji budou charakterizovat. Uživatelé budou moci zobrazit již existující labely, vytvářet si vlastní labely nebo je upravovat popřípadě mazat.

FR5 - **Správa dokumentů**

- Softwarový systém bude umožňovat uživatelům nahrávat soubory, přesouvat je, mazat soubory, které nahráli sami a přejmenovávat soubory. V detailu dokumentu bude umožněno označování dat pomocí labelů. Systém bude podporovat pouze dokumenty ve formátech .pdf, .jpg a .png.

FR6 - **Zobrazení poslední aktivity dokumentu**

- Uživatelé budou moci zjistit, který uživatel naposledy modifikoval dokument.

FR7 - **Zobrazení a řazení dokumentů**

- Aplikace umožní uživatelům zobrazit dokumenty v dané složce a seřadit je podle labelů.

FR8 - **Možnost sdílení dokumentů**

- Aplikace bude umožňovat uživatelům sdílet jednotlivé dokumenty s ostatními uživateli. Uživatel, kterému byl dokument sdílen, ho uvidí v sekci "Ostatní soubory" v pracovišti. Tato osoba bude moci dokument přetáhnout do své vlastní složky.

FR9 - **Vytváření jednorázových odkazů pro sdílení dokumentu**

- Systém bude umožňovat uživatelům vytvářet dočasné odkazy pro sdílení dokumentů. Tyto odkazy budou mít životnost 15 minut od momentu jejich vytvoření. Uživatel, který otevře daný odkaz, nemusí být přihlášený a může si dokument pouze prohlédnout, nikoliv jej modifikovat. Pokud se pokusí otevřít již neplatný odkaz, systém přesměruje ne-přihlášeného uživatele na přihlašovací stránku a přihlášeného uživatele na dashboard.

FR10 - **Možnost nahrání více dokumentů pro zpracování**

- Systém bude umožňovat uživatelům nahrávat větší počty dokumentů najednou, a to pomocí asynchronního zpracování jednotlivých dokumentů.

FR11 - **Vytěžení dat z dokumentu pomocí OCR**

- Systém bude využívat OCR pro extrakci dat z dokumentů pro možné budoucí zpracování těchto dokumentů.

FR12 - **Vyplňování pomocí vytěžených dat**

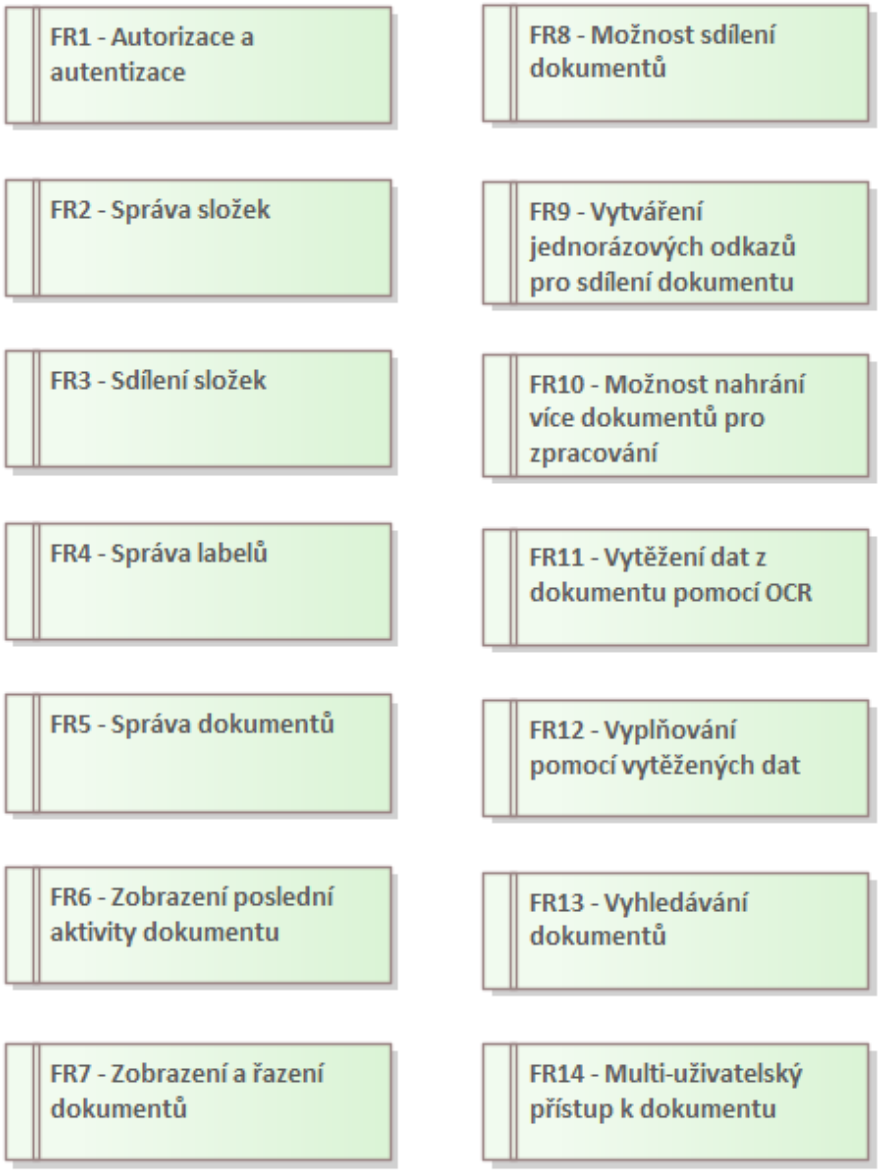
- Systém umožní uživatelům vyplňovat data v dokumentech pomocí interaktivního kliknutí na konkrétní text vyfoceného/naskenovaného dokumentu.

FR13 - **Vyhledávání dokumentů**

- Systém umožní uživatelům vyhledávat dokumenty pomocí názvů dokumentů nebo pomocí přiřazených labelů.

FR14 - **Multi-uživatelský přístup k dokumentu**

- Systém bude umožňovat modifikaci dokumentu pouze prvnímu uživateli, který daný dokument otevře. Ostatním uživatelům, kteří mají dokument také otevřený, bude systém signalizovat, že jsou pouze čtenáři a neumožní jim modifikovat dokument.



Obrázek 2.1: Funkční požadavky

Nefunkční požadavky

NFR1 - **Dostupnost aplikace**

- Webová aplikace by měla být dostupná v různých webových prohlížečích, jako jsou Google Chrome, Firefox, Edge a Safari.

NFR2 - **Intuitivní design**

- Aplikace by měla být navržena tak, aby byla snadno pochopitelná a intuitivně použitelná pro uživatele.

NFR3 - **Responzivní design**

- Aplikace by měla být optimalizována pro použití na mobilních zařízeních.

NFR4 - **Zabezpečení dat**

- Aplikace by měla zajišťovat správné ověřování přístupových práv uživatelů k jednotlivým dokumentům a složkám.

NFR5 - **Stabilita aplikace**

- Softwarová aplikace by měla být schopna rychle reagovat na uživatelské požadavky s maximální dobou odezvy do dvou sekund, s výjimkou procesu extrakce dat z dokumentů.

NFR6 - **Zabezpečení profilů**

- Aplikace by měla být navržena tak, aby poskytovala ochranu osobních údajů uživatelů.

NFR7 - **Dostupnost aplikace po 24 hodin**
NFR8 - **Hypertext Transfer Protocol Secure protokol**

- Webová aplikace by měla používat bezpečnostní protokol HTTPS pro zabezpečení komunikace mezi webovým serverem a klientem.

NFR9 - **Dobře škálovatelná**

- Program by měl být vyvíjen s ohledem na možnost dalšího rozšiřování o nové funkce.

NFR10 - **Testovatelná**

- Pro zajištění stability aplikace je důležité, aby byla navržena tak, aby byla snadno testovatelná.

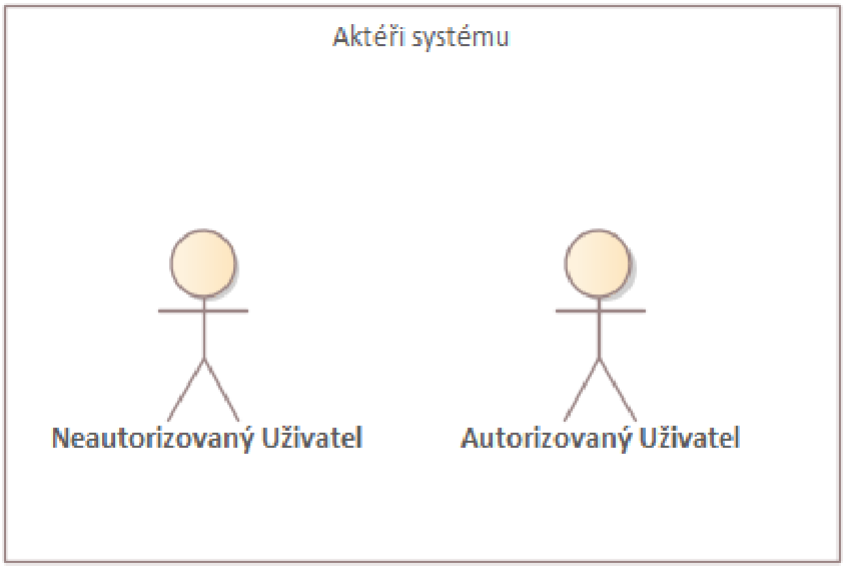
Obrázek 2.2: Nefunkční požadavky

Kapitola 3

Analýza use casů

Use casy jsou klíčovým nástrojem pro definování požadavků na softwarový systém. Vznikají během analýzy požadavků a představují seznam interakcí mezi aktéry a systémem. V rámci této kapitoly budou popsány jednotlivé případy užití, včetně popisu užití, aktorů a, v případě potřeby, i scénářů pro komplexnější případy.

- Ve vztahu k této práci budou existovat pouze dva typy aktérů: autorizovaní uživatelé a neautorizovaní uživatelé 3.1.



Obrázek 3.1: Aktoři systému

UC1 - Registrovat se

Popis:

- Uživatel si chce vytvořit účet, aby mohl využívat systém.

Scénář:

- Uživatel si přes domovskou stránku rozklikne tlačítko registrace.
- Na registrační stránce uživatel zadá všechny potřebné údaje a potvrdí registraci.
- Po úspěšné registraci systém přesměruje uživatele na dashboard stránku.

Aktéři:

- Neautorizovaný uživatel.

UC2 - Otevřít jednorázový dočasný odkaz

Popis:

- Uživatel chce zobrazit dokument.

Scénář:

- Uživatel klikne na dočasný odkaz, po kterém systém přesměruje uživatele na náhled dokumentu
- Pokud odkaz už nebude již platný, systém přesměruje uživatele na dashboard stránku pokud je uživatel autorizován, jinak systém přesměruje uživatele na přihlašovací stránku
- Po úspěšné registraci systém přesměruje uživatele na dashboard stránku

Aktéři:

- Neautorizovaný uživatel a autorizovaný uživatel.

UC3 - Přihlásit se

Popis:

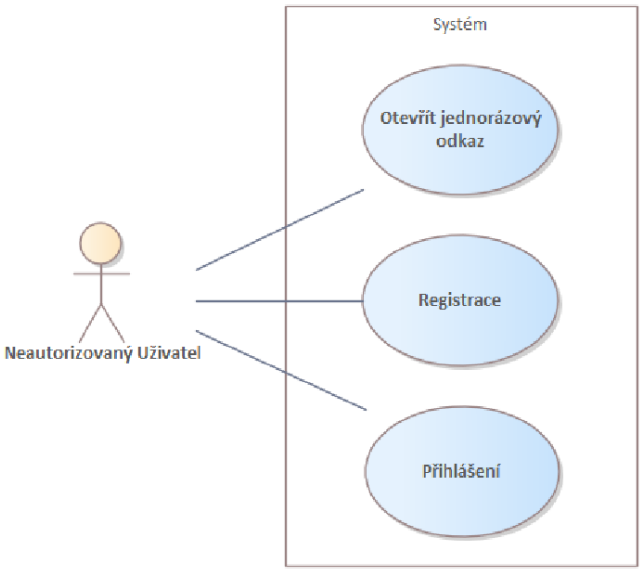
- Uživatel se chce přihlásit do aplikace.

Scénář:

- Uživatel vyplní uživatelské jméno nebo email a heslo poté klikne na přihlásit.
- Pokud uživatel zadal validní informace, systém přesměruje uživatele na dashboard stránku.
- Při zadání nevalidních údajů systém upozorní uživatele o neúspešné autorizaci.

Aktéři:

- Neautorizovaný uživatel.



Obrázek 3.2: Use case - Diagram

UC4 - Odhlásit se

Popis:

- Uživatel chce opustit systém a nechce zůstat přihlášený.

Aktéři:

- Autorizovaný uživatel.

UC5 - Zobrazit osobní údaje

Aktéři:

- Autorizovaný uživatel.

UC6 - Úpravit osobní údaje

Popis:

- Uživatel si chce změnit svoje jméno.

Aktéři:

- Autorizovaný uživatel.

UC7 - Úpravit profilovou fotku

Popis:

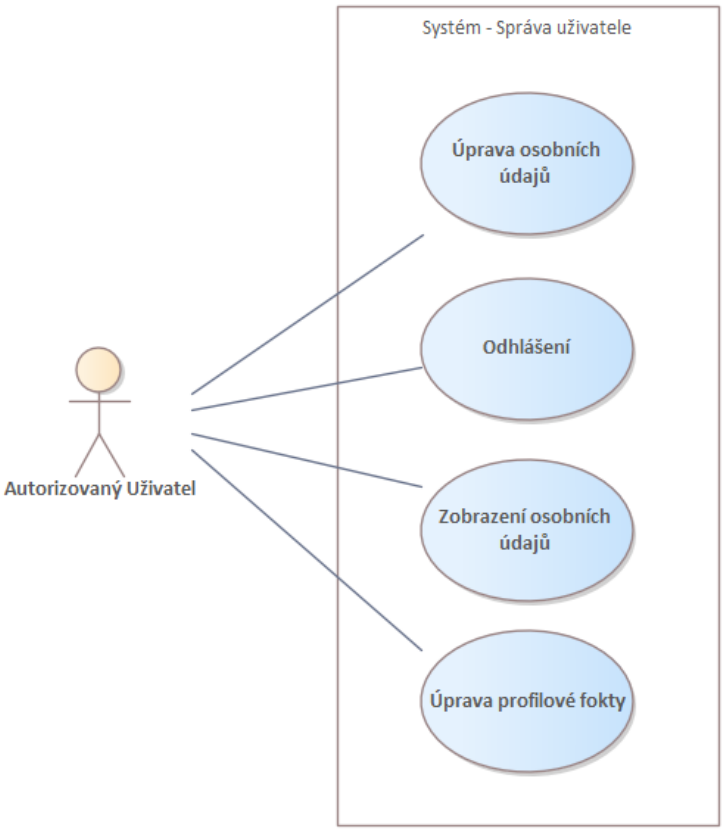
- Uživatel si chce změnit profilovou fotku.

Scénář:

- Uživatel si vybere barvu pozadí a barvu fontu.

Aktéři:

- Autorizovaný uživatel.



Obrázek 3.3: Use case - Diagram - Správa uživatele

Správa složek

UC8 - Vytvořit novou složku

Popis:

- Uživatel se rozhodne vytvořit složku, do které by mohl ukládat dokumenty. Aby tuto složku vytvořil, klikne na tlačítko "Vytvořit složku"a zadá její název. Poté klikne na tlačítko "Potvrdit". V případě, že uživatel nezadá název složky, systém ho upozorní, že nezadal název. Pokud zadá uživatel název, který již v systému existuje, systém ho upozorní na duplicitu názvu.

Aktéři:

- Autorizovaný uživatel.

UC9 - Zobrazit složky

Aktéři:

- Autorizovaný uživatel.

UC10 - Smazat složku

Aktéři:

- Autorizovaný uživatel.

UC11 - Přejmenovat složku

Popis:

- Uživatel si chce přejmenovat název složky.

Scénář:

- Pokud uživatel smaže název, systém upozorní uživatele o prázdném názvu.
- V případě duplicity názvu složky bude uživatele upozorněn systém.

Aktéři:

- Autorizovaný uživatel.

UC12 - Sdílet složku

Popis:

- Uživatel chce sdílet složku včetně všech dokumentů s ostatními uživateli.

Scénář:

- Uživatel klikne na přidat uživatelé a zadá emaily uživatelů.

Aktéři:

- Autorizovaný uživatel.

UC13 - Odebrat uživatele ze složky
Popis:

- Uživatel chce odebrat uživatelé ze složky.

Scénář:

- Uživatel si zobrazí všechny účastníky složky a odebere konkrétního účastníka.

Aktéři:

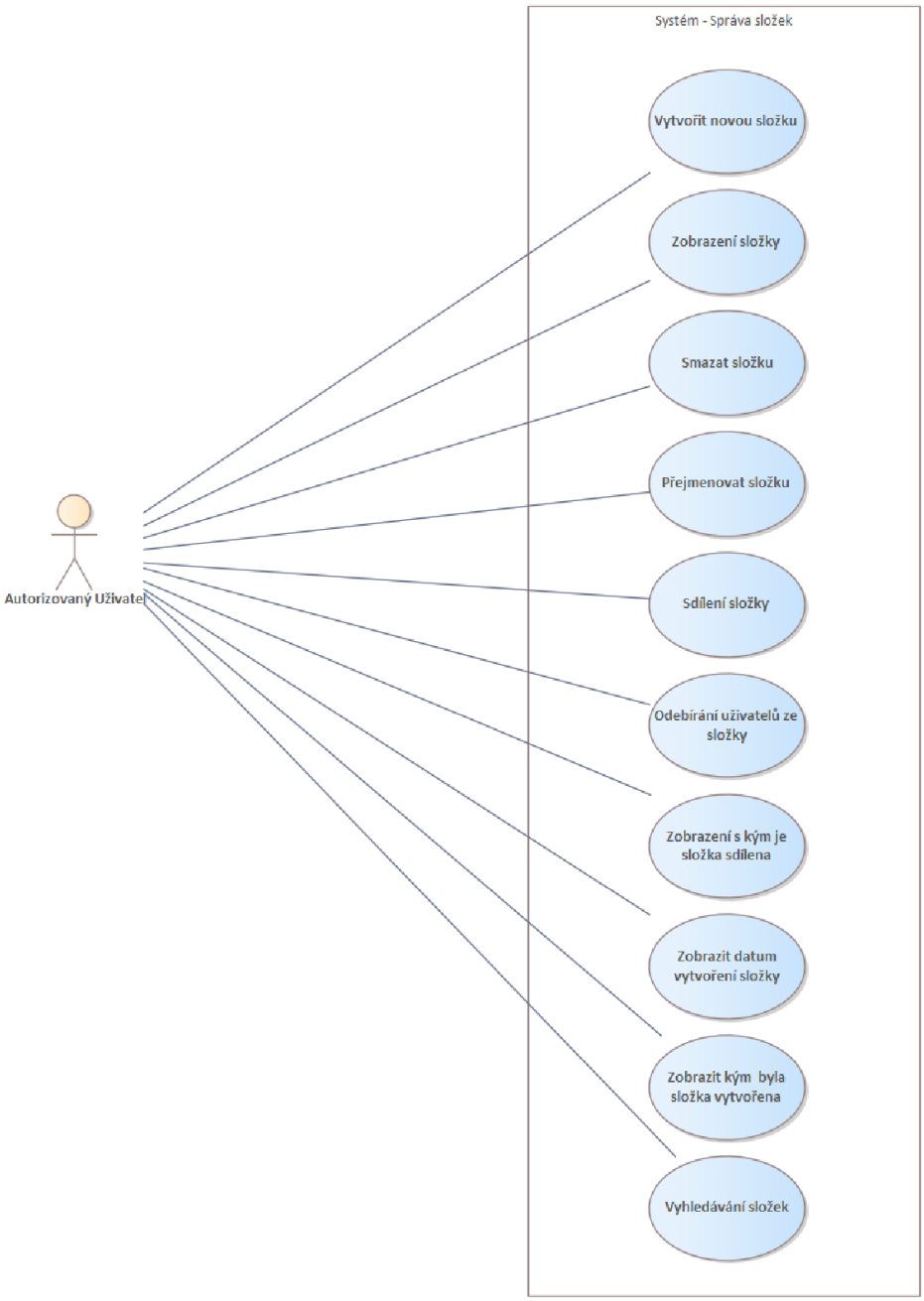
- Autorizovaný uživatel

UC14 - Zobrazit kým byla složka vytvořena
Aktéři:

- Autorizovaný uživatel.

UC15 - Vyhledat složku
Aktéři:

- Autorizovaný uživatel.



Obrázek 3.4: Use case - Diagram - Správa dokument

Správa dokument

UC16 - Nahrát dokumentu
Popis:

- Uživatel si chce nahrát dokument nebo dokumenty do systému.

Aktéři:

- Autorizovaný uživatel.

UC17 - Přesunout dokument do jiné složky
Popis:

- Uživatel si chce přesunout dokument nebo dokumenty do jiné složky.

Scénář:

- V případě, že se v rámci složky budou vyskytovat dokumenty se shodným názvem, bude uživatele upozorněn systém na tuto duplicitu a automaticky přidá k názvu pořadové číslo, aby byl dokument rozeznatelný od ostatních s identickým názvem. Toto opatření slouží ke zjednodušení orientace uživatele v rámci složky a ke zvýšení její přehlednosti.

Aktéři:

- Autorizovaný uživatel.

UC18 - Zobrazit detail dokumentu
Popis:

- Uživatel si přeje zobrazit detaily konkrétního dokumentu a podívat se na data, která jsou na něm uvedena.

Aktéři:

- Autorizovaný uživatel.

UC19 - Vyplnit data v dokumentu
Popis:

- Uživatel si bude chtít vyplnit data v detailu dokumentu.

Aktéři:

- Autorizovaný uživatel.

UC20 - Upravit data v detailu dokumentu
Popis:

- Uživatel si bude chtít upravit detail dokumentu pro snazší vyhledávání.

Aktéři:

- Autorizovaný uživatel.

UC21 - Anotovat data
Popis:

- Uživatel nebude chtít vyplňovat data ručně, ale bude chtít vyplňovat data pomocí anotování.

Aktéři:

- Autorizovaný uživatel.

UC22 - Smazat dokument
Aktéři:

- Autorizovaný uživatel.

UC23 - Sdílet dokumentu
Popis:

- Uživatel se rozhodne sdílet konkrétní dokument s ostatními uživateli. Chce tak umožnit ostatním uživatelům přístup k obsahu dokumentu a možnost jej společně upravovat.
- Dokument co je sdílený s ostatními uživateli, se zobrazí v pracovišti "Sdílené soubory"uživatele. Uživatelé tak mohou snadno a rychle najít sdílený dokument a začít s ním pracovat.

Scénář:

- Uživatel vybere požadovaný dokument ze seznamu a klikne na tlačítko "Zobrazit detaily", klikne na tlačítko "Sdílet"a vybere požadované uživatele nebo zadá jejich emaily a potvrdí své volby tlačítkem "Potvrdit".

Aktéři:

- Autorizovaný uživatel.

UC24 - Vytvořit jednorázový odkaz pro sdílení
Popis:

- Uživatel bude chtít sdílet jinému uživateli dokument, ale pouze pro náhled na 15 minut.

Scénář:

- Uživatel rozklikne detail dokumentu a klikne na "sdílet"a "jednorázový odkaz", systém vytvoří jednorázový odkaz a ten může uživatel následně zkopírovat do schránky.

Aktéři:

- Autorizovaný uživatel.

UC25 - Zobrazit poslední aktivitu
Popis:

- Uživatel chce vědět, kdo naposled pracoval na dokumentu.

Aktéři:

- Autorizovaný uživatel.

UC26 - Vyhledat dokumenty
Popis:

- Uživatel si přeje vyhledat konkrétní dokument nebo dokumenty za pomoci názvu, labelů nebo textu obsaženého v dokumentech.

Aktéři:

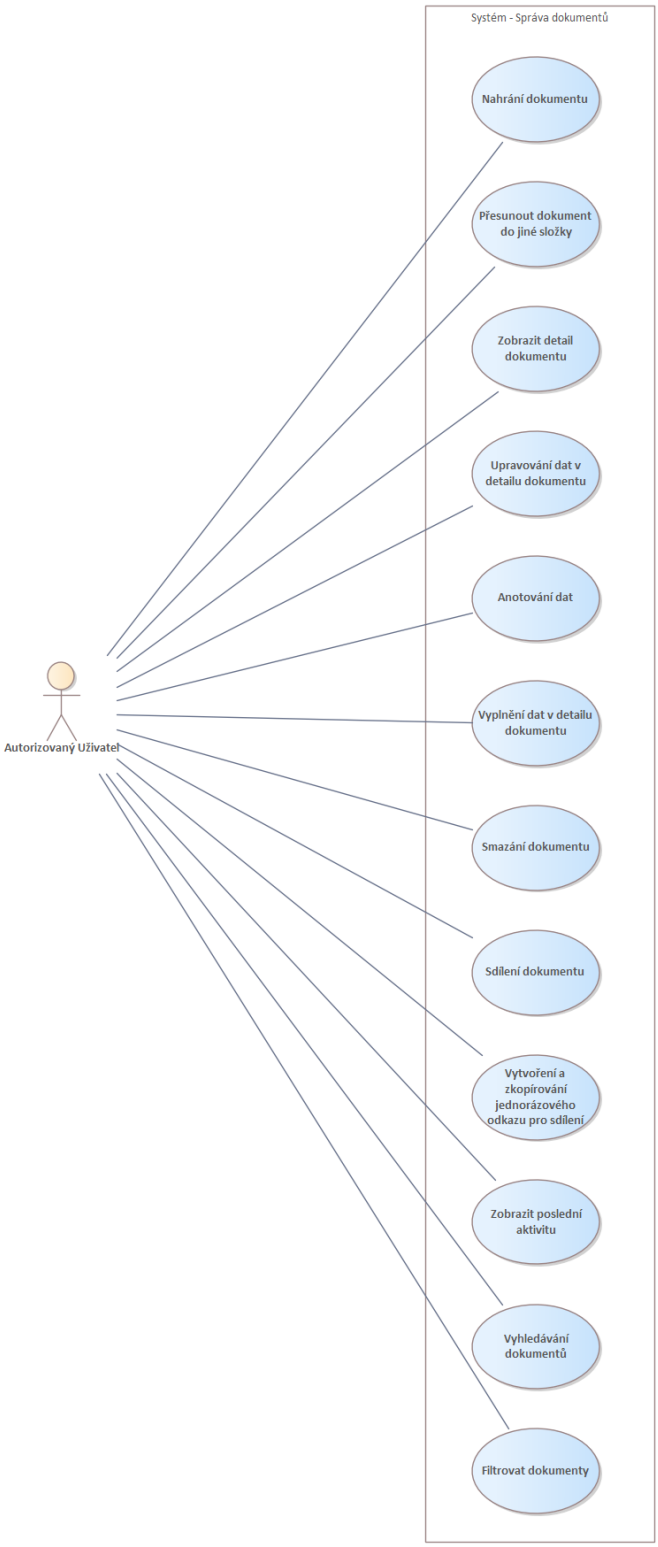
- Autorizovaný uživatel.

UC27 - Filtrovat dokumenty
Popis:

- Uživatel si bude chtít vyfiltrovat dokumenty pomocí labelů.

Aktéři:

- Autorizovaný uživatel.



Obrázek 3.5: Use case - Diagram - Správa dokumentu

Správa labelů

UC28 - Zobrazit label

Aktéři:

- Autorizovaný uživatel.

UC29 - Vytvořit label

Popis:

- Uživatel si chce vytvořit label a pojmenovat si ho.

Scénář:

- Pokud uživatel nevyplní název, systém upozorní uživatele o prázdném názvu. Pokud uživatel zadá název již existujícího labelu, systém upozorní uživatele o duplicitním názvu.

Aktéři:

- Autorizovaný uživatel.

UC30 - Smazat label

Scénář:

- Uživatel si zobrazí všechny labely, vybere daný label, který chce smazat a klikne na tlačítko “smazat”, systém následně upozorní uživatele o smazání dat, které jsou spojené s daným labelem.

Aktéři:

- Autorizovaný uživatel.

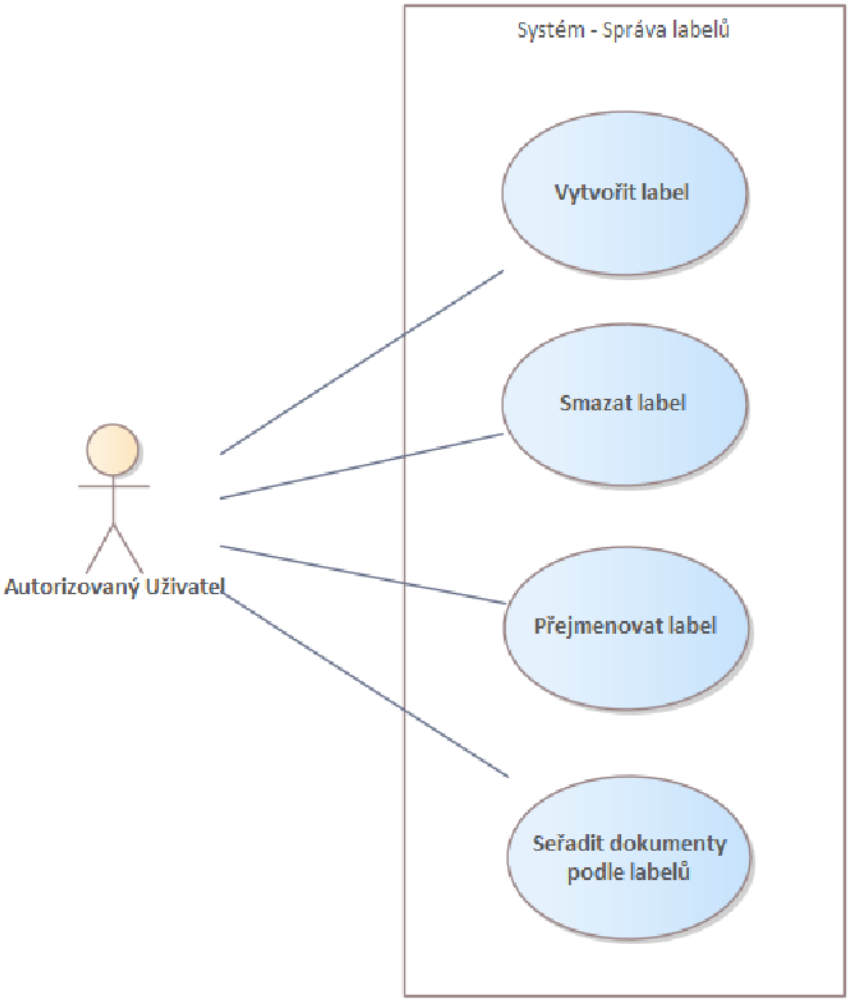
UC31 - Přejmenovat label

Scénář:

- Pokud uživatel smaže název labelu, systém upozorní uživatele o prázdném názvu. Pokud uživatel zadá název již existujícího labelu, systém upozorní uživatele o duplicitním názvu.

Aktéři:

- Autorizovaný uživatel.



Obrázek 3.6: Use case - Diagram - Správa labelů

UC32 - Zobrazit “Sdílené složky”
Aktéři:

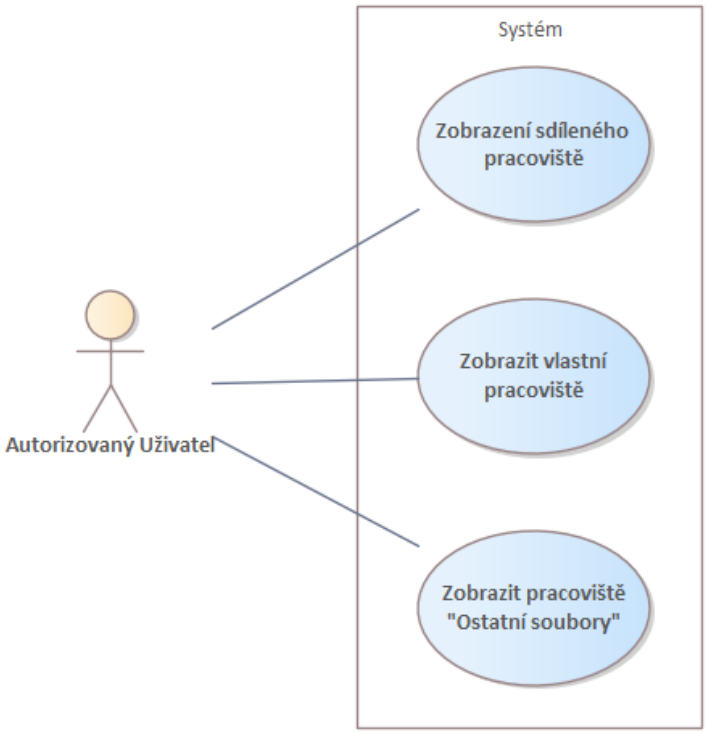
- Autorizovaný uživatel.

UC33 - Zobrazit “Vlastní složky”
Aktéři:

- Autorizovaný uživatel.

UC34 - Zobrazit “Ostatní soubory”
Aktéři:

- Autorizovaný uživatel.



Obrázek 3.7: Use case - Diagram

Kapitola 4
Analýza technologie

Před samotnou implementací je klíčové pečlivě zvážit volbu technologií pro daný projekt. Analýza technologií je důležitou součástí tohoto procesu, která umožňuje vybrat nejvhodnější nástroje pro daný účel. Při výběru technologií je důležité zohlednit funkčnost aplikace a její cílové použití. V této kapitole se budeme zabývat především webovými technologiemi vzhledem k funkčnosti a požadavkům systému. Technologie, na které se v této kapitole zaměříme, budou rozděleny do následujících skupin:

- Backend
- Databáze
- Frontend
- Autentizace
- Cloudové úložiště
- OCR

4.1 Backend

Backend je oblast aplikace, která je uživateli skrytá a zajišťuje většinu požadavků aplikace. Jeho úkolem je ukládání a získávání dat z databáze a poskytování těchto dat aplikaci. V následující kapitole budou podrobně analyzovány vybrané programovací jazyky, jejich rámce, využití, výhody a nevýhody.

4.1.1 Java

Java je jedním z nejoblíbenějších programovacích jazyků, který se vyznačuje tím, že je objektově orientovaný. Je kompilovaným jazykem, což znamená, že jeho zdrojový kód musí být přeložen do strojového kódu kompilátorem před spuštěním. Výhodou kompilace je vysoká rychlost a snadná detekce chyb v zdrojovém kódu, neboť při nalezení chyby kompilace spadne a vývojář je o ní informován. Java se používá pro vývoj her, enterprisových systémů,

webových aplikací, big data, internet of things, desktopových aplikací atd. Díky multiplatformnímu virtuálnímu stroji Java JVM je Java nejen široce používaná v technologiích, ale také snadno škálovatelná a především vysoce přenosná. JVM přenáší do počítače instrukce specifické pro danou platformu, což znamená, že jakmile vývojář vytvoří kód v Javě, může ho snadno přenést do jiného systému na téměř jakémkoli výpočetním zařízení. Mezi společnosti, které používají Javu pro vývoj, patří například Netflix, Google Android, Minecraft nebo Spotify [1].

Výhody:

- Objektově orientovaný.
- Velká komunita a dokumentace na internetu.

Nevýhody:

- Jeho výkon je slabší ve srovnání s ostatními jazyky.
- Při vývoji je třeba každou změnu v kódu znovu nasadit.

4.1.2 Kotlin

Stejně jako Java, Kotlin je objektově orientovaným programovacím jazykem. Je moderní, stručný a bezpečný a umožňuje kompilaci kódu do bytecodu a jeho spouštění na virtuálním stroji (JVM). Kotlin je kompatibilní s ekosystémem Javy, takže je možné s ním používat oblíbené frameworky, jako je Spring boot, který bude v následující kapitole podrobněji analyzován. Kotlin se používá pro vývoj server-side aplikací a mobilních aplikací [2][3].

Výhody:

- Je kompatibilní se stávajícím kódem Javy.
- Je snadno udržovatelný.

Nevýhody:

- Pomalejší kompilace.

4.1.3 Spring Boot framework pro Javu a Kotlin

Frameworky lze charakterizovat jako softwarové knihovny, které pomáhají a urychlují vývoj systému. Skládají se ze tříd, šablon, komponent a dalších struktur, které lze využít.

Spring boot framework je jedním z nejpoužívanějších frameworků s rozsáhlým ekosystémem. Poskytuje možnost vytváření enterprisových aplikací, webových aplikací a mikroservisních aplikací[4].

Výhody:

- Populární a stabilní framework.
- Volnost ve využití modulů.
- Spring Initializr pro vytvoření projektu.
- Obsáhlá dokumentace.
- Velká komunita.

Nevýhody:

- Komplexní.
- Pro menší projekty je příliš náročná.

4.1.4 PHP

Hypertext Preprocessor, též známý jako PHP, je skriptovací jazyk určený pro použití na straně serveru. Je interpretovaným jazykem a poskytuje širokou škálu frameworků, což z něj činí oblíbený jazyk pro vývojáře při tvorbě dynamických webových stránek za rozumné náklady [5].

Výhody:

- Rychlejší pro vývoj, jelikož se nemusí kompilovat.
- Vytvořeno pro vývoj webových stránek.
- Levnější na vývoj a údržbu.

Nevýhody:

- Nejedná se o kompilovaný jazyk tudíž vývojář se nedozví o chybě při vývoji.
- Nepřehledná dokumentace.

4.1.5 Laravel

Laravel je open-source framework pro Hypertext Preprocessor (PHP), jehož jádrem je pro většinu frameworků tradiční návrhový vzor Model-View-Controller (MVC). Je frameworkem, který vyžaduje minimální konfiguraci a při tvorbě webových aplikací opakovaně využívá existující komponenty z různých frameworků. Laravel je proslulý svou jednoduchostí a rychlostí při kódování a skvěle se hodí pro vytváření efektivních webových aplikací v jazyce PHP [6].

Výhody:

- Je neustále aktualizován.
- Umožňuje integraci libovolné aplikace nebo platformy třetí strany prostřednictvím rozhraní API.
- Má značný ekosystém dalších nástrojů.

Nevýhody:

- Některé aplikace vytvořené v Laravelu mohou mít nedostatečnou rychlost načítání na mobilních zařízeních.
- Migrace starších systémů do Laravelu je komplikovaná.

4.1.6 Symphony

Symfony je open-source framework pro webové aplikace v jazyce PHP, který se může používat na různých platformách a opírá se o vzor MVC. Má rozsáhlou komunitu a široké spektrum open-source projektů. Stejně jako Laravel je využitelný pro vývoj aplikací PHP [7].

Výhody:

- Symfony je pravidelně aktualizován.
- Jedná se o Open source technologii.

Nevýhody:

- Vyžaduje značný čas na testování a zároveň zpomaluje proces vývoje.

4.1.7 Python

Python je interpretovaný programovací jazyk, který je široce využíván pro vývoj webových stránek, aplikací a softwaru, stejně jako pro analýzu dat. Jeho údržba je nízko nákladová. Při vývoji webových aplikací se často používají frameworky, jako například Django nebo Flask. Velké společnosti, jako například Google, Microsoft a Dropbox, také využívají Python pro své projekty [8].

Výhody:

- Jednoduché na naučení.
- Nezávislá na platformě.

Nevýhody:

- Horší výkon než u ostatních již zmíněných programovacích jazyků.

4.1.8 Ruby

Ruby je interpretovaný skriptovací jazyk s plnou podporou objektově orientovaného programování. Je často používán pro vývoj webových aplikací, přičemž se využívá architektura Model-View-Controller (MVC). Ruby se osvědčil jako vhodný pro vytváření systémů pro správu obsahu a tvorbu webových stránek [9].

Výhody:

- Jedná se o interpretovaný jazyk, tudíž změny ve zdrojovém kódu se hned projeví.

- Jednoduché na naučení.

Nevýhody:

- Pomalý výkon.
- Není tak populární.

4.1.9 Node.js

Node.js je open-source platforma pro vývoj server-side aplikací, která je postavena na javascriptovém enginu. Je navržena hlavně pro vývoj webových aplikací a jejím vlastností je, že běží pouze v jednom vlákně, což vyžaduje použití asynchronního programování. Výsledkem je, že zpracování jedné žádosti není závislé na dokončení jiné žádosti. Žádosti jsou ukládány do fronty First in first out (FIFO) a Node.js funguje na událostmi řízeném (event-driven) modelu.

Výhody:

- Jednoduché k naučení jazyku Javascript.
- Lehký přechod mezi frontendem a backendem.
- Menší nároky na výkon.

Nevýhody:

- Nevhodné pro náročné aplikace.
- Nestabilní Application Programming Interface (API) - API se často změní a není zpětně kompatibilní.

4.2 Databáze

Databáze jsou nezbytnou součástí webových aplikací a slouží k ukládání dat. Je důležité si udělat analýzu databázových systémů, neboť každý typ databáze má své výhody a nevýhody. Databázové systémy lze rozdělit na relační a nerelační. V této kapitole se budeme věnovat srovnání těchto dvou typů databází a budeme také popisovat konkrétní vybrané databáze.

4.2.1 Relační databáze

Relační databáze jsou založené na tabulkách, které se skládají ze sloupců a řádků. Sloupce představují atributy, zatímco řádky obsahují záznamy. Důležitým atributem je primární klíč, který je unikátní, nenulový a neměnný. V relačních databázích se využívá relační algebra a normalizačních pravidel pro správu a zpracování dat.

Výhody:

- Snadné v tabulce filtrovat záznamy.
- Umožňuje relace mezi tabulkami.
- Data jsou snadno udržovatelná.

Nevýhody:

- Pomalejší výpis dat než u NoSQL databází.
- Pomalejší ukládání dat.

Mezi SQL databázové systémy patří:

- PostgreSQL
  - PostgreSQL je objektově-relační databázový systém, který se zaměřuje na rozšiřitelnost a dodržování standardů. Umožňuje definici datových typů a je vysoce ceněn pro svou spolehlivost, bezpečnost a vysoký stupeň konformity s Structured Query Language (SQL) standardem
- MySQL
  - MySQL je relační databázový systém, který používá tabulky jako základní komponentu. Je to nejpopulárnější databáze, kterou v roce 2019 používalo 39% vývojářů [10]. Je ceněn pro svoji rychlost, jednoduchost a schopnost snadno integrovat s mnoha webovými aplikacemi.

4.2.2 Nerelační databáze

Nerelační databáze, také známé jako NoSQL databáze, neukládají data v tabulkách, ale používají flexibilní datový model, který umožňuje kombinovat různé struktury dat bez nutnosti měnit schéma databáze. Existují dva hlavní typy nerelačních databází: klíč-hodnota a dokument-orientované.
Klíč-hodnota přiřazuje každé hodnotě unikátní klíč, který může být číslo, řetězec nebo další klíč-hodnota pár.
Dokument-orientované databáze ukládají data v dokumentech ve formátu JavaScript Object Notation (JSON), binární JSON (BSON) nebo Extensible Markup Language (XML). Každý dokument obsahuje klíč a hodnotu, podobně jako u klíč-hodnota typu.

Výhody:

- Rychlejší zápis a výpis dat.
- Horizontální škálování.
- Možnost přidání atributu bez zásahu do tabulky.

Nevýhody:

- Velká redundance dat.

Mezi NoSQL databázové systémy patří:

- Redis
  - Redis (Remote Dictionary Server) ukládá data ve formátu klíč-hodnota. Jeho hlavní výhody jsou vysoká rychlost a schopnost pracovat s vysokými objemy dat. Redis podporuje mnoho datových typů, včetně řetězců, seznamů, množin a hash tabulek.
- MongoDB
  - Ukládá data ve formě dokumentů, podobně jako JSON soubory. Tyto dokumenty jsou uloženy v kolekcích a obsahují klíč-hodnota páry. MongoDB podporuje širokou škálu datových typů, včetně binárních dat, a umožňuje rychle indexovat a vyhledávat data. Je také schopný pracovat s velkými objemy dat a podporuje replikaci a škálování na více serverů.
- ElasticSearch
  - Elasticsearch je vyhledávací a analytický nástroj, který ukládá data ve formátu JSON v dokumentech. Jeho hlavní předností je rychlá indexace a fulltextové vyhledávání.
  - Umožňuje rychle vyhledávat a analyzovat velké množství dat a je schopný pracovat s různorodými datovými typy.

4.3 Frontend

Frontend je oproti backendu viditelný pro koncového uživatele. Jde o technologii, která poskytuje uživatelům rozhraní pro interakci s aplikací. Základní technologie používané v rámci frontendu jsou Hyper Text Markup Language (HTML), Cascading Style Sheets (CSS) a JavaScript. V případě větších projektů se v poslední době stále častěji využívají frameworky, které zjednodušují a urychlují vývoj aplikace. Cílem této kapitoly bude analyzovat vybrané frameworky a popsat jejich vlastnosti, výhody a nevýhody.

4.3.1 Angular

Angular je open-source frontendový framework vyvinutý společností Google pro tvorbu webových aplikací. Používá programovací jazyk TypeScript, který je založený na JavaScriptu. Pomáhá vytvářet interaktivní single-page aplikace díky svým funkcím, jako jsou šablony, modularizace a zpracování rozhraní REST API [11].

Výhody:

- Angular je multiplatformní.
- Využívá MVC architekturu.

Nevýhody:

- Omezené možnosti Search engine optimization (SEO).
- Složitější na naučení než ostatní frameworky.

4.3.2 React

React, také známý jako framework React, byl vyvinut společností Facebook. Je určen pro vývoj webových aplikací a je navržen tak, aby pomáhal vývojářům vytvářet rychlé single-page aplikace a uživatelská rozhraní. Je založen na konceptu komponent a umožňuje vývojářům snadno přidávat nebo odebírat funkce v aplikaci bez nutnosti přepisovat celý kód [12].

Výhody:

- Opakovaně použitelné komponenty.
- Zpětná kompatibilita.
- Silná komunita.
- Jednoduší na naučení než již zmíněný Angular 4.3.1.

Nevýhody:

- Nedostatečná dokumentace.
- Problémy se Search engine optimization (SEO).

4.3.3 Vue

Vue.js je open-source JavaScriptový framework pro vývoj uživatelských rozhraní a single-page aplikací. Byl vyvinut v roce 2016 jedním z vývojářů Angular. Je známý svou krátkou křivkou učení, kterou umožňuje použití rozhraní Option API. To umožňuje začátečníkům rychle se naučit framework a začít s ním pracovat. Vue.js je navržen tak, aby byl snadno použitelný a mohl být integrován do stávajících projektů. Je také lehký a rychlý, což umožňuje rychle načítat stránky a zlepšuje uživatelský zážitek.
Výhody:

- Jednoduché na naučení.
- Rychlé vykreslování.
- Paměťove nenáročné a výkonné.

Nevýhody:

- Nemá tak velký ekosystém jako React.
- Není multiplatformní.

4.3.4 Next

Next.js je open-source JavaScriptový framework, který umožňuje vytvářet rychlé a uživatelsky přívětivé statické webové stránky a webové aplikace pomocí Reactu. Jeho hlavní výhodou je schopnost vytvářet hybridní aplikace, které obsahují jak stránky vykreslované na straně serveru, tak staticky generované stránky. To umožňuje vývojářům využít výhody obou přístupů a vytvářet aplikace s rychlým načítáním a dobrou SEO optimalizací. Next.js také poskytuje několik nástrojů pro automatizaci procesu vývoje a deployování aplikací, což usnadňuje a urychluje celý proces [13].

Výhody:

- Vyšší výkon.
- Rychlý vývoj.
- Zlepšená SEO.

4.4 Autentizace

Pro webové aplikace, které jsou určeny pro určitou skupinu uživatelů, je velmi důležité zabezpečit autentizaci uživatelů pro přístup do aplikace. Existuje mnoho způsobů, jak autentizovat uživatele, a v této kapitole se budeme zabývat těmi nejvyužívanějšími.

4.4.1 Basic Auth

Přihlašování pomocí uživatelského jména a hesla je historicky nejstarším způsobem autentizace. Tento způsob vyžaduje, aby uživatel zadal své přihlašovací údaje, které jsou pak zakódované pomocí base64 v hlavičce požadavku na úrovni Hypertext Transfer Protocol (HTTP) protokolu a poslány na server pro ověření.

Výhody:

- Rychlé na implementaci.

Nevýhody:

- V každém kroku požadavku se odešle kompletní ověření včetně hesla.

4.4.2 JSON Web Token (JWT) / Bearer token

JSON Web Token je otevřeným standardem (RFC 7519), který se používá pro autentizaci uživatelů. JWT je digitálně podepsaný token, který může být využíván uživateli aplikace i neuživateli. Server používá token k ověření identity uživatele a získání informací o něm z payloadu tokenu a porovnání s databází. JWT je často používán pro Single sign-on (SSO), což umožňuje uživateli přihlásit se pouze jednou a pak přistupovat k více aplikacím bez nutnosti opakovaného přihlašování. JWT je také často používán k autentizaci uživatelů při přístupu k API.

Výhody:

- Uživatel po ověření získá podepsaný token, který neobsahuje informace o hesle.

4.4.3 OAuth 2.0

OAuth 2.0 není typickým způsobem autentizace, ale mechanismem pro autorizaci aplikací třetích stran k přístupu k uživatelským prostředkům. Autorizační server vydává přístupový token, který je náhodným řetězcem a je také uložen v databázi.
Výhody:

- Uživatelé se budou moci přihlásit pomocí Google účtu nebo i Facebook účtu.
- Přijatý token bude mít autentizaci i autorizaci.

Nevýhody:

- Složitější na implementaci.

4.5 Cloudové úložiště

Pro ukládání nestrukturovaných dat, jako jsou dokumenty, média, analytická data a další, se používají objektová úložiště. Dvě konkurenční platformy, které se často používají pro tento účel, jsou AWS S3 a GCS.

4.5.1 Amazon Simple Storage Service

Amazon Simple Storage Service (S3) je služba pro ukládání dat poskytovaná společností Amazon. Služba je určena převážně pro ukládání nestrukturovaných dat a umožňuje ukládat data jako objekty v rámci bucketů. Amazon S3 poskytuje řadu funkcí, jako je přidávání metadatových značek objektům, nastavení přístupu a sdílení dat pro ostatní uživatele, monitorování trendů aktivity a využití v reálném čase. Služba AWS S3 nabízí také bezplatnou verzi, která je omezená na 5 GB úložiště, 20000 GET requestů, 2000 POST, PUT, COPY requestů a 15 GB přenosu dat každý měsíc.

Výhody:

- Vysoce škálovatelná platforma.
- Široká možnost integrace.

Nevýhody:

- Vyšší cenový žebřík.

4.5.2 Google Cloud Storage

Google Cloud Storage (GCS) je platformou pro ukládání dat na úrovni objektů poskytovanou společností Google. Podobně jako Amazon S3 ukládá data jako objekty v rámci bucketů, které jsou kontejnery v cloudu, které lze přiřadit k různým třídám úložiště. Bezplatná verze GCS poskytuje 15 GB úložiště pro ukládání dat. GCS poskytuje také škálovatelné a vysokoodolné úložiště pro ukládání velkého množství dat a umožňuje snadný přístup k datům pomocí webového rozhraní nebo API. GCS navíc nabízí integraci s ostatními službami Google, jako je například Google BigQuery pro zpracování velkých objemů dat.

Výhody:

- Poskytuje rychlé vyhledávání dat.
- Snadné a flexibilní použití.
- Integrace s Google službami jako je Google Drive.

Nevýhody:

- Neposkytuje tolik funkcí jako u AWS S3.
- Backend API nelze integrovast s dalšími cloud provideři.

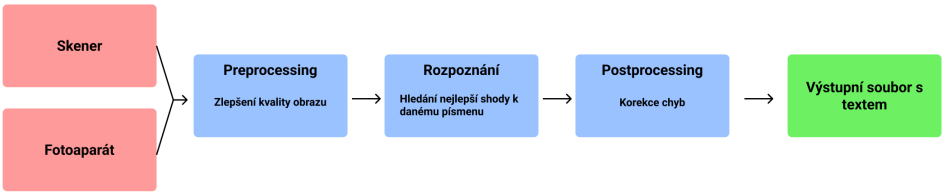
4.6 Optical Character Recognition

Optical Character Recognition (OCR) je technologie, která umožňuje počítačům rozpoznávat a extrahovat text ze skenovaných nebo fotografovaných obrazů. OCR dokáže rozpoznat nejen text napsaný pomocí počítače, ale také ručně psaný text. Tento text je poté převeden do počítačové podoby, ve které jej lze nadále zpracovávat a využívat. OCR je velmi užitečná technologie, která umožňuje automatizovat procesy zpracovávání a ukládání dokumentů, a tím šetřit čas a náklady.

Idea optického rozpoznávání znaků se začala zaznamenávat již v roce 1914, kdy Emanuel Goldberg vynalezl stroj, který dokázal přečíst znaky a převést je na standardní telegrafický kód. Současně byl vyvinut i Optophone, ruční skener, který při přejetí nad znakem vytvářel tóny. V roce 1984 byl vyvinut první pásový skener společnosti Caere Corporation pro ministerstvo zahraničí v USA. O tři roky později se OCR již používalo k čtení cenovek.

OCR systémy se skládají většinou z několika komponent 4.1, kterým prochází vyčtený znak z obrazu. Jedná se o komponenty optické, lokalizace, odstranění členitosti, defragmentace, preprocessing, extrakce vzhledu a rozpoznávání. Je důležité si definovat obraz po nahráním do OCR. Obraz je tvořen pixely, každý pixel reprezentuje určitou hodnotu, například 8bitové pixely mají hodnoty v rozmezí 0 až 255. Pixely jsou na obrazu rozmístěny do mřížky zvané bitmapa neboli rastr, z něhož vyplývá, že každý pixel má své souřadnice v mřížce.

Obvykle se skládá z několika komponent, které procházejí vyčtený znak z obrazu. Tyto komponenty zahrnují optickou část, lokalizaci, odstranění členitosti, defragmentaci, preprocessing, extrakci vzhledu a rozpoznávání. Je důležité definovat obraz po jeho nahrání do OCR. Obraz se skládá z pixelů, přičemž každý pixel představuje určitou hodnotu. Například 8bitové pixely mají hodnoty v rozmezí 0 až 255. Pixely jsou na obrazu rozmístěny do mřížky zvané bitmapa nebo rastr, což znamená, že každý pixel má své souřadnice v této mřížce.



Obrázek 4.1: Průběh zpracování OCR

První fáze je předzpracování obrazu, jedná se o moduly optické a lokalizace. Tato fáze dokáže otočit dokument, pokud je to nutné, odstranit skvrny a šum. Součástí první fáze je i binarizace, což je převedení obrazu na 2 bitová data, tedy černobílý obraz. Systém pak ještě identifikuje text v obrazu a oddělí je od různých logických objektů (grafika, sloupce, linky) a následně rozdělí text

do jednotlivých znaků.

Druhá fáze je rozpoznávání znaků, jedna z nejtěžších fází v OCR. Existuje řada metod, jak tyto znaky rozpoznat, ale v této kapitole se budeme zabývat dvěma nejznámějšími metodami, a to soft computingem a převodem do maticového objektu. Nejprve se zaměříme na metodu soft computing, která je považována za nejúčinnější metodu v OCR systémech. Algoritmus této metody se skládá z několika částí. Prvním krokem je vytvoření modelu. Následuje získání dostatečného množství trénovacích dat a trénink modelu, při kterém se model učí kvalifikovat nová data. Druhou metodou rozpoznávání znaků je převod znaku do maticového objektu. Tato metoda spočívá v převodu pixelů, které jsou černé, na hodnotu 1. Algoritmus poté porovnává matici s databází matic reprezentujících znaky s různými písmy, aby zjistil, se kterým znakem má nejvíce společného [14].

Třetí fáze rozpoznávání textu se zaměřuje na filtrování slov, která se nenašla v daném slovníku nebo databázi. Je to finální fáze, jejímž cílem je dosáhnout co nejlepších výsledků. Proto je důležité zvolit správný model pro konkrétní jazyk. To může způsobovat problémy, pokud dokument obsahuje slova v jiném jazyce. Znalost gramatiky také může zlepšit výsledek, například pomocí určení, zda se jedná o podstatné jméno, přídavné jméno, sloveso atd.

Existuje mnoho OCR nástrojů, jak open-source, tak i online. Online nástroje fungují na principu, že uživatel nahraje dokument a nástroj vytěží text a umožní uživateli stáhnout textový dokument. Open-source řešení nám naopak umožňuje nahlédnout do zdrojového kódu a případně jej upravit. V této kapitole budou vybrány dva nástroje na analýzu a cílem je ukázat, který z nich bude vhodnější pro implementaci této práce. Prvním nástrojem bude Google Vision, druhým pak Amazon Textract.

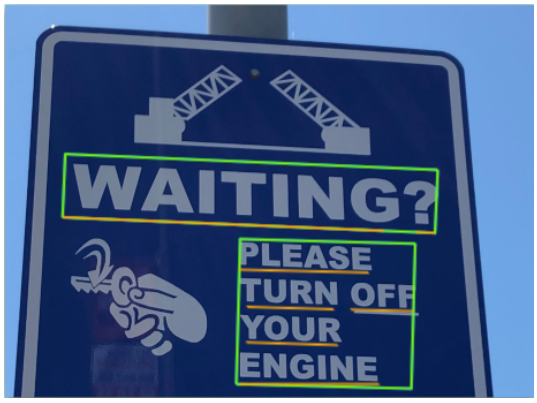
4.6.1 Google Vision API

Mezi existující OCR nástroje patří také Google Vision. Tento nástroj nabízí výkonné přetrénované modely prostřednictvím REST API. Google Vision umí rozpoznat objekty, tváře, strojově psaný text i ručně psaný text. Obsahuje dvě metody, které pomáhají při rozpoznávání textu: TEXT-DETECTION a DOCUMENT-TEXT-DETECTION.

TEXT-DETECTION je funkce, která detekuje a rozpoznává text z obrazu. Například, pokud se použije na obrázek, výstupem bude JSON obsahující celý vytěžený řetězec textu a jednotlivá slova včetně jejich ohraničení. 4.2

DOCUMENT-TEXT-DETECTION je metoda, která se nepracuje s obrázky, ale s dokumenty. Jeho zpracování trvá déle než u předchozí metody TEXT-DETECTION, ale výstupem je JSON obsahující informace o bloku, odstavci a jednotlivých řetězcích textu.

- Google Vision API umožňuje i offline asynchronní zpracování balíčku až 2000 obrázků
- Google Vision nabízí také zdarma nástroj, avšak s omezením na 1000 zpracování za měsíc



Obrázek 4.2: Ukázka Google Vision OCR
Zdroj: `www.cloud.google.com/vision/docs/ocr`

Výhody:

- Zaručená stabilita od Googlu.
- Podpora mnoho formátů obrázků.
- Bohatá dokumentace.
- Poskytuje bezplatnou verzi.

Nevýhody:

- Není open-source, pouze prostřednictvím API.

4.6.2 Amazon Textract

- Amazon Textract je nástroj, který dokáže extrahovat text, rukopis a data z naskenovaných dokumentů. Podporuje obrázky formátu .jpg a .png [15].
- Nabízí bezplatnou verzi po dobu 12 měsíců, přičemž free tier obsahuje až 5000 zpracování za měsíc.

Výhody:

- Zaručená stabilita od Amazonu.
- Poskytuje bezplatnou verzi.

Nevýhody:

- Podporuje pouze .jpg a .png.
- Google Vision 4.6.1 poskytuje bohatší integraci.

Kapitola 5

Analýza architektonických vzorů

Následující kapitola se bude zabývat architekturou aplikace. Jejím hlavním cílem architektury je vytvořit strukturu programu a zjednodušit nejen vývoj aplikace, ale také její budoucí údržbu. Tímto způsobem lze dosáhnout vysoké úrovně škálovatelnosti architektury, což umožňuje provádět změny v aplikaci na konkrétních modulech. Toto přispívá ke zvýšení efektivity a snižuje náročnost na čas a zdroje při údržbě aplikace.

Je třeba zmínit, že existuje mnoho softwarových architektonických vzorů, z nichž autor práce vybral pro další studium těchto čtyři:

- Layered architecture
- Event drive architecture
- Microservice architecture
- Monolithic architecture

5.1 Layered architecture

Layered architecture, také možné najít pod názvem "Vícevrstvá architektura", je architektura, která rozděluje aplikaci do více vrstev, přičemž každá vrstva plní určitou funkci. Tato architektura umožňuje jasné oddělení různých logických celků a ulehčuje tak vývoj a údržbu aplikace. Vícevrstvá architektura je často používána v enterprise aplikacích, kde je nutné zajistit vysokou míru škálovatelnosti a rozšiřitelnosti.

Architektura rozděluje aplikaci do následujících vrstev:

- Prezentační vrstva
  - Obsahuje uživatelské rozhraní. Jejím úkolem je zajistit dobré uživatelské prostředí.
- Business vrstva
  - Jak název napovídá, tato vrstva obsahuje logiku aplikace. Jedná se o vrstvu, která odděluje uživatelské rozhraní od výpočetní části aplikace. Tento princip nám umožňuje nezávisle měnit logiku aplikace bez vlivu na ostatní vrstvy.

- Vrstva datového propojení
  - Jejím úkolem je zajistit persistenci dat v aplikaci.

Výhody:

- Snadnější implementace.
- Lehčí údržba aplikace díky volnému propojení.
- Poskytuje abstrakci odpovědností prostřednictvím oddělení mezi jednotlivými vrstvami.

Nevýhody:

- Malá škálovatelnost.
- Data musí projít každou vrstvou, i když se nemusí přenášet z konkrétních vrstev.

5.2 Event driven architecture

Architektura řízená událostmi funguje na principu posílání dat do centrální jednotky, která zpracuje data a odesílá je jako událost ke konkrétní službě zpracování. Tato služba pak může s událostí pracovat nebo z ní získat potřebné informace a odeslat ji dále.

Výhody:

- Výstup v reálném čase.
- Snadná škálovatelnost.
- Odolnost proti poruchám a vysoká dostupnost.

Nevýhody:

- Zvýšená složitost pro poskytovatele API.
- Nejednoznačnost analýzy API.

5.3 Microservice architecture

Jak název napovídá, tato aplikace místo jednoho velkého programu obsahuje více mikroslužeb. Architektura funguje na principu komponentizace jednotlivých služeb, přičemž každá z nich má jinou zodpovědnost. To je důležité, protože změny v jedné službě by neměly mít vliv na ostatní služby.

Výhody:

- Vysoká modularita.
- Snadná úprava jedné služby.
- Selhání jedné služby neovlivní celou aplikaci.

Nevýhody:

- Riziko poruchy nebo selhání komunikace mezi službami.
- Delší doba implementace.
- Aplikace s menší složitostí bude obtížnější rozdělit na mikroslužby.
- Komplexnější nasazení na produkci.

5.4 Monolithic architecture

Jedná se zřejmě o nejjednodušší způsob implementace. Je vhodné pro startupy a malé týmy vývojářů. Na rozdíl od mikroslužbové architektury jsou všechny části programu propojené.

Výhody:

- Snadný vývoj.
- Snadný nasazení aplikace.

Nevýhody:

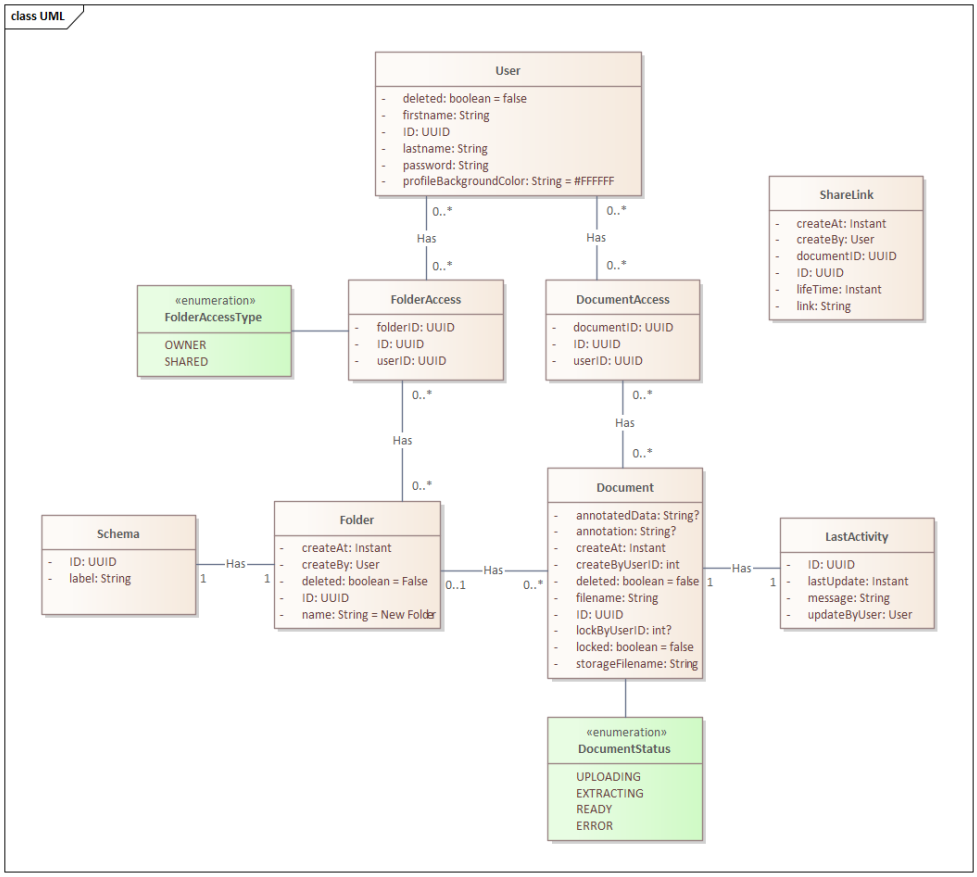
- Obtížnější na modernizaci pokud se jedná a komplexnější projekt.
- Omezená flexibilita.

Kapitola 6

Návrh

6.1 Doménový model

Doménový model obsahuje 8 entit a 2 enumerační typy 6.1. Doménový model byl vytvořen na základě funkčních požadavků 2 a analýzy případu užití 3. Níže v kapitole budou podrobně popsány jednotlivé enumerační typy a entity, včetně jejich atributů a vztahů mezi ostatními entitami.



Obrázek 6.1: Doménový model

User

- Entita User prezentuje uživatele, mezi jeho atributy patří:
  - `id`: Identifikační číslo uživatele, které je unikátní
  - `firstname`: Křestní jméno uživatele
  - `lastname`: Přijímení uživatele
  - `password`: Heslo uživatele, který bude uložen zahashován
  - `deleted`: Atribut deleted označuje zda daný účet je smazan nebo ne, v defaultním stavu je hodnota nastavena na False
  - `profileBackgroundColor`: Barva na pozadí profilu, výchozí barva je nastavena na #FFFFFF
- K entitě User je vázána tabulka **FolderAccess** 6.1 a **DocumentAccess** 6.1, kde uživatel může mít několik přístupů jak do složky, tak k dokumentu.

Folder

- Entita Folder obsahuje tyto atributy:
  - `id`: Identifikační unikátní číslo
  - `name`: Název složky
  - `createAt`: Čas vytovření složky
  - `createBy`: Jakým uživatelem byl vytvořen
  - `deleted`: Označuje zda složka je smazána, defaultní hodnota je nastavena na False
- K Folderu se váže entita **Schema** 6.1, kde každá složka má právě pouze jedno schéma.

Schema

- Entita schema jak z názvu už vypovídá reprezentuje schéma. Mezi jeho atributy patří:
  - `id`: Unikátní identifikační číslo
  - `label`: pole názvů jednotlivých labelů
- Ve vztahu mezi entitou Schema patří již zmíněný **Folder** 6.1, kde jedno schéma patří k jednomu folderu.

Document

- Entita Document obsahuje tyto atirbuty:
  - `id`: Identifikační unikátní číslo dokumentu
  - `filename`: Název dokumentu
  - `storageFilename`: Název dokumentu v cloud úložišti
  - `createAt`: Čas, kdy byl dokument vytvořen
  - `createBy`: Kým byl dokument vytvořen
  - `annotation`: Annotation jsou vytěžená data z dokumentu pomocí OCR
  - `annotatedData`: Data, která byla označena uživatelem
  - `deleted`: Defaultně nastavený na False, označuje zda dokument není smazán
  - `locked`: Označuje zda je v dokumentu jiný uživatel, výchozí hodnota je False
  - `lockByUserID`: Kým byl dokument uzamčen
- Entita Document je vázána na enum **DocumentStatus** 6.1, kde DocumentStatus popisuje status dokumentu. Je dále vázána na **Folder** 6.1, **DocumentAccess** 6.1 a **LastActivity** 6.1.
- Dokument buď patří do nějaké složky nebo může být samostatná. Dokument může mít několik DokumentAccessu neboli k dokumentu může mít přístup několik uživatelů. Každý dokument má dále provázanost právě na jednu LastActivityEntitu.

DocumentStatus

- Enum DocumentaStatus reprezentuje stavy dokumentu, dokument se může nacházet ve stavu:
  - `UPLOADING`: Dokument se nahrává do apikace a do cloudového úložiště
  - `EXTRACTING`: Z dokumentu se vytěžujou data pomocí OCR
  - `READY`: Dokument byl úspěšně nahrán, uložen a vytěžen
  - `ERROR`: Dokument se z určitých problémů nemohl nahrát, uložit nebo vytěžit

LastActivity

- LastActivity entita představuje poslední aktivitu dokumentu, tedy je přímo vázána na konkrétní dokument. Mezi atributy patří:
  - `id`: Identifikační číslo
  - `lastUpdate`: Čas poslední změny

- `message`: Změna, která byla provedena
- `updateByUser`: Jakým uživatelem byla změna provedena

FolderAccess

- Entita FolderAccess reprezentuje přístup do složky.
- Uživatel, který má folderAccess dané složky, bude mít do složky přístup a tím i přístup do všech dokumentů ve složce.
- K entitě se váže i enum FolderAccessType, která označuje konkrétní typ přístupu.
  - `id`: Identifikační číslo

FolderAccessType

- Enum, který má typy:
  - `OWNER`: Uživatel, který složku vytvořil bude mít FolderAccessType OWNER
  - `SHARED`: Uživateli, kterému byl sdílena složka bude mít přístup typu SHARED, uživateli se pak sdílené složky zobrazí v pracovišti "Sdílené složky"

DocumentAccess

- DocumentAccess podobně jako FolderAccess je přístup k dokumentu. Slouží pro uživatelé, které nemají přístup do složky a dokument jim byl pouze sdílen.
- Uživatel kterému byl sdílen dokument se dokument zobrazí v pracovišti "Sdílené soubory".
  - `id`: Identifikační číslo

ShareLink

- Entita ShareLink představuje sdílený přístup přes odkaz. Mezi atributy patří:
  - `id`: Identifikační číslo
  - `link`: Odkaz na dokument

- `createAt`: Čas, kdy byl odkaz vytvořen
- `lifetime`: Doba platnosti odkazu
- `documentUUID`: UUID dokumentu, ke kterému je vytovřen odkaz
- `createBy`: Kým byl odkaz vytvořen

6.2 Architektura

Tato aplikace využívá vícevrstvé architektury, která se hodně využívá pro vývoj aplikací v oblasti enterprise, jelikož umožňuje oddělení funkčnosti aplikace do různých vrstev. Tyto vrstvy se obvykle dělí na klientskou, logickou a databázovou vrstvu.

- Klientská vrstva má za úkol zpracovat uživatelské interakce a prezentovat data uživateli. V aplikaci bude klientská vrstva implementována pomocí jazyka JavaScript s frameworkem Next.js.
- Logická vrstva, také nazývaná businessová vrstva, bude napsána v jazyce Java s využitím frameworku Spring Boot. Tato vrstva se stará o zpracování dat, výpočty a implementaci aplikační logiky. V této vrstvě bude probíhat většina zpracování dat a bude také zajištěna komunikace s databázovou vrstvou.
- Databázová vrstva se stará o ukládání dat a jejich zpřístupnění aplikaci. Pro tuto vrstvu bude využita open source databáze PostgreSQL a Elastic Database.

Komunikace mezi logickou vrstvou a klientskou vrstvou bude zajištěna pomocí GraphQL rozhraní, které umožňuje efektivní a flexibilní komunikaci mezi vrstvami aplikace.

Využití vícevrstvé architektury je výhodné pro oddělení jednotlivých funkcionalit aplikace do samostatných vrstev. Tento přístup umožňuje snadnější údržbu kódu, rychlejší vývoj nových funkcionalit a usnadňuje testování aplikace.

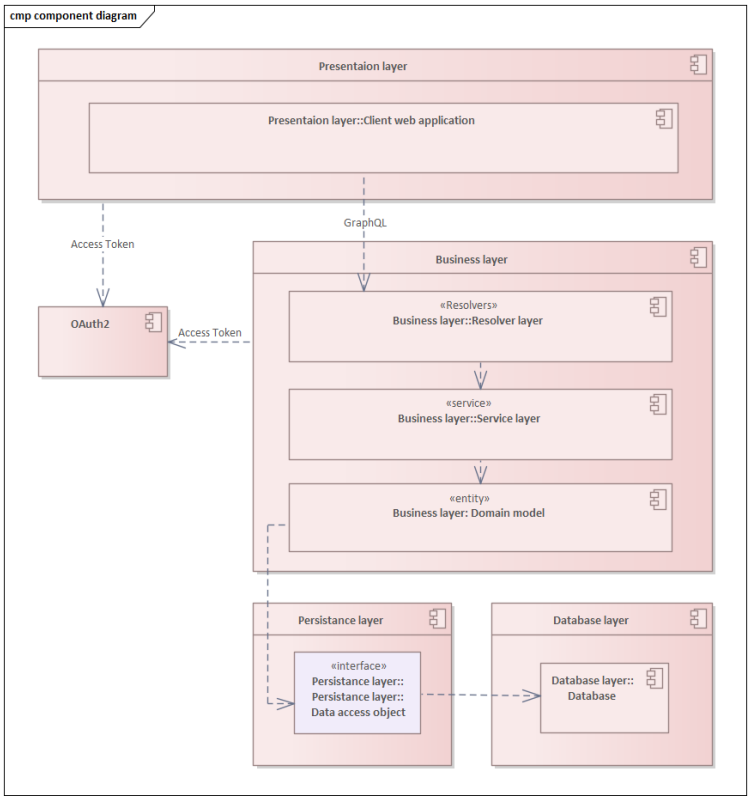
Diagram komponent 6.2 popisuje architekturu softwarového systému a zahrnuje několik vrstev. V první vrstvě, nazvané prezentní vrstva, se nachází webový klient, který komunikuje s uživatelem a zobrazuje mu výstup na obrazovce. Webový klient může být realizován pomocí webové aplikace nebo mobilní aplikace.

Další vrstvou je business vrstva, která se skládá z několika komponent. Resolver vrstva přijímá požadavky od webového klienta a přeposílá je business vrstvě. Service vrstva poskytuje funkčnosti a aplikační logiku, zpracovává požadavky a vrací výsledky zpět do resolver vrstvy. Domain model vrstva obsahuje objekty, které reprezentují datové entity v systému 6.1.

Komunikace mezi prezentní vrstvou a business vrstvou je realizována pomocí dot GraphQL.

Perzistentní vrstva zajišťuje ukládání a načítání dat ze systému. Obsahuje Data Access Object (DAO) vrstvu, která představuje datový přístupový objekt, a databázovou vrstvu, která obsahuje DB komponentu, která se připojuje k databázi.

V rámci systému se také nachází cross cutting komponenta OAuth2, která je zodpovědná za autentizaci uživatele. Diagram komponent popisuje celkovou architekturu systému a umožňuje snadné porozumění jednotlivým komponentám a vztahům mezi nimi. Také ukazuje, jak jsou jednotlivé vrstvy propojené a jakým způsobem probíhá komunikace mezi nimi.



Obrázek 6.2: Component diagram

6.3 Návrh uživatelského rozhraní

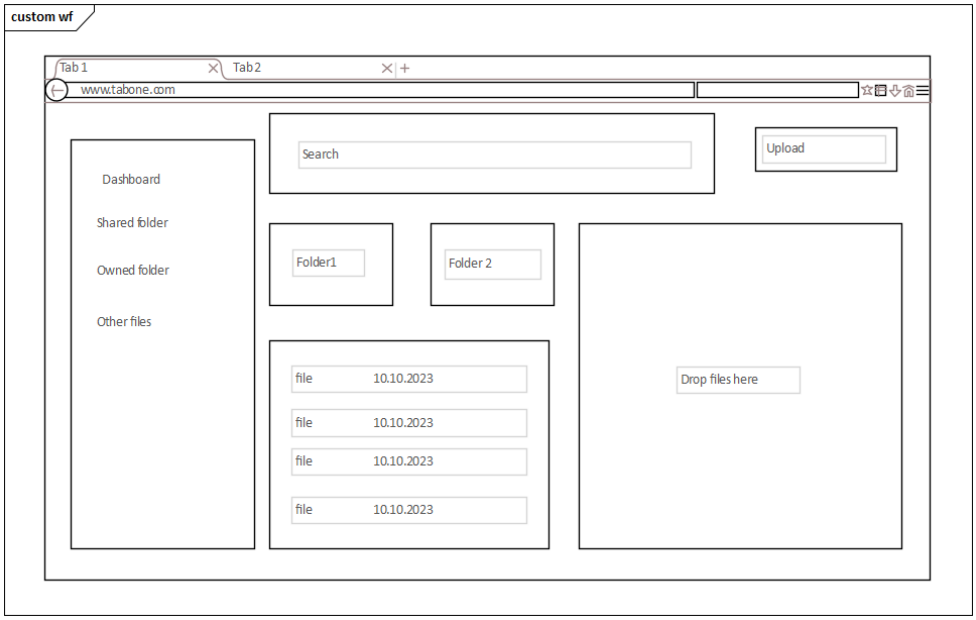
Před vývojem aplikace je nezbytné promyslet si vzhled grafického uživatelského rozhraní. Je důležité určit vhodné rozložení stránek, ovládacích prvků, vhodné barvy a font písma, aby bylo uživatelské rozhraní co nejpřívětivější. Předtím, než se začne pracovat na designu, se většinou vytvářejí wireframy, které představují koncept a určují základní rozvržení jednotlivých stránek. Na základě wireframů se poté vytváří finální design uživatelského rozhraní. V této kapitole budou představeny wireframy a na nich postavený výsledný prototyp.

Celkový design aplikace je založen na principu Flat Design a využívá temný režim. Flat Design je designový přístup, který se zaměřuje na minimalismus. Upřednostňuje plochý vzhled s jednoduchými a srozumitelnými ikonami a prvky, zatímco temný režim zajišťuje pohodlné uživatelské prostředí v různých podmínkách osvětlení.

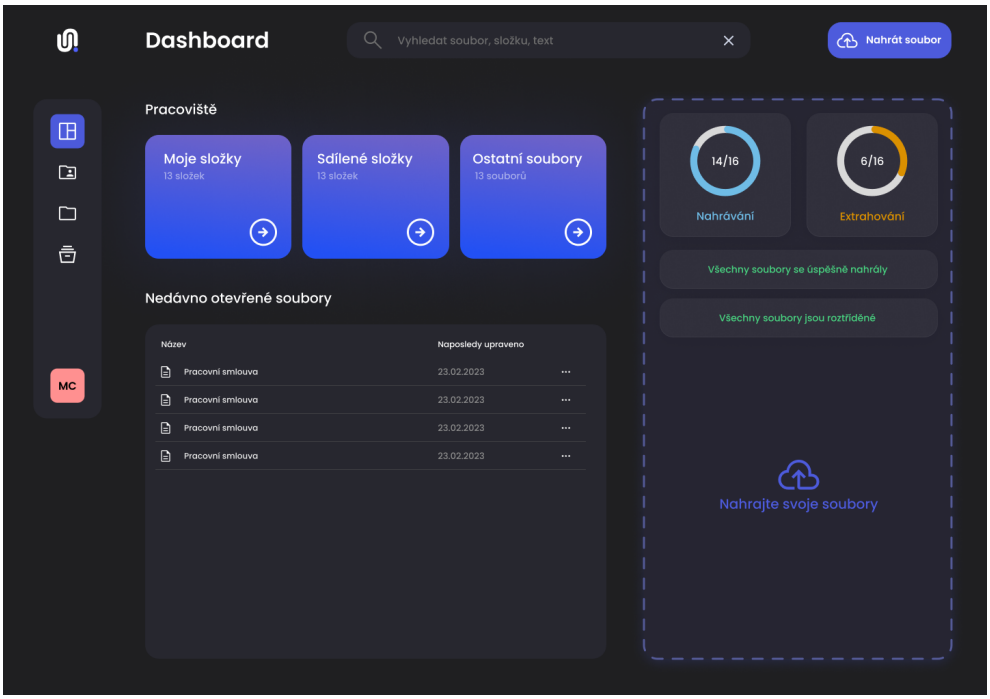
Obrazovka dashboardu 6.4 a celkově celá aplikace je rozdělena na dvě hlavní části - levou a pravou. Pravá část má statický charakter, zatímco levá část slouží jako informativní a hlavní oblast. V pravé části dashboardu je umístěn statický prvek, který uživateli umožňuje nahrávat dokumenty pomocí techniky drag and drop. Dále vizualizuje počet souborů, které se právě nahrávají, a počet souborů, které se extrahují. Levá část obrazovky obsahuje tři podkategorie, kam se uživatel může prokliknout. Tyto podkategorie slouží jako navigační prvky, které umožňují uživateli přecházet v aplikaci. Součástí dashboardu je také seznam nedávno otevřených souborů, který je uživateli k dispozici. Tento seznam poskytuje rychlý přístup k nedávno používaným dokumentům a usnadňuje navigaci mezi nimi.

Stejně jako u dashboardu je obrazovka zobrazení složek 6.6 také rozdělena na dvě části, kde pravá část obrazovky zobrazuje pouze detail dané vybrané složky a v levé části je seznam složek uživatele. Podobné pravidlo platí i pro zobrazení souborů 6.8

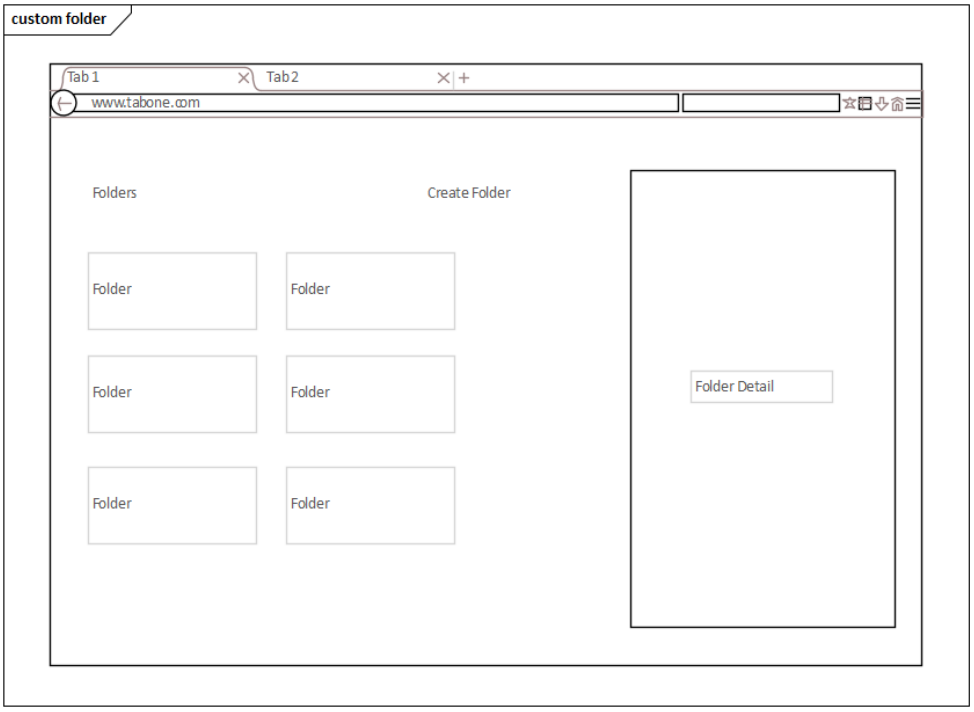
Obrazovka souboru 6.10 je taktéž rozdělena na dvě části, v levé části vidí uživatel detail dokumentu. V levé části je pak zobrazený dokument, kde uživatel může kliknutím myší a posouváním zanotovat text v dokumentu.



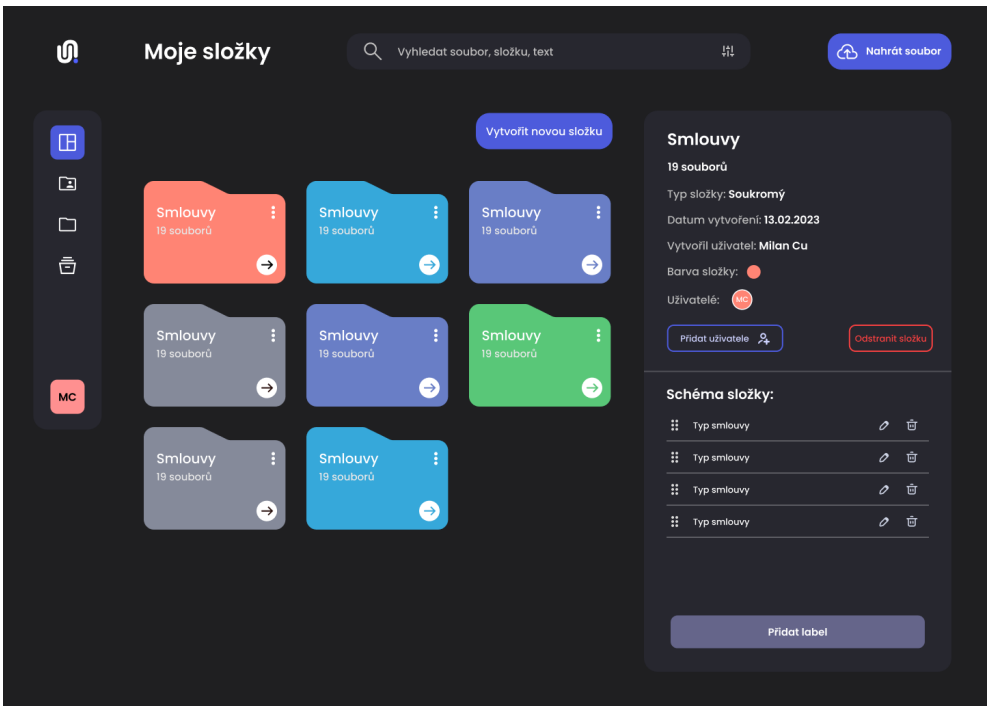
Obrázek 6.3: Wireframe - Dashboard



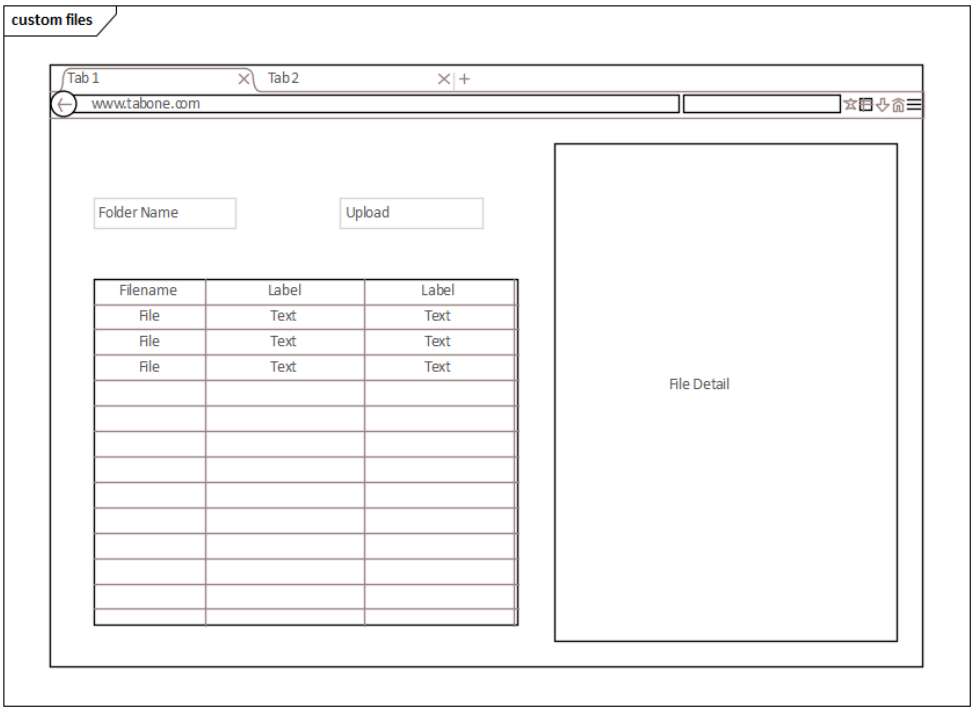
Obrázek 6.4: Prototyp - Dashboard



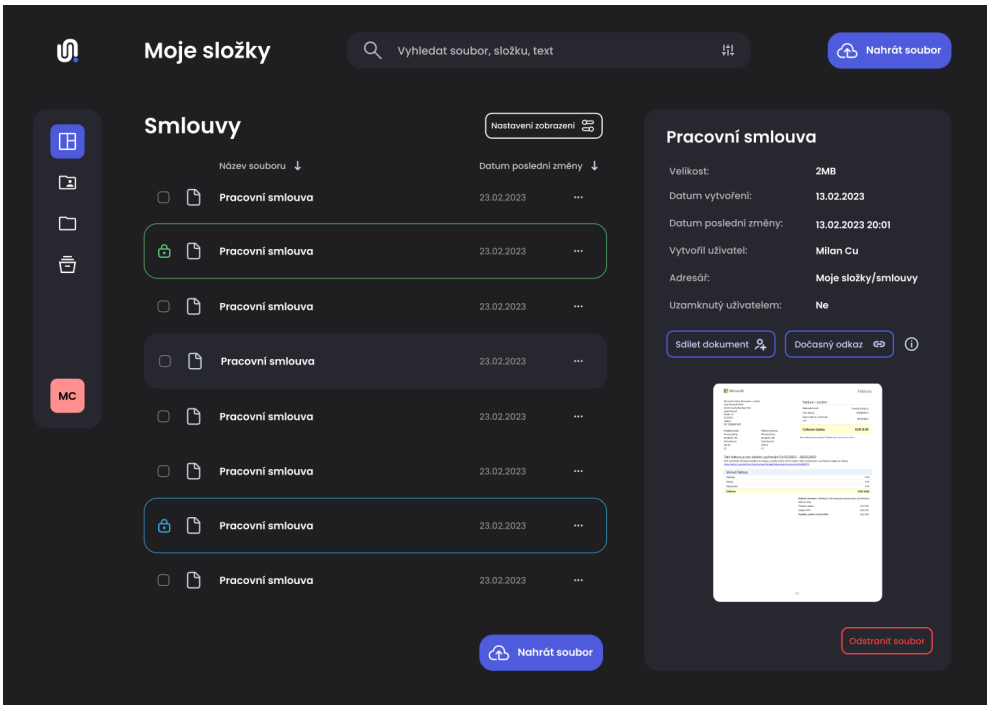
Obrázek 6.5: Wireframe - Zobrazení složek



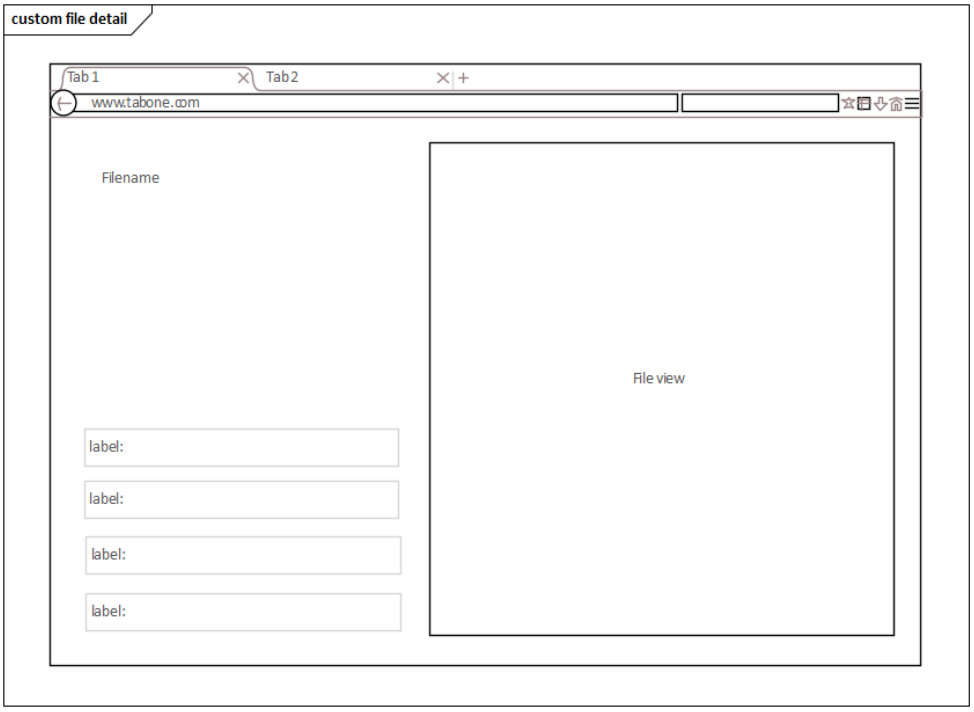
Obrázek 6.6: Prototyp - Zobrazení složek



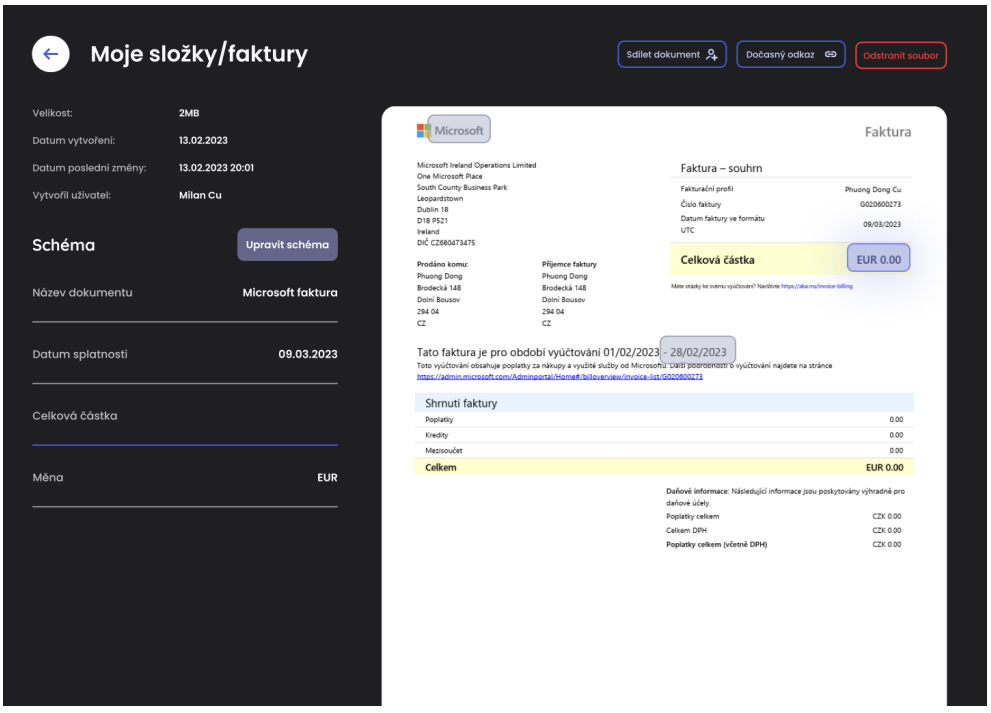
Obrázek 6.7: Wireframe - Zobrazení souborů



Obrázek 6.8: Prototyp - Zobrazení souborů



Obrázek 6.9: Wireframe - Zobrazení souboru



Obrázek 6.10: Prototyp - Zobrazení souboru

Kapitola 7

Implementace

7.1 Backend

Backend aplikace byl vyvíjen v programovacím jazyce Kotlin 4.1.2 a frameworkem Spring Boot. 4.1.3 Do projektu byly dále přidány moduly pro GraphQL, Elasticsearch nebo Google Cloud. V téhle kapotile si rozebereme strukturu backendu aplikace a ukážeme si ukázky kódu.

7.1.1 OCR API

Funkce extractTextAndSave 7.1 je implementací extrakce textu pomocí rozhraní Google Cloud Vision API a následného uložení extrahovaných informací do databáze. Funkce přijímá dva parametry - file a documentId, file je soubor, který uživatel nahrál, documentId je ID dokumentu, který je v databázi. Nejprve funkce optimalizuje obrázek získaný z parametru soubor pomocí metody optimizeImage 7.2. Poté je vytvořena instance třídy Image s použitím bajtových dat obrázku. Poté je vytvořen seznam požadavků, který obsahuje požadavek na rozpoznání textu v obrázku. Instance třídy ImageAnnotatorClient je použita k odeslání všech požadavků a získání extrahovaných textových informací v odpovědi. Pokud se v odpovědi vyskytne chyba, nastaví se stav dokumentu na `DocumentStatus.ERROR`, zaznamená se chybová zpráva a vyhodí se výjimka VisionException. Funkce je asynchronní z důvodu, že se dokument vytěžuje delší dobu a aby uživatel nemusel čekat až se dokument vytěží, bude vytěžování probíhat asynchronně na pozadí.

Pro dosažení vyšší kvality extrakce obrázků je implementována funkce pro optimalizaci 7.2, která provádí rasterizaci a diskretizaci daného obrázku, následně provádí škálování na šířku 850 pixelů.

7.1.2 GraphQL

Pro implementaci GraphQL byla využita knihovna od java-kickstart. Pro využití graphql je nutné na backendu definovat schéma. GraphQL schema je zpravidla vytvářena v serverové části aplikace a je poskytována klientům jako dokumentace API nebo pro frontend vývojářům. Schema je tedy zásadním prvkem při vývoji a dokumentaci GraphQL API.

```
@Async
fun extractTextAndSave(file: ApplicationPart, documentId: UUID) {
    val imgBytes = optimizeImage(file)
    val img: Image = Image.newBuilder().setContent(imgBytes).build()
    val requests: MutableList<AnnotateImageRequest> = ArrayList()
    val feat: Feature = Feature.newBuilder().setType(Feature.Type.TEXT_DETECTION).
     build()
    val request: AnnotateImageRequest = AnnotateImageRequest.newBuilder().
     addFeatures(feat).setImage(img).build()
    val entityAnnotation: MutableList<EntityAnnotation> = ArrayList()
    requests.add(request)

    val document = documentRepository.findById(documentId)

    ImageAnnotatorClient.create().use { client ->
        val response: BatchAnnotateImagesResponse = client.batchAnnotateImages(
     requests)
        val responses: List<AnnotateImageResponse> = response.responsesList
        for (res in responses) {
            if (res.hasError()) {
                document?.documentStatus = DocumentStatus.ERROR
                documentRepository.save(document!!)
                log.error { "Error:␣${res.error.message}" }
                throw VisionException(res.error)
            }
            for (annotation in res.textAnnotationsList) {
                entityAnnotation.add(annotation)
            }
        }
    }

    document?.allTextDescription = entityAnnotation[0].description
    document?.customAnnotations = mapToCustomAnnotation(entityAnnotation)
    document?.documentStatus = DocumentStatus.READY
    documentRepository.save(document!!)
    log.info { "Extracted␣text␣for␣document␣with␣id:␣${document.id},␣Document␣
     status:␣${document.documentStatus}" }
}
```

Obrázek 7.1: Ukázka OCR vytěžování

```
<dependency>
    <groupId>com.graphql-java-kickstart</groupId>
    <artifactId>graphql-spring-boot-starter</artifactId>
    <version>15.0.0</version>
</dependency>
```

Obrázek 7.3: Knihovna pro Graphql

```
@Throws(IOException::class)
fun optimizeImage(file: ApplicationPart): ByteString? {
    val image = ImageIO.read(file.inputStream)
    var outputStream = ByteArrayOutputStream()
    ImageIO.write(image, "jpg", outputStream)
    outputStream.close()
    val newWidth = 850
    val newHeight = (image.height.toDouble() * newWidth / image.width).roundToInt()
    val resizedImage = BufferedImage(newWidth, newHeight, BufferedImage.
     TYPE_INT_RGB)
    val g = resizedImage.createGraphics()
    g.drawImage(image, 0, 0, newWidth, newHeight, null)
    g.dispose()
    outputStream = ByteArrayOutputStream()
    ImageIO.write(resizedImage, "jpg", outputStream)
    val imageBytes: ByteArray = outputStream.toByteArray()
    outputStream.close()

    return ByteString.copyFrom(imageBytes)
}
```

Obrázek 7.2: Ukázka Optimalizace obrázku

```
type Document{
    id: UUID!
    filename:String!,
    storageFilename:String!,
    createAt:Instant!,
    createByUser:UUID!,
    customAnnotations:[Annotation]!,
    annotatedData:[AttributeKeyValuePair]!,
    allTextDescription:String!,
    documentStatus:DocumentStatus!,
    imgLink:String!,
    lockByUser:UUID!,
    deleted:Boolean!,
    isLocked:Boolean!,
    folderId:UUID
}
```

Obrázek 7.4: Ukázka definice typu pro dokument

```
type Mutation {
    createUser:Boolean

    #FILE
    uploadImage(files:[Upload], folderId:String):Boolean
    updateAnnotation(annotations:[AttributeKeyValuePairInput]!, documentId:UUID):
     Boolean

    #FOLDER
    createFolder(name:String!, labels:[String]):Folder
    addUser(email:String!, folderId:UUID!):Folder

    #SCHEMA
    updateSchema(schema:[String], folderId:UUID):Folder
}
```

Obrázek 7.5: Ukázka mutátorů

7.1.3 Business Layer

Jak již z názvu vyplývá tato vrstva obsahuje rozhraní a implementace, které řeší logiku celé aplikace. Jedná se o soubory:

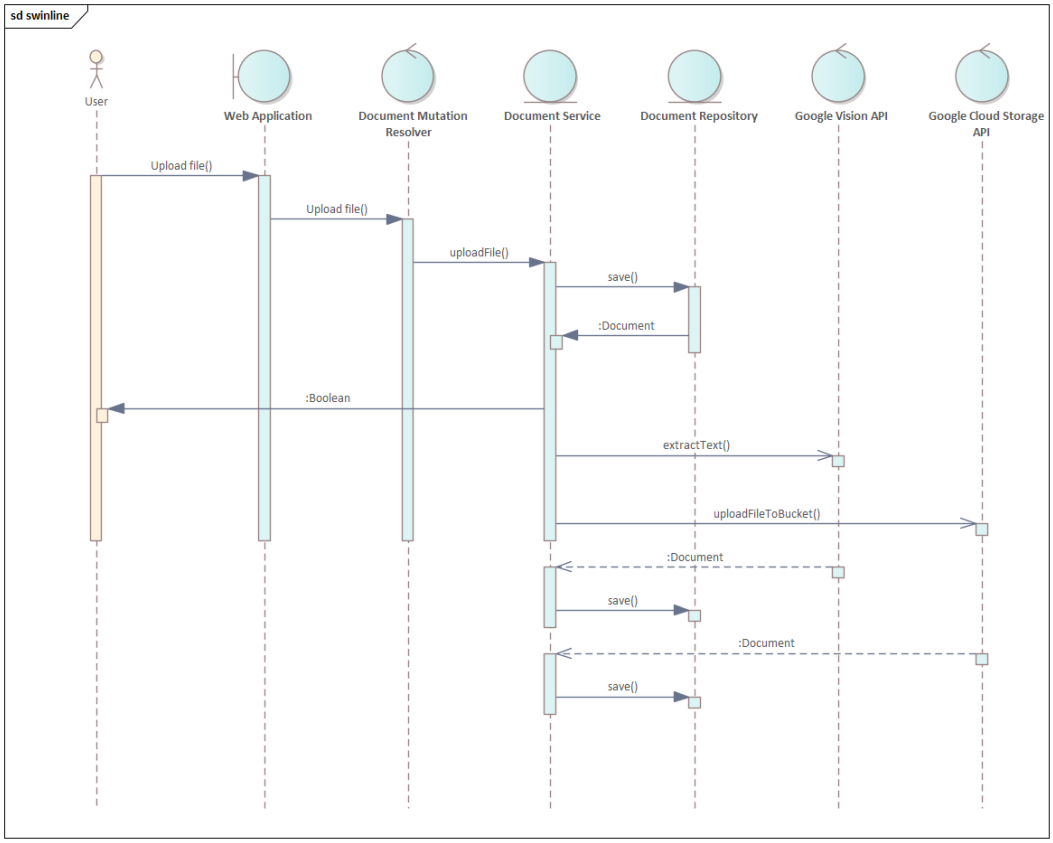
- DocumentAccessService
- DocumentService
- FolderAccessService
- FolderService
- GCPBucketUtil
- OAuth2UserService
- SchemaService
- ShareLinkService
- UserService

```
@Component
class GCPBucketUtil(
    @Value("\${gcp.bucket.name}")
    private val bucketName: String,
    @Value("\${gcp.url}")
    private val bucketLink: String,
    private val documentRepository: DocumentRepository
) {

    private val storage: Storage = StorageOptions.getDefaultInstance().service

    fun uploadFileToBucketAndSave(file: ApplicationPart, documentId: UUID) {
        val filename = file.submittedFileName.plus(documentId)
        val blobId = BlobId.of(bucketName, filename)
        val blobInfo = BlobInfo.newBuilder(blobId).setContentType(file.contentType).
     build()
        storage.create(blobInfo, file.inputStream.readAllBytes())

        val document = documentRepository.findById(documentId)
        document?.storageFilename = filename
        document?.imgLink = "$bucketLink/$bucketName/$filename"

        documentRepository.save(document!!)
        log.info { "Uploaded␣document␣with␣id:␣${document.id}␣to␣GCP,␣Document␣
     status:␣${document.documentStatus}" }
    }
}
```

Obrázek 7.6: Ukázka kódu pro ukládání dokumentu do cloudu

Tento kód 7.6 obsahuje třídu GCPBucketUtil, která slouží k nahrávání souborů do Google Cloud Storage a ukládání informací o nahraných souborech do databáze. Tato třída je označena jako @Component, což znamená, že se jedná o Spring bean a bude vytvořena a spravována Spring kontejnerem. Metoda uploadFileToBucketAndSave přijímá soubor a identifikátor dokumentu

a nahrává soubor do Google Cloud Storage s názvem, který se skládá z názvu původního souboru a identifikátoru dokumentu. Poté aktualizuje informace o uložení souboru do databáze, včetně názvu souboru a odkazu na soubor v Google Cloud Storage.



Obrázek 7.7: Sekvenční diagram nahrávání dokumentu

Sekvenční diagram je grafické zobrazení interakcí mezi objekty nebo komponentami systému v čase. V tomto konkrétním případě 7.7, sekvence popisuje následující proces:

- Uživatel nahraje dokument do aplikace.
- Frontend pošle soubor pomocí GraphQL mutation na backend.
- Document Mutation Resolver zavolá metodu uploadFile v Document Service.
- Document Service vytvoří novou instanci dokumentu a zavolá na documentRepository metodu save(), která uloží dokument do databáze.
- Document Service spustí metodu extractTextAndSave asynchronně pro extrakci textu a metodu uploadFileToBucket pro uložení do Google Cloud bucket.

7.1.4 Data Layer

V rámci data layeru se často pracuje s objekty, které se nazývají entity. Tyto objekty představují reprezentaci dat v aplikaci. Využívá se Objektově relační mapování (ORM) nástroje, které umožňují mapování entit na tabulky a záznamy v databázi. Kromě entit data layer obsahuje také datové objekty, které slouží k přenosu dat mezi různými vrstvami aplikace. Jedná se o entity:

- CustomSchema
- Document
- DocumentAccess
- Folder
- FolderAccess
- ShareLink
- User
- AttributeKeyValue
- CustomAnnotation

7.1.5 Repository layer

Tato vrstva poskytuje rozhraní pro práci s daty. Jedná se o rozhraní, které definuje základní Create, Read, Update, Delete (CRUD) operace pro práci s daty. Implementace tohoto rozhraní pak zajišťuje konkrétní přístup k datům pomocí Objektově relační mapování (ORM).

```
@Repository
interface DocumentAccessRepository : JpaRepository<DocumentAccess, Long> {

    fun findById(id: UUID): DocumentAccess?

    fun findDocumentAccessByDocumentIdAndUserId(
        @Param("documentId") documentId: UUID,
        @Param("userId") userId: UUID
    ): DocumentAccess?
}
```

Obrázek 7.9: Ukázka repository - documentAccessRepository

Tento kód 7.9 definuje rozhraní DocumentAccessRepository, které rozšiřuje rozhraní JpaRepository. JpaRepository je součástí frameworku Spring Data JPA a poskytuje základní metody pro práci s databází. Rozhraní DocumentAccessRepository definuje dvě metody pro přístup k entitě DocumentAccess v databázi. Metoda findById slouží k nalezení dokumentu podle jeho identifikátoru, který je předán jako parametr metody. Metoda

findDocumentAccessByDocumentIdAndUserId slouží k nalezení přístupového záznamu pro dokument, který je identifikován pomocí identifikátoru dokumentu a identifikátoru uživatele. Anotace @Repository nad rozhraním DocumentAccessRepository označuje tuto třídu jako Spring Bean.

7.1.6 Security

```
@Bean
fun filterChain(http: HttpSecurity): SecurityFilterChain {
    http
        .cors().configurationSource(corsConfigurationSource()).and()
        .csrf().disable()
        .authorizeHttpRequests()
        .requestMatchers("/login", "/error", "/loginSuccess").permitAll()
        .anyRequest().authenticated()
        .and()
        .oauth2Login()
        .defaultSuccessUrl("/loginSuccess")
        .and()
        .oauth2Client()
        .and()
        .addFilterBefore(oAuth2Filter, UsernamePasswordAuthenticationFilter::class.
     java)
    return http.build()
}
```

Obrázek 7.10: filter chain

Tento kódu 7.10 definuje konfiguraci bezpečnostního filtru pro security. Používá SecurityFilterChain, který obsahuje filtry pro ověření OAuth 2.0 přihlášení a zabezpečení aplikace před neoprávněným přístupem. Konfigurace obsahuje několik pravidel pro autorizaci požadavků. Endpointy, jako jsou /login, /error a /loginSuccess nejsou autorizované. Dále se nastavuje konfigurace pro OAuth 2.0 přihlášení pomocí metody oauth2Login(), která definuje výchozí úspěšnou URL pro přihlášení.

```
@Document(indexName = "document")
@Where(clause = "deleted=false")
class Document(
    @Id
    var id: UUID = UUID.randomUUID(),

    @Field(type = FieldType.Text)
    var filename: String,

    @Field(type = FieldType.Text)
    var storageFilename: String? = null,

    @Field(type = FieldType.Date)
    @DateTimeFormat(iso = DateTimeFormat.ISO.DATE_TIME)
    var createAt: Instant? = Instant.now(),

    var createByUser: UUID,

    @Field(type = FieldType.Nested)
    var customAnnotations: List<CustomAnnotation> = ArrayList(),
    @Field(type = FieldType.Nested)
    var annotatedData: List<AttributeKeyValueModel> = ArrayList(),

    @Field(type = FieldType.Text)
    var allTextDescription: String? = null,

    @Field(type = FieldType.Text)
    var documentStatus: DocumentStatus? = DocumentStatus.NEW,

    @Field(type = FieldType.Text)
    var imgLink: String? = null,

    var lockByUser: UUID? = null,

    var deleted: Boolean? = false,

    var isLocked: Boolean? = false,

    var folderId: UUID? = null,

    @ManyToMany
    @JoinTable(
        name = "DOCUMENT_ACCESS",
        joinColumns = [JoinColumn(name = "DOCUMENT_ID")],
        inverseJoinColumns = [JoinColumn(name = "ACCESS_ID")]
    )
    var documentAccesses: MutableList<DocumentAccess> = ArrayList()
)
```

Obrázek 7.8: Ukázka entity - dokument

7.2 Frontend

Při implementaci frontendu se využíval Next.js, což je framework pro React, který poskytuje mnoho funkcí pro tvorbu moderních webových aplikací, jako je server-side rendering. Pro tvorbu komponent byla využitá knihovna styled-components, která umožňuje vytvářet komponenty s inline styly. Pro komunikaci s backendem se použila knihovnu Apollo GraphQL, která umožňuje snadnou a efektivní komunikaci s GraphQL API. Díky použití této knihovny bylo možné definovat a používat GraphQL dotazy a mutace. V rámci Atomic design patternu se rozdělila aplikace na atomické komponenty, jako jsou tlačítka, textová pole a obrázky, a postupně je kombinoval do složitějších molekul a organismů. Tyto komponenty byly dále použity v layoutech a stránkách.

7.2.1 GraphQL Fetching

```
export default function App({Component, pageProps}: AppProps) {
    return (
        <AuthContext>
            <ApolloProvider client={client}>
                <Layout>
                    <Component {...pageProps} className={poppins.className}/>
                </Layout>
            </ApolloProvider>
        </AuthContext>
    )
}

const client = new ApolloClient({
    link: createUploadLink({
        uri: process.env.FETCH_URL,
        headers: {
            Authorization: `Bearer ${token}`
        },
    }),
    cache: new InMemoryCache(),
});
```

Obrázek 7.11: Ukázka GraphQL Apollo clienta

Ukázka kódu 7.11 definuje hlavní kořenovou komponentu aplikace v Next.js. Hlavním cílem této komponenty je poskytnout GraphQL klienta (ApolloProvider), která bude použita pro všechny stránky. Klient Apollo GraphQL s nastavením propojení na GraphQL server na adrese http://localhost:8080/graphql, který je definovaný v .env.local souboru a autorizačním tokenem předaným v hlavičce požadavku. Tento klient je poté poskytnut jako props do ApolloProvider komponenty.

```
export const GET_FILE = gql‘
    query GetDocument($id: String) {
        getDocument(id: $id) {
            id
            filename
            imgLink
            folderId
            createAt
            createByUser
            annotatedData{
                key
                value
            }
            customAnnotations {
                description
                x
                y
                width
                height
            }
        }
    }
‘;
```

Obrázek 7.12: Ukázka definování query

```
const {loading, error, data} = useQuery(GET_FILE, {variables: {id: id}});
```

Obrázek 7.13: Ukázka graphql dotazování na backend

7.2.2 Komponenty

Jak již bylo zmíněno, při vývoji byl použit návrhový vzor atomic design, tudíž celá aplikace byla rozdělena na komponenty, konkrétně na atomic, molecules a organisms. K tvorbě komponent byla použita knihovna styled-components.

```
interface LoadingCircleProgressProps {
    color: string
    executing: number
}

export const StyledCircle = styled.circle<LoadingCircleProgressProps>‘
  fill: none;
  stroke: ${props => props.color};
  stroke-width: 10px;
  stroke-dasharray: 282.6;
  stroke-dashoffset: ${props => props.executing};

  animation: loading 1s ease forwards;

  @keyframes loading {
    0% {
      stroke-dashoffset: 282.6;
    }
  }
‘
```

Obrázek 7.14: Ukázka komponenty

Kapitola 8
Testování

Testování je nevyhnutelnou částí vývoje každé aplikace. V naší implementační části jsme se zaměřili na tři různé metody testování: unit test, end to end testy a usability testing.

Výsledky testování byly zaznamenány a analyzovány. Během testování byly zjištěny určité nedostatky a chyby, které byly následně opraveny. Výsledky testování nám poskytly důležité informace o funkčnosti a použitelnosti aplikace, což nám umožnilo přijmout rozhodnutí o případných úpravách a vylepšeních.

Celkově lze říci, že testování je nezbytnou součástí každého vývojového procesu a umožňuje nám zajistit, že aplikace splňuje požadavky a očekávání uživatelů.

8.1 Unit testing

Unit test se zaměřuje na testování malých částí kódu, např. funkcí nebo metod, aby se ověřilo, že pracují správně a nezpůsobují neočekávané chyby. Testování jednotek bylo použito pro ověření funkcionality jednotlivých klíčových metod. Během testování servisů bylo použito mockování a simulace vstupů a výstupů.

V ukázce 8.1 kódu je funkce `initSecurityContext()` s anotací `@BeforeEach`, která označuje, že tato funkce se provede před každým jednotlivým testem. Tato funkce slouží k inicializaci bezpečnostního kontextu pro jednotkové testování backendové části aplikace.

Funkce `testUploadDocument()` testuje funkčnost metody `uploadDocument()` z třídy `documentService`. Nejprve se vytvoří instance `File`, která představuje soubor, který bude nahrán. Poté se vytvoří instance `ApplicationPart`, která představuje část multipart souboru. Jako první parametr se předává název části souboru (v tomto případě null, protože není potřeba specifikovat název) a jako druhý parametr se předává vytvořená instance `File`. Volá se metoda `uploadDocument()` na `documentService`, která je testována. Jako parametry se předává `folderId` a soubor, který má být nahrán. Následuje ověření pomocí metody `assertEquals()`, která porovnává očekávanou hodnotu s aktuální hodnotou.

```
 @BeforeEach
fun initSecurityContext() {
    val user = Utils.createUser()
    userRepository.save(user)
    val oauth2User = GoogleOAuth2AuthenticationToken(
        email = user.email,
        authorities = Collections.singleton(SimpleGrantedAuthority("user"))
    )
    val authentication = UsernamePasswordAuthenticationToken(
        oauth2User,
        null,
        oauth2User.authorities
    )
    SecurityContextHolder.getContext().authentication = authentication
}

@Test
fun testUploadDocument() {
    val file = File("src/test/kotlin/cz/milancu/app/beunlost/service/impl/
     Screenshot_3.png")
    val multipartFile = ApplicationPart(null, file)

    documentService.uploadDocument(folderId = folderId, file = multipartFile)

    assertEquals(1, documentRepository.count())
}

@Test
fun renameDocument() {
    val newName = "hodlcube"
    val documentId = UUID.randomUUID()
    val document = Document(
        filename = "filename",
        createByUser = userId,
        documentStatus = DocumentStatus.EXTRACTING,
        folderId = folderId,
        id = documentId
    )
    documentRepository.save(document)

    documentService.renameDocument(documentId, newName)
    val result = documentService.findDocumentById(documentId)

    assertEquals(newName, result.filename)
}
```

Obrázek 8.1: Ukázka unit testů

8.2 End to End testy

End to end testování je proces testování softwarové aplikace z hlediska uživatele. Tento typ testování simuluje reálné interakce uživatele s aplikací a testuje její funkcionalitu a spolehlivost. Pro realizaci e2e testů byla použita technologie Cypress. End to end testy jsou navrženy tak, aby simulovaly celý proces uživatele v aplikaci od začátku do konce. Pro end to end test byl vytvořen jeden rozsáhlý scénář:

- Přihlásit se.
- Vytvoření složky s názvem: "Test folder".

- Vytvořit si label s názvem: "Test label".
- Nahrání souboru z dashboard stránky přes drag and drop file.
- Vybrat si složku kam soubor nahrát.
- Zobrazit si nahraný soubor.
- Z anotovat si datum splacení na dokumentu.
- Vrátit se na vytvořený soubor a přidat label s libovolným názvem.
- Sdílet složku s uživatelem admin@unlost.cz.
- Odhlásit se a přihlásit se pod jiným uživatelem (admin@unlost.cz).
- Zkontrolovat zda byla uživateli sdílena složka.
- Otevřít soubor.
- Upravit hodnotu.
- Odhlásit se.
- Přihlásit se zpátky na uživatele.
- Zkontrolovat zda se hodnota uložila.

V ukázce kódu 8.2 je znázorněna funkce, která simuluje nahrávání souborů skrze drag and drop.

```
cy.fixture('Screenshot_2.png').then(fileContent => {
    const file = new File([fileContent], 'Screenshot_2.png', { type: 'image/png' })
    ;

    const fileList = {
        length: 1,
        item: file
    };

    cy.log(fileList)
    cy.get('.DragAndDrop-style__StyledDragAndDrop-sc-5b92576c-0')
        .trigger('dragover', {dataTransfer: {}})
        .then(() => {
            cy.get('.DragAndDrop-style__StyledDragAndDrop-sc-5b92576c-0')
                .trigger('drop', {
                    dataTransfer: {
                        files: fileList,
                    },
                })
        });
    cy.get('#folders').select("f686324c-79bf-4674-9cb5-520084066679")
    cy.get('.UploadFileModal-style__ButtonWrapper-sc-a35b7145-7␣>␣.Button-
    style__StyledButton-sc-4ba6b3f3-1').click()
});
```

Obrázek 8.2: Ukázka Cypress pro nahrávání souboru

V ukázce kódu 8.3 je znázorněna simulovace anotování dat v dokumentu, kdy se na daný element klikne `.trigger("mousedown")` s danou pozicí a posune se kurzor `.trigger("mousemov")`.

```
cy.get('#annotationLayer')
    .trigger('mousedown', {clientX: 110, clientY: 98, force: true})
    .trigger('mousemove', {clientX: 156, clientY: 104, force: true})
    .trigger('mouseup', {force: true});
});
```

Obrázek 8.3: Ukázka Cypress anotování v dokumentu

8.3 Usability testing

Testování použitelnosti se zaměřuje na ověření, zda je aplikace snadno použitelná a intuitivní pro uživatele. Tato metoda testování zahrnuje testování návrhu uživatelského rozhraní a způsobu, jakým jsou uživatelské akce interpretovány. Pro tuto metodu testu si autor práce vytvořil scénář a požádal si 4 lidi aby provedli určité úkoly v aplikaci dle scénáře zatím co jsou pozorovány autorem práce.

8.3.1 Scénář

Uživatelům byl zadán scénář, podle kterého aplikaci testovali. Testeři na dále měli volnou ruku, aby mohli aplikaci otestovat. Scénář:

- Přihlásit se.
- Vytvoření složky.
- Vytvořit si label s libovolným názvem.
- Nahrání souboru, které bylo uživateli dodáno, z dashboard stránky přes drag and drop file.
- Vybrat si složku(která byla uživatelem vytvořena) pro nahrání souboru.
- Zobrazit si nahraný soubor.
- Z anotovat si datum splacení na dokumentu.
- Vrátit se na vytvořený soubor a přidat label s libovolným názvem.
- Sdílet složku s uživatelem admin@unlost.cz.

8.3.2 Tester 1

U prvního testera se při pokusu o import souborů vyskytly komplikace. Tester omylem přidal více souborů než jeden a nebyla jiná možnost odstranit soubor než zrušit celé nahrávání. Dále uživatel nepoznal, že se daný dokument už nahrál.

8.3.3 Tester 2

U druhého testeru fungovala importní bez jakýkoliv komplikací. Problém nastal, když chtěl tester vyhledávat dokument pomocí obsaženého textu v dokumentu. Vyhledávání v danou chvíli nevracel žádný záznam. Chyba byla pak upravena. Zbytek testování proběhl bez problémů.

8.3.4 Tester 3

Třetí tester měl problémy především s nahráváním souborů, kde uživatel nepozná, že se dokument již úspěšně nahrál.

8.3.5 Tester 4

Čtvrtý a poslední tester se setkal s podobnými potížemi jako předchozí testeři. Nahrávání dokumentů nebylo tak intuitivní. Dále tester upozornil na chybějící snackbar [1] při uložení dokumentu.

8.3.6 Shrnutí testování

Výsledky testu použitelnosti byly velmi přínostné, protože při sledování uživatelů se odhalilo několik problémů:

- Nahrávání souborů nebylo intuitivní.
- Chybějící snackbar pro úspěšné uložení dokumentu.
- Chyba při vyhledávání dokumentu podle obsaženého textu.

[1] Snackbar - komponenta rozhraní uživatele, která slouží k zobrazení krátkých zpráv nebo oznámení.

Kapitola 9

Závěr

Cílem bakalářské práce byla analýza existujícího řešení zaměřený na správu dokumentů s využitím technologie OCR. Dále jsem provedl průzkum a seznámenil se s technologií OCR a jejími nástroji. Součástí této bakalářské práce byla i implementace, která byla postavena na analýze a návrhu ze semestrálního projektu. Aplikace se skládá z backendové části a webového rozhraní, které umožňují autentizaci a autorizaci, správu dokumentů, extrakci textového obsahu z dokumentu pomocí OCR, anotaci dat v dokumentech, sdílení dokumentů a fulltextové vyhledávání.

Součástí práce byla i navržena testovací strategie k ověření správné funkčnosti aplikace. Při analýze existujících řešení jsem se zaměřil na pět aplikací. V rámci analýzy byla zprostředkována schůzka s produkt manažerem a vrchním technikem firmy Wlow, aby mi představili svůj produkt a seznámili s technickým řešením jejich enterprise aplikace. Důkladnější analýza aplikace byla provedena díky testovacímu účtu od WFlow, který mi poskytli.

Na základě analýzy technologií a architektury jsem si vybral programovací jazyk Kotlin a framework Spring Boot pro implementaci backendu aplikace, zatímco pro frontend React spolu s frameworkem Next.js. Pro ukládání strukturovaných dat byla zvolena PostgreSQL databáze, zatímco pro nestrukturovaná data byla použita databáze od ElasticSearch díky výkonnému fulltextovému vyhledávání. Pro OCR a cloudové úložiště souborů jsem použil nástroje od Google, konkrétně Google Vision a Google Cloud Storage. Aplikace byla vyvíjena ve vícevrstvé architektuře, a pro komunikaci mezi backendem a frontendem byl využit Graphql.

Součástí práce bylo i otestování aplikace, kde jsem požádal čtyři kolegové, aby otestovali aplikaci podle scénáře a provedli test použitelnosti. Díky testům bylo zjištěno, že aplikace není v určitých situacích dostatečně intuitivní.

Všechny definované cíle práce byly splněny včetně všech klíčových požadavků. V budoucnosti by bylo možné vytvořit mobilní aplikaci, zlepšit uživatelské rozhraní pro vyšší přehlednost nebo přidání automatické extrahování textu, tudíž by uživatel nemusel dokumenty anotovat, pouze tedy zkontrolovat zda se hodnota správně nakopírovala z dokumentu. Aplikace by bylo také vhodné publikovat tedy nasadit na nějakou platformu.

Literatura

[1] Rollbarnew, "Most popular java backend frameworks for 2021," Jan 2022.

[2] "Kotlin tutorial: Kotling programming language - javatpoint."

[3] Admin, "Java vs kotlin: Every difference you should know in 2022," Jan 2022.

[4] "Advantages of spring boot," Oct 2022.

[5] L. Singla, "What is php for web development and why should you use it?," Dec 2022.

[6] H. Kundariya, "Why choose laravel for web development?," Mar 2022.

[7] "Symphony v/s laravel - which php development framework to choose?," Jun 2022.

[8] T. Madeira, "Why use python for web development?," Apr 2021.

[9] M. Nowak, "Why ruby on rails is still a good choice in 2022 [updated]," Dec 2022.

[10] V. H. s. https://www.vas hosting.cz, "Velké srovnání mysql (mariadb) a postgresql."

[11] A. Bampakos, "Why choose the angular framework," Nov 2022.

[12] D. Jadhav, "Why choose react for frontend?," Oct 2021.

[13] M. Gathoni, "Why you should migrate to next.js," Aug 2022.

[14] F. K. is a data science professional with demonstrated history with both academic, professional to deliver valuable insights via data analytics, and advanced data-driven methods. She's specialized in Computer Vision., "A comprehensive guide to optical character recognition (ocr)," Nov 2019.

[15] "Amazon textract is a machine learning (ml) service that automatically extracts text, handwriting, and data from scanned documents.," 1978.

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Cu** Jméno: **Phuong Dong** Osobní číslo: **499203**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Webová aplikace pro správu dokumentů s využitím OCR pro extrakci textu

Název bakalářské práce anglicky:

Web application for document management using OCR for text extraction

Pokyny pro vypracování:

Cílem bakalářské práce je provést analýzu existujícího řešení zaměřeného na správu dokumentů s využitím technologie OCR. Dále bude proveden průzkum a seznámení se s technologií OCR a jejími nástroji, které budou integrovány do navrhované aplikace.
Tato aplikace se bude skládat z backendové části a webového rozhraní a bude umožňovat:
- autentizaci a autorizaci uživatelů,
- správu dokumentů,
- provádění OCR s extrakcí textového obsahu,
- anotaci dat v dokumentech, sdílení dokumentů,
- fulltextové vyhledávání.

V rámci bakalářské práce bude navržena testovací strategie k ověření správné funkčnosti aplikace.

Seznam doporučené literatury:

Richards, Mark. Fundamentals of Software Architecture: An Engineering Approach. O'Reilly Media, 2020.
Rice, Stephen V., George Nagy, and Thomas A. Nartker. Optical Character Recognition: An Illustrated Guide to the Frontier. Boston, MA: Springer Science + Business Media, 2013.
“What Is Amazon Textract? - Amazon Textract.” n.d. https://docs.aws.amazon.com/textract/latest/dg/what-is.html.
“Detect Text in Images.” n.d. Google Cloud. https://cloud.google.com/vision/docs/ocr.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Kyrylo Bulat katedra počítačů FEL

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

Datum zadání bakalářské práce: **14.02.2023** Termín odevzdání bakalářské práce: **26.05.2023**

Platnost zadání bakalářské práce: **22.09.2024**

Ing. Kyrylo Bulat podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

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

. | Datum převzetí zadání

Podpis studenta