UNIVERZITA PARDUBICE

NÁZEV FAKULTY

BAKALÁŘSKÁ PRÁCE

2025 Petr Šafrata

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Virtuální úložiště s pokročilým vyhledáváním

Bakalářská práce

2025 Petr Šafrata

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2024/2025

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Petr Šafrata**
Osobní číslo:	**I22134**
Studijní program:	**B0688A140009 Informační technologie**
Téma práce:	**Virtuální úložiště s pokročilým vyhledáváním**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Cílem této bakalářské práce je navrhnout a implementovat webovou aplikaci pro správu a pokročilé vyhledávání souborů. V uložišti by mělo být možné full-textově vyhledávat v uložených souborech (například v textových souborech, souborech typu word, excel, pdf apod.). Aplikace by také měla umět prohledávat v textu obrázků pomocí technologie Tesseract OCR. Aplikace bude implementována v jazyce Java s frameworkem Spring Boot pro backend a API a celý systém bude kontejnerizován pomocí Dockeru.

Rozsah pracovní zprávy: **min. 30 stran**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

PECINOVSKÝ, Rudolf. Java 21: kompletní příručka jazyka. Knihovna programátora. Praha: Grada Publishing, 2023. ISBN 978-80-247-0599-6.
CARNELL, John. Spring microservices in action: a multiplatform approach to building chatbots. Shelter, Island, NY: Manning Publications Co., 2017. ISBN 16-172-9398-9.
KUĆ, Rafał a Marek ROGOZIŃSKI. Elasticsearch server. S.l.: Packt Publishing Limited, 2013. ISBN 1849518440. WARBURTON, Richard. Java 8 lambdas. Beijing: O'Reilly, 2014. ISBN 1449370772.

Vedoucí bakalářské práce: **Ing. Jan Merta, Ph.D.**
Katedra softwarových technologií

Datum zadání bakalářské práce: **15. prosince 2024**
Termín odevzdání bakalářské práce: **16. května 2025**

L.S.

prof. Ing. Petr Doležel, Ph.D. v.r.
děkan

Ing. Jan Panuš, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 28. února 2025

Prohlašuji:

Práci s názvem **Virtuální uložiště s pokročilým vyhledáváním** jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 14. 05. 2025

Petr Šafrata v.r.

PODĚKOVÁNÍ

Na tomto místě bych rád poděkoval vedoucímu své bakalářské práce, panu Ing. Janu Mertovi, Ph.D. za jeho cenné rady, odborné vedení a podporu během celé doby zpracování. Dále děkuji své rodině, přítelkyni a přátelům za trpělivost, podporu a motivaci jak při tvorbě této práce, tak po celou dobu studia.

ANOTACE

Bakalářská práce se věnuje analýze poskytovatelů cloudových služeb a popisu jednotlivých technologií v rámci teoretické části. V praktické části se zabývá návrhem a implementací webové aplikace pro správu souborů s podporou pokročilého fulltextového vyhledávání. Aplikace umožnuje uživatelům nahrávat různé typy souborů a vyhledávat v jejich obsahu. Implementace je provedena v jazyce Java s využitím frameworku Spring Boot pro tvorbu backendových REST API. Celý systém je navržen jako kontejnerizované řešení pomocí platformy Docker. Výsledkem práce je funkční webová aplikace inteligentního uložiště, které kombinuje moderní technologie a umožnuje efektivní správu a vyhledávání dokumentů.

KLÍČOVÁ SLOVA

webová aplikace, Spring Boot, fulltextové vyhledávání, Tesseract OCR, OCR, rozpoznávání textu, rozpoznávání řeči, převod řeči na text, Vosk, Docker, kontejnerizace, Java, Elasticsearch, správa souborů

TITLE

Virtual storage with advanced search

ANNOTATION

The bachelor thesis is devoted to the analysis of cloud service providers and description of individual technologies within the theoretical part. The practical part deals with the design and implementation of a web application for file management with support for advanced full-text search. The application allows users to upload different types of files and search their contents. The implementation is done in Java using the Spring Boot framework for creating backend REST APIs. The entire system is designed as a containerized solution using the Docker platform. The result of the work is a functional web application of an intelligent repository that combines modern technologies and enables efficient document management and retrieval.

KEYWORDS

web application, Spring Boot, fulltext search, Tesseract OCR, OCR, text recognition, speech recognition, speech to text, Vosk, Docker, containerization, Java, Elasticsearch, file management

OBSAH

SEZNAM ILUSTRACÍ A TABULEK

SEZNAM ZKRATEK A ZNAČEK

REST	Representational State Transfer	Přenos reprezentačního stavu
OMR	Optical Mark Recognition	Optické rozpoznání značek
ICR	Intelligent Character Recognition	Inteligentní rozpoznání znaků
ASR	Automatic Speech Recognition	Automatické rozpoznání řeči
NLP	Natural Language Processing	Zpracování přirozeného jazyka
DLS	Domain Specific Language	Jazyk specifický pro danou oblast
API	Application Programming Interface	Aplikační programové rozhraní
OCR	Optical Character Recognition	Optické rozpoznání znaků
CLI	Command-Line Interface	Rozhraní příkazového řádku
HMM	Hidden Markov Models	Skryté Markovské modely
DTO	Data Transfer Object	Objekt přenosu dat
SMB	Server Message Block	Blok zpráv serveru
SaaS	Software as a Service	Software jako služba
PaaS	Platform as a Service	Platforma jako služba
IaaS	Infrastructure as a Service	Infrastruktura jako služba
JPA	Java Persistence API	Rozhraní Java Persistence API
JSON	JavaScript Object Notation	
HTML	Hypertext Markup Language	Hypertextový značkovací jazyk
CSS	Cascading Style Sheets	Kaskádové styly
AI	Artificial Intelligence	Umělá inteligence
NFS	Network File Systém	Síťový souborový systém
GCP	Google Cloud Platform	
AWS	Amazon Web Services	
ES	Elasticsearch	
DB	Databáze	
JS	JavaScript	
JDK	Java Development Kit	
JVM	Java Virtual Machine	
docker	Docker klient	

ÚVOD

Od počátku vývoje informačních systému je jejich nedílnou součástí ukládání dat. Jako první médium pro ukládání dat můžeme považovat děrné štítky. Dalším milníkem byl přechod na magnetická uložiště, nejprve se jednalo o magnetické pásky a později o magnetické disky, které umožnily přímý přístup k datům. Postupem času se vyvíjely různé nové formy pevných disků, optických medií a dalších druhů paměti. Na počátku 21. století se rozmohla masová digitalizace, nástup nové generace pevných disků, rozvoj polovodičových pamětí a v posledních letech cloud computing. Cloud computing má však svůj původ již mnohem dříve, předpokládá se, že jej v 60. letech 20. stolení vynalezl JCR Licklider. Od této doby vznikají první služby, které umožnují nahrávání a sdílení dat přes internet. Vývoj těchto služeb ještě podpořilo rozšíření vysokorychlostního připojení k internetu v 90. letech. V roce 2006 nastal průlom, kdy Amazon spustil službu Simple Storage Service, která představila revoluční koncept uložiště s prakticky neomezenou škálovatelností. Později se přidávají i další technologičtí giganti jako Microsoft nebo Google.

Nyní žijeme v době, kdy jsou cloudová uložiště nedílnou součástí moderních informačních systémů a používají se v různých oblastech. Ať už se jedná o osobní použití, komerční sféru nebo vládní či vzdělávací instituce. V každém z těchto odvětví mohou být cloudová uložiště použita trochu jiným způsobem, nicméně hlavní cíl zůstává stejný, tedy bezpečné ukládání, sdílení a přístup k datům odkudkoliv a kdykoliv.

Cílem této bakalářské práce je navrhnout a implementovat webovou aplikaci, která umožní snadné nahrávání a správu souborů v rámci virtuálního uložiště a zajistí pokročilé vyhledávání těchto souborů a dat v nich. Zároveň je také cílem zajistit snadnou instalaci celého systému.

Teoretická část této práce se věnuje představení cloudových uložišť, průzkumu velkých poskytovatelů těchto systémů na trhu a analýze jejich řešení. Zbytek této části je pak věnován popisu použitých technologií.

Praktická část práce se věnuje implementaci aplikace v jazyce Java s využitím frameworku Spring Boot pro backend API. Pro vytěžení dat z nahrávaných souborů je použita technologie Tesseract OCR pro obrázky, Tika pro PDF soubory a Vosk pro audionahrávky. Elasticsearch pak slouží jako databáze pro ukládání dat a umožnuje fulltextové vyhledávání v souborech. Celý systém je kontejnerizován pomocí nástroje Docker pro snadnou instalaci.

1 Analýza dostupných řešení

1.1 Amazon Web Service

Společnost Amazon je v dnešní době předním lídrem v oblasti poskytování cloudových služeb a platforem na světě. Jejich služba Amazon Web Service (AWS) nabízí více, než 200 plně vybavených služeb pro výpočetní výkon, uložiště, databáze, umělou inteligenci, strojové učení atd. V oblasti zabezpečení AWS poskytuje širokou sadu bezpečnostních nástrojů a také podporuje velké množství bezpečnostních standardů a certifikátů. Infrastruktura AWS je navržena tak, aby splňovala bezpečnostní požadavky armád, bank a dalších institucí, které pracují s vysoce citlivými daty. Díky vysokému tempu nejnovějších inovací, spolehlivosti a zabezpečení má AWS miliony zákazníků po celém světě. Mezi nejvýznamnější zákazníky můžeme zařadit společnosti jako Netflix, NASA, BMW či Coca-Cola [1].

Jak již bylo uvedeno, tak jednou z hlavních služeb AWS jsou uložiště. Do této kategorie patří služby S3, EBS, EFS.

1.1.1 Amazon Simple Storage Service

Amazon Simple Storage Service (Amazon S3) je služba pro ukládání objektů, která nabízí špičkovou škálovatelnost, dostupnost dat, zabezpečení a výkon. Amazon S3 funguje na principu ukládání dat jako objektů v segmentech (viz. Obrázek 1). Objekt je pak soubor a veškerá jeho metadata, která tento soubor popisují. Aby bylo možné data uložit do Amazon S3, je nejprve nutné, vytvořit segment a příslušnou oblast v AWS. Poté již stačí do segmentu nahrát data jako objekty. Každý objekt má klíč, což je jedinečný identifikátor objektu v rámci segmentu. Amazon S3 poskytuje velké množství funkcí, které si zákazníci mohou jednoduše nakonfigurovat, aby podporovaly jejich potřeby a konkrétní případy využití [2].

Obrázek 1: **Grafické znázornění fungování S3**[1]

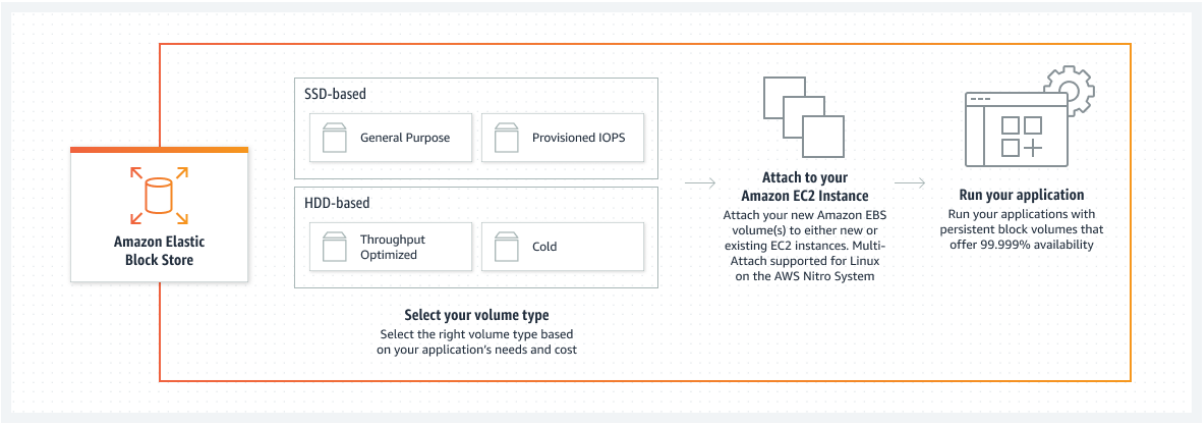
1.1.2 Amazon Elastic Block Storage

Amazon Elastic Block Storage (Amazon EBS) je služba, která poskytuje škálovatelné a vysoce výkonné blokové uložiště, které lze používat s instancemi Amazon EC2. Pomocí Amazon EBS je možné vytvářet a spravovat jednotlivé prostředky blokového uložiště (viz. Obrázek 2). Tyto prostředky zahrnují Amazon ESB Volumes a Amazon ESB Snapshots.

- Amazon ESB Volumes jsou úložné svazky připojené k instanci Amazon EC2. Po připojení může být svazek využíván jako klasický pevný disk připojený k počítači. Je tedy možné zde ukládat soubory nebo instalovat aplikace.
- Amazon ESB Snapshots jsou bodové zálohy svazků, které existují nezávisle na samotném svazku. Mohou být tedy vytvářeny pro zálohování dat na svazcích a je možné je kdykoliv obnovit.

Typické použití této služby jsou databázové, souborové a aplikační servery [3].

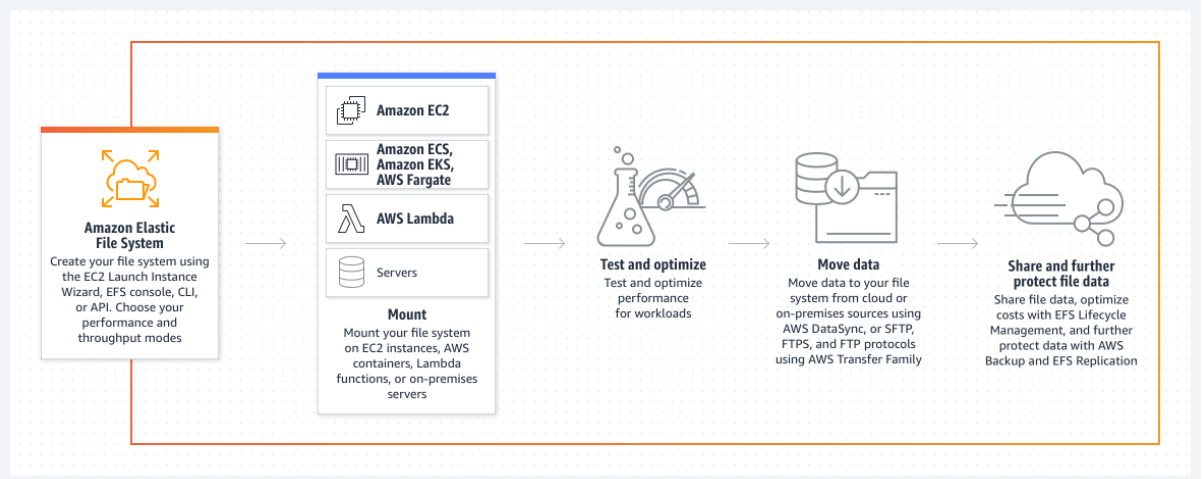
[1] Amazon. Online. In: aws.amazon.com. 2025. Dostupné také z: https://aws.amazon.com/s3/



Obrázek 2: **Grafické znázornění fungování EBS**[2]

1.1.3 Amazon Elastic File Systém

Amazon Elastic File System (Amazon EFS) služba, která poskytuje jednoduchý a elastický souborový systém bez serveru. Je tedy možné sdílet data a soubory bez nutnosti zajišťovat či spravovat výkon a kapacitu uložiště (viz. Obrázek 3). Amazon EFS nabízí vysokou škálovatelnost bez narušení aplikací, kdy se velikost automaticky zvětšuje či zmenšuje podle toho, zda jsou přidávány nebo odebírány soubory. Zároveň služba disponuje jednoduchým rozhraním, které je přístupné z internetu, takže mohou být souborové systémy snadno vytvářeny a konfigurovány. Veškerou infrastrukturu pro ukládání souboru spravuje služba automaticky [4].



Obrázek 3: **Grafické znázornění fungování EFS**[3]

[2] Cloudzero. Online. In: cloudzero.com. 2025. Dostupné také z: https://www.cloudzero.com/blog/amazon-ebs-optimization/
[3] Amazon. Online. In: amazonaws.cn. 2025. Dostupné také z: https://www.amazonaws.cn/en/efs/

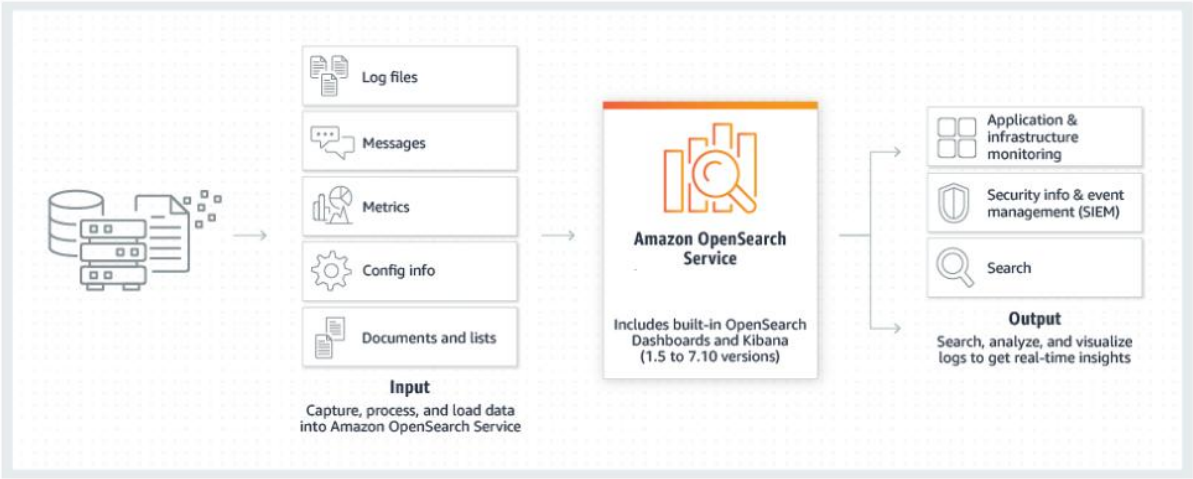
1.1.4 Technologie pro vyhledávání a OCR

Amazon v rámci AWS poskytuje mnoho různých služeb, ve kterých nabízí mimo jiné fulltextové vyhledávání či OCR.

1.1.4.1 Amazon OpenSearch Service

Amazon OpenSearch Service je služba, která zajišťuje provoz clusterů OpenSearch v rámci cloudu AWS [5].

OpenSearch je pak open-source vyhledávací engine, který je distribuován pod licencí Apache 2.0 a je založen na knihovně Apache Lucene. K předním vlastnostem tohoto enginu patří vysoká škálovatelnost a podpora velkého množství vyhledávacích algoritmů jako je vyhledávání pomocí K-nejbližších sousedů, dotazování pomocí SQL syntaxe, detekce anomálií, Machine Learning Commons, Trace Analytics, fulltextové vyhledávání a další (viz. Obrázek 4) [6].



Obrázek 4: **Grafické znázornění fungování Amazon OpenSearch Service**[4]

1.1.4.2 Amazon CloudSearch

Amazon CloudSearch je jednoduší vyhledávací služba, plně spravovaná v cloudu, která podporuje funkce jako je fulltextové vyhledávání, booleovské vyhledávání, vyhledávání podle předpony, posilování termínů, fasetování, zvýrazňování zásahů a návrhy automatického doplňování [7].

[4] Amazon. Online. In: docs.aws.amazon.com. 2025. Dostupné také z: https://docs.aws.amazon.com/opensearch-service/latest/developerguide/what-is.html.

1.1.4.3 Amazon Textract

Amazon Textract je služba, díky které je možné v rámci aplikací provádět detekci a analýzu textů v dokumentech. Amazon Textract umožnuje detekovat text v ručně psaných dokumentech, ale také umí extrahovat text, formuláře, tabulky z dokumentů se strukturovanými daty pomocí API rozhraní. Zároveň je možné pomocí této služby zpracovávat faktury a účtenky s využitím API AnalyzeExpense. Amazon Textract je založen na vysoce škálovatelné technologii hlubokého učení [8].

1.1.5 Typické workflow

Soubory různého typu (např. PDF, Word, obrázky) jsou ukládány do Amazon S3. Po uložení souboru dojde k vytěžení obsahu daného souboru pomocí Amazon Textract. Výsledný text je poté indexován do Amazon OpenSearch, které umožnuje fulltextové vyhledávání.

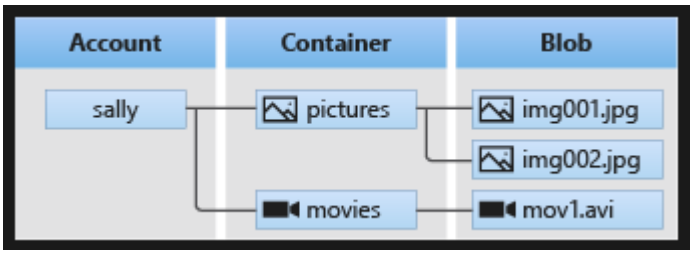
1.2 Microsoft Azure

Dalším velkým hráčem na poli s poskytováním cloudových služeb je společnost Microsoft a jejich platforma Azure, což je cloudová platforma, která je navržena tak, aby co nejvíce zjednodušila proces vytváření moderních aplikací. Azure poskytuje široké portfolio cloudových služeb a produktů, které zahrnují uložiště, databáze, výpočetní výkon, umělou inteligenci, síťové služby atd. Zároveň Azure poskytuje své služby ve více než 60 regionech po celém světě, ve kterých se nachází více než 300 vysoce zabezpečených datových center. Z pohledu lokality se tedy jedná o nejdostupnější cloudovou službu. Díky velké flexibilitě a škálovatelnosti lze jakékoliv služby v rámci Azure snadno rozšířit nebo omezit podle potřeb klientů. Pokud jde o zabezpečení, tak Azure splňuje velké množství mezinárodních standardů a certifikací. Mimo to je Azure chráněn integrovanými bezpečnostními řešeními a specializovaný tým expertů denně monitoruje systémy a zjišťuje zranitelnosti. Microsoft zároveň vynakládá velké množství financí na bezpečnost. Díky tomu službu Azure využívají miliony zákazníků po celém světě [9]. Do kategorie uložiště dat poté patří služby jako Azure Blob Storage, Azure Files a Disk Storage.

1.2.1 Azure Blob Storage

Azure Blob Storage je služba pro ukládání objektů. Toto uložiště je optimalizována pro ukládání velkého množství nestrukturovaných dat. Za nestrukturovaná data se považují data, která se neřídí určitým datovým modelem či definicí. Může se tedy jednat např. o textová nebo binární data. Uživatelé nebo aplikace mohou k objektům v uložišti přistupovat pomocí http nebo https protokolů a Azure Rest API nebo knihovny Azure Storage, která je dostupná

pro mnoho programovacích jazyků včetně .NET a Java. Další možnosti připojení zahrnují protokoly jako SFTP nebo NFS. Z pohledu struktury se uložiště dělí na tři typy prostředků. Prvním je účet v uložišti, který poskytuje jedinečný prostor v rámci Azure. Dalším prostředkem je kontejner, který organizuje sadu blobů podobně jako adresář v souborovém systému. V rámci jednoho účtu v Azure může být neomezený počet kontejnerů a kontejner může ukládat neomezený počet blobů. Bloby jsou pak nejmenší prostředek, kdy se může jednat o blokové bloby, která ukládají binární nebo textová data (viz. Obrázek 5), append bloby, které jsou optimalizovány pro operace append či stránkové bloby, které ukládají soubory s náhodným přístupem [10].

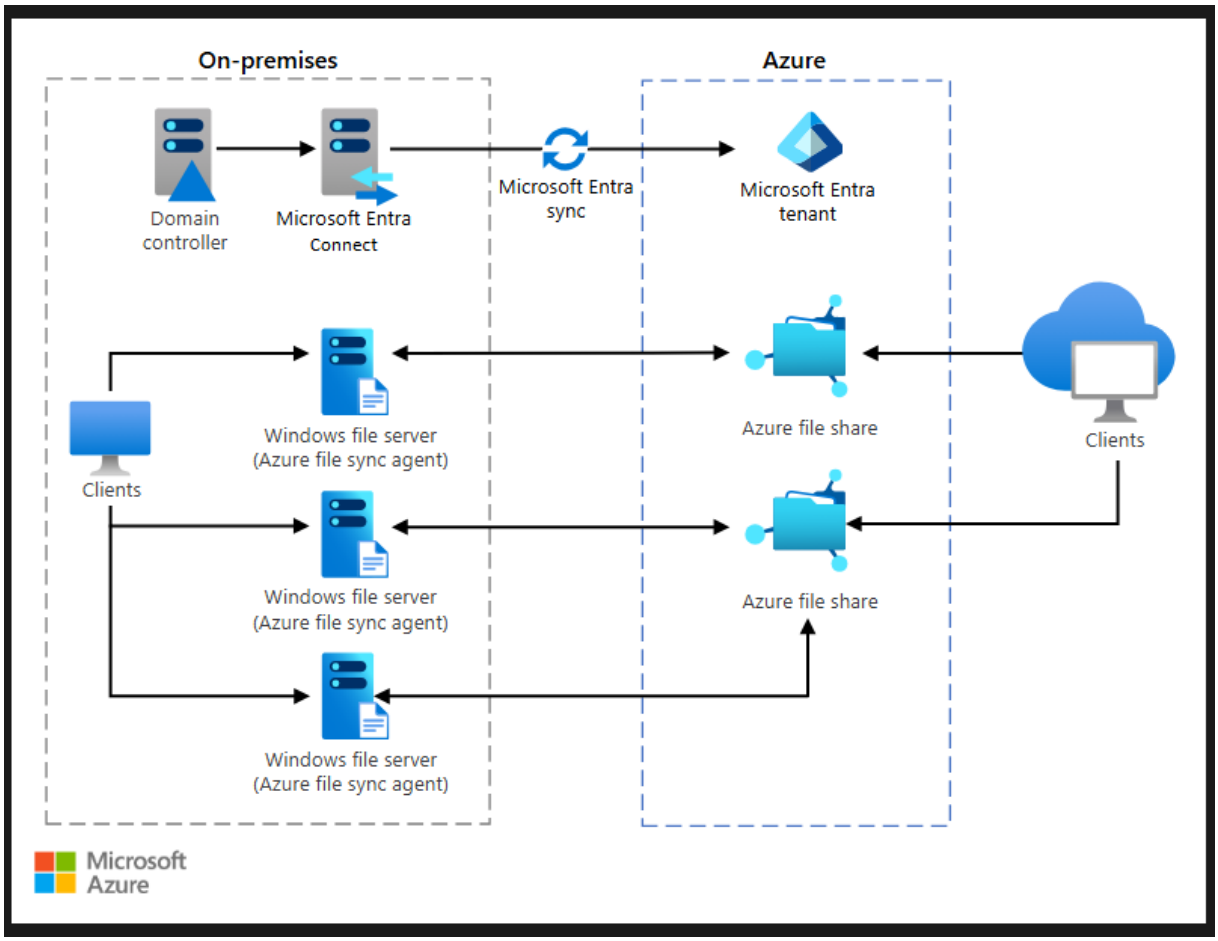


Obrázek 5: **Schéma vztahů mezi prostředky**[5]

1.2.2 Azure Files

Služba Azure Files poskytuje plně spravované cloudové uložiště ve formě sdílení souborů, které je dostupné pomocí protokolů SMB a NFS, nebo pomocí rozhraní Azure Files REST API. Sdílené soubory lze pak připojovat současně jak v cloudu, tak i v lokálních nasazeních (viz. Obrázek 6). Výhodou této služby je, že sdílené soubory pomocí SMB Azure lze navíc ukládat do mezipaměti na serverech Windows pomocí funkce Azure File Sync. To umožnuje přistupovat k datům přímo tam, kde jsou aktuálně využívána, např. v místní pobočce organizace nebo v datovém centru nacházejícím se v blízkosti koncových uživatelů [11].

[5] Introduction to Azure Blob Storage. Online. In: Learn.microsoft.com. 2025. Dostupné také z: https://learn.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction.



Obrázek 6: **Architektura rozšíření souborových služeb pomocí Azure File Sync a Azure Files**[6]

1.2.3 Azure Disk Storage

Azure Disk Storage je služba, která nabízí virtualizované blokové uložiště pro virtuální servery. Disky uložiště jsou podobné fyzickým diskům, ale jsou virtualizované a abstrahované od fyzické infrastruktury. Azure disponuje velkou nabídku disků, které si uživatelé mohou připojit ke svým virtuálním strojům. Z pohledu uživatele stačí vybrat typ disku a kapacitu a o vše ostatní se postará Azure, včetně vytvoření virtuálního disku, zabezpečení a obecné konfigurace. Tato služba se využívá především pro provozování virtuálních serverů, databázových a aplikačních serverů a obecně prostředí, které mají vysoké nároky [12].

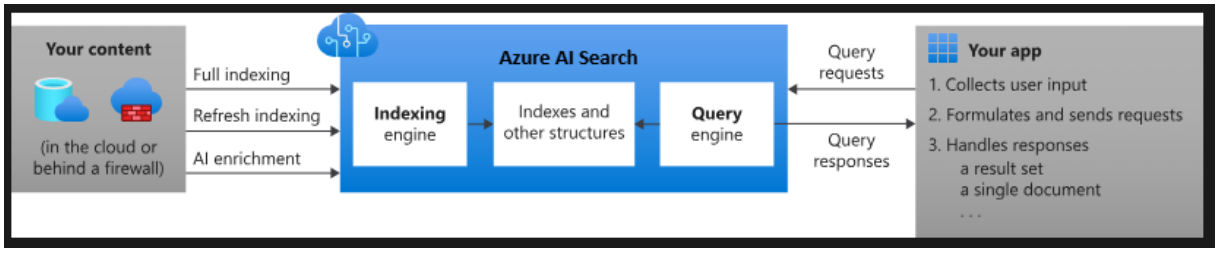
1.2.4 Technologie pro vyhledávání a OCR

Společnost Microsoft v rámci své cloudové platformy Azure poskytuje řadu pokročilých služeb zaměřených na pokročilé zpracování dat. Mezi které patří služby OCR i inteligentní vyhledávání.

[6] Use Azure file shares in a hybrid environment. Online. In: Learn.microsoft.com. 2025. Dostupné také z: https://learn.microsoft.com/en-us/azure/architecture/hybrid/azure-file-share

1.2.4.1 Azure AI Search

Azure AI Search je služba, která umožnuje vyhledávání informací z heterogenního obsahu, který je vkládán do vyhledávacího indexu. Data jsou poté zpřístupněna pomocí dotazů či aplikací (viz. Obrázek 7). Služba je vybavena komplexní sadou pokročilých vyhledávacích technologií. Mezi základní technologie patří fulltextové, vektorové a hybridní vyhledávání. Díky tomu lze Azure Cognitive Search použít pro běžné vyhledávání v dokumentech, zjišťování informací, ale i pro Retrieval-augmented generation, což je technika, která umožňuje generativním modelům umělé inteligence získávat a začleňovat nové informace. Při vytváření vyhledávací služby je možné si sužbu upravit podle vlastních požadavků a konkrétních potřeb projektu [13].



Obrázek 7: **Architektura Azure AI Search**[7]

1.2.4.2 Azure AI Vision

Služba Azure AI Vision zpřístupňuje pokročilé algoritmy pro zpracování obrázků a informací z nich získaných, na základě vizuálních vlastností. Tuto službu lze rozdělit do několika částí podle typu zpracování (viz. Tabulka 1) [14].

Optical Character Recognition (OCR)	Služba dokáže extrahovat text z obrázků
Image Analysis	Služba dokáže z obrazu získávat různé informace, dle vizuálních prvků. Může se jednat o objekty, obličeje atd.
Face	Služba poskytuje detekci a rozpoznávání lidské tváře
Video Analysis	Služba poskytuje prostorovou analýzu videa, díky které je možné analyzovat přítomnost a pohyb osob na videu

Tabulka 1: **Rozdělení služeb (zdroj: autor práce)**

[7] What's Azure AI Search? Online. In: Learn.microsoft.com. 2025. Dostupné také z: https://learn.microsoft.com/en-us/azure/search/search-what-is-azure-search.

1.2.5 Typické workflow

Soubory různých typů jsou ukládány do Azure Blob Storage. Po uložení souboru se automaticky aktivuje Azure Function, případně jiný serverless nástroj, který aktivuje službu Azure AI Vision. Služba pak extrahuje veškerý text ze souboru. Azure AI Search poté načte text a metadata souboru vytvoří vyhledávací index dle definovaných pravidel pro fulltextové či jiné vyhledávání.

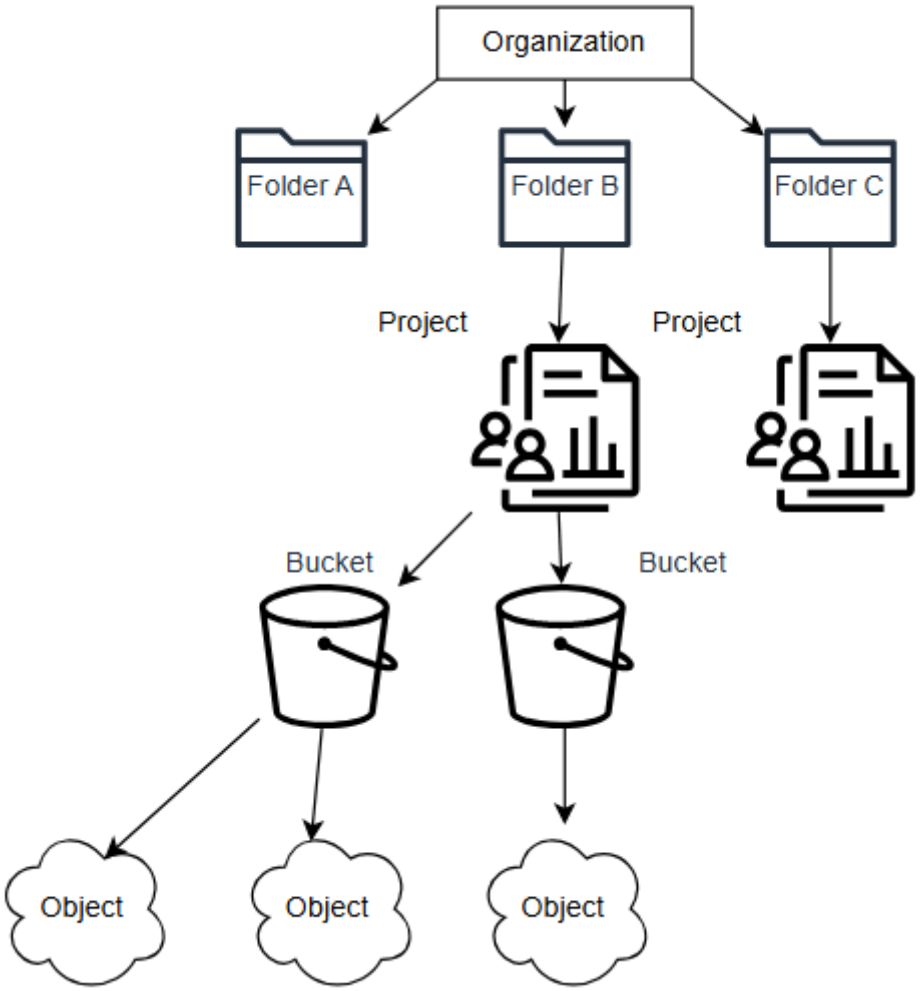
1.3 Google Cloud Platform

Google Cloud Platform (GCP) je cloudová platforma od společnosti Google, nabízí podobnou škálu služeb jako dříve uvedené AWS či Azure. GCP je složen ze souborů fyzických prostředků, což zahrnuje počítače a pevné disky a virtuálních prostředků jako jsou virtuální počítače umístěné v datových centrech. Datová centra jsou poté k dispozici v regionech jako jsou Asie, Austrálie, Evropa, Afrika, Severní a Jižní Amerika a Blízký východ. V každém regionu poté existuje soubor několika zón, které jsou vzájemně izolovány. Toto rozdělení je výhodné z důvodu geografické a fyzické redundance v případě selhání a snížení latence, díky umístění blíže ke klientům [15]. Google také klade velký důraz na bezpečnost, provozuje vlastní specializovaný tým složený z předních odborníků na bezpečnost, kteří udržují bezpečností systémy, vyvíjí nové kontrolní procesy a budují bezpečnostní infrastrukturu. Zároveň Google také úzce spolupracuje s komunitou výzkumníků v oblasti zabezpečení. Podílejí se např. na projektu zranitelnosti nultého dne. Zároveň také provozují vlastní program Bug Hunters, který nabízí velké finanční odměny za naleznutí zranitelností v systémech [16]. Google zároveň stejnou infrastrukturu a bezpečnostní modely využívá i pro své vlastní služby a produkty jako např. Gmail, Fotky Google atd. [17].

1.3.1 Cloud Storage

Cloud Storage je služba pro ukládání objektů v rámci platformy Google Cloud Platform. Objekt je zde definován jako neměnný díl dat tvořených souborem libovolného formátu. Objekty jsou následně ukládány do kontejnerů, které mohou být následně spojovány s projekty a projekty mohou být seskupené pod nějakou organizací. Organizace je nejvyšší jmenný prostor v rámci Google Cloud. Každá organizace poté může vytvářet různé projekty pro své aplikace. Každý projekt má svou vlastní sadu Cloud Storage API a zdroje. Každý z projektů poté může obsahovat více kontejnerů neboli bucketů, pro ukládání jednotlivých objektů (viz. Obrázek 8). S cloudovým uložištěm je poté možné pracovat více způsoby. Je možné využít Google Cloud Console, která poskytuje vizuální rozhraní pro správu dat v prohlížeči nebo využít klientské

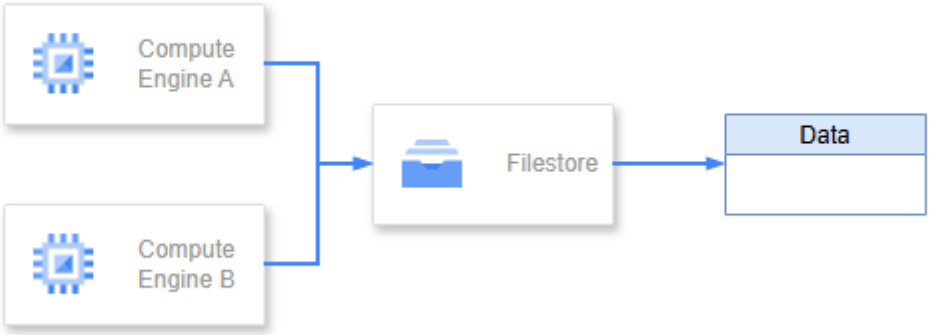
knihovny, která umožnují pracovat s daty v rámci vybraných programovacích jazyků např. C#, Java, Node.js, PHP atd. Samozřejmě je také možné využívat rozhraní REST API [18].



Obrázek 8: **Architektura Cloud Storage (zdroj: autor práce)**

1.3.2 Filestore

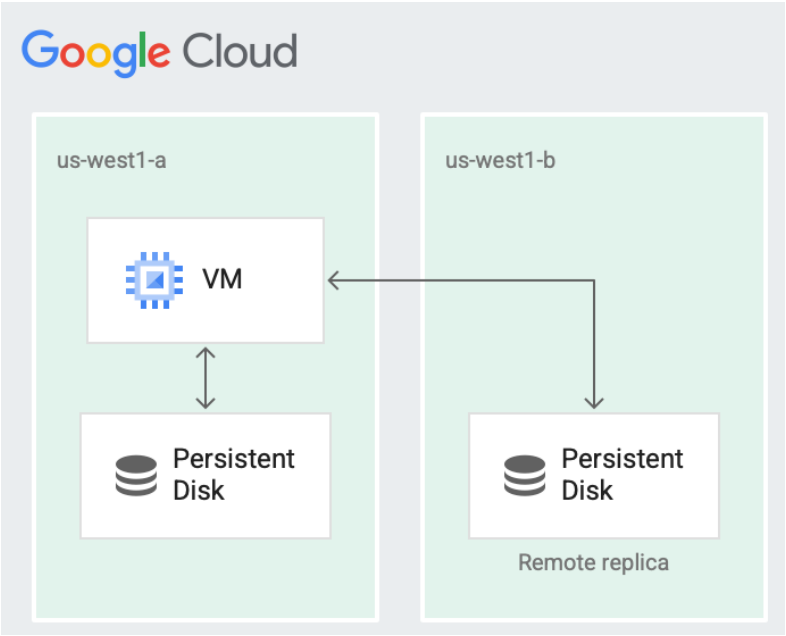
Filestore je plně spravovaná služba souborového uložiště v rámci GCP. Tento typ uložiště podporuje více souběžných instancí aplikací, které se připojují ke stejnému systému souboru. V rámci této služby je nabízeno několik úrovní, které se odlišují funkcemi, výkonem a kapacitou. Každá úroveň je poté přizpůsobena konkrétnímu použití. Základní úroveň je vhodná pro sdílení souborů, vývoj softwaru, webhosting a základní AI. Regionální a podniková úroveň jsou vhodné pro kritické pracovní zátěže, které vyžadují vysokou dostupnost. Zónová úroveň se pak především používá pro dálkové výpočty, pokročilou AI a velké datové soubory. Filestore podporuje sadu protokolů NFS, konkrétně NFSv3, který je k dispozici pro všechny úrovně služby a NFSv4.1, který je dostupný všechny úrovně, kromě základní [19].



Obrázek 9: **Architektura Filestore (zdroj: autor práce)**

1.3.3 Persistent Disks

Persistent Disks je služba, která v rámci GCP poskytuje trvanlivá úložná síťová zařízení, ke kterým se dá přistupovat podobně jako k fyzickým diskům. Persistent disk svazky však nejsou připojeny fyzicky k žádnému počítači, ale jsou k instanci připojeny jako síťová bloková zařízení. Data v rámci Persistent disk svazku jsou rozdělena na několik fyzických disků. Fyzické disky a distribuce dat je následně spravována automaticky pomocí Compute Engine, aby byl zajištěn optimální výkon a redundance. Google nabízí poměrně velkou nabídku pro trvalé disky, kde je možné vybrat si disk dle vlastních preferencí [20].



Obrázek 10: **Příklad použití Persistent disků**[8]

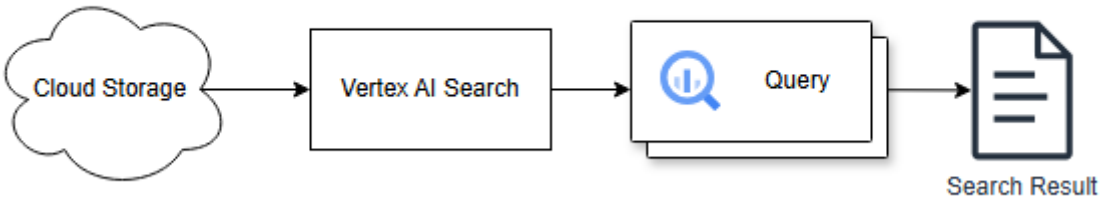
[8] About Persistent Disk. Online. In: Cloud.google.com. 2025. Dostupné také z: https://cloud.google.com/compute/docs/disks/persistent-disks.

1.3.4 Technologie pro vyhledávání a OCR

Google má také v rámci platformy GCP několik specializovaných technologií a služeb, pro vyhledávání a OCR. Služby jsou navržené pro snadnou integraci do webových, mobilních a enterprise aplikací.

1.3.4.1 Vertex AI Search

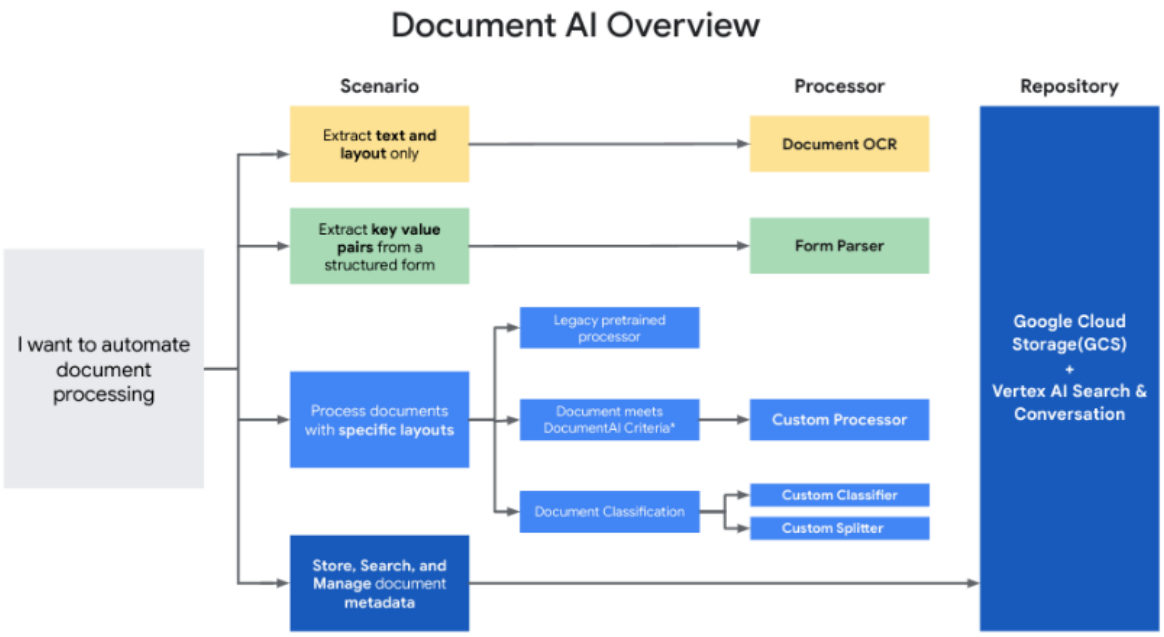
Vertex AI Search je služba, která spojuje sílu hlubokého vyhledávání, moderního zpracování přirozeného jazyka pomocí velkých jazykových modelů. Služba umožňuje vyhledávat v datech webových stránek, strukturovaných a i nestrukturovaných dat. Mezi základní technologie patří jak fulltextové vyhledávání, tak i sémantické vyhledávání. Díky tomu služba umožnuje vyhledávat v databázích, produktech nebo dokumentech. Pro využití služby stačí vytvořit aplikaci, kterou je následně možné připojit k uložišti. Následně jsou do uložiště vložena data, která jsou indexována. Vertex AI Search je možné implementovat pomocí použití konzole Google Cloud, Vertex AI Agent API nebo skombinováním obou služeb [21].



Obrázek 11: **Architektura Vertex AI Search (zdroj: autor práce)**

1.3.4.2 Document AI

Document AI představuje pokročilou OCR platformu, která je určena ke zpracování dokumentů. Umožnuje převod informací z nestrukturovaných dokumentů do podoby strukturovaných dat. Díky této transformaci je následně snazší data pochopit, lépe analyzovat a patřičně využít. Document AI umožňuje digitalizovat dokumenty, pomocí OCR je získán text a další data. Následně je extrahován tento text extrahován, stejně jako další informace ze souboru. Pokud se jedná o strukturovaná data, např. v tabulkách, je možné identifikovat páry klíč-hodnota. Poté dojde ke klasifikaci dokumentu. Každou část digitalizace dokumentu vykonává jiný procesor [22].



Obrázek 12: **Příklad využití Document AI se scénáři a procesory**[9]

1.3.5 Typické workfow

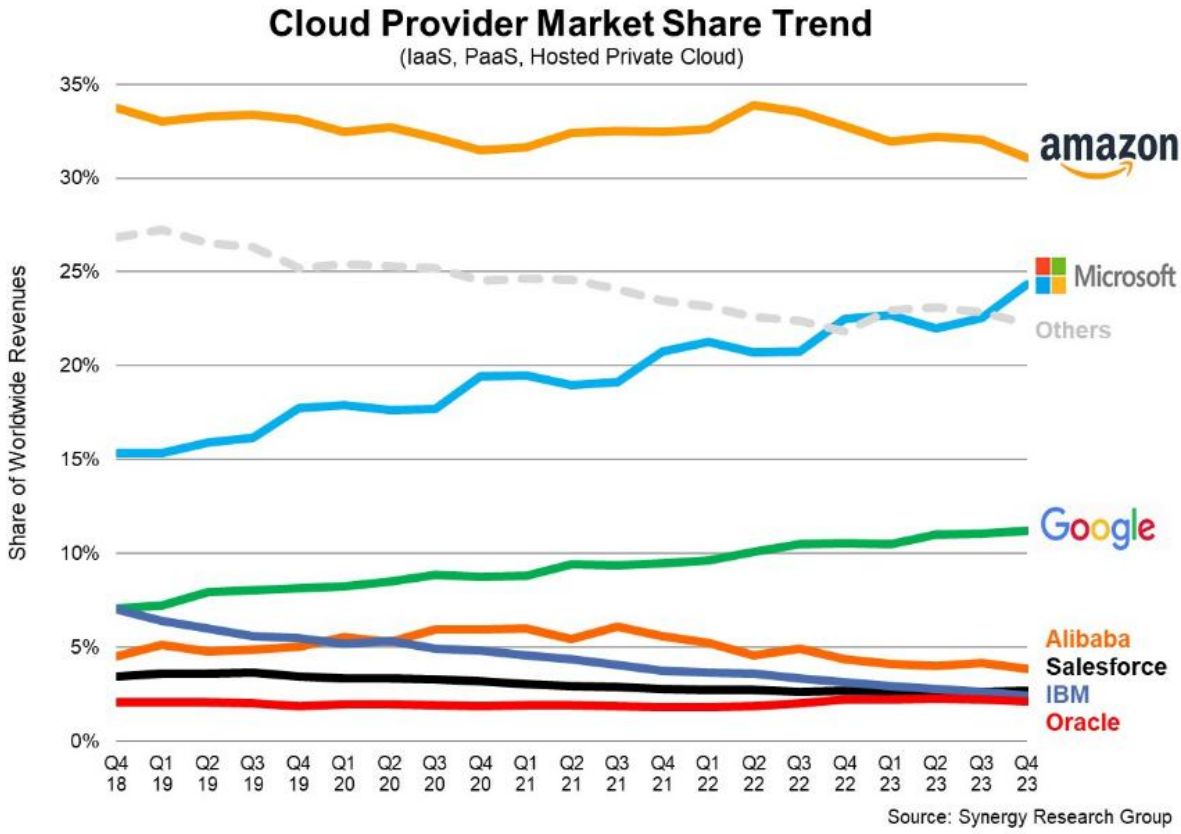
Soubory různých typů jsou nejprve nahrány do Google Cloud Storage, Následně dojde s pomocí služby Document AI k automatickému zpracování – provede se rozpoznání textu a extrakce strukturovaných dat. Výsledná data jsou uložena a dále indexována pomocí služby Vertex AI Search. Díky této integraci je posléze možné nalézt relevantní informace, a to i ve velkém množství nestrukturovaných souborů.

1.4 Další poskytovatelé

Amazon, Microsoft a Google samozřejmě nejsou jediní poskytovatelé cloudových řešení na trhu, nicméně společně tvoří více jak 60 % jeho zastoupení (viz. Obrázek 13). Existují však i další důležité společnosti, které nabízí cloudová řešení, nicméně jejich ekosystém služeb není tak rozsáhlý jako u výše uvedených společností. Mezi tyto společnosti patří Alibaba, která dominuje na asijském trhu a podporuje hlavní operace, velké, střední a malé podniky nebo také vlády či neziskové organizace. Dále pak společnost IBM se svým IBM Cloud, který nabízí více než 170 produktů pro lokální, hybridní i multicloudové nasazení. Stejně jako velká trojka se i IBM zaměřuje spravovanou infrastrukturu jako službu (IaaS) a služby migrace do cloudu. Platforma DigitalOcean Cloud společnosti DigitalOcean poskytuje platformu pro nasazení a škálování vývojářských projektů, mezi hlavní produkty patří uložiště

[9] Document AI overview. Online. In: Cloud.google.com. 2025. Dostupné také z: https://cloud.google.com/document-ai/docs/overview.

a výpočetní výkon. Podobně jako jiné společnosti poskytují také model IaaS, často přizpůsobený potřebám malých a středních firem. Společnost Salesforce nabízí široké spektrum cloudových nástrojů využitelných v marketingu, obchodu a službách. Tencent Cloud poskytuje sadu různorodých cloudových řešení a patří mezi přední poskytovatele v Číně. Současně ale působí i v dalších více než dvaceti regionech po celém světě a můžeme jej nalézt jak v Evropě, tak i v Americe. Posledním významnějším poskytovatelem je společnost Oracle, která jako první jako první nabízela modely cloudových služeb IaaS, PaaS a SaaS v rámci jediné platformy [23].



Obrázek 13: **Zastoupení společností na trhu**[10]

1.5 Srovnání velkých poskytovatelů

Jak již bylo uvedeno, AWS, Azure a GCP jsou tři největší cloudové platformy, které dohromady tvoří většinový podíl na trhu (viz. Obrázek 13). Nicméně pro potencionální výběr vhodné platformy je důležité pochopit silné a slabé stránky každé z nich a zároveň mít přehled o všech službách, které v rámci různých odvětví cloud computingu nabízejí (viz. Tabulka 2).

[10] Cloud Market Growth Stays Strong in Q2 While Amazon, Google and Oracle Nudge Higher. Online. In: Srgresearch.com. 2025. Dostupné také z: https://www.srgresearch.com/articles/cloud-market-growth-stays-strong-in-q2-while-amazon-google-and-oracle-nudge-higher.

	AWS	Azure	GCP
Výpočetní výkon	EC2, Lambda, Lightsail, Batch, APP Runner, ECS	Virtual Machines, Azure Functions, App Services, Kubernetes Service, Container Apps, Batch, LVM	Compute Engine, Cloud Functions, App Engine, VMware Engine, Cloud Run, Recommender, Google Kubernetes Engine
Uložiště	S3, EBS, EFS, Backup, Glacier, Elastic Data Recovery	Data Lake Storage, Disk Storage, File Storage, Container Storage, Blob Storage, Azure NetApp Files	Cloud Storage, Persistent Disk, Filestore, Storage Transfer Service, Transfer Appliance, Nearline
Databáze	RDS, SimpleDB, DynamoDB, DocumentDB, Aurora, Redshift	Azure Arc, Azure Stack, Azure SQL, Cosmos DB, Cache for Redis, Database Migration Service, Azure Database for MySQL, Azure Database for PostgreSQL	Cloud SQL, Firestore, Bigtable, Memorystore, Spanner, Datastream, AlloyDB
Strojové učení	AmazonQ, SageMaker, Augmented AI, Bedrock, AmazonQ, CodeGuru, Comprehend, Forecast	Azure ML, Cognitive Search, OpenAI Service, AI Speech, AI Vision, AI Content Safety, DevOps	Gemini, Looker, Vertex AI Platform, AutoML, Dialogflow, Natural Language, Vision AI, Video AI
Bezpečnost a identita	IAM, Shield, Cognito, Detective, Inspector, GuardDuty	Application Gateway, Microsoft Sentinel, Key Vault, Azure Firewall, Web Application Firewall, Azure DDoS Protection	IAM, Cloud IDS, Cloud Armor, Identity Platform, Cloud Key Management, Cloud Data Loss Prevention, Identity-Aware Proxy
Ceník Pay-as-you-go	✓	✓	✓

Tabulka 2: **Tabulka srovnání platforem na základě služeb**

1.5.1 Výhody a nevýhody AWS

AWS nabízí širokou škálu služeb, které jsou vhodné pro různé použití a odvětví. Zároveň se jedná o globálně rozšířenou platformu, díky čemuž mají její služby nízkou latenci v souladu s regionálními předpisy o datech. Některé služby v rámci platformy však mohou být náročnější pro pochopení, a tedy mohou vyžadovat specifické dovednosti. Cenový model AWS je sice flexibilní, ale i tak může být komplikovaný, což může vést k neočekávaným nákladům. Nicméně AWS nabízí 12měsíční bezplatné období pro více než 40 služeb, které umožnují získat cenné praktické zkušenosti s platformou [24]. Platforma však stále vede v oblastech

výpočetních služeb, uložišť, infrastruktury, IoT a edge computingu a bezserverových výpočtů. V rámci výpočetních služeb AWS EC2 nabízí nejširší škálu různých typů instancí, od univerzálních, až po GPU a vysoce výkonné výpočty. V oblasti uložišť Amazon S3 stále nabízí bezkonkurenční trvanlivost, zásady životního cyklu a bezpečností funkce. Co se týká infrastruktury, tak AWS má největší globální dostupnost s více než 30 regiony a více než 100 zónami dostupnosti [25].

1.5.2 Výhody a nevýhody Azure

Za největší výhodu platformy Azure a jejích služeb je bezproblémová a snadná integrace s dalšími produkty společnosti Microsoft. To tedy z Azure dělá vhodnou volbu pro podniky, které již nějakým způsobem využívají další služby této společnosti. Díky důrazu na hybridní cloudová řešení, je následná správa dat flexibilnější. Platforma také poskytuje velkou škálu služeb, a díky některým technologiím, vyhází vstříc i náročným zákazníkům. Azure je však oproti jiným platformám náročnější na naučení a kvůli svému zaměření především na podniková řešení, může být nevhodný pro menší podniky. Ačkoliv se Azure prezentuje jasným platebním modelem, tak jeho struktura je i přesto poměrně složitá, a to může vést opět k neočekávaným nákladům [24]. Azure oproti ostatním poskytovatelům vyniká především v oblastech databázových služeb, DevOps a vývojářských nástrojů, hybridní a podnikové integrace, bezpečnosti a dodržování předpisů. Azure SQL Database a Cosmos DB poskytují bezproblémovou integraci s prostředím Microsoft, takže jsou ideálním řešením pro podniky. Azure DevOps poskytuje komplexní sadu pro CI/CD, řízení verzí a monitorování. Azure Arc a Azure Stack nabízejí nejvyspělejší hybridní cloudová řešení, která umožňují firmám provozovat pracovní zátěže v lokálním prostředí, v cloudu nebo ve více prostředích. Azure Security Center a Sentinel nabízejí pokročilou detekci hrozeb založenou na umělé inteligenci [25].

1.5.3 Výhody a nevýhody GCP

GCP je díky své otevřené architektuře vhodný pro nasazení v multi-cloud a hybridních cloudech. Hraje také významnou roli při vývoji technologií s otevřeným kódem. Platforma klade velký důraz na umělou inteligenci a zároveň vyniká v oblasti Kubernetes s pokročilou technologií pro kontejnerové aplikace. Nicméně platforma má stále nejmenší zastoupení na trhu, což může vést k menšímu počtu integrací aplikací třetích stran. V rámci GCP jsou ceny také flexibilní, společnost však doporučuje, aby si potencionální uživatelé vyžádali cenovou nabídku. Výhodou je také to, že oproti jiným platformám dostávají noví zákazníci 300 dolarů v kreditech, které mohou využít v rámci platformy [24]. Platforma tedy především vyniká

v oblasti umělé inteligence a strojového učení, pro které nabízí nejlepší nástroje a také v oblasti cen, je tato platforma cenově nejtransparentnější a nabízí různé slevy [25].

2 Optické rozpoznávání znaků

Optické rozpoznávání znaků (OCR) je technologie, která extrahuje a převádí data z naskenovaných obrázků, obrázků z fotoaparátu a obrázkových souborů PDF. Následně z obrázků vyčleňuje písmena a skládá je do slov, a následně tato slova skládá do vět čímž umožnuje přistoupit k původnímu obsahu původního obrázku [26]. Jedná se tedy o proces, který převádí text z obrázku do strojově čitelného textového formátu [27].

OCR využívá kombinaci hardwaru a softwaru k převodu fyzicky tištěných dokumentů na strojově čitelný text. Jako hardware se typicky používá skener, který kopíruje nebo čte text a software se následně stará o pokročilé zpracování. OCR může zároveň využívat AI pro implementaci pokročilých metod jako je inteligentní rozpoznávání znaků [26].

2.1 Využití

V dnešní poměrně moderní době stále velké množství organizací zpracovává informace z tištěných medií, může se jednat např. o papírové formuláře, faktury, smlouvy a další dokumenty firemních procesů. Práce s těmito papírovými dokumenty je však značně neefektivní a zabírá velké množství času. Technologie OCR umožnuje digitalizaci těchto papírových dokumentů, tím že převádí naskenované textové obrázky na textová data, se kterými lze následně efektivněji pracovat, což zvyšuje produktivitu [27]. OCR se využívá v mnoha odvětvích, včetně vzdělávání, bankovnictví, zdravotnictví logistiky, dopravy atd. Mezi další důležité, ale méně známé případy použití této technologie patří pomoc pomoct pro nevidomé a zrakově postižené osoby [26]. S využitím OCR je také možné text z naskenovaných dokumentů integrovat dále do dalších systému, například databáze, Elasticsearch, systémy pro big-data atd. Díky tomu je poté snadné v informacích obsažených v dokumentu vyhledávat a pracovat s nimi.

2.2 Princip funkce

V první fázi procesu jsou nejprve veškeré stránky dokumentu zkopírovány a následně převedeny pomocí OCR enginu do dvoubarevného nebo černobílého formátu. Výsledný obrázek nebo bitmapa se poté analyzován z hlediska rozložení světlých a tmavých oblastí. Světlé části jsou programem rozpoznány jako pozadí a tmavé jsou vyhodnoceny jako znaky, které je třeba dále rozpoznat. V druhé fázi pak probíhá odstranění cizích pixelů, to zahrnuje opravu nesprávného zarovnání obrázku během skenování, odstranění grafických pravidel a rámečků. Poté dochází k určení, zda je součástí textu písmo. Další fází je poté již rozpoznání textu, tmavé části obrázku jsou analyzovány s cílem identifikovat jednotlivé znaky,

tedy písmena abecedy, symboly, či číslice. Tato fáze obvykle probíhá postupně, buď po jednotlivých znacích, slovech nebo blocích textu. Rozpoznání znaků je následně realizováno pomocí jednoho ze dvou algoritmů, rozpoznávání vzorů nebo rozpoznávání znaků. Algoritmus rozpoznání vzorů funguje na principu, kdy byl OCR program předem naučen na příkladech textů různých písmen a formátu, za cílem rozpoznání znaků a porovnání se vzorem naskenovaného dokumentu nebo obrázku. Aby tato funkce fungovala, musí být znaky v písmu, na kterém byl program OCR naučen. Algoritmus rozpoznání znaků se používá v případě, že OCR program analyzuje písmo, na které nebyl naučen. OCR aplikuje pravidla, týkající se konkrétního písmena nebo čísla k rozpoznávání znaků. V principu to funguje tak, že každý znak má nějaké rysy, např. počet šikmých čar, počet průsečíků, smyček nebo křivek ve znaku. Když je znak pomocí pravidel identifikován, je následně převeden na kód ASCII, se kterým již umí počítačové systémy pracovat. Komplexnější OCR také dokážou v rámci fáze rozpoznání rozložení analyzovat strukturu dokumentu. Dojde tedy k rozdělení stránky na prvky jako jsou bloky textů, tabulky nebo obrázky. Poslední fází je pak následné zpracování, informace jsou uloženy jako digitální soubor, který je možné upravovat, např PDF. Některé systémy uchovávají jak vstupní obrázek, tak i verze po OCR pro snadnější porovnání a úplnou správnost dokumentu [26].

2.3 Typy OCR

OCR můžeme rozdělit do několika typů, podle toho, jakým způsobem analyzují text a následně s ním pracují:

- Jednoduché OCR – analýza spočívá v porovnání vzorů znak po znaku a porovnání naskenovaných znaků s uloženými glyfy, avšak vzhledem k velkému množství různých kombinací písmen, jazyků a abeced, je tato analýza omezena pouze na některé dokumenty.
- Optické rozpoznávání značek (OMR) – používá se pro identifikaci zaškrtnutých políček a dalších značek, podpisů na formulářích, log, symbolů a vodoznaků. Všechny lze identifikovat porovnáním s uloženými obrázky, podobně jako u jednoduchého OCR.
- Inteligentní rozpoznání znaků (ICR) – využívá umělou inteligenci, pomocí hlubokého učení se program učí číst podobně jako lidé. Neuronová síť opakovaně prochází text a hledá charakteristické rysy.
- Inteligentní rozpoznání slov – pokročilejší model ICR, kdy ale umělá inteligence dokáže rozpoznávat celá slova [26].

3 Automatické rozpoznání řeči

Automatické rozpoznání řeči (ASR) je počítačové rozpoznávání řeči nebo převod řeči na text, je schopnost, díky které mohou programy převádět lidskou řeč do textové podoby. Existuje celá řada aplikací, které technologii ASR umožnují, přičemž pokročilejší systémy často využívají strojové učení a také metody umělé inteligence. Tyto systémy analyzují zvuk na základě daného jazyka, pracují se strukturou vět, gramatickými pravidly, výslovností a dalšími vlastnostmi, aby dokázali porozumět obsahu lidské řeči, kterou následně převádějí do textové podoby [28].

3.1 Princip funkce

Rozpoznávání řeči je založeno na několika na sebe navazujících krocích. Zpracování začíná zachycením zvukového signálu, který je následně dále analyzován za účelem identifikace příkazu. Systémy pracují s vektorovými reprezentacemi, které následně procházejí dekódovacím procesem, který má za cíl převést zvukový vstup do srozumitelného textu. Při zpracování využívá dekodér slovník výslovnosti, akustické a jazykové modely na základě, kterých určuje příslušný výstup. ASR se hodnotí podle chybovosti a rychlosti, chybovost však může ovlivnit celá řada faktorů např. přízvuk, výslovnost, výška tónu atd. Mezi dlouhodobé cíle těchto systému je dosažení míry chybovosti, která je běžná i v rámci lidských konverzací. K rozpoznání řeči do textu a zvýšení přenosit se pak používají různé algoritmy.

Zpracování přirozeného jazyka (NLP) není sice specifický algoritmus rozpoznávání řeči, ale je spíše o oblast umělé inteligence, která se zaměřuje na interakci člověka a stroje prostřednictvím řeči a textu. V dnešní době mono mobilní zařízení využívá tuto oblast, např. iPhone a jejich Siri.

Skryté markovské modely (HMM) jsou modely, které vycházení z modelu Markovova řetězce, který stanovuje, že pravděpodobnost daného stavu závisí na aktuálním stavu, nikoliv na jeho předchozích stavech. Markovovy řetězce se používají pro pozorovatelné událostí, jako je např. vstup textu, HMM umožnují zahrnout do pravděpodobnostního modelu skryté události, jako jsou značky částí řeči. V ASR se využívají sekvenční modely, které přiřazují značky k jednotlivým jednotkám, což jsou např. slabiky, části slov atd.

N-gramy jsou nejjednodušší typ jazykového modelu LM, který přiřazuje pravděpodobnost jednotlivým větám nebo frázím. N-gram je tedy v podstatě posloupnost N slov, které budou s pravděpodobností za sebou [28].

3.2 Využití

V dnešní moderní době se ASR využívá poměrně běžně v různých odvětvích. Patří sem např. automobilový průmysl, kdy je umožněno komunikovat s autem a dávat mu příkazy. Dále můžeme zahrnout takzvané virtuální agenty, kteří jsou integrováni do běžných lidských životů pomocí mobilních zařízení a dalších asistentů. Jedná se např. o Apple Siri, Google Assistant, Amazon Alexa a další. Jako další příklady můžeme uvést chatboty, kteří také dokážou interagovat s hlasovými příkazy nebo např. zabezpečení a hlasovou autentizaci [28].

4 Elasticsearch

Elasticsearch je opensource distribuovaný, RESTful vyhledávácí a analytický nástroj škálovatelného datového uložiště a vektorové databáze postavený na platformě Apache Lucene [29] [30]. Je optimalizován pro rychlost a relevanci v produkčním měřítku. Pomocí Elasticsearch je možné vyhledávat, indexovat, ukládat a analyzovat data všech typů a velikostí téměř v reálné čase [31]. Používá standartní rozhraní RESTful API a JSON [29].

4.1 Princip funkce a vyhledávání

Do Elasticsearch (ES) mohou být data zasílána ve formě dokumentů JSON pomocí API rozhraní. ES automaticky uloží zaslaný dokument a přidá do něj odkaz pro vyhledávání v indexu clusteru. Evidovaný dokument je pak možné vyhledávat a načítat pomocí rozhraní API [32].

Index je základní jednotkou v rámci ES, jedná se o logický prostor názvů pro ukládání dat, která mají podobné vlastnosti. Index představuje kolekci dokumentů, které jsou jednoznačně identifikovatelné názvem nebo aliasem. Tento jedinečný název je důležitý, protože se používá k zaměření indexu při vyhledávacích dotazech. Datový proud je pak abstrakce indexu pro data, která jsou opatřená pouze časovými razítky. Index je zde automaticky generován na základě skrytých záložních indexů [33].

ES serializuje a následně ukládá data ve formátu JSON dokumentů (viz. Obrázek 14). Tento dokument se skládá z množiny polí, které mají strukturu klíč-hodnota a obsahují konkrétní data. Každý dokument je následně možné přesně identifikovat pomocí unikátního ID, které může být vytvořeno manuálně nebo automaticky vygenerováno [33].

```
{
    "_index": "files_prod",
    "_id": "QsSCQ5YBozgSNQc03Kmb",
    "_score": 1,
    "_source": {
      "_class": "cz.upce.file.FileDocument",
      "fileName": "test2.wav",
      "fileType": "audio/wav",
      "fileSize": 632472,
      "fileUploadDate": "1744889371505",
      "fileContent": "hello my name is john",
      "filePath": "D:\\virtual-storage-root\\storage\\test2.wav",
      "fileOwnerId": 1,
      "folderId": ""
    }
}
```


Obrázek 14: **Příklad Elasticsearch dokumentu (zdroj: autor práce)**

Indexovaný dokument zahrnuje jak samotná data, tak i metadata. Metadata představují systémové informace, které slouží k uložení údajů popisujících daný dokument, může jít např. o identifikátor dokumentu nebo typ obsahu. V ES mají pole metadat předponu podtržítko. Zároveň každý index obsahuje také mapování nebo schéma pro indexování polí v dokumentech. Pomocí mapování se definuje datový typ každého pole, způsob indexování a způsob uložení pole. V rámci ES existují dva systémy mapování. Dynamické mapování, ES automaticky detekuje datové typy a vytváří mapování. Explicitní mapování, mapování je definováno pomocí předem zadaných datových typů pro každé pole [33].

Pro vyhledávání v rámci ES se používá API, které interaguje s dotazovacím jazykem [34]. Jako primární dotazovací jazyk se používá Domain Specific Language (DLS) založený na JSON pro definici dotazů. V rámci DLS pak existují dvě dotazovací klauzule. Listové dotazovací klauzule hledají konkrétní hodnotu v určitém poli, jako je tomu u dotazů na shodu, termín nebo rozsah. Tyto dotazy lze použít samostatně. Složené dotazovací klauzule obalují jiné listové nebo složené dotazy a používají se k logickému spojení více dotazů. Některé typy dotazů se však mohou provádět pomaleji kvůli způsobu jejich implementace, což může ovlivnit stabilitu clusteru. Tyto dotazy lze rozdělit do kategorií dotazy na scripty, dotazy na číselná pole, data, boolean, neindexovaná klíčová slova atd. DLS má poté koncový bod _search, který přijímá několik typů vyhledávání [35].

4.2 Typy vyhledávání

Vyhledávání v rámci Elasticsearch je možné rozdělit na několik typů, podle toho, jaký vyhledávací nástroj či algoritmus se použije:

- Fulltextové vyhledávání je vyhledávání v textu, který byl analyzován a indexován. Podporuje frázové nebo proximitní dotazy, fuzzy shody a další.
- Vyhledávání podle klíčových slov je vyhledávání podle přesné shody polí pro klíčová slova.
- Sémantické vyhledávání vyhledává v sémantických polích textu pomocí hustého nebo řídkého vektorového vyhledávání.
- Vektorové vyhledávání je vyhledávání podobně hustých vektorů pomocí algoritmu kNN pro embeddings.
- Geoprostorové vyhledávání je vyhledávání míst a výpočet prostorových vztahů pomocí geoprostorových dotazů [34].

5 Použité technologie

5.1 Spring Boot

Spring Boot je opensourcový framework, vývojářům výrazně usnadňuje tvorbu mikroslužeb a webových serverových aplikací v programovacím jazyce Java. Vychází z oblíbeného Spring frameworku, který je postaven na principu předem předpřipravených částí kódu. Tyto části kódu lze pak snadno rozšiřovat a upravovat podle specifických potřeb dané aplikace. Díky tomuto je následný vývoj aplikace nejen rychlejší, ale také méně náchylný na chyby, což zvyšuje bezpečnost [36]. Mezi hlavní důvody popularity tohoto frameworku patří možnost snadného vkládání závislostí, což umožnuje vývojářům definovat si potřebné závislosti, bez nutnosti je manuálně spravovat, o správu se stará Spring kontejner. Tento mechanismus je základem pro tvorbu modulárních aplikací z navzájem propojených komponent. Spring framework zároveň také nabízí vestavěnou podporu pro časté úkoly, jako jsou validace, převod dat, pánování úloh nebo práce s událostmi [37].

Mikroslužby představují architektonický přístup ve kterém je vyvíjená aplikace rozdělena na samostatné části neboli komponenty. Každá část pak plní konkrétní funkci a je nezávislá na ostatních, nicméně části aplikace spolu navzájem komunikují prostřednictvím definovaných rozhraní API. Díky tomu, že jsou na sobě jednotlivé části aplikace nezávislé nedojde při selhání jedné z nich k pádu celé aplikace [36] [38].

Spring Boot je pak navržen tak, aby co nejvíce zjednodušil konfiguraci, nastavení a nasazení Spring aplikací. To je umožněno třemi hlavními funkcemi Spring Bootu, které zahrnují automatickou konfiguraci, názorový přístup a samostatné aplikace. Automatická konfigurace využívá tzv. přednastavené závislosti, které výrazně šetří čas při nastavování. Díky tomu je možné např. nakonfigurovat databázi nebo server bez nutnosti manuálního zadávání všech parametrů. Dále Spring Boot umožnuje využívat názorový přístup, který slouží k přidávání a konfigurování nejdůležitějších vlastností. Ostatní vlastnosti jsou automaticky přednastaveny podle potřeb projektu. Spring Boot rovněž poskytuje velké množství startovacích závislostí tzv. Spring Starterů, což jsou balíčky, které obsahují přednastavené komponenty pro různé typy aplikací. Může se tedy jednat o komponenty pro práci s webem, zabezpečením nebo databázemi. Celkově Spring Boot zahrnuje více než 50 startovacích balíčků a výrazně usnadňuje také integraci s nástroji třetích stran. [37].

V rámci tabulky (Tabulka 3) je znázorněna architektura Spring Boot aplikace. Tabulka shrnuje jednotlivé vrstvy aplikace, jejich odpovědnosti a používané anotace. Na obrázku níže (Obrázek 15) je poté uveden konkrétní příklad zdrojového kódu.

Část aplikace	Popis
Projektová struktura	Vytvořená pomocí Maven nebo Gradle. Závislosti se definují pomocí spring-boot-starter-* balíčků.
Aplikační třída	Obsahuje anotaci @SpringBootApplication. Slouží jako výchozí bod běhu aplikace.
REST kontrolery	Třídy s anotací @RestController. Zpracovávají http požadavky a vracejí odpovědi.
Datová vrstva	Používají anotace @Entity a @Repository. Pracuje s databází pomocí Spring Data JPA.
Business logika	Obsahuje anotaci @Service. Definuje logiku operací a aplikační pravidla.

Tabulka 3: **Přehled architektury aplikace Spring Boot (zdroj: autor práce)**

```
@SpringBootApplication
public class MyApp {
    public static void main(String[] agrs) {
        SpringApplication.run(MyApp.class, agrs);
    }
}

@RestController
@RequestMapping("/api")
public class HelloController() {
    @GetMapping("/hello")
    public String sayHello() {
        return "Hello from Spring Boot";
    }
}
```

Obrázek 15: **Příklad kódu aplikace ve Spring Boot (zdroj: autor práce)**

5.1.1 Java

Java je oblíbený objektově orientovaný programovací jazyk, který se používá k vývoji mobilních, webových i podnikových aplikací. Díky své multiplatformní povaze lze jednou napsaný kód spustit na různých operačních systémech bez nutnosti provádění úprav. To je

umožněno díky virtuálnímu počítači Java (JVM), který překládá kód do spustitelné podoby. Java je využívána miliardami zařízení po celém světe [39] [41].

5.2 Docker

Docker je otevřená platforma umožňující vývoj, distribuci a spouštění aplikací. Hlavní výhodou je schopnost oddělit jednotlivé aplikace od infrastruktury, díky čemuž je výrazně zjednodušen a urychlen celý proces dodání softwaru. S využitím této platformy při vývoji, testování a nasazování lze výrazně zkrátit čas potřebný k uvedení softwaru do provozu [40].

5.2.1 Funkce platformy

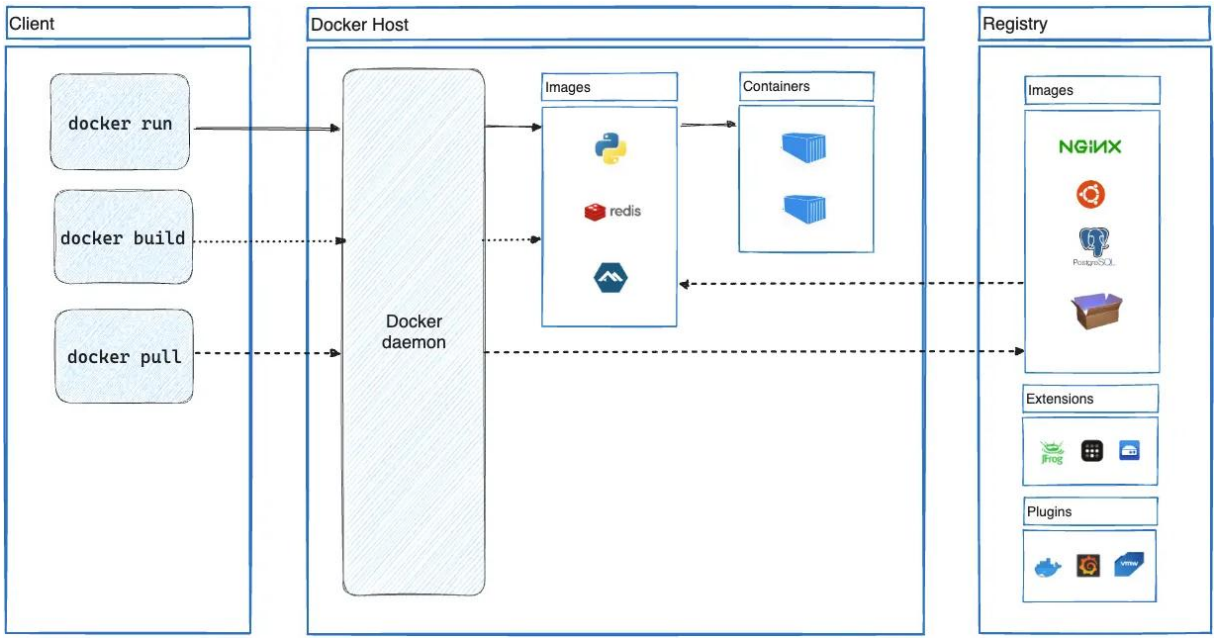
Docker platforma poskytuje prostředí, ve kterém je umožněno zabalovat aplikace a spouštět je v rámci jednotlivých kontejnerů, což jsou izolovaná prostředí, která běží nezávisle na hostitelském systému. Díky izolaci je možné provozovat více kontejnerů paralelně bez vzájemného ovlivnění. Jednotlivé kontejnery obsahují veškeré potřebné komponenty pro běh aplikace. Velkou výhodou je i možnost přenosu a sdílení kontejnerů mezi jednotlivými systémy bez nutnosti úprav [40].

5.2.2 Využití

Využití kontejnerů výrazně usnadňuje vývojový cyklus aplikací, protože vývojáři pracují v jednotném prostředí. Kontejnerizace navíc podporuje moderní přístupy jako je CI/CD. Díky lehkosti, flexibilitě a přenositelnosti kontejnerů je možné efektivně spravovat pracovní zátěže, škálování či rušení aplikací a služeb, a to podle aktuálních potřeb. Tyto operace lze provádět téměř v reálném čase. Docker je ideální pro prostředí s vysokou hustotou a pro malá a střední nasazení, ve kterých je nutné dosáhnout co nejvyšší efektivity [40].

5.2.3 Architektura

Docker je postaven na modelu klient-server. Komunikace mezi těmito částmi probíhá prostřednictvím UNIX socketů, REST API nebo síťového rozhraní. Serverová část tzv. Docker démon má za úkol sestavování, spuštění a distribuci kontejnerů (viz. Obrázek 16). Klient i démon mohou běžet na stejném zařízení, případně se klient může připojovat také vzdáleně. Pro správu komplexnějších aplikací, skládajících se z více kontejnerů je vyžíván nástroj Docker Compose [40].



Obrázek 16: **Architektura Dockeru**[11]

Démon Dockeru (dockerd) naslouchá příchozím požadavkům přes rozhraní API a zajišťuje správu klíčových komponent systému, jako jsou kontejnery, svazky, sítě a obrazy. Kromě toho může také spolupracovat i s dalšími démony a podílet se na správně služeb v rámci větší infrastruktury.

Klient Dockeru (docker) slouží jako nástroj, pomocí kterého uživatelé komunikují s démonem. Při zadání příkazu (např. docker run) je vytvořen požadavek, který je následně předán na démon k provedení příslušné operace. Klient se může připojovat k více démonům, buď lokálně nebo vzdáleně, komunikace pak probíhá přes definované API.

Pro spuštění kontejnerů jsou využívány obrazy, které se uchovávají v registrech dockeru. Tyto registry je možné rozdělit na veřejné a soukromé. Nejznámějším veřejným registrem je Dockeru Hub, ze kterého Docker automaticky stahuje ve výchozím nastavení obrazy. Ke stažení obrazu slouží příkaz docker pull, případně docker run, kdy dojde ke stažení obrazu z nakonfigurovaného registru, zatímco příkaz docker push pak nahrává obraz zpět do nakonfigurovaného registru [40].

[11] What is Docker? Online. In: Docs.docker.com. 2025. Dostupné také z: https://docs.docker.com/get-started/docker-overview/.

5.2.4 Objekty

Docker pracuje s několika klíčovými objekty, které jsou základem jeho fungování. Mezi tyto objekty patří obrazy, kontejnery, sítě, svazky, zásuvné moduly a další komponenty, které jsou nezbytné pro provoz prostředí a aplikací v nich.

Obraz (image) slouží v prostředí Dockeru jako šablona, která obsahuje instrukce potřebné ke spuštění kontejneru. Většina obrazů vychází z jiných obrazů, které jsou dále upraveny podle konkrétních požadavků. Například je možné vytvořit obraz na základě systému Debian a doplnit jej o webový server nebo vlastní aplikaci. Jednotlivé obrazy pak mohou být uloženy ve věřených či privátních registrech, odkud jsou následně stahovány. Je zde také možnost vytvoření kompletně nového obrazu s využitím Dockerfile souboru, ve kterém je popsán proces sestavení kontejneru krok za krokem. Každý příkaz v tomto souboru reprezentuje samostatnou vrstvu. Pokud tedy dojde k úpravě tohoto souboru, je po opětovném spuštění upravena pouze změněná vrstva, což přispívá k efektivitě celého systému a vyšší rychlosti v pórování s ostatními virtualizačními technologiemi.

Kontejnery jsou spustitelné instance, vytvořené z obrazů, které je možné pomocí API nebo CLI vytvářet, spouštět, kopírovat, mazat, přesouvat či duplikovat (viz. Obrázek 17). Zároveň každý kontejner je možné připojit k jedné nebo více sítím, přidat k nim uložiště nebo jej použít k tvorbě nového obrazu. Ve výchozím nastavení jsou jednotlivé kontejnery dobře izolovány vůči sobě navzájem i vůči hostiteli. Tuto izolaci je však možné ovlivňovat. Kontejner je definován pomocí své bitové kopie a konfiguracemi, které jsou mu poskytnuty při jeho spuštění. Při mazání kontejneru jsou smazány všechny jeho změny, které nejsou uloženy v připojeném uložišti [40].

```
$ docker run -i -t ubuntu /bin/bash
```

Obrázek 17: **Příklad vytvoření kontejneru ubuntu a připojení se do jeho terminálu (zdroj: autor práce)**

6 Návrh aplikace uložiště

6.1 Cíle aplikace

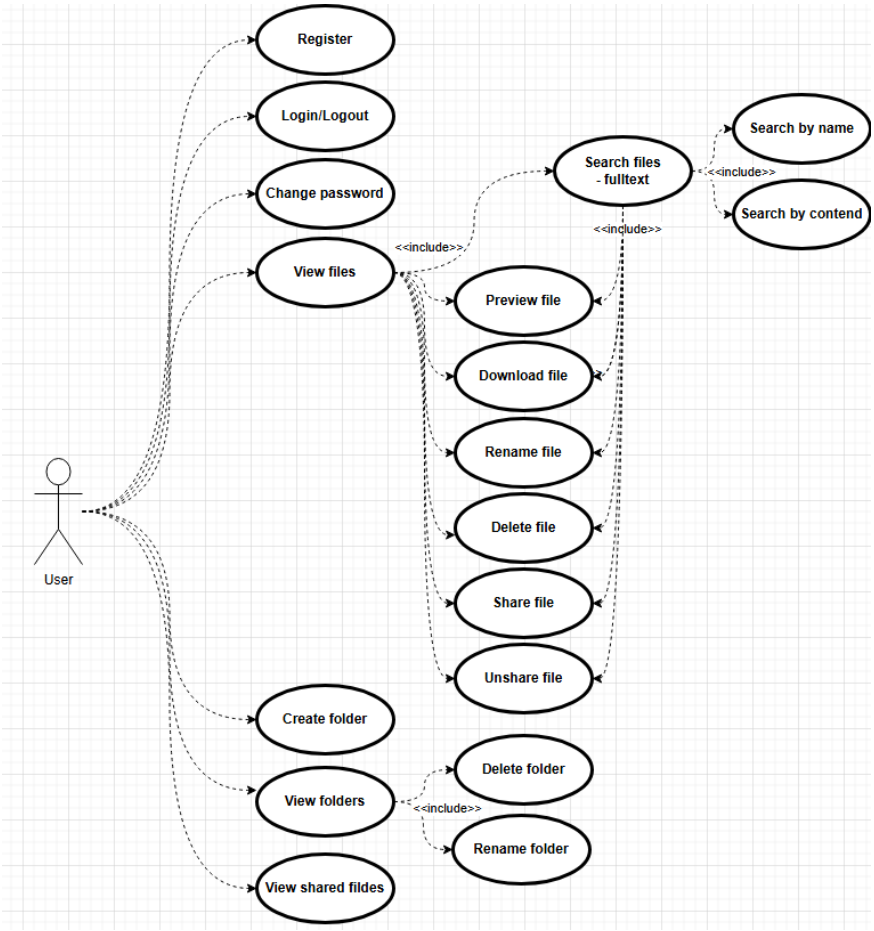
Hlavním cílem při tvorbě aplikace bylo poskytnou uživatelům jednoduchý a intuitivní nástroj, který se svými funkcionalitami blíží aplikacím poskytovatelů cloudových uložišť. Aplikace umožňuje uživatelům nahrávání, sdílení, správu a vyhledávání v rámci souborů a jejich obsahů prostřednictvím webového rozhraní. Podporovány jsou běžné typy dokumentů, jako jsou PDF, Word, Excel, obrázky či zvukové soubory. Systém využívá následující technologie pro zpracování obsahu souborů:

- Apache Tika – extrakce textů z dokumentů
- Tesseract OCR – extrakce textů z obrázků
- Vosk a ffmpeg – převod hlasu na text
- Elasticsearch – ukládání a fulltextové vyhledávání obsahu

Hlavním záměrem aplikace je tedy uživateli umožnit nahrávat a spravovat své soubory, rychle soubory vyhledávat pomocí názvu nebo pomocí obsahu uvnitř souboru a sdílet soubory s ostatními uživateli.

6.2 Use-case diagram

Níže uvedený diagram (Obrázek 18) popisuje základní funkce, které mohou uživatelé provádět v rámci aplikace.



Obrázek 18: **Use-case diagram uživatele (zdroj: autor práce)**

6.3 Funkční požadavky

V rámci této sekce jsou vymezeny konkrétní funkcionality, které musí aplikace poskytovat koncovému uživateli či administrátorovi. Požadavky vycházejí ze zadání bakalářské práce, analýz dalších aplikací a potřeb uživatelů

Uživatelské funkce:

- Registrace a přihlášení uživatele do systému
- Nahrávání souboru
- Tvorba a správa složek (přejmenování, mazání)
- Náhled soubor (obrázky, PDF, audio)
- Stahování, přejmenování a mazání souborů

- Sdílení souborů s jinými uživateli aplikace
- Zrušení sdílení
- Vyhledávání podle názvu
- Vyhledávání podle obsahu

Administrátorské funkce:

- Zobrazení stavu aplikace
- Zobrazení systémových metrik

6.4 Nefunkční požadavky

Nefunkční požadavky přímo nesouvisejí s konkrétními funkcionalitami ani se zadáním bakalářské práce, jde spíše o kvalitu a dostupnost systému. Jsou zde uvedeny tedy pouze věci, které by bylo vhodné vylepšit, případně dopracovat v dalších verzích aplikace.

- Výkon
- Škálovatelnost
- Vylepšená bezpečnost
- Grafické rozhraní
- Lokalizace

7 Architektura systému

7.1 Architektonický návrh

Aplikace je rozdělena do několika modulů, které spolu vzájemně komunikují a utváří komplexní celek.

7.1.1 Frontend

Jedná se o klientskou část aplikace, která zajišťuje uživatelské rozhraní, interakci s uživatelem a volání backendových REST API. Frontend využívá technologie HTML, CSS s využitím Bootstrap a Javascript.

7.1.2 Backend

Serverová část aplikace postavena v jazyce Java s využitím frameworku Spring Boot. Backend zajišťuje autentizaci a autorizaci uživatelů, správu souborů a složek, rozhraní pro vyhledávání, komunikaci s databází, elasticsearch a zpracování dat.

7.1.3 Databáze

Slouží pro ukládání informací o uživatelích jako jsou přihlašovací údaje.

7.1.4 Elasticsearch

Vyhledávací nástroj, který poskytuje fulltextové vyhledávání nad obsahem dokumentů.

7.1.5 OCR

Využíván pro extrakci textů z obrázků, případně z PDF dokumentů, které obsahují obrázky.

7.1.6 Vosk

Služba, která poskytuje převod řeči v audio souborech na text.

7.1.7 Ffmpeg

Nástroj, který umožnuje převádět audio soubory do vhodných formátu, využívá se společně se službou Vosk.

7.2 Nasazení pomocí Dockeru

Pro zajištění snadného nasazení, přenositelnosti a oddělení jednotlivých komponent je aplikace provozována v kontejnerech dockeru.

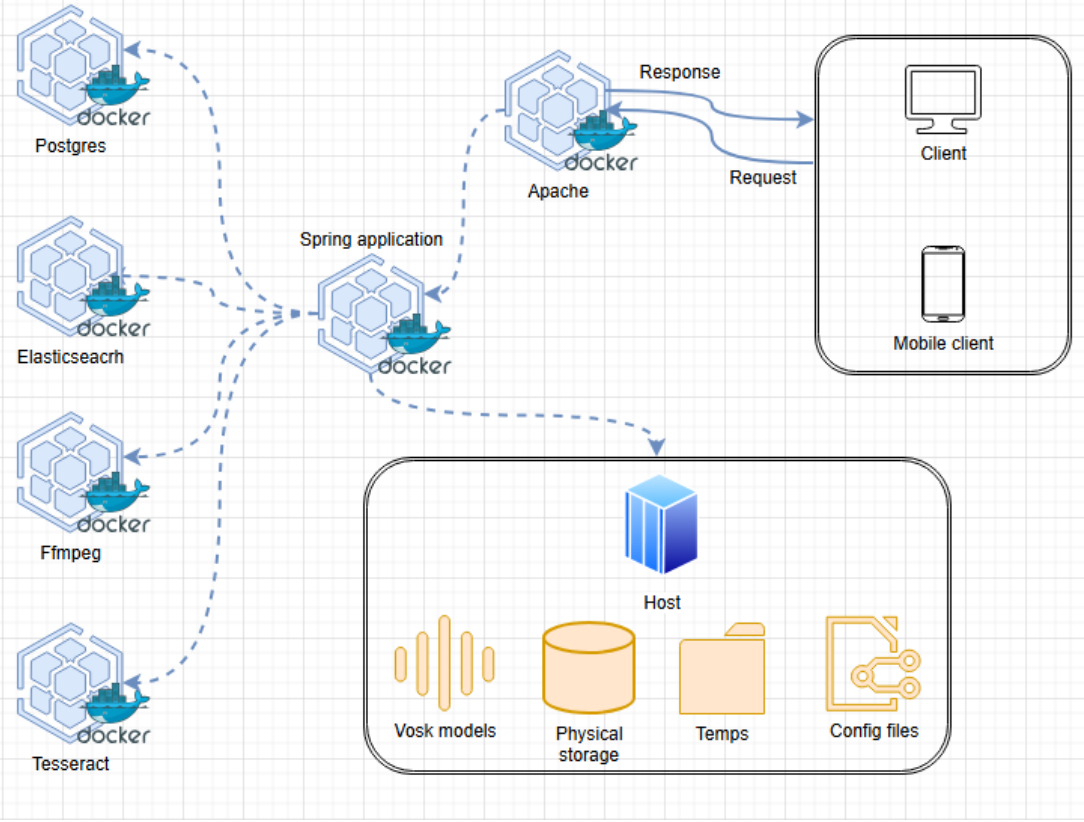
- Aplikační kontejner – obsahuje kompletní Spring Boot aplikaci, která komunikuje s ostatními službami

- Postgres kontejner – poskytuje objektově-relační databázový systém, ke kterému je aplikace připojena pomocí JPA (Hibernate) + PostgreSQL JDBC driver
- Elasticsearch kontejner – nastroj, který poskytuje fulltextové vyhledávání nad indexovaným obsahem souborů. Komunikace s aplikací probíhá přes REST API.
- Ffmpeg kontejner – poskytuje nástroj ke konverzi audio formátu před zpracováním. Aplikace se přímo připojuje do kontejneru pomocí CLI a provádí konverzi.
- Tesseract kontejner – poskytuje nástroj pro extrakci textů z obrázků. Aplikace se přímo připojuje do pomocí CLI a následně provádí extrakci.
- Apache kontejner – poskytuje webový server, který je používán jako reverzní proxy a přesměrovává požadavky z domény na aplikaci.

Služba Vosk není samostatně kontejnerizována, aplikace využívá knihovnu vosk-api, pomocí které komunikuje s lokálními modely na disku. Tyto modely jsou mountovány do aplikačního kontejneru, aby k nim měla aplikace přístup.

7.2.1 Diagram nasazení

Níže na obrázku (Obrázek 19) je znázorněna architektonická struktura systému a způsob propojení mezi jednotlivými kontejnery a adresáři na hostitelském systému.



Obrázek 19: **Architektura systému (zdroj: autor práce)**

8 Uživatelský návrh a chování

8.1 Uživatelé aplikace

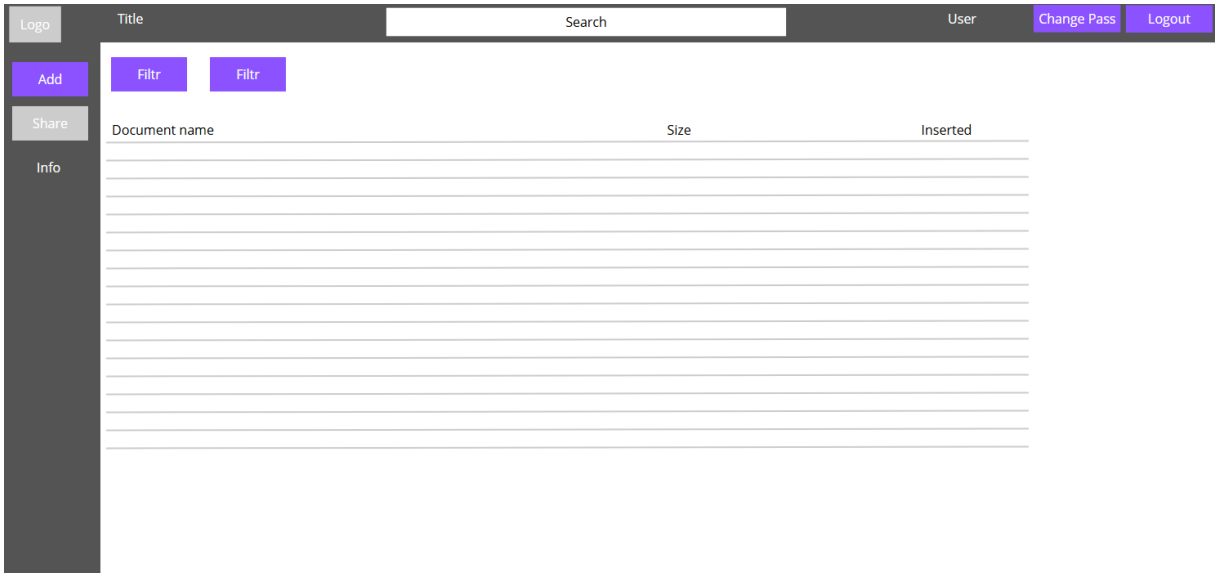
Do aplikace se může registrovat jakýkoliv uživatel. Uživatelé mají následně přístup ke všem základním funkcím uložiště, jako je nahrávání souboru, sdílení a rušení sdílení souborů, vyhledávání dle názvu a obsahu, správa souboru a složek atd.

Aplikace zároveň při nasazení má v databázi definovaného jednoho administrátorského uživatele, který má kromě výše uvedených funkcí přístup k administračnímu rozhraní, ve kterém může sledovat stav aplikace a její metriky. Administrátor je v aplikaci identifikován dle ID 1 v databázi.

8.2 Návrh uživatelského rozhraní

Návrh uživatelského rozhraní (UX) vychází z požadavků na jednoduchost, srozumitelnost, přehlednost a efektivní práci s dokumenty. Cílem UX je tedy prostředí, které bude pro běžné uživatele intuitivní bez potřeby školení, a zároveň umožní rychlý přístup k systémovým funkcím, tedy náhradní dokumentu, vyhledávání zobrazení atd.

Pro vizualizaci návrhu byl vytvořen jednoduchý wireframe (viz. Obrázek 20), který ukazuje rozmístění prvků na hlavním panelu uložiště uživatele. Tento návrh sloužil jako podklad pro následnou implementaci uživatelského rozhraní.



Obrázek 20: **Wireframe pro UX (zdroj: autor práce)**

9 Implementace klíčových komponent

9.1 Backend a Spring Boot

Backend aplikace je postaven v jazyce Java (JDK 17) s využitím frameworku Spring Boot (verze 3.4.2), který umožnuje rychlý vývoj REST služeb, snadnou integraci s databází, práci s bezpečností a dalšími externími službami

Struktura backendové části je složena z několika vrstev.

9.1.1 Controller

Controller slouží jako vstupní bod http požadavků, mapuje jednotlivé endpontiy, zpracovává vstupy od uživatele a vrací odpovědi. Zároveň také obsahuje části business logiky. V rámci aplikace jsou využívány následující controllery:

- *ErrorController* – slouží ke zpracování chybových http odpovědí a přesměrování uživatele na odpovídající chybovou stránku
- *PageController* – mapuje URL na šablony a vrací název šablony, která se má vykreslit
- *FileController* – zpracovává http požadavky na soubory, provádí implementované operace a vrací uživateli odpovědi.
- *FolderController* – zpracovává http požadavky na složky, provádí implementované operace a vrací uživateli odpovědi
- *UserController* – zpracovává http požadavky na uživatele, provádí implementované operace a vrací uživateli odpovědi
- *InfoController* – zpracovává http požadavky na systém, provádí implementované operace a vrací uživateli odpovědi

```java
@GetMapping("/preview/{id}")  petrsafrata
public ResponseEntity<Resource> previewFile (@PathVariable String id, Authentication authentication){
    Optional<FileDocument> fileDocuments = fileService.findFileById(id);
    if (fileDocuments.isPresent()) {
        SecurityPrincipalUser principal = (SecurityPrincipalUser) authentication.getPrincipal();
        Integer userId = (int) principal.getUser().getId();
        if (!userId.equals(fileDocuments.get().getFileOwnerId())
                && (fileDocuments.get().getFileSharedUsersIds() == null
                || !fileDocuments.get().getFileSharedUsersIds().contains(userId))) {
            return ResponseEntity.status(HttpStatus.FORBIDDEN).build();
        }
        Path filePath = uploadDir.resolve(fileDocuments.get().getFileName());
        try {
            Resource resource = new UrlResource(filePath.toUri());
            if (resource.exists()) {
                switch (fileDocuments.get().getFileType()) {
                    case "image/jpeg":
                        return ResponseEntity.status(HttpStatus.OK).contentType(MediaType.IMAGE_JPEG).body(resource);
                    case "image/png":
                        return ResponseEntity.status(HttpStatus.OK).contentType(MediaType.IMAGE_PNG).body(resource);
                    case "application/pdf":
                        return ResponseEntity.status(HttpStatus.OK).contentType(MediaType.APPLICATION_PDF).body(resource);
                    default:
                        return ResponseEntity.status(HttpStatus.UNSUPPORTED_MEDIA_TYPE).build();
                }
            } else {
                logger.warn("Can not preview file.");
                return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
            }
        } catch (MalformedURLException e) {
            ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body(e.getMessage());
        }
    }
    return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
}
```

Obrázek 21: **Ukázka implementace metody previewFile v rámci FileControlleru (zdroj: autor práce)**

9.1.2 Service

Services obsahují části business logiky, která je volána z controllerů, integraci s repositories a externími službami. Aplikace využívá následující services:

- *InfoServices* – obsahuje logiku pro zpřístupnění systémových informací aplikace a také generuje systémové release notes do html podoby pomocí knihovny flexmark
- *FileServices* – obsahuje metody pro práci s *FileRepository*
- *FolderService* – obsahuje metody pro práci s *FolderRepository*
- *SecurityUserDetailService* – obsahuje logiku načtení uživatele z databáze
- *TesseractService* – obsahuje implementaci a napojení na službu Tesseract OCR
- *TikaService* – obsahuje implementaci a napojení na službu Tika a Tesseract OCR
- *VoskService* – obsahuje implementaci a napojení na službu Vosk

```java
@Service  4 usages  petrsafrata
public class TikaService {

    private static final Logger logger = LoggerFactory.getLogger(TikaService.class);  2 usages

    public String doTikaExtractWithOcr(File fie, TesseractService tesseractService) {  1 usage  petrsafrata
        AutoDetectParser parser = new AutoDetectParser();
        BodyContentHandler bodyHandler = new BodyContentHandler();
        OcrExtractor ocrExtractor = new OcrExtractor(tesseractService);
        ParseContext context = new ParseContext();

        context.set(EmbeddedDocumentExtractor.class, new OcrExtractor(tesseractService));
        Metadata metadata = new Metadata();

        try (InputStream inputStream = new FileInputStream(fie)) {
            parser.parse(inputStream, new EmbeddedContentHandler(bodyHandler), metadata, context);
        } catch (TikaException | IOException | SAXException e) {
            logger.error("Tika extract error: " + e.getMessage());
            throw new RuntimeException(e);
        }

        String tikaText = bodyHandler.toString();
        String ocrText = ocrExtractor.getOcrResult();
        return tikaText + "\n" + ocrText;
    }
```

Obrázek 22: **Ukázka implementace extrakce textu pomocí TikaService (zdroj: autor práce)**

9.1.3 Repository

Repositories jsou rozhraní, sloužící pro komunikaci aplikace s databází a s Elasticseach. Aplikace využívá následující repositories:

- *FileRepository* – slouží pro práci se souborovými dokumenty, které jsou uloženy v indexu Elasticsearch. Rozhraní dědí z *ElasticsearchRepository*.
- *FolderRepository* – slouží pro práci se logickými složkami, které jsou uloženy v indexu Elasitcsearch. Rozhraní dědí z *ElasticsearchRepository*.
- *UserRepository* – slouží pro správu uživatelů uložených v databázi. Rozhraní dědí z *JpaRepository*.

```java
public interface FileRepository extends ElasticsearchRepository<FileDocument, String> {  2 usages  petrsafrata

    @Query("{\"bool\": {\"must\": [ {\"wildcard\": {\"fileName.keyword\": {\"value\": \"*?0*\"}}}, {\"term\": {\"fileOwnerId\": \"?1\"}} ]}}")  1 usage  petrsafrata
    List<FileDocument> findByFileName(String query, Integer ownerId);

    @Query("{\"bool\": {\"must\": [ {\"match\": {\"fileContent\": {\"query\": \"?0\"}}}, {\"term\": {\"fileOwnerId\": \"?1\"}} ]}}")  1 usage  petrsafrata
    List<FileDocument> findByFileContent(String query, Integer ownerId);

    @Query("{\"bool\": {\"must\": [ {\"wildcard\": {\"fileName.keyword\": {\"value\": \"*?0*\"}}}, {\"term\": {\"fileSharedUsersIds\": \"?1\"}} ]}}")  1 usage  petrsafrata
    List<FileDocument> findByFileNameAndShareId(String query, Integer shareId);

    @Query("{\"bool\": {\"must\": [ {\"match\": {\"fileContent\": {\"query\": \"?0\"}}}, {\"term\": {\"fileSharedUsersIds\": \"?1\"}} ]}}")  1 usage  petrsafrata
    List<FileDocument> findByFileContentAndShareId(String query, Integer shareId);

    @Query("{\"bool\":{\"must\":[{\"wildcard\":{\"fileName.keyword\":{\"value\":\"*?0*\"}}},{\"term\":{\"folderId.keyword\":\"?1\"}}, {\"term\": {\"fileOwnerId\": \"?2\"}} ]}}")
    List<FileDocument> findByFileNameAndFolder(String query, String folder, Integer ownerId);

}
```

Obrázek 23: **Ukáza implementace FileRepository (zdroj: autor práce)**

9.1.4 Data Transfer Object (DTO)

Slouží pro přenos dat mezi backendem a frontendem, v rámci aplikace jsou využívány následující DTO:

- *FolderDto* – přenos dat složky
- *UserDto* – přenos dat uživatele

```
public class UserDto {  3 usages   petrsafrata

    private String email;  1 usage
    private String password;  1 usage

    public String getEmail() {   petrsafrata
        return email;
    }

    public String getPassword() {   petrsafrata
        return password;
    }
}
```

Obrázek 24: **Ukázata implementace UserDto (zdroj: autor práce)**

9.1.5 Entity

Entity reprezentují konkrétní tabulky v databázi a slouží pro práci s perzistencí dat prostřednictvím Spring Data JPA. V rámci aplikace je využívána pouze entita *UserEntity* pro práci s tabulkou users v databázi.

```
@Entity  21 usages  petrsafrata
@Table(name = "users")
public class UserEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;

    @Column(unique = true, nullable = false)  2 usages
    private String email;

    @Column(nullable = false)  2 usages
    private String password;

    public UserEntity() {  petrsafrata
    }

    public long getId() {  petrsafrata
        return id;
    }

    public String getEmail() {  petrsafrata
        return email;
    }

    public String getPassword() {  petrsafrata
        return password;
    }

    public void setEmail(String email) {  petrsafrata
        this.email = email;
    }

    public void setPassword(String password) {  petrsafrata
        this.password = password;
    }
}
```

Obrázek 25: **Ukázka implementace UserEntity (zdroj: autor práce)**

9.1.6 Document

Documents reprezentují konkrétní index v rámci Elasticsearch a slouží jako datový model pro jeho správu a vyhledávání. V rámci aplikace jsou využívány následující documents

- *FileDocument* – reprezentuje index dokumentů v rámci Elasicsearch a jeho jednotlivá pole
- *FolderDocument* – reprezentuje index složek v rámci Elasitcsearch a jeho jednotlivá pole

```java
@Document(indexName = "folders_prod")  14 usages  petrsafrata
public class FolderDocument {

    @Id
    private String folderId;

    @Field(type = FieldType.Text)  2 usages
    private String folderName;

    @Field(type = FieldType.Integer)  2 usages
    private Integer folderOwnerId;

    @Field(type = FieldType.Text)  2 usages
    private String parentFolderId;

    public String getFolderId() {  2 usages  petrsafrata
        return folderId;
    }

    public void setFolderId(String folderId) {  no usages  petrsafrata
        this.folderId = folderId;
    }

    public String getParentFolderId() {  no usages  petrsafrata
        return parentFolderId;
    }

    public void setParentFolderId(String parentFolderId) {  1 usage  petrsafrata
        this.parentFolderId = parentFolderId;
    }

    public Integer getFolderOwnerId() {  2 usages  petrsafrata
        return folderOwnerId;
    }

    public void setFolderOwnerId(Integer folderOwnerId) {  1 usage  petrsafrata
        this.folderOwnerId = folderOwnerId;
    }

    public String getFolderName() {  no usages  petrsafrata
        return folderName;
    }

    public void setFolderName(String folderName) {  2 usages  petrsafrata
        this.folderName = folderName;
    }
}
```

Obrázek 26: **Ukázka implementace FolderDocument (zdroj: autor práce)**

9.2 Integrace PostgreSQL

Databázová vrstva je v systému realizována pomocí PostgreSQL, který běží v samostatném kontejneru. Aplikace pak s databází komunikuje prostřednictvím JPA (Hibernate)

a PostgreSQL JDBC driver. Díky tomu je zajištěno snadné mapování entit na relační struktury a jednoduchá práce s uloženými daty.

Aplikace používá klasické mapování JPA pomocí anotací Entity a dalších. Pro přístup k uživatelům v databázi slouží *UserRepository*, které dědí z *JpaRepository* a umožnuje dotazování na data pomocí metod.

9.3 Integrace Elasticsearch

V rámci aplikace je umožněno fulltextové vyhledávání jak v názvech souborů, tak i v jejich obsahu. K tomu je využívána služba Elasticsearch, která je nasazena v samostatném kontejneru dockeru. Aplikace pak se službou komunikuje prostřednictvím http požadavků, skrze *ElasticsearchClient*. Při nahrávání souboru je obsah souboru extrahován pomocí jedné z uvedených služeb a následně je tento obsah uložen do indexu ES. Typy polí v indexu jsou definovány tak, aby umožňovali efektivní fulltextové vyhledávání. V rámci vyhledávání se pracuje s následujícími dotazy:

- Wildcard – používá se pro vyhledávání dle názvu souboru v poli fileName.keyword, což je nemodifikovaná hodnota názvu souboru
- Match – používá se pro analyzované fulltextové vyhledávání v obsahu souboru, pole fileContent je rozděleno na tokeny, ve kterých se hledá relevantní shoda

Tímto způsobem je možné prohledávat dokumenty dle slovních spojení nebo klíčových výrazů, a to napříč různými typy souborů.

```
@Configuration  petrsafrata
public class ElasticSearchConfig {

    @Value("${app.elasticsearch.url}")
    private String elasticsearchHost;

    @Bean  petrsafrata
    public ElasticsearchClient elasticsearchClient() {
        RestClient restClient = RestClient.builder(HttpHost.create(elasticsearchHost)).build();
        return new ElasticsearchClient(new RestClientTransport(restClient, new JacksonJsonpMapper()));
    }
}
```

Obrázek 27: **Integrace Elasticsearch (zdroj: autor práce)**

9.4 Integrace Tesseract OCR

Součástí systému je možnost extrakce textů z obrázků pomocí nástroje Tesseract OCR, který běží v samostatném kontejneru dockeru. Aplikační kontejner s tímto kontejnerem komunikuje pomocí CLI příkazů. Zároveň oba kontejnery sdílí temp složku, která slouží

jako výměnné uložiště pro dočasné soubory používané při OCR. Při využití OCR aplikace pracuje následujícím způsobem:

- Nahrání obrázkového souboru do systému
- Vytvoření dočasné kopie pro OCR
- Připojení do Tesseract kontejneru pomocí CLI a provedení OCR
- Získání výsledného textu a jeho uložení do ES

OCR je také využíváno společně s knihovnou Apache Tika, která umožňuje automatickou extrakci textového obsahu ze souboru. Díky napojení na OCR je možné ze souboru získat i textový obsah obrázků. Pro zajištění této komunikace je využita vlastní implementace třídy Tika *EmbeddedDocumentExtractor*, která využívá *TesseractService*. Výsledný text je poté složen z textů uvedených na obrázcích v dokumentu, extrahovaných prostřednictvím OCR a z textu dokumentu, který extrahovala Tika.

```java
@Service  8 usages  petrsafrata
public class TesseractService {

    private static final Logger logger = LoggerFactory.getLogger(TesseractService.class);  2 usages

    public String doTesseractOcr(File file) {  2 usages  petrsafrata
        try {
            String containerPath = "/temp/" + file.getName();
            ProcessBuilder pb = new ProcessBuilder( ...command: "docker", "exec", "tesseract", "tesseract", containerPath, "stdout", "-l", "eng");
            Process process = pb.start();

            String result = new BufferedReader(
                    new InputStreamReader(process.getInputStream(), StandardCharsets.UTF_8)
            ).lines().collect(Collectors.joining( delimiter: "\n"));

            String error = new BufferedReader(
                    new InputStreamReader(process.getErrorStream(), StandardCharsets.UTF_8)
            ).lines().collect(Collectors.joining( delimiter: "\n"));

            int exitCode = process.waitFor();

            if (exitCode != 0) {
                logger.error("Tesseract exited with code: " + exitCode + "\nError: " + error);
                throw new RuntimeException("Tesseract exited with code: " + exitCode + "\nError: " + error);
            }
            return result;

        } catch (Exception e) {
            logger.error("Tesseract error: " + e.getMessage());
            throw new RuntimeException("Tesseract error:", e);
        }

    }
```

Obrázek 28: **Integrace Tesseract OCR (zdroj: autor práce)**

```java
public class OcrExtractor implements EmbeddedDocumentExtractor {  4 usages  petrsafrata

    private final TesseractService tesseractService;  2 usages
    private final StringBuilder stringBuilder = new StringBuilder();  2 usages
    private static final Path tempDir = resolveTempDir();  1 usage

    private static Path resolveTempDir() {  1 usage  petrsafrata
        if (System.getProperty("os.name").toLowerCase().contains("win")) {
            return Paths.get( first: "D:/virtual-storage-root/temp");
        }
        return Paths.get( first: "/temp");
    }

    public OcrExtractor(TesseractService tesseractService) {  2 usages  petrsafrata
        this.tesseractService = tesseractService;
    }

    @Override  no usages  petrsafrata
    public boolean shouldParseEmbedded(Metadata metadata) {
        String type = metadata.get(Metadata.CONTENT_TYPE);
        return type != null && type.startsWith("image/");
    }

    @Override  no usages  petrsafrata
    public void parseEmbedded(InputStream inputStream, ContentHandler contentHandler, Metadata metadata, boolean b) throws SAXException, IOException {
        File tempFile = File.createTempFile( prefix: "ocr-", suffix: ".tmp", tempDir.toFile());
        try (OutputStream outputStream = new FileOutputStream(tempFile)) {
            byte[] buffer = new byte[8192];
            int bytesRead;
            while ((bytesRead = inputStream.read(buffer)) != -1) {
                outputStream.write(buffer,  off: 0, bytesRead);
            }
        }

        String ocrResult;
        try {
            ocrResult = tesseractService.doTesseractOcr(tempFile);
        } catch (RuntimeException e) {
            ocrResult = "OCR failed" + e.getMessage();
        }
        finally {
            tempFile.delete();
        }
        stringBuilder.append(ocrResult).append("\n");
    }
```

Obrázek 29: **Vlastní implementace EmbeddedDocumentExtractor (zdroj: autor práce)**

9.5 Integrace Vosk a Ffmpeg

Tato část systému umožnuje automatické rozpoznání řeči z nahraných zvukových souborů, pomocí knihovny Vosk. Ffmpeg zde slouží pouze jako nástroj pro konverzi zvukových souborů do vhodného formátu, vosk umí pracovat pouze se soubory ve formátu WAV, které mají 16 kHz. Vosk a Ffmpeg jsou od sebe navzájem odděleny. Ffmpeg je služba běžící v samostatném kotejneru dockeru. Aplikační kontejner s tímto kontejnerem komunikuje pomocí CLI příkazů. Zároveň oba kontejnery sdílí temp složku, která slouží jako výměnné uložiště pro dočasné soubory používané při konverzi formátu zvukových souborů. Vosk je implementován přímo v aplikací s využitím knihovny vosk. Model pro rozpoznávání, který knihovna využívá, je připojen přímo do aplikačního kontejneru. Při využití rozpoznání řeči aplikace pracuje následujícím způsobem:

- Nahrání zvukového souboru
- Vytvoření dočasné kopie pro konverzi
- Připojení do Ffmpeg kontejneru a konverze kopie zvukového souboru
- Rozpoznání řeči nad konvertovaným souborem
- Získání textu a uložení do ES

```java
@Service  4 usages  petrsafrata
public class AudioConvertor {

    private static final Logger logger = LoggerFactory.getLogger(AudioConvertor.class);  4 usages

    public File convertToWav(File file, Path tempDir) throws IOException, InterruptedException {  3 usages  petrsafrata
        File convertedWaw = File.createTempFile( prefix: "convertedWaw",  suffix: ".wav", tempDir.toFile());
        convertedWaw.deleteOnExit();

        String inputFilePath = "/temp/" + file.getName();
        String outputFilePath = "/temp/" + convertedWaw.getName();

        ProcessBuilder ffmpegBuilder = new ProcessBuilder( ...command: "docker", "exec", "ffmpeg", "ffmpeg", "-y", "-nostdin", "-i"
                , inputFilePath, "-ac", "1", "-ar", "16000", "-sample_fmt", "s16", outputFilePath);
        ffmpegBuilder.redirectErrorStream(true);
        Process ffmpeg = ffmpegBuilder.start();
        logger.info("FFmpeg audio convertor started");

        try (BufferedReader reader = new BufferedReader(new InputStreamReader(ffmpeg.getInputStream()))) {
            String outputLine;
            while ((outputLine = reader.readLine()) != null) {
                logger.info(outputLine);
            }
        }

        int statusCode = ffmpeg.waitFor();
        if (statusCode != 0) {
            logger.error("FFmpeg audio convertor error with code: " + statusCode);
            throw new IOException("FFmpeg audio convertor error with code: " + statusCode);
        }
        logger.info("FFmpeg audio convertor finished");
        return convertedWaw;
    }
}
```

Obrázek 30: **Integrace Ffmpeg (zdroj: autor práce)**

```java
@Service  4 usages  petrsafrata
public class VoskService {

    private static final Logger logger = LoggerFactory.getLogger(VoskService.class);  no usages
    private static final String modelPath = resolveModelPath();  1 usage

    private static String resolveModelPath() {  1 usage  petrsafrata
        if (System.getProperty("os.name").toLowerCase().contains("win")) {
            return "D:/virtual-storage-root/vosk-model/vosk-model-en-us-0.22";
        }
        return "/model";
    }

    public String doVoskAudio (File wavFile) throws IOException {  3 usages  petrsafrata
        LibVosk.setLogLevel(LogLevel.INFO);
        Model model = new Model(modelPath);
        InputStream ais = new FileInputStream(wavFile);
        Recognizer recognizer = new Recognizer(model,  sampleRate: 16000);

        int nbytes;
        byte[] b = new byte[4096];
        while ((nbytes = ais.read(b)) >= 0) {
            if (recognizer.acceptWaveForm(b, nbytes)) {
                recognizer.getResult();
            } else {
                recognizer.getPartialResult();
            }
        }

        JSONObject jsonResult = new JSONObject(recognizer.getResult());
        wavFile.delete();
        return jsonResult.getString( name: "text");
    }

}
```

Obrázek 31: **Integrace Vosk (zdroj: autor práce)**

9.6 Autentizace a zabezpečení

V rámci systému je implementována autentizace a autorizace pomocí frameworku Spring Security, který zajišťuje ochranu endpointů, správu přístupů a bezpečné přihlašování.

Uživatelská hesla jsou ukládána do databáze v hashované podobně. K tomu slouží algoritmus BCrypt (viz. Obrázek 32), který je bezpečný a odolný proti útokům hrubou silou.

```
@Bean  petrsafrata
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
```

Obrázek 32: **Hashovaní hesel pomocí algoritmu BCrypt (zdroj: autor práce)**

Přihlášení do aplikace poté probíhá prostřednictvím jednoduchého formuláře, ve kterém jsou zadány přihlašovací údaje (e-mail, heslo). Tyto údaje jsou následně ověřeny proti databázi. Pokud jsou údaje správné, je vytvořena http session s identitou přihlašovaného uživatele, což umožní přístup do zabezpečené části aplikace. Každý uživatel může mít vytvořenou maximálně jednu http session, kterou si prohlížeč pomatuje po dobu 20 minut, po uplynutí této doby dojde k automatickému odhlášení uživatele z bezpečnostních důvodů. Pro načítání uživatelů z databáze je využita vlastní implementace *UserDetailsService* (viz. Obrázek 33), která načítá uživatele prostřednictvím *UserRepository* a vrací identitu přihlášeného uživatele jako *SecurityPrincipalUser* (viz. Obrázek 35), což je vlastní implementace *UserDetails*.

```
@Service  petrsafrata
public class SecurityUserDetailService implements UserDetailsService {

    @Autowired
    private UserRepository userRepository;

    @Override  no usages  petrsafrata
    public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
        Optional<UserEntity> user = userRepository.findUserByEmail(email);
        if (user.isEmpty()) {
            throw new UsernameNotFoundException(email);
        }

        return new SecurityPrincipalUser(user.get());
    }
}
```

Obrázek 33: **Načtení uživatele z databáze (zdroj: autor práce)**

V aplikaci jsou důvěrné endpointy zabezpečeny, pomocí *SecurityFilterChain* (viz. Obrázek 34) je určeno, kdo má přístup, k jakým částem aplikace. V aktuální implementaci mají nepřihlášení uživatelé přístupné pouze stánky pro přihlášení či registraci, společně s endpotiny a soubory, které tyto stránky využívají. Přihlášený uživatel má poté přístup ke většině stránek a endpointů v rámci aplikace. Administrátor aplikace má povolen přístup na stránky administrátorského panelu a k jednotlivým endpointům, které jsou zde využívány. Administrátorský účet je v databázi identifikován pomocí ID 1. Při ověřování uživatelů se kontroluje, zda jejich ID odpovídá administrátorskému účtu.

```
@Bean  petrsafrata
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
            .csrf(csrf -> csrf.disable())
            .authorizeHttpRequests(auth -> auth
                    .requestMatchers("/login","/register").permitAll()
                    .requestMatchers("/users_prod/login", "/users_prod/register").permitAll()
                    .requestMatchers("/loginAndRegisterStyles.css", "/loginJs.js", "/registerJs.js", "/logo_app_transparent.png").permitAll()
                    .requestMatchers("/actuator/**", "/administration", "/administrationJs.js").access((authentication, context) -> {
                        if (authentication.get() == null || !authentication.get().isAuthenticated()) {
                        return new AuthorizationDecision( granted: false);
                        }

                        SecurityPrincipalUser user = (SecurityPrincipalUser) authentication.get().getPrincipal();
                        boolean hasAccess = user.getUser().getId() == 1;

                        return new AuthorizationDecision(hasAccess);
                    })
                    .anyRequest().authenticated()
            )
            .logout(logout -> logout
                    .logoutUrl("/logout")
                    .logoutSuccessUrl("/login?logout")
                    .invalidateHttpSession(true)
                    .deleteCookies( ...cookieNamesToClear: "JSESSIONID")
                    .permitAll()
            )
            .rememberMe(rem -> rem
                    .tokenValiditySeconds(1200)
            )
            .exceptionHandling(ex -> ex.authenticationEntryPoint(new LoginUrlAuthenticationEntryPoint( loginFormUrl: "/login")))
            .sessionManagement(session -> session
                    .sessionCreationPolicy(SessionCreationPolicy.IF_REQUIRED)
                    .maximumSessions(1)
            )
            .formLogin(form -> form.disable());
    return http.build();
}
```

Obrázek 34: **Implementace přístupů (zdroj: autor práce)**

Zároveň je většina endpointů v aplikaci zabezpečena proti neoprávněné manipulaci (viz. Obrázek 36). Probíhá zde ověření, zda je uživatel přihlášený nebo zda se jeho ID shoduje s ID vlastníka manipulovaného souboru či složky. Z přihlášeného uživatele se stane objekt *SecurityPrincipalUser*, ze kterého je možné načíst ID a zjistit, zda je evidováno v rámci databáze, případně jej porovnat s ID, které je uvedeno u souboru či složky jako vlastník.

```java
public class SecurityPrincipalUser implements UserDetails {  50 usages  petrsafrata

    private final UserEntity user;  4 usages
    public SecurityPrincipalUser(UserEntity user) {  1 usage  petrsafrata
        this.user = user;
    }

    public UserEntity getUser() {  petrsafrata
        return user;
    }

    @Override  no usages  petrsafrata
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return Collections.emptyList();
    }

    @Override  petrsafrata
    public String getPassword() {
        return user.getPassword();
    }

    @Override  petrsafrata
    public String getUsername() {
        return user.getEmail();
    }
}
```

Obrázek 35: **Vlasní implementace UserDetails (zdroj: autor práce)**

```java
@DeleteMapping("/delete/{id}")  petrsafrata
public ResponseEntity<?> deleteFile(@PathVariable String id, Authentication authentication) {
    Optional<FileDocument> fileDocuments = fileService.findFileById(id);
    if (fileDocuments.isPresent()) {
        SecurityPrincipalUser principal = (SecurityPrincipalUser) authentication.getPrincipal();
        Integer userId = (int) principal.getUser().getId();
        if (!userId.equals(fileDocuments.get().getFileOwnerId())) {
            return ResponseEntity.status(HttpStatus.FORBIDDEN).build();
        }
```

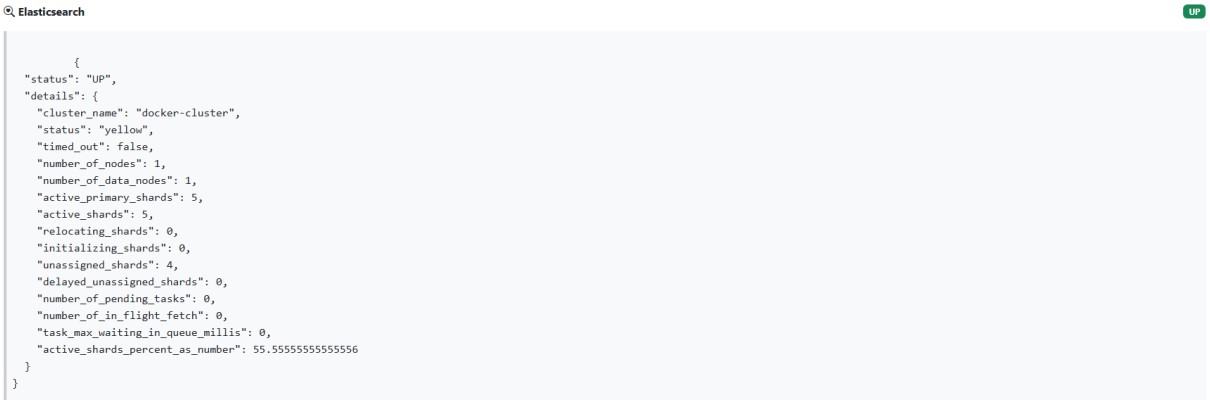
Obrázek 36: **Ukázka zabezpečení proti neoprávněné manipulaci (zdroj: autor práce)**

9.7 Administrace a monitoring

Systém využívá framework Spring Boot Actuator pro monitoring a diagnostiku aplikace. Díky tomu je v rámci aplikace možné sledovat technické metriky systému, jeho základní stavové informace a stavové informace komponent. Actuator automaticky vystavuje množství diagnostických endpointů pomocí http a REST API.

V rámci aplikace jsou zpřístupněny pro administrátorský účet všechny endpointy, které Actuator nabízí. V rámci administrátorského grafického rozhraní se poté pracuje pouze s endpointy health (viz. Obrázek 37) a metrics. Popis jednotlivých endpointů je možné dohledat na oficiálních stránkách frameworku, níže jsou uvedeny pouze nejvíce využívané:

- Health – základní přehled o tom, zda aplikace běží a zda jsou dostupné služby
- Metrics – přehled metrik o CPU, paměti, garbage collectoru, requestech atd.
- Beans – přehled všech Spring Bean komponent
- Env – přehled veškerých enviromentálních proměnných aplikace

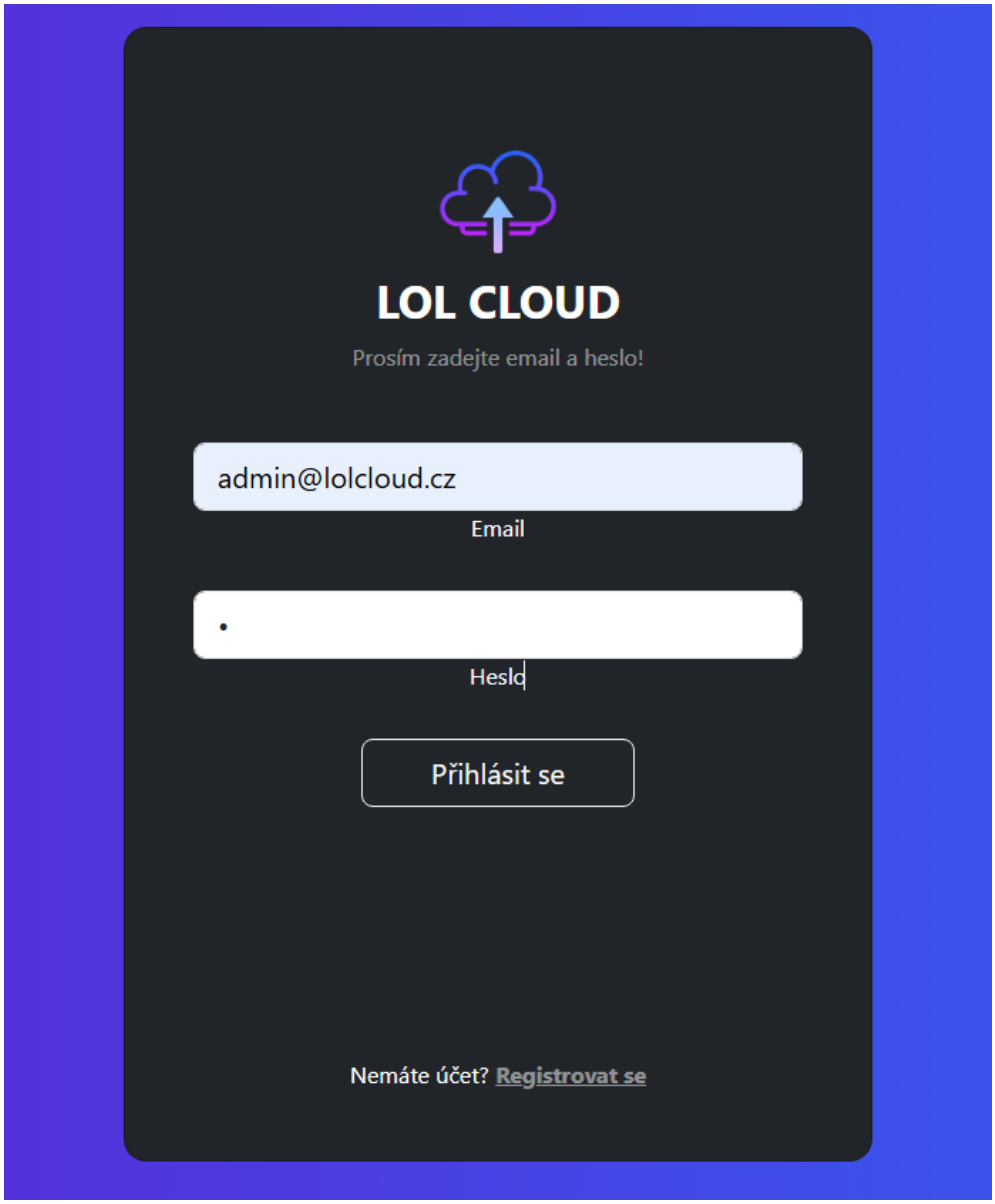


Obrázek 37: **Ukázka dostupnosti služby v administrátorském panelu (zdroj: autor práce)**

10 Ukázka uživatelského rozhraní

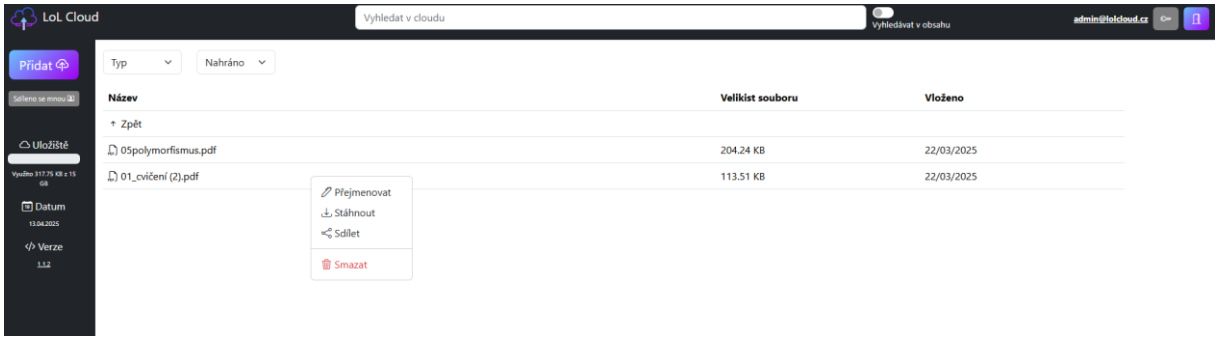
Výsledná aplikace nabízí přehledné a intuitivní uživatelské rozhraní, které bylo implementováno v souladu s požadavky a cíli definovanými v kapitole 8.2. Návrh kladl důraz na jednoduchost, snadnou orientaci a rychlí přístup k hlavním funkcím systému. V této části jsou prezentovány klíčové obrázky aplikace spolu s popisem jejich funkcionality.

Přihlašovací stránka obsahuje přihlašovací formulář (viz. Obrázek 38), do kterého uživatel zadá své přihlašovací údaje (e-mail, heslo). Pokud jsou zadané údaje neplatné, zobrazí se chybová hláška o neúspěšném pokusu o přihlášení. Zároveň je z této stránky dostupný proklik na registrační formulář, kde je možné vytvořit si nový účet.



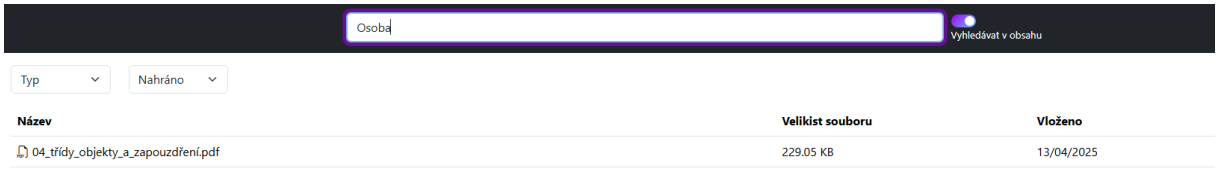
Obrázek 38: **Přihlašovací stránka (zdroj: autor práce)**

Po úspěšném přihlášení dojde k přesměrování uživatele na hlavní panel uložiště (viz Obrázek 39). Na tomto panelu jsou poté dostupné všechny uživatelovi nahrané soubory a vytvořené složky. Uživatel zde může provádět operace jako nahrávaní souboru, procházení složek, vyhledávání atd. Další operace se soubory či složkami jsou poté dostupné z kontextového menu. Tyto operace zahrnují přejmenování, sdílení, stahování a smazání v rámci konkrétního souboru a přejmenování či mazání v rámci konkrétní složky. Uživatel má také možnost prokliku na soubory, které jsou mu sdíleny jiným uživatelem. Zobrazují se zde také klíčové informace o uložišti, například aktuální využití uživatelem nebo systémová verze.



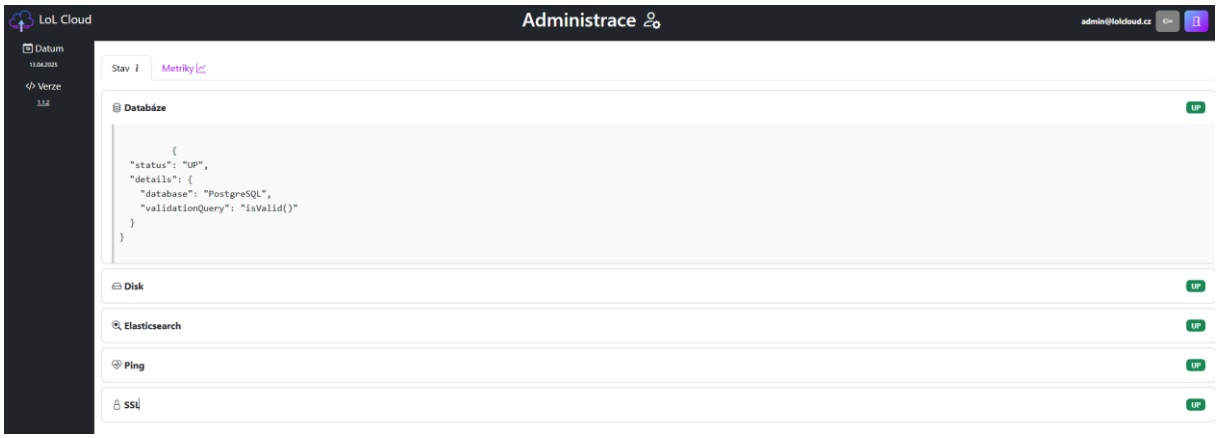
Obrázek 39: **Hlavní panel uložiště (zdroj: autor práce)**

Vyhledávání je zajištěno pomocí jednoduchého textového pole, které je uprostřed navigačního panelu stránky (viz. Obrázek 40). Uživatel do textového pole jednoduše zadá, co chce vyhledat a klikne na klávesu enter. Vyhledaný výsledek se mu poté zobrazí v rámci tabulky se soubory. Ve výchozím nastavení je aplikováno vyhledávání podle názvů souboru. Pokud chtějí uživatelé vyhledávat v obsahu, je nutné zakliknout přepínač, umístěný na levé straně vyhledávacího pole.



Obrázek 40: **Vyhledávání (zdroj: autor práce)**

Administrátorský učet má administrátorský panel, v němž je možné sledovat stav aplikace a její metriky (viz. Obrázek 41). Do administrátorského panelu je možné přepnout se kliknutím na uživatelské jméno, které je zobrazeno na levé straně navigačního menu. Poté dojde k přesměrování na administrátorský panel.



Obrázek 41: **Administrátorský panel (zdroj: autor práce)**

ZÁVĚR

Cílem této bakalářské práce bylo navrhnout a implementovat webovou aplikaci pro inteligentní ukládání a správu souborů, která využívá moderní technologie pro extrakci dat, správu souboru a umožňuje vyhledávání na základě názvů či obsahu souborů. V rámci implementace této aplikace byly integrovány klíčové komponenty jako Spring Boot, ProstreSQL, Elasticsearch. Zároveň součástí řešení byla také integrace externích nástrojů pro extrakci dat ze souborů. Pomocí Tesseract OCR je možné extrahovat text z obrázků, zatímco Vosk umožnuje převod řeči na text v rámci zvukových souborů. Celý systém byl nadále kontejnerizován pomocí Dockeru a rozdělen na jednotlivé služby, což zajistilo částečnou škálovatelnost, snadnou přenositelnost systému a izolaci jednotlivých služeb.

Teoretická část práce byla zaměřena na popis a analýzu a porovnání existujících cloudových platforem, především těch od společností Amazon, Microsoft a Google. Dále byla zahrnuta analýza trhu s cloudovými službami a podrobné vysvětlení technologií využitích v rámci řešení jako je OCR, ASR, Docker, Elasticsearch nebo Spring Boot. Praktická část se poté soustředí na návrh a implementaci vlastní aplikace, přičemž detailně popisuje jednotlivé komponenty systému, jeho architekturu a chování z pohledu uživatele.

Aplikace umožnuje nahrávat a zpracovávat různé typy souborů (např. PDF, obrázky a zvukové soubory) a následně vyhledávat v jejich obsahu. Díky využití technologie OCR a převodu řeči na text dochází k efektivní extrakci nestrukturovaných dat ze souborů.

Během vývoje byly identifikovány a překonány technické výzvy, zejména správné nastavení prostředí pro běh všech služeb v kontejnerech Dockeru, komunikace mezi kontejnery, efektivní práce se soubory a bezpečné zpracování dat a celkové zabezpečení systému.

Výsledná aplikace je připravena pro reálné nasazení do produkčního prostředí a díky své modularitě a rozšiřitelnosti umožnuje i další vývoj, například vylepšení uživatelského rozhraní, bezpečnosti, škálovatelnosti, přidání podpory pro vícejazyčnost nebo přidání pokročilého zpracování obsahu pomocí strojového učení. Přesto již nyní aplikace splnila svůj účel možnosti využití open-source technologií při vývoji cloudových systémů a zároveň vytváří pevný základ pro další vývoj v této oblasti.

POUŽITÁ LITERATURA

[1] AMAZON. Cloud computing with AWS. *https://aws.amazon.com/what-is-aws/*. 2025.

[2] AMAZON. Amazon S3. *https://aws.amazon.com/s3/?nc=sn&loc=0*. 2025.

[3] AMAZON. Amazon ESB. *https://docs.aws.amazon.com/ebs/latest/userguide/what-is-ebs.html*. 2025.

[4] AMAZON. Amazon EFS. *https://docs.aws.amazon.com/efs/latest/ug/whatisefs.html*. 2025.

[5] AMAZON. What is Amazon OpenSearch Service. *https://docs.aws.amazon.com/opensearch-service/latest/developerguide/what-is.html*. 2025.

[6] AMAZON. What is OpenSearch. *https://aws.amazon.com/what-is/opensearch/*. 2025.

[7] AMAZON. Transition from Amazon CloudSearch to Amazon OpenSearch Service. *https://aws.amazon.com/blogs/big-data/transition-from-amazon-cloudsearch-to-amazon-opensearch-service/*. 2025.

[8] AMAZON. What is Amazon Textract. *https://docs.aws.amazon.com/textract/latest/dg/what-is.html*. 2025.

[9] MICROSOFT. What is Azure. *https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-azure*. 2025.

[10] MICROSOFT. Introduction to Azure Blob Storage. *https://learn.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction*. 2025.

[11] MICROSOFT. What is Azure Files. *https://learn.microsoft.com/en-us/azure/storage/files/storage-files-introduction*. 2025.

[12] MICROSOFT. Introduction to Azure managed disks. *https://learn.microsoft.com/en-us/azure/virtual-machines/managed-disks-overview*. 2025.

[13] MICROSOFT. What's Azure AI Search? *https://learn.microsoft.com/en-us/azure/search/search-what-is-azure-search*. 2025.

[14] MICROSOFT. What is Azure AI Vision? *https://learn.microsoft.com/en-us/azure/ai-services/computer-vision/overview*. 2025.

[15] GOOGLE. Google Cloud overview. *https://cloud.google.com/docs/overview*. 2025.

[16] GOOGLE. Google security overview. *https://cloud.google.com/docs/security/overview/whitepaper*. 2024.

[17] GOOGLE. Google infrastructure security design overview. *https://cloud.google.com/docs/security/infrastructure/design*. 2024.

[18] GOOGLE. Product overview of Cloud Storage. *https://cloud.google.com/storage/docs/introduction*. 2025.

[19] GOOGLE. Filestore overview. *https://cloud.google.com/filestore/docs/overview*. 2025.

[20] GOOGLE. About Persistent Disk. *https://cloud.google.com/compute/docs/disks/persistent-disks*. 2025.

[21] GOOGLE. Introduction to Vertex AI Search. *https://cloud.google.com/generative-ai-app-builder/docs/enterprise-search-introduction*. 2025.

[22] GOOGLE. Document AI overview. *https://cloud.google.com/document-ai/docs/overview*. 2025.

[23] CODY SLINGERLAND. 13 Top Cloud Service Providers Globally In 2025. *https://www.cloudzero.com/blog/cloud-service-providers/*. 2024.

[24] NISAR AHMAD. AWS vs. Azure vs. Google Cloud: Cloud Services Compared 2025. *https://www.channelinsider.com/cloud-computing/aws-vs-azure-vs-google-cloud/*. 2024.

[25] ANKIT. Azure vs. AWS vs. GCP (2025): Which Cloud Provider Excels in Which Area? . *https://medium.com/@ankitcodepract/azure-vs-aws-vs-gcp-2025-which-cloud-provider-excels-in-which-area-%EF%B8%8F-97102ef7e9a9*. 2025.

[26] IBM. What is optical character recognition (OCR)? *https://www.ibm.com/think/topics/optical-character-recognition*. 2024.

[27] AMAZON. What is OCR (Optical Character Recognition)? *https://aws.amazon.com/what-is/ocr/*. 2025.

[28] IBM. What is speech recognition? *https://www.ibm.com/think/topics/speech-recognition*. 2021.

[29] ELASTIC. The heart of the Elastic Stack. *https://www.elastic.co/elasticsearch*. 2025.

[30] KUĆ, Rafał a Marek ROGOZIŃSKI. *Elasticsearch Server*. B.m.: Packt Publishing Limited, 2013. ISBN 1849518440.

[31] ELASTIC. What is Elasticsearch? *https://www.elastic.co/guide/en/elasticsearch/reference/current/elasticsearch-intro-what-is-es.html*. 2025.

[32] AMAZON. What is Elasticsearch? *https://aws.amazon.com/what-is/elasticsearch/*. 2025.

[33] ELASTIC. Indices, documents, and fields. *https://www.elastic.co/guide/en/elasticsearch/reference/current/documents-indices.html*. 2025.

[34] ELASTIC. Search and analyze data. *https://www.elastic.co/guide/en/elasticsearch/reference/current/search-analyze.html*. 2025.

[35] ELASTIC. Query DSL. *https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl.html*. 2025.

[36] MICROSOFT. Co je Java Spring Boot? *https://azure.microsoft.com/cs-cz/resources/cloud-computing-dictionary/what-is-java-spring-boot*. 2025.

[37] IBM. What is Java Spring Boot? *https://www.ibm.com/think/topics/java-spring-boot*. 2025.

[38] CARNELL, John. *Spring microservices in action: a multiplatform approach to building chatbots*. Shelter, Island, NY: Manning Publications Co., 2017. ISBN 16-172-9398-9.

[39] PECINOVSKÝ, Rudolf. *Java 21: kompletní příručka jazyka*. Praha: Grada Publishing, 2023. ISBN 978-80-247-0599-6.

[40] DOCKER. What is Docker? *https://docs.docker.com/get-started/docker-overview/*. 2025.

[41] MICROSOFT. Co je Java? *https://azure.microsoft.com/cs-cz/resources/cloud-computing-dictionary/what-is-java-programming-language*. 2025.

SEZNAM PŘÍLOH

Příloha A: Zdrojový kód aplikace

Příloha B: Kontejnerizovaná a spustitelná aplikace

PŘÍLOHA A: Zdrojový kód aplikace

Tato příloha obsahuje archivovaný projekt s názvem VirtualStorage, vytvořeným v prostředí IntelliJ IDEA. Součástí jsou veškeré zdrojové kódy aplikace a konfigurační soubory aplikace, nikoliv však spustitelné prostředí.

PŘÍLOHA B: Kontejnerizovaná a spustitelná aplikace

Tato příloha obsahuje plně kontejnerizované řešení aplikace. Součástí archivu je adresářová struktura a také soubor docker-compose.yml, který obsahuje veškeré potřebné konfigurace pro kontejnery. Správně vytvořené kontejnery a adresářová struktura jsou nezbytné pro spuštění aplikace, jak v rámci vývojového, tak produkčního prostředí. Archiv zároveň obsahuje soubor README.md, který popisuje správný postup nasazení pomocí Docker Compose.