



**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**  
BRNO UNIVERSITY OF TECHNOLOGY

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**  
FACULTY OF INFORMATION TECHNOLOGY

**ÚSTAV INTELIGENTNÍCH SYSTÉMŮ**  
DEPARTMENT OF INTELLIGENT SYSTEMS

**WEB PRO ZOBRAZOVÁNÍ ARCHIVÁLIÍ**  
WEB FOR DISPLAYING ARCHIVE MATERIALS

**DIPLOMOVÁ PRÁCE**  
MASTER'S THESIS

**AUTOR PRÁCE**  
AUTHOR

**Bc. PAVEL ŠESTÁK**

**VEDOUCÍ PRÁCE**  
SUPERVISOR

**Ing. JAROSLAV ROZMAN, Ph.D.**

**BRNO 2024**

## Zadání diplomové práce



Ústav: Ústav inteligentních systémů (UITS)  
Student: **Šesták Pavel, Bc.**  
Program: Informační technologie a umělá inteligence  
Specializace: Softwarové inženýrství  
Název: **Web pro zobrazování archiválí**  
Kategorie: Web  
Akademický rok: 2023/24

153890

### Zadání:

1. Nastudujte funkčnost a vzhled webů různých archivů v České republice. Zaměřte se na části, které zobrazují skeny archiválí. Nastudujte moderní webové technologie pro responzivní ovládání webu.
2. Na základě nastudovaných znalostí vyberte nejvhodnější technologie a navrhněte web, který umožní zobrazovat požadované archiválie. Na základě rešerše archivů umožněte uživatelům co nejpohodlnější procházení skenů (stejné zvětšení obrázku při přechodu na další sken, ovládání klávesami, atd.). Web by měl být také snadno rozšířitelný o další kategorie archiválí a nově přidané archiválie. Proto navrhněte skripty pro automatickou aktualizaci vybraných informací z archivů v České republice.
3. Podle návrhu web implementujte. Umožněte také vyhledávání archiválí podle obcí, tzn. je nutné udělat napárování obcí obsažených v archivech s databází obcí RÚIAN.
4. Proveďte důkladné otestování na skupině uživatelů a navrhněte připadné úpravy do budoucna.

### Literatura:

- Matriční rozcestník, Česká genealogická a heraldická společnost v Praze,  
<http://www.genealogie.cz/aktivity/digitalizace/>, [cit. 15.9.2022]

Při obhajobě semestrální části projektu je požadováno:

- První dva body zadání

Podrobné závazné pokyny pro vypracování práce viz <https://www.fit.vut.cz/study/theses/>

Vedoucí práce: **Rozman Jaroslav, Ing., Ph.D.**

Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.

Datum zadání: 1.11.2023

Termín pro odevzdání: 17.5.2024

Datum schválení: 6.11.2023

## Abstrakt

Cílem diplomové práce je sjednotit data jednotlivých archivů do jednoho systému, v rámci kterého bude tato data uživatel moci vyhledávat a prohlížet. Především se jedná o naskeované archiválie a jejich metadata. Práce se věnuje popisu jednotlivých typů archiválií a analýze aktuálních systémů. V práci jsou porovnány čtyři scrapovací systémy, a to na základě velikosti datových sad, chybějících hodnot a konzistence mezi jednotlivými datovými sadami. Systém pro zobrazování archiválií je následně namodelován pomocí behaviorálních a strukturálních UML diagramů. Součástí práce je i grafický návrh vyhotovený v nástroji Figma. Práce rovněž obsahuje popis postupu implementace, testování a nasazení pomocí Dockeru na školní infrastrukturu.

## Abstract

The aim of the thesis is to unify the data of the individual archives into one system, within which the user will be able to search and view the data. In particular, it is about scanned archival materials and their metadata. The thesis describes the different types of archival data and analyses the current systems. The thesis compares four scraping systems on the basis of dataset size, missing values and consistency between datasets. The archival imaging system is then modeled using behavioral and structural UML diagrams. The work also includes a graphical design produced in Figma. The thesis also includes a description of the implementation, testing and deployment procedure using Docker on the school infrastructure.

## Klíčová slova

UML, Figma, ReactJS, Webová aplikace, datová analýza, Docker, kontajnerizace, OpenSeaDragon, FabricJS, archivnictví, matriky

## Keywords

UML, Figma, ReactJS, Web application, data analysis, Docker, containerization, OpenSeaDragon, FabricJS, archives, matrices

## Citace

ŠESTÁK, Pavel. *Web pro zobrazování archiválií*. Brno, 2024. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jaroslav Rozman, Ph.D.

# Web pro zobrazování archiválií

## Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Jaroslava Rozmana, Ph.D. Další informace mi poskytla Bc. Iveta Brabcová. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....  
Pavel Šesták  
13. května 2024

## Poděkování

Tímtoto bych chtěl poděkovat svému vedoucímu Ing. Jaroslavu Rozmanovi, Ph.D., za vedení této práce, pomoc se strukturováním a formátováním závěrečné práce. Vedoucí práce mě dále seznámil s cílovou doménou řešené problematiky a poskytoval zpětnou vazbu ve všech fázích vývoje softwaru. Také bych chtěl poděkovat Bc. Ivetě Brabcové za poskytnuté informace z oblasti archivnictví.

# Obsah

<b>1</b>	<b>Úvod</b>	<b>7</b>
<b>2</b>	<b>Archiválie</b>	<b>8</b>
2.1	Berní rula . . . . .	8
2.2	Lánové rejstříky . . . . .	8
2.3	Matriky . . . . .	8
2.4	Rektifikační akta . . . . .	9
2.5	Pozemkové knihy . . . . .	9
2.6	Sčítací operáty . . . . .	9
2.7	Soupis poddaných dle víry . . . . .	10
2.8	Urbáře . . . . .	10
<b>3</b>	<b>Webové aplikace pro zpřístupnění digitalizovaných archiválií</b>	<b>11</b>
3.1	Národní archiv . . . . .	11
3.2	Moravský zemský archiv . . . . .	12
3.3	Zemský archiv v Opavě . . . . .	14
3.4	Státní oblastní archiv v Třeboni . . . . .	15
3.5	Státní oblastní archiv v Litoměřicích . . . . .	16
3.6	Státní oblastní archiv v Plzni . . . . .	17
3.7	Státní oblastní archiv v Hradci Králové . . . . .	18
3.8	Státní oblastní archiv v Praze . . . . .	20
3.9	Digitalizace evropských archivů . . . . .	21
3.10	Matricula online . . . . .	23
3.11	Zhodnocení dostupných webových aplikací . . . . .	24
<b>4</b>	<b>Formáty pro efektivní zobrazování velkých snímků na webu</b>	<b>26</b>
4.1	Deep Zoom Image . . . . .	26
4.2	Zoomify . . . . .	27
4.3	International Image Interoperability Framework . . . . .	27
<b>5</b>	<b>Webové technologie</b>	<b>29</b>
5.1	Serverové technologie . . . . .	30
5.2	Technologie pro ukládání a poskytování dat . . . . .	30
5.3	Klientské technologie . . . . .	30
<b>6</b>	<b>Datová analýza</b>	<b>32</b>
6.1	Datová sada Dominika Popa . . . . .	32
6.2	Datová sada Jakuba Sokolíka . . . . .	32

6.3	Datová sada Jana Valuška . . . . .	33
6.4	Datová sada obsahující URL snímky . . . . .	33
6.5	Porovnání datových sad . . . . .	33
6.5.1	Moravský zemský archiv . . . . .	34
6.5.2	Archiv hlavního města Praha . . . . .	34
6.5.3	Státní oblastní archiv v Hradci Králové . . . . .	34
6.5.4	Státní oblastní archiv v Litoměřicích . . . . .	35
6.5.5	Zemský archiv v Opavě . . . . .	35
6.5.6	Státní oblastní archiv v Plzni . . . . .	35
6.5.7	Státní oblastní archiv v Praze . . . . .	35
6.5.8	Státní oblastní archiv v Třeboni . . . . .	35
6.5.9	Výsledky datové analýzy . . . . .	35
<b>7</b>	<b>Analýza požadavků</b>	<b>36</b>
7.1	Diagram případů užití . . . . .	36
7.2	Systémové diagramy sekvence . . . . .	38
7.3	Doménový model . . . . .	41
7.4	Použitá architektura a technologie . . . . .	43
7.4.1	Relační databáze . . . . .	43
7.4.2	Databázové indexy . . . . .	43
7.4.3	Plné textové vyhledávání . . . . .	46
7.4.4	Nerelační databáze Elasticsearch . . . . .	46
7.5	Tvorba interaktivního prototypu . . . . .	47
<b>8</b>	<b>Návrh</b>	<b>48</b>
8.1	Entity-relationship diagram . . . . .	48
8.2	Diagramy struktury . . . . .	50
8.2.1	Vysokoúrovňový diagram komponent . . . . .	50
8.2.2	Diagram komponent Webového scraperu . . . . .	51
8.2.3	Diagram komponent klientské aplikace . . . . .	51
8.2.4	Diagram komponent serverové části aplikace . . . . .	54
8.3	Tvorba konečného automatu přechodů . . . . .	56
8.4	Tvorba grafického návrhu . . . . .	57
<b>9</b>	<b>Implementace</b>	<b>62</b>
9.1	Stahování a aktualizace dat . . . . .	62
9.1.1	Lokality . . . . .	62
9.2	Implementace serverové části aplikace . . . . .	66
9.2.1	Ukládání dat . . . . .	66
9.2.2	Databázové indexy . . . . .	67
9.2.3	Bezpečnost . . . . .	69
9.2.4	Odesílání e-mailů . . . . .	70
9.3	Dokumentace . . . . .	70
9.4	Implementace klientské aplikace . . . . .	71
9.5	Implementace proxy pro snímky archiválií . . . . .	73
9.6	Kontejnerizace . . . . .	74
9.6.1	Docker . . . . .	74
9.6.2	Docker compose . . . . .	75

9.6.3 Architektura Docker . . . . .	76
<b>10 Testování</b>	<b>77</b>
10.1 Testování modelu v rámci programu Figma . . . . .	78
10.2 Testování serverové části . . . . .	78
10.3 Zátěžové testování . . . . .	79
10.3.1 Zátěžové testování na serveru Perun . . . . .	80
10.3.2 Zátěžové testování na serveru Radegast . . . . .	82
10.3.3 Výsledky porovnání . . . . .	84
10.4 Testování odbornou veřejností . . . . .	85
<b>11 Závěr</b>	<b>86</b>
<b>Literatura</b>	<b>88</b>
<b>A Výstupy datové analýzy</b>	<b>92</b>
A.1 Formáty datových sad . . . . .	92
A.2 Grafy pro jednotlivé archivy . . . . .	97
<b>B Simulace očních vad pohledů prototypu v nástroji Figma</b>	<b>111</b>
B.1 Simulace očních vad pro hlavní stranu . . . . .	111
B.2 Simulace očních vad pro prohlížeč archiválií . . . . .	118
<b>C Ukázka grafického uživatelského rozhraní výsledné aplikace</b>	<b>123</b>
<b>D Nasazení</b>	<b>128</b>

# Seznam obrázků

3.1	Vyhledávání archiválií v systému Národního archivu . . . . .	11
3.2	Prohlížeč archiválií v systému Národního archivu . . . . .	12
3.3	Vyhledávání archiválií v systému Moravského zemského archivu . . . . .	13
3.4	Prohlížeč archiválií v systému Moravského zemského archivu . . . . .	14
3.5	Vyhledávání archiválií v systému Zemského archivu v Opavě . . . . .	14
3.6	Prohlížeč archiválií v systému Zemského archivu v Opavě . . . . .	15
3.7	Vyhledávání archiválií v systému SOA v Třeboni . . . . .	15
3.8	Prohlížeč archiválií v systému SOA v Třeboni . . . . .	16
3.9	Vyhledávání archiválií v systému SOA v Litoměřicích . . . . .	17
3.10	Vyhledávání archiválií v systému SOA v Plzni . . . . .	17
3.11	Prohlížeč archiválií v systému SOA v Plzni . . . . .	18
3.12	Vyhledávání archiválií v systému SOA v Hradci Králové . . . . .	19
3.13	Prohlížeč archiválií v systému SOA v Hradci Králové . . . . .	20
3.14	Vyhledávač archiválií v systému SOA v Praze . . . . .	20
3.15	Prohlížeč archiválií v systému SOA v Praze . . . . .	21
3.16	Vyhledávač archiválií v systému Monasterium . . . . .	22
3.17	Prohlížeč archiválií v systému Monasterium . . . . .	22
3.18	Vyhledávač archiválií v systému Matricula . . . . .	23
3.19	Prohlížeč archiválií v systému Matricula . . . . .	24
3.20	Diagram funkcionality . . . . .	25
4.1	Rozložení obrázku na dlaždice v rámci technologie Deep Zoom <sup>1</sup> . . . . .	27
4.2	Architektura pro použití IIIF <sup>2</sup> . . . . .	28
4.3	Zpracování obrázku pomocí IIIF <sup>2</sup> . . . . .	28
6.1	Porovnání velikosti jednotlivých datových sad . . . . .	33
7.1	Diagram případů užití . . . . .	37
7.2	Systémové diagramy sekvence – vyhledání archiválie . . . . .	38
7.3	Systémové diagramy sekvence - prohlížení archiválie . . . . .	39
7.4	Systémové diagramy sekvence – uživatelský profil . . . . .	40
7.5	Doménový model . . . . .	42
7.6	Vizualizace B-stromu <sup>3</sup> . . . . .	44
7.7	Vizualizace hashovací tabulky <sup>4</sup> . . . . .	45
7.8	Screenshot z prototypu . . . . .	47
8.1	Entity-relationship diagram . . . . .	49
8.2	Vysokoúrovňový diagram komponent . . . . .	50
8.3	Diagram komponent webového scraperu . . . . .	51

8.4	Diagram komponent klienta . . . . .	53
8.5	Diagram komponent serverové části aplikace . . . . .	55
8.6	Konečný automat přechodů . . . . .	56
8.7	Návrhový režim ve Figmě . . . . .	57
8.8	Návrh titulní strany ve Figmě . . . . .	58
8.9	Seznam matrik . . . . .	59
8.10	Seznam archiválií pro ostatní typy archiválií . . . . .	59
8.11	Detailní pohled na archiválii . . . . .	60
8.12	Detailní pohled na archiválii . . . . .	61
8.13	Prohlížeč snímků archiválie . . . . .	61
9.1	Vizualizace clusterizačního algoritmu <sup>5</sup> . . . . .	64
9.2	Struktura JWT tokenu <sup>6</sup> . . . . .	69
9.3	Ukázka použití JWT tokenu <sup>7</sup> . . . . .	70
9.4	Architektura Redux <sup>8</sup> . . . . .	72
9.5	Ukázka archiválie s anotacemi . . . . .	73
9.6	Ukázka komunikace pomocí proxy . . . . .	74
10.1	Výsledky automatických testů . . . . .	79
10.2	Zatížení serveru Perun v průběhu testu . . . . .	80
10.3	Průměrná doba odpovědi . . . . .	80
10.4	Percentil průměrné doby odpovědi . . . . .	81
10.5	Přehled latence . . . . .	81
10.6	Datová propustnost . . . . .	82
10.7	Zatížení serveru Radegast v průběhu testu . . . . .	82
10.8	Průměrná doba odpovědi . . . . .	83
10.9	Percentil průměrné doby odpovědi . . . . .	83
10.10	Přehled latence . . . . .	84
10.11	Datová propustnost . . . . .	84
A.1	Analýza chybějících hodnot pro MZA . . . . .	97
A.2	Analýza duplicitních hodnot pro MZA . . . . .	97
A.3	Vennův diagram pro MZA . . . . .	98
A.4	Kontrola konzistence pro MZA . . . . .	98
A.5	Analýza chybějících hodnot pro Národní archiv . . . . .	99
A.6	Analýza duplicitních hodnot pro Národní archiv . . . . .	99
A.7	Kontrola konzistence pro Národní archiv . . . . .	100
A.8	Analýza chybějících hodnot pro SOA v Hradci Králové . . . . .	100
A.9	Analýza duplicitních hodnot pro SOA v Hradci Králové . . . . .	101
A.10	Vennuv diagram pro SOA v Hradci Králové . . . . .	101
A.11	Kontrola konzistence pro SOA v Hradci Králové . . . . .	102
A.12	Analýza chybějících hodnot pro SOA v Litoměřicích . . . . .	102
A.13	Analýza duplicit pro SOA v Litoměřicích . . . . .	103
A.14	Vennuv diagram pro SOA v Litoměřicích . . . . .	103
A.15	Analýza chybějících hodnot pro ZA v Opavě . . . . .	104
A.16	Analýza duplicit pro ZA v Opavě . . . . .	104
A.17	Vennuv diagram pro ZA v Opavě . . . . .	105
A.18	Kontrola konzistence pro ZA v Opavě . . . . .	105
A.19	Analýza chybějících hodnot pro SOA v Plzni . . . . .	106

A.20 Analýza duplicit pro SOA v Plzni . . . . .	106
A.21 Vennuv diagram pro SOA v Plzni . . . . .	107
A.22 Kontrola konzistence pro SOA v Plzni . . . . .	107
A.23 Analýza chybějících hodnot pro SOA v Praze . . . . .	108
A.24 Analýza duplicit pro SOA v Praze . . . . .	108
A.25 Vennuv diagram pro SOA v Praze . . . . .	109
A.26 Analýza chybějících hodnot pro SOA v Třeboni . . . . .	109
A.27 Analýza duplicit pro SOA v Třeboni . . . . .	110
A.28 Vennuv diagram pro SOA v Třeboni . . . . .	110
B.1 Hlavní strana - simulace achromatopsie . . . . .	111
B.2 Hlavní strana - simulace krátkozrakosti . . . . .	112
B.3 Hlavní strana - simulace fotofobie . . . . .	113
B.4 Hlavní strana - simulace Deutanopie . . . . .	114
B.5 Hlavní strana - simulace Diplopie . . . . .	114
B.6 Hlavní strana - simulace Zeleného zákalu . . . . .	115
B.7 Hlavní strana - simulace Protanopie . . . . .	116
B.8 Hlavní strana - simulace Tritanopie . . . . .	117
B.9 Hlavní strana - simulace Šedého zákalu . . . . .	117
B.10 Prohlížeč archiválií - simulace achromatopsie . . . . .	118
B.11 Prohlížeč archiválií - simulace krátkozrakosti . . . . .	118
B.12 Prohlížeč archiválií - simulace fotofobie . . . . .	119
B.13 Prohlížeč archiválií - simulace Deutanopie . . . . .	119
B.14 Prohlížeč archiválií - simulace Diplopie . . . . .	120
B.15 Prohlížeč archiválií - simulace Zeleného zákalu . . . . .	120
B.16 Prohlížeč archiválií - simulace Protanopie . . . . .	121
B.17 Prohlížeč archiválií - simulace Tritanopie . . . . .	121
B.18 Prohlížeč archiválií - simulace Šedého zákalu . . . . .	122
C.1 Titulní strana . . . . .	123
C.2 Seznam archiválií . . . . .	124
C.3 Detailní pohled na archiválii . . . . .	124
C.4 Zobrazení lokalit dané archiválie . . . . .	125
C.5 Prohlížeč zdigitalizovaných archiválií . . . . .	125
C.6 Formulář . . . . .	126
C.7 Uživatelský profil . . . . .	126
C.8 Vyhledávání obcí . . . . .	127
D.1 Diagram nasazení . . . . .	129

# Kapitola 1

## Úvod

Lidstvo již ušlo velmi dlouhou a nelehkou cestu, na které čelilo nejedné náročné překážce. To, že jsme stále tu, značí, že se nám je všechny povedlo překonat. Nyní závisí na nás, zda se budou opakovat stejné chyby a nebo dojde k posunu a dělaní chyb vlastních. Jak pravil George Santayana: „*Ti, kteří si nepamatují minulost, jsou odsouzeni k tomu, aby si ji zopakovali.*<sup>1</sup>“ V dnešní době digitalizace je možné se podrobnosti ze života našich předků dozvědět z pohodlí našich domovů. Je tedy na nás informaticích poskytnout tuto službu co nejkvalitněji a umožnit tak nejen odborné veřejnosti, ale i začínajícímu badateli nahlédnout do historie. Digitalizování archivního materiálu má také pozitivní dopad na jejich životnost a přístup k digitalizovaným kopím prodlužuje životnost originálů. Bohužel v rámci České republiky doposud neexistuje centralizované řešení a každý archiv poskytuje pouze své zdigitalizované materiály, což komplikuje vyhledávání a nutí uživatele přecházet mezi více uživatelskými rozhraními. Cílem této práce je tedy sjednotit nashromážděná data a poskytnout je uživateli skrze reaktivní webové rozhraní.

Diplomová práce se skládá z dvanácti kapitol popisujících kompletní vývojový cyklus tvořeného systému. První dvě kapitoly seznámí čtenáře s jednotlivými typy archivního materiálu a systémy zpřístupňujícími archiválie v rámci České republiky. Čtvrtá kapitola je věnována technologiím optimalizujícím načítání velkých snímků. Kapitola pět popisuje dostupné architektury a webové technologie pro tvorbu systémů. Šestá kapitola je zaměřena na analýzu datových sad, které jsou výstupem dostupných scrapovacích systémů. Následující dvě kapitoly se venují popisu požadavků, vymezení funkcionality a návrhu architektury pro nově vznikající systém. Ze životního cyklu software vyplývá, že následuje implementace s testováním, které by mělo být součástí vývoje libovolného softwarového díla. Na projektu se po konzultaci s vedoucím práce podílel i Ota Malík [38], a to na implementaci serverové části aplikace, rozšiřování scrapovacích modulů a implementaci alternativního uživatelského rozhraní. Poslední dvě kapitoly této práce se venují nasazení systému na školní infrastrukturu a shrnutí výsledků celé práce.

---

<sup>1</sup><https://citaty.net/citaty/24176-george-santayana-ti-kteri-si-nepamatuji-minulost-jsou-odsouzeni-k/>

# Kapitola 2

## Archiválie

*„Za archiválii [52] považujeme psané, obrazové a zvukové památky dokumentární povahy, vzniklé ze soustavné organické činnosti svých původců, kteří je již ve svých registraturách nepotřebují, avšak byly pro svou společenskou, politickou či vědeckou důležitost vybrány a určeny k trvalé úschově.“*

### 2.1 Berní rula

Berní rula [37, 2] je nejstarší katastr na historickém území Čech. Název je odvozen od staročeského slova berně, které znamená daň. Jedná se o soupis hospodářů na poddanských usedlostech, jejich pozemkové držby a seznam chovaných hospodářských zvířat. Berní rula vznikla v polovině 17. století a měla za cíl co nejdetajněji zmapovat skutečný stav půdy a vyměřit daňové povinnosti pro jednotlivé usedlosti. Výsledky se využívaly k odhadu výši výběru do státní pokladny. Berní rula je rozdělena po panstvích, kde je ke každému panství uveden soupis poddaných obcí. Jako jednotka se používal jeden „osedlý“, což byla osoba trvale žijící na daném území. Jednotliví hospodáři jsou v ní seřazeni podle velikosti pozemků. U každého pole se kromě celkové výměry evidoval i údaj o tom, jaká část pole byla skutečně oseta.

### 2.2 Lánové rejstříky

Lánové rejstříky [37, 6] obsahují soupis všech hospodářů na území Moravy a jedná se o jistý ekvivalent berní ruly pro Moravu. Lánové rejstříky byly vydány na začátku 2. poloviny 17. století. Standardně se zde uváděla výměra osévané půdy. Lánové rejstříky byly psány tehdejším úředním jazykem, němčinou. Rozloha pole se uváděla ve starých rakouských plošných mírách, Metz a Achtel. Metz neboli měřice činila 0,19 hektaru a achtel byl potom  $\frac{1}{8}$  měřice. Obdobně jako v berní rule jsou seznamy seřazeny podle velikosti vlastněné půdy.

### 2.3 Matriky

Matriky [37, 9] jsou nejdůležitějším zdrojem informací pro genealogy, jelikož v nich na lezneme seznamy narození, křtů, svateb, úmrtí a pohřbů. Slovo matrika pochází z latinského *matricula*, výrazu označujícího seznamy duchovních. Nejstarší dochované matriky jsou evangelické a pochází až z poloviny 16. století. V roce 1784 se matriční knihy staly úředním dokumentem a byla přesně stanovena jejich struktura, tzn. jak mají být vedeny.

V 19. století začaly vznikat i civilní matriky určené pro nevěřící a civilní sňatky. V roce 1949 bylo rozhodnuto, že matriky přebere stát a spravovat je budou národní výbory. V archivech jsou pro badatele zpřístupněny pouze neživé matriky, to jsou takové, kde uběhlo 100 let od posledního zapsaného narození nebo 75 let od posledního sezdání nebo úmrtí.

## 2.4 Rektifikační akta

Rektifikační akta [37, 7] vznikla za účelem revize stávajících lánových rejstříků, které obsahovaly mnoho chyb. Rektifikační akta zahrnovala poddanskou i vrchnostenskou půdu pro účely zdanění. U jednotlivých hospodářů jsou uvedeny informace o tom, jaké pozemky drží a jakou mají rozlohu. Pro Českou berní rulu vznikla obdobná písemnost s názvem tereziánský katastr.

## 2.5 Pozemkové knihy

Pozemkové knihy [37, 29] jsou také známé pod názvem gruntovnice nebo gruntovní knihy. Jedná se o ekvivalent dnešního katastru nemovitostí a najdeme v nich informace o držení a přepisu majetku. Pozemkové knihy byly vedeny v rámci jednotlivých panství a velkostatků. Dnes je nalezneme ve státních oblastních archivech. V rámci archivů jsou většinou uloženy v samostatných fondech, případně ve fondech panství nebo velkostatků. Tyto knihy se dochovaly z různých období a ne vždy na sebe nutně navazují. Některé pozemkové knihy pocházejí již z 16. století, na jiných panstvích se dochovaly až od počátku 19. století. Důvodem pro chybějící knihy nejspíše ale nebyl nedbalý přístup vrchnostenské správy, jelikož bylo v jejím zájmu tyto záznamy uchovávat. Používaly se k evidenci poplatků, které plynuly z převodu majetku. Častou příčinu jejich nedochování lze spatřovat např. ve změnách vrchnosti daného panství, která si pozemkovou knihu vzala s sebou při stěhování. Jazyk, kterým byly knihy vedeny, se odvíjel od úředního jazyka dané vrchnosti a podle oblasti, kde byla kniha vedená. Na našem území se jedná převážně o češtinu a němčinu. Pozemkové knihy se mohly vést vložkovým systémem nebo chronologicky. Ve vložkovém systému byl pro každý pozemek vyčleněn určitý počet stran, kam se dané změny zapisovaly. Chronologický systém uchovává změny v pořadí, jak se chronologicky udaly za sebou, a pro badatele je složitější na orientaci. Jelikož jsou pozemkové knihy často starší než matriky, tak mohou sloužit pro genealogické účely z období a lokalit, které nejsou zmapovány žádnou z dostupných matrik. Zápisy v pozemkových knihách slouží i ke zmapování movitého vybavení, jež je v rámci každého přepisu uvedeno. Kromě věcných informací se zde lze dočíst i takové zajímavosti, že pozemky bývaly v historii často přepisovány na nejmladšího syna, čímž je nabourávána tradiční představa o tom, že dědí vždy nejstarší syn.

## 2.6 Sčítací operáty

První sčítání lidu [37, 39] neboli *census*, bylo na českém území provedeno v roce 1754 na základě patentu vydaného Marií Terezií v předchozím roce. Bohužel výstupy těchto sčítání byly nepřesné, jelikož obsahovaly agregovaná data bez jmen poddaných. Změna ve struktuře formuláře pro sčítání lidu přišla až o více než sto let později a již se více podobala těm, které známe dnes. I přes modernější strukturu původních formulářů zůstala ale metoda sběru a zpracování informací stejná, a proto byli zaznamenáni jen domácí obyvatelé, a ne skutečné počty přítomných osob. Například v Praze se podle sčítání nacházelo 68 tisíc

lidí, avšak ve skutečnosti zde žilo okolo 140 tisíc osob. V roce 1869 přišel říšský zákon, který nastavil periodu pro sčítání lidu na 10 let. Seznam sbíraných údajů již obsahoval, zda osoba je přítomna v místě sčítání, pohlaví, věk, státní příslušnost, rodinný stav, náboženské vyznání a obcovací řeč. V roce 1880 přibyly otázky ohledně gramotnosti a v roce 1890 se začal zjišťovat i majitel domu. Jedním z cílů sčítání lidu bylo zjištění ekonomického postavení obyvatel, proto bylo v rámci sčítání rozlišováno, zda člověk je výdělečně činný či nikoliv a ve kterém odvětví pracuje. Sčítací operáty jsou v rámci archivů uloženy v separátních fonitech. Mohou sloužit k ověření informací z matriky, případně i k nalezení informací z míst a období, kde žádná matrika není k dispozici.

## 2.7 Soupis poddaných dle víry

Soupis poddaných dle víry [1] vznikl na popud patentu vydaného roku 1650, který nařizoval vrchnosti v Čechách pořídit soupis poddaných na základě příslušnosti ke katolické církvi. Na vypracování těchto seznamů měla vrchnost šest týdnů. Každý soupis začínal majitelem panství, případně jeho správcem, a následoval seznam poddaných seřazených podle obcí. V seznamu nemuselo být uvedeno duchovenstvo a vojáci. Předdefinovaná struktura obsahovala jméno, stav, povolání, věk a údaj o náboženském vyznání. Základní jednotkou soupisu byla rodina, u jejichž členů se zapisoval vztah vůči hospodáři. Na konec soupisu se připojovaly informace o stavu kostelů a farních budov. Toto nařízení trvalo jen velmi krátkou dobu, jelikož zanedlouho stačilo vytvořit pouze soupis nekatolíků, a proto seznam pro Čechy není kompletní a pro Moravu a Slezsko není vypracován vůbec.

## 2.8 Urbáře

Urbáře [37, 24] obsahují informace o vlastnictví pozemků, daňové a pracovní zatížení našich předků. Urbáře jsou obdobně jako berní rula členěny podle panství a závisely na nich, v jakém stavu se dochovaly, jestli vůbec. Nejstarší urbáře jsou církevní a pocházejí z 2. poloviny 14. století. V případě, že se urbáře dochovaly z daného panství pro delší časové období, tak se dají využít k analýze vývoje daňového zatížení. V případě, že pro dané území existují i pozemkové knihy, tak se informace dají kombinovat. Pro genealogii se nejedná o tak cenný zdroj informací, jelikož obsahují pouze jméno a příjmení hospodáře. Urbáře jsou uloženy ve fonitech panství a velkostatků v oblastních archivech.

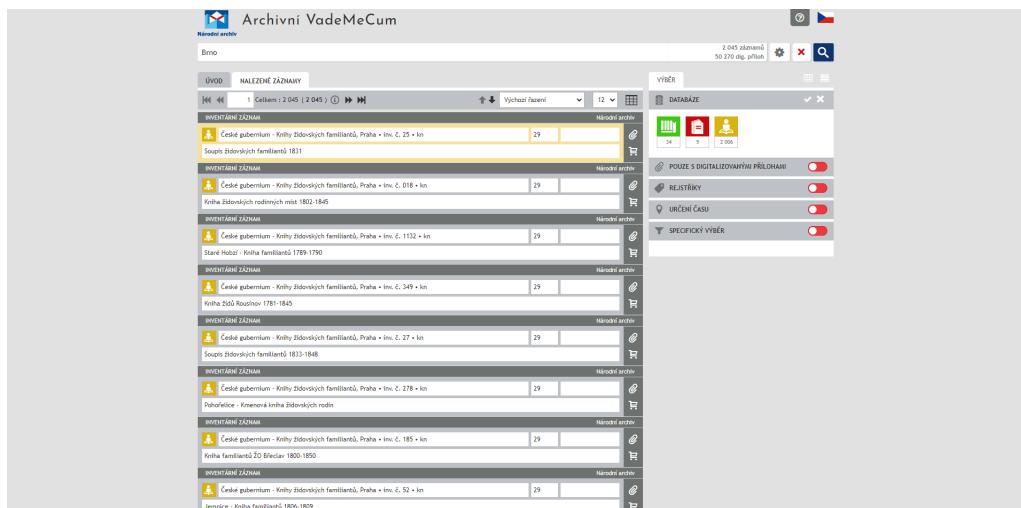
## Kapitola 3

# Webové aplikace pro zpřístupnění digitalizovaných archiválií

V této sekci jsou představeny aktuálně používané systémy jednotlivých archivů, které zpřístupňují digitalizované archiválie. Následně bude provedeno porovnání jednotlivých prohlížečů snímků archiválií, vyhodnoceny budou jak jejich klady, tak i zápory. Dále budou porovnány technologie, na nichž jsou dané systémy postaveny, a zda se vždy jedná o řešení na míru nebo společně sdílí nějaké aplikační jádro.

### 3.1 Národní archiv

Národní archiv má vlastní elektronickou badatelnu [Vademecum](#)<sup>1</sup>, která veřejnosti poskytuje přístup ke sbírkám a fondům spravovaných Národním archivem. V rámci aplikace jsou spravovány archivní fondy, archivní pomůcky a inventární záznamy. Aplikace umožňuje v daných archiváliích vyhledávat pomocí více kritérií jako například pomocí rejstříku, časového období či podle toho, zda daná archiválie obsahuje zdigitalizované přílohy.

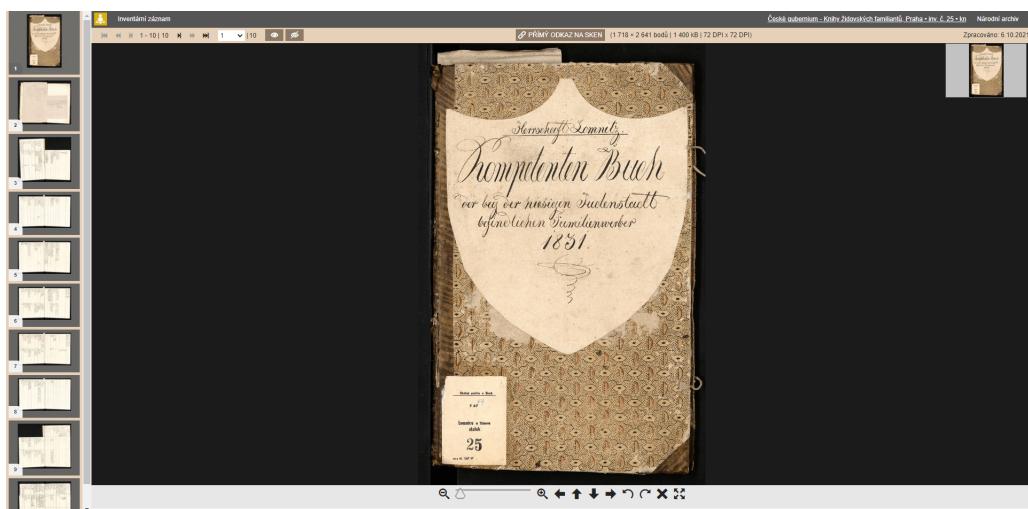


Obrázek 3.1: Vyhledávání archiválií v systému Národního archivu

<sup>1</sup><https://vademecum.nacr.cz/vademecum>

Podle analýzy zdrojového kódu to vypadá, že aplikace běží na Apache/2.4.54, které je hostované na systému Debian. Požadavek na vyhledání se posílá na SearchBean, což díky jmenným konvencím naznačuje, že serverová část aplikace je napsána v Javě. V rámci analýzy zdrojového kódu byly nalezeny čitelné komentáře, což naznačuje, že kód byl tvořen ručně a podle odkazů (například `header.jspf`) to vypadá, že pro klientskou část aplikace se využívá technologie Java Server Pages. Z toho vyplývá, že celá aplikace je napsána v Javě a nevyužívá architekturu, kde by zobrazovací logika byla oddělena a komunikovala pomocí API. Na klientské části se dále vyskytují JavaScriptové knihovny jako jQuery a Zoomify pro prohlížení archiválií.

Prohlížeč archiválií má dobrý poměr mezi ovládacími prvky a plochou pro zobrazení archiválie. Mezi archiváliemi se lze orientovat a navigovat pomocí postranního menu s náhledy nebo pomocí horního navigačního menu. Aplikace podporuje klávesové zkratky a umožňuje i stažení snímku po vyplnění CAPTCHA. Aplikace dále umožňuje i rotovat sken po malém úhlu. Možnost resetovat nastavení a vrátit snímek do původní pozice zde chybí.



Obrázek 3.2: Prohlížeč archiválií v systému Národního archivu

## 3.2 Moravský zemský archiv

Moravský zemský archiv své matriční knihy badatelům zpřístupňuje na portálu [Actapublica](#)<sup>2</sup>. Jednotlivé matriky lze vyhledávat například podle obce, původce nebo čísla knihy. Vyhledávač obcí je doplněn o našetrvání okresů, což umožňuje vybrat správnou obec při shodě jmen mezi obcemi.

<sup>2</sup><https://www.mza.cz/actapublica/matrika/hledani>

The screenshot shows the search interface for historical records. At the top, there are search fields for 'Vyhledávání' (Search), 'Aktuality' (News), 'Dokumenty' (Documents), 'Nápověda' (Help), and 'Kontaktní formulář' (Contact form). The user has selected 'Czech' language ('Čeština') and is logged in ('Přihlásit se').

**Vyhledávání**

- podle obce**: Název obce (Name of town) - Brno, Vyhledat (Search)
- podle původce**: Typ původce (Type of origin) - vyberte typ původce ze seznamu --, Název obce (Name of town) - České Knínice, Vyhledat (Search)
- podle čísla knihy**: Číslo knihy (Book number) - 210, Číslo snímku (Image number) - 274, Vyhledat nebo zobrazit (Search or view)
- Převodník starých odkazů**: Původní odkaz (Original link) - Brno, Převést na nový odkaz (Convert to new link)

**Aktuální hledání**  
Matrิกy s obci, jejíž český, německý nebo historický název obsahuje:

**Nalezené matrิกy**

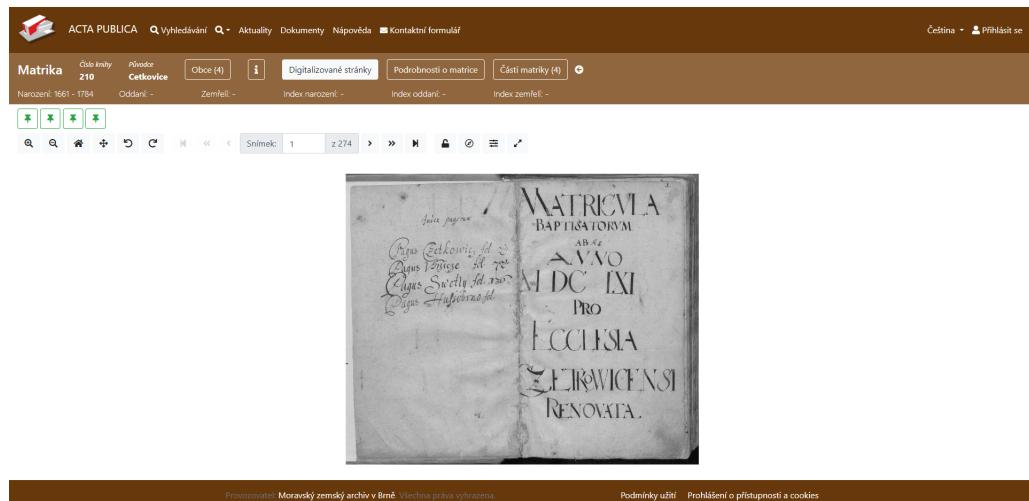
číslo knihy	původce typ původce	Narození od-do index narození	Oddání od-do index oddání	Zemření od-do index zemření	Obce	Počet snímků	
210	Cetkovice římskokatolická díreva	1661 - 1784	-	-	4	274	Zobrazit
211	Cetkovice římskokatolická díreva	-	1784 - 1835	-	5	216	Zobrazit
216	Cetkovice římskokatolická díreva	-	-	1661 - 1784	4	98	Zobrazit
217	Cetkovice	-	-	1784 - 1835	5	99	Zobrazit

Obrázek 3.3: Vyhledávání archiválií v systému Moravského zemského archivu

Podle analýzy komunikace, kde se odesílá ASP.NET\_SessionId, lze předpokládat, že aplikace běží na dotnetu s knihovnou ASP.NET. Nástrojem WhatRuns bylo identifikováno běhové prostředí IIS 7.5 na Windows serveru. Podle struktury volání, kde se vrací celé HTML soubory obsahující čitelné komentáře, lze usoudit, že obdobně jako u Národního archivu se zde jedná o monolitickou aplikaci, která řeší jak aplikační logiku, tak zobrazování. Aplikace pravděpodobně běží na aplikačním rámci ASP.NET MVC. V klientské části aplikace využívá JavaScriptové knihovny jako jQuery, Bootstrap a Popper.JS. Architektura je totožná s předchozím archivním systémem. Pro zobrazování archiválií používá knihovnu [OpenSeaDragon<sup>3</sup>](#) a jako formát skenů používá Deepzoom.

Prohlížeč archiválií je vybaven kvalitním režimem celé obrazovky a je zde možné zobrazit archiválii přes celé okno. Aplikace nenabízí náhled dalších snímků a navigace po archiválii je tedy možná pouze pomocí horního navigačního menu. Aplikace umožňuje rotaci archiválie o 90° a nastavení jasu a kontrastu pro lepší čitelnost archiválie. Kladně hodnotím záložky v rámci archiválie pro snazší vyhledávání. Aplikace nenabízí možnost stáhnutí snímku v plné kvalitě. Není sice odchycen event na canvasu, který by samotné stažení znemožnil, ale použití formátu DeepZoom dělá výsledný stažený snímek dále nepoužitelný.

<sup>3</sup><https://openseadragon.github.io>



Obrázek 3.4: Prohlížeč archiválií v systému Moravského zemského archivu

### 3.3 Zemský archiv v Opavě

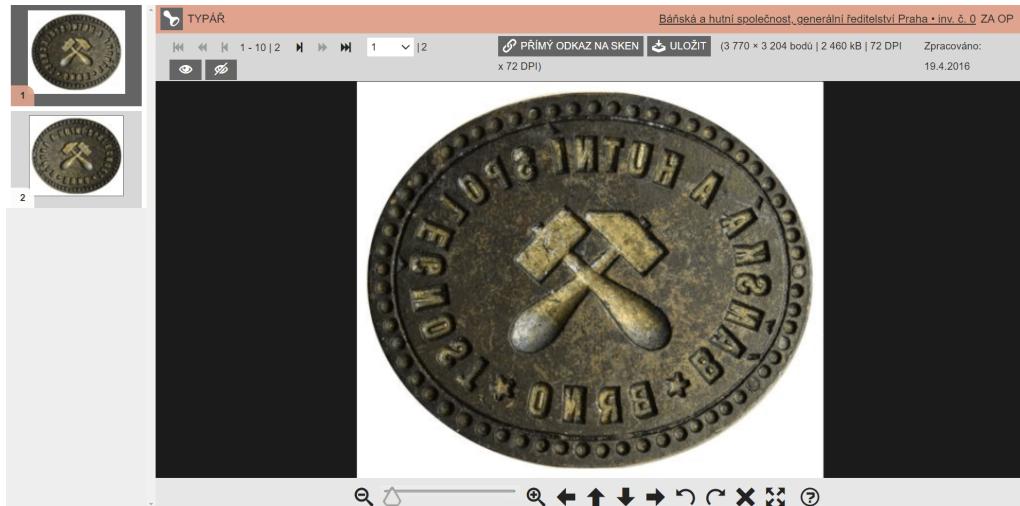
Zemský archiv v Opavě zpřístupňuje svůj [digitální archiv](#)<sup>4</sup>, který má obdobné rozložení i formát zobrazování dat jako Vademecum od Národního archivu. Navíc je zde možnost v rámci filtrování vybrat typ archiválií, kde lze volit mezi matrikami, mapami, kronikami a mnoha dalšími kategoriemi. V rámci archiválií se vyhledává pomocí jednoho plně textového vyhledávače.

Obrázek 3.5: Vyhledávání archiválií v systému Zemského archivu v Opavě

Nalezení SearchBean a odkazu na soubory s příponou jspf napovídá, že se jedná nejen o vzhledově podobné, ale i o aplikace postavené na stejném základu s jistou mírou přizpůsobení. Aplikace Zemského archivu v Opavě běží na Apache Tomcat 4.1+. Obsahuje javascriptové knihovny jako například jQuery, less a jistou míru vlastního javascript kódu. Obdobně jako Národní archiv používá knihovnu Zoomify pro efektivní zobrazování skenů

<sup>4</sup><https://digi.archives.cz>

archiválií. Dle mého názoru se jedná o dost podobné aplikace, které sdílí své klady, ale i zápory, které je zužují.



Obrázek 3.6: Prohlížeč archiválií v systému Zemského archivu v Opavě

### 3.4 Státní oblastní archiv v Třeboni

Státní oblastní archiv v Třeboni zpřístupňuje své materiály pomocí aplikace [Digiarchiv](#)<sup>5</sup>. Rozložení i vzhled aplikace je originální a nepodobá se doposud porovnávaným systémům. V kartě Hledání je možnost filtrovat jednotlivé archiválie podle typu archiválie, archivu i datace. Dále umožňuje vyhledávání nejen pomocí inventárního čísla, ale i plné textové vyhledávání podle klíčového slova.

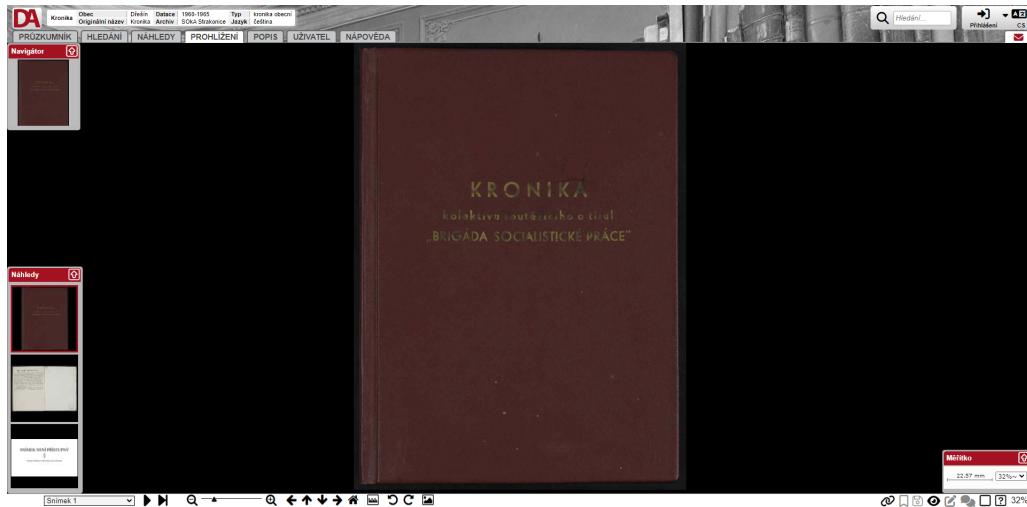
A screenshot of the SOA search interface. The top navigation bar includes links for 'DIGIARCHIV', 'HLEDÁNÍ', 'UZIVATEL', and 'NAPOVEDA'. The search form on the left has fields for 'TYP : VŠE', 'ARCHIV : VŠECHNY', 'DATACE : VŠE', 'ROZŠÍŘENÉ HLEDÁNÍ', 'PŘISTUPOVÉ BODY (KLÍČOVÁ SLOVA)', 'Inventární číslo', and 'Hledání slova'. The search results are listed in a grid format. Each result card contains details about a specific chronicle (Kronika) from a town (Obec), its original name (Originální název), date (Datace), type (Typ), inventory number (Inventární číslo), and the archive it belongs to (Archiv). There are also small thumbnail images of the documents and a link to 'celý popis' (full description).

Obrázek 3.7: Vyhledávání archiválií v systému SOA v Třeboni

Aplikace je hostována na Apache s PHP ve verzi 8.1.10. Z javascriptových knihoven je zde využito Tipped a jQuery. Samotná část webu, která je určena k prohlížení archiválií, je

<sup>5</sup><https://digi.ceskearchivy.cz>

zabalena v iframu a používá javascriptovou knihovnu ckeditor. Aplikace pro zobrazování skenů využívá dlaždicový formát, který je dostupný ze složky cgi-bin, což značí, že jsou dlaždice snímků generovány externím skriptem na webovém serveru. Pro kompletaci snímků v prohlížeči se používá knihovna viewer-cs, ve které se vyskytuje klíčové slovo Zoomify a IIIF, tudíž lze předpokládat, že právě tyto formáty jsou v rámci systému použity.



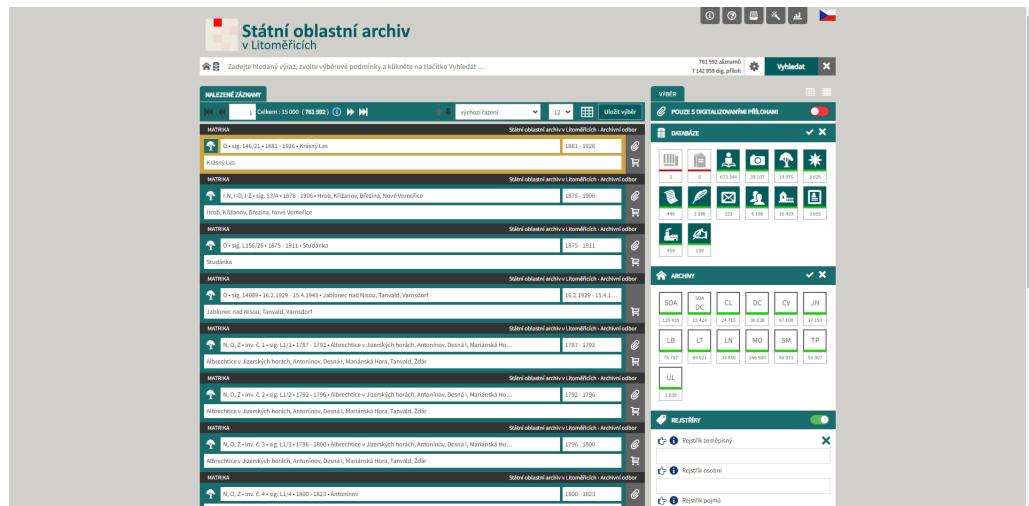
Obrázek 3.8: Prohlížeč archiválií v systému SOA v Třeboni

Samotný prohlížeč nechází dostatek prostoru pro zobrazení skenu archiválie. Na pravé straně obsahuje náhledy dalších skenů a v dolní části je umístěno bohaté nastavení včetně filtrů, jež lze na snímek aplikovat. Hlavní nevýhodou prohlížeče je přechod mezi archiváliemi, který zapříčiní přenačtení celého webu a samotná operace trvá okolo jedné vteřiny. Tato odezva je negativně citelná v případě, kdy badatel chce rychle projít celou archiválií nebo hledá konkrétní informaci.

### 3.5 Státní oblastní archiv v Litoměřicích

Státní oblastní archiv v Litoměřicích poskytuje aplikaci *Vademecum*<sup>6</sup>. Jak již název napovídá, jedná se o totožnou aplikaci, jako má Národní archiv a Zemský archiv v Opavě, přičemž se více podobá digitálnímu archivu z Opavy. Aplikace běží na Javě, využívá podpůrné javascriptové knihovny a formát Zoomify pro zobrazení snímků.

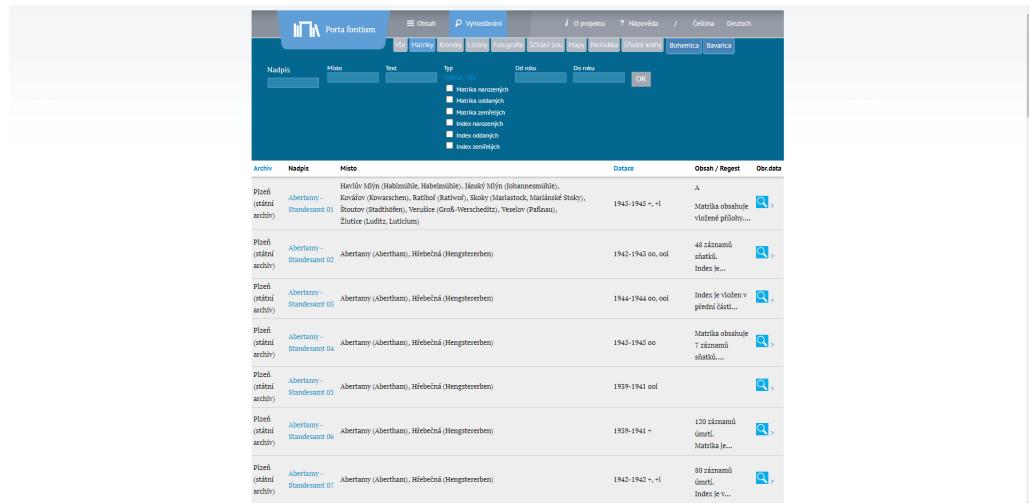
<sup>6</sup><http://vademecum.soalitomerice.cz/vademecum>



Obrázek 3.9: Vyhledávání archiválií v systému SOA v Litoměřicích

### 3.6 Státní oblastní archiv v Plzni

Státní oblastní archiv v Plzni poskytuje své archiválie prostřednictvím [Porta Fontium](#)<sup>7</sup>. Systém nabízí rozdělení jednotlivých záznamů do hierarchické struktury podle původce a oblasti. Toto dělení je podobné systému [Demos](#)<sup>8</sup>. Dále zde lze vyhledávat pomocí místa, textu a roku. Aplikace umožňuje procházet všechny archiválie nebo filtrovat konkrétní typ archiválií.



Obrázek 3.10: Vyhledávání archiválií v systému SOA v Plzni

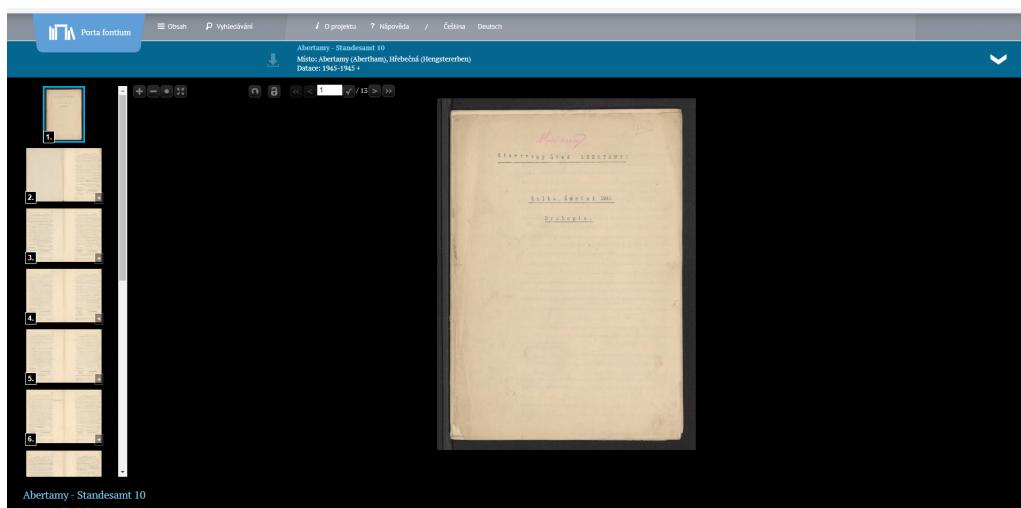
Analýzou komunikace bylo zjištěno, že běží na CMS Drupal 7, který je napsaný v jazyce PHP. Snímky jsou uloženy ve složce fcgi-bin, což nám obdobně jako u Státního oblastního archivu v Třeboni značí, že jsou generovány externím skriptem. Pro dlaždicový formát obrázků zde byl zvolen formát Deepzoom. Z javascriptových knihoven jsou použity Drupal.JS

<sup>7</sup><https://www.portafontium.eu>

<sup>8</sup><http://radegast.fit.vutbr.cz>

a jQuery.

Prohlížeč snímků archiválií je dobře rozložen, obsahuje standardní navigaci nad snímkem archiválie i náhled dalších stran v levém panelu. Z mého pohledu plně nevyužívá plochu pro snímek, která je na stránce k dispozici. Tento nedostatek je negován kvalitním režimem celé obrazovky, který kromě subtilního horního menu zobrazuje archiválii na celé obrazovce. Rychlosť načítání je dostatečná a přechod mezi snímkami je plynulý. Systém postrádá možnost úpravy jasu a kontrastu snímků společně s libovolnou rotací snímků.

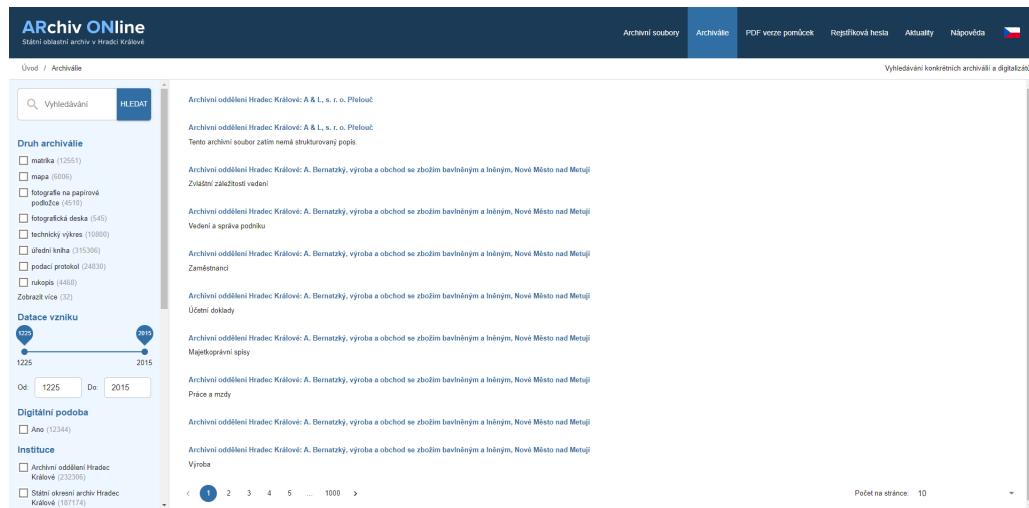


Obrázek 3.11: Prohlížeč archiválií v systému SOA v Plzni

### 3.7 Státní oblastní archiv v Hradci Králové

Státní oblastní archiv v Hradci Králové poskytuje své materiály skrze aplikaci [Archivy Online](#)<sup>9</sup>. Aplikace disponuje moderním uživatelským rozhraním a bohatým nastavením pro filtrování archivních záznamů.

<sup>9</sup><https://aron.vychodoceskearchivy.cz>

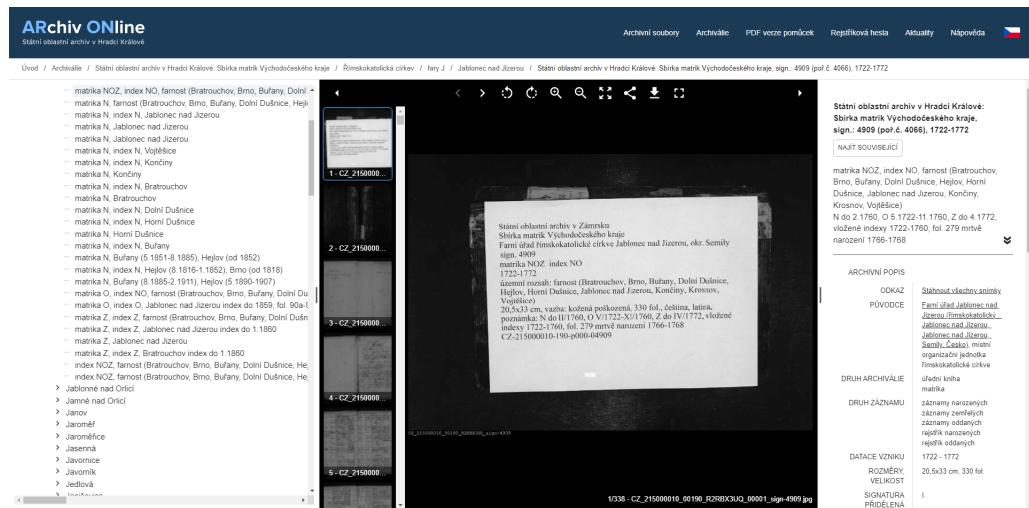


Obrázek 3.12: Vyhledávání archiválů v systému SOA v Hradci Králové

ARON je první aplikace v tomto seznamu, jež využívá separátní aplikaci pro zobrazování klientské části, která komunikuje s API. Klientská část aplikace je napsána v reaktivním javascriptovém aplikačním rámci React.JS ve verzi 16. Po zaslání separátního dotazu na API byla obdržena upravená odpověď 404. Z odpovědi se dá usoudit, že serverová část aplikace běží na Javě, vzhledem k moderní volbě technologie pro frontend se dá očekávat aplikační rámec nad Javou jako například Spring. Pro zobrazování snímků archiválů je zde využita javascriptová knihovna [OpenSeaDragon<sup>10</sup>](#) a jednotlivé snímky jsou uloženy v dlaždicovém formátu Deepzoom.

Prohlížeč snímků archiválů obsahuje mnoho meta informací, takže nezbývá dostatek prostoru pro samotný snímek. Malou plochu pro snímek na detailu archiválie kompenzuje režim celé obrazovky, kde se již dá s daným snímkem phodnotně pracovat. Aplikace umožňuje daný snímek stáhnout ve vysoké kvalitě a obsahuje základní manipulační nástroje, mezi něž například patří rotace obrazu. Zde jsem bohužel narazil na problém, kdy snímek lze otáčet pouze po 90° a daná archiválie byla otočena o 45°, což znemožňovalo natočení řádků do vodorovné polohy. Aplikace dále podporuje klávesové zkratky a jako celek funguje velmi obstojně.

<sup>10</sup><https://openseadragon.github.io>



Obrázek 3.13: Prohlížeč archiválií v systému SOA v Hradci Králové

### 3.8 Státní oblastní archiv v Praze

Státní oblastní archiv v Praze poskytuje svoje archiválie skrze eBadatelnu<sup>11</sup>. Matriky aplikace zvládají vyhledávat pomocí lokality, názvu a původce. Dále je možné specifikovat časový rozsah a jednotlivé výsledky jsou zobrazeny pomocí standardní tabulky.

Obrázek 3.14: Vyhledávač archiválií v systému SOA v Praze

<sup>11</sup><https://ebadatelna.soapraha.cz>

Na základě analýzy bylo zjištěno, že aplikace využívá webový aplikační rámec Apache Wicket, který je napsaný v Javě. Z JavaScriptových knihoven využívá AJAX pro asynchronní nahrávání obsahu a jQuery.

Prohlížeč snímků archiválií nabízí možnost rotace a navigace mezi snímky. Možnost náhledu dalších snímků zde chybí. Při změně snímku probíhá přenačtení celé aplikace a snímky jsou stahovány rovnou v plné kvalitě. Tato kombinace tvoří velmi pomalé rozhraní a změna snímku zde trvá v řádu několika vteřin. Libovolná manipulace s archiválií jako například přiblížení nebo pohyb není plynulá, což snižuje uživatelskou přívětivost pro badatele při práci s tímto systémem.



Obrázek 3.15: Prohlížeč archiválií v systému SOA v Praze

### 3.9 Digitalizace evropských archivů

Digitalizace evropských archivů poskytuje aplikaci [Monasterium<sup>12</sup>](https://www.monasterium.net/mom/fonds) zpřístupňující archiválie z více států včetně České republiky. Aplikace neobsahuje pokročilé vyhledávání a záznamy jsou organizovány pouze podle státu, města a fondu.

<sup>12</sup><https://www.monasterium.net/mom/fonds>

The screenshot shows a search results page for 'Results, Page 1'. At the top, there's a navigation bar with links for Home, Fonds, Collections, Search, Help, MyArchive, Register, Login, and language selection (English). Below the navigation is a search bar with the text 'cces'. The main content area displays two search results. Each result includes a thumbnail image, a title ('Charter: 012-l/14'), a date ('Date: 02.07.1221'), and a brief description. There are also links for 'View Charter In context', 'Browse in search results', 'PDF...', and 'Export'. On the left side, there are filters for 'by date' (selected) and 'relevance ranking', and a 'categories' section with various checkboxes like 'Abstract', 'Manuscript', 'Ceremony', etc. At the bottom, there are 'filter' checkboxes for 'Only hits with images' and 'Only hits with annotations', and a 'Fonds' section.

Obrázek 3.16: Vyhledávač archiválií v systému Monasterium

Aplikace je napsána v Javě a běží v kontejneru Jetty ve verzi 9.4.14. Z JavaScriptových knihoven využívá pouze jQuery a Leaflet pro zobrazení map.

Prohlížeč archiválií zde disponuje omezenou sadou funkcí. Kromě navigace mezi stránkami neumožňuje upravit žádné nastavení, neposkytuje režim celé obrazovky a pohyb po archiválii zde nefunguje pomocí táhnutí archiválie, ale pomocí vertikálního a horizontálního scroll baru. Jediná možnost interakce se snímkem je pomocí scroll barů a jednoho slideru pro nastavení přiblížení. Snímky nepoužívají žádný dlaždicový systém a jsou přednacítány dopředu pro celou archiválii, což ze začátku prohlížení archiválie způsobuje prodlevu. Celková plocha pro prohlížení snímku je velmi malá a při kliknutí na daný snímek se pouze zobrazí v prohlížeči jako zdroj.

The screenshot shows a detailed view of a manuscript page from 'Charter: Archiv Metropolitní kapituly u sv. Vita 012-l/14'. At the top, there's a navigation bar with links for Home, Fonds, Collections, Search, Help, MyArchive, Register, Login, and language selection (English). Below the navigation is a breadcrumb trail: 'Fonds > CZ-APL > AMK > 012-l/14'. The main content area shows a large image of the manuscript page with some text in Latin. Below the image, there are buttons for 'Add bookmark' and 'Edit charter (old editor)'. At the bottom, there are sections for 'Graphics' (with a zoom tool), 'Abstract' (with a date '02.07.1221, Schatzberg'), and a note about the document's origin ('Přemysl I., král český, obnovuje pražskému kostelu svobody a vraci mu hrad Podivín, a').

Obrázek 3.17: Prohlížeč archiválií v systému Monasterium

### 3.10 Matricula online

Dalším zástupcem mezinárodního prohlížeče matrik je [Matricula](#)<sup>13</sup>. Aplikace umožňuje vyhledávat pomocí místa a časového rozsahu. Dále je zde možné klasické prohledávání podle státu a oblasti. Při volbě místa je aktualizována mapa, která zobrazuje místa, odkud archiválie pochází. Zobrazení matrik je provedeno standardní tabulkou s možností filtrování druhu matriky a časového období.

The screenshot shows a search interface for historical baptism records. At the top, there's a header with the CompGen logo and project details. Below it is a table with columns for Signature, Type of record, and Date. The table contains eight rows of data, each representing a baptism entry from Aachen.

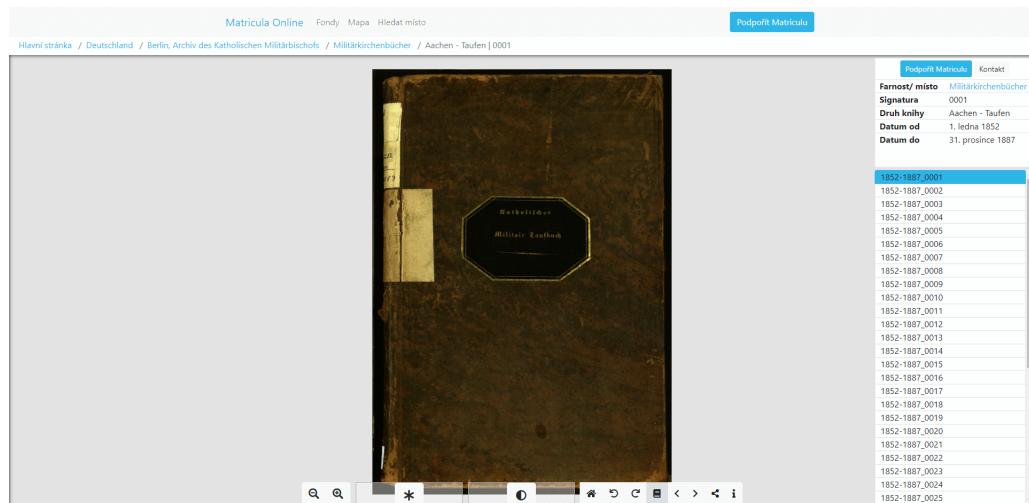
Signature	Druh matriky	Datum (1730 - 1959)
0001	Aachen - Taufen	1852-1887
0002	Aachen - Trauungen	1852-1880
0003	Aachen - Verstorbene	1849-1887
0004	Aachen - Taufen	1899-1920
0005	Aachen - Trauungen	1898-1917
0006	Aachen - Trauungen	1917-1918
0007	Aachen - Verstorbene	1898-1918
0008	Aachen - Taufen, Trauungen, Verstorbene	1937-1942

Obrázek 3.18: Vyhledávač archiválií v systému Matricula

Aplikace je hostována na Apache 2.4.18, které běží na Ubuntu. V rámci JavaScriptových knihoven využívá jQuery, Bootstrap a OpenLayers pro tvorbu map.

Prohlížeč snímků archiválií nepoužívá dlaždicový formát, avšak načítání snímků je relativně rychlé. Aplikace nenabízí režim celé obrazovky, nicméně plocha pro zobrazení archivního snímku je relativně velká oproti konkurenčním nástrojům. Aplikace umožňuje snímek otáčet po 90° a měnit jas s kontrastem. U prohlížení jsem narazil na chybu, která v případě rychlé změny snímků archiválie způsobí to, že aplikace dále na jakoukoliv změnu snímku přestane reagovat a je nutné aplikaci přenačíst.

<sup>13</sup><https://data.matricula-online.eu>



Obrázek 3.19: Prohlížeč archiválií v systému Matricula

### 3.11 Zhodnocení dostupných webových aplikací

Po představení jednotlivých webových aplikací je nutné identifikovat jejich klíčové vlastnosti a sestavení seznamu vlastností, které by prohlížeč archiválií měl dle mého názoru obsahovat. Při prohlížení archiválií jsou preferovány systémy asynchronně načítající další stránky a kde nedochází k přenačtení celého rozhraní. Dalším důležitým aspektem je plocha rozhraní, jež jednotlivé systémy alokovaly pro samotné zobrazení archiválie. Za velmi důležitou vlastnost systému považuji režim zobrazení na celou obrazovku, během kterého musí být maximalizována plocha pro zobrazení dané archiválie.

Za mandatorní prvky prohlížeče archiválií považuji ovládací panel a náhled dalších snímků, což ne všechny systémy poskytovaly. Některé systémy umožňovaly efektivní práci pomocí předdefinovaných klávesových zkratek, avšak u spousty systémů chyběla intuitivní nápověda, kde by se uživatel naučil se zkratek rychle pracovat.

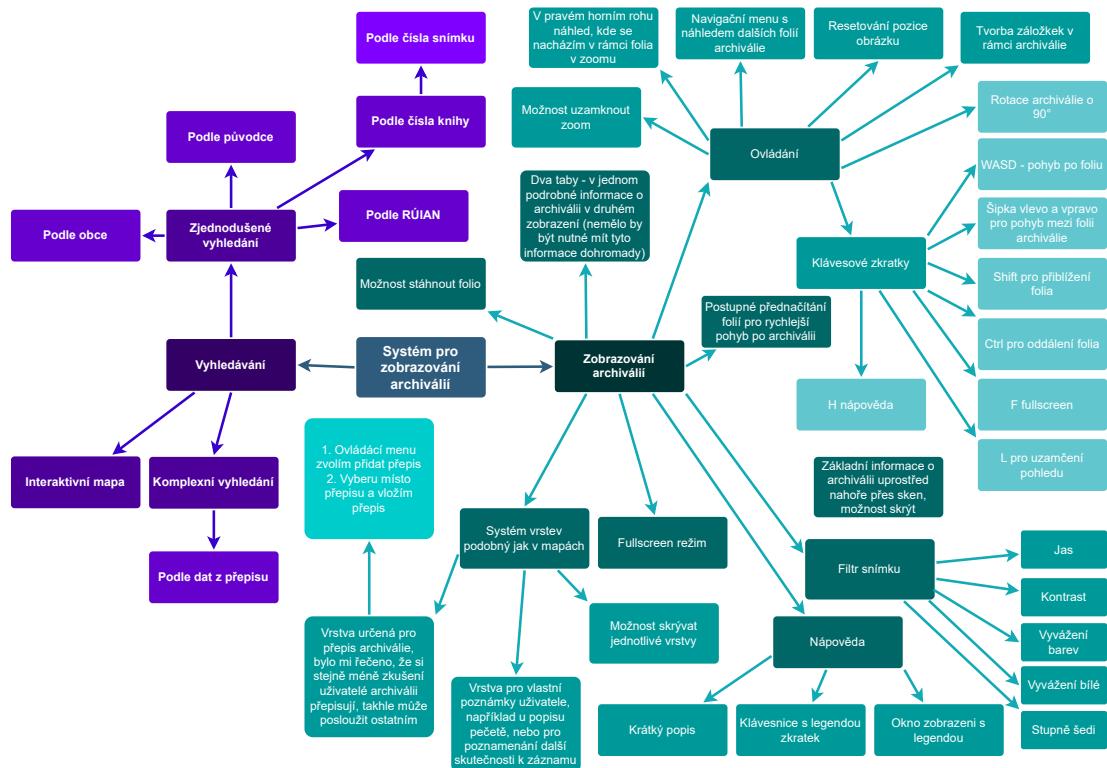
Jelikož výsledná kvalita zdigitalizovaných skenů není konzistentní, tak je vhodné, aby prohlížeč umožňoval rotovat snímky o libovolný stupeň, což většina systémů neumožňovala. V systému ARON existuje archiválie, která je naskenována pod úhlem  $45^\circ$ , ale aplikace umožňuje rotovat archiválii pouze po  $90^\circ$ , což znemožňuje srovnat řádky textu do vodorovné polohy. Další kladnou vlastností některých systémů je možnost manuální úpravy jasu a kontrastu skenů. Některé archiválie nejsou naskenovány v čitelné podobě a toto nastavení může v mnoha případech zlepšit čitelnost. Při práci s archiválií se uživatel často zaměřuje pouze na jistou část skenu, na níž má přiblíženo. Z tohoto důvodu má většina systémů funkci Zachovat zobrazení, která při změně skenu zachová aktuální přiblížení a pozici v rámci archiválie. V rámci systému Actapublica byly definovány záložky, což usnadňovalo vyhledávání v matrikách, kde mohou být záznamy z více obcí.

Vyhledávání konkrétní archiválie se liší napříč systémy, které nabízejí různé možnosti od hledání podle určitých atributů až po plné textové vyhledávání. Velmi užitečná je funkce, kdy systém při našepťávání názvu obcí zobrazil také informaci o kraji a okrese. Tuto funkcionality ocení badatelé v případě, kdy existuje více obcí se stejným názvem a potřebují je

od sebe odlišit.

K jednoznačné identifikaci obce lze využít identifikátor RÚIAN [21]. Registr územní identifikace adres a nemovitostí je jedinečným identifikátorem v rámci České republiky pro územní prvky, územní evidenční jednotky a adresy. Kromě jedinečné identifikace se dále RÚIAN používá k určení jejich vlastníků a zjištění podrobnějších informací o jednotlivých parcelách a budovách. Celý informační systém spravuje Český úřad zeměměřický a katastrální. Tento systém se kromě správy nemovitostí dále používá pro urbanismus a územní plánování.

Pro lepší přehlednost je plánovaná funkcionality systému rozdělena do kategorií a zaznamenána v diagramu.



Obrázek 3.20: Diagram funkcionality

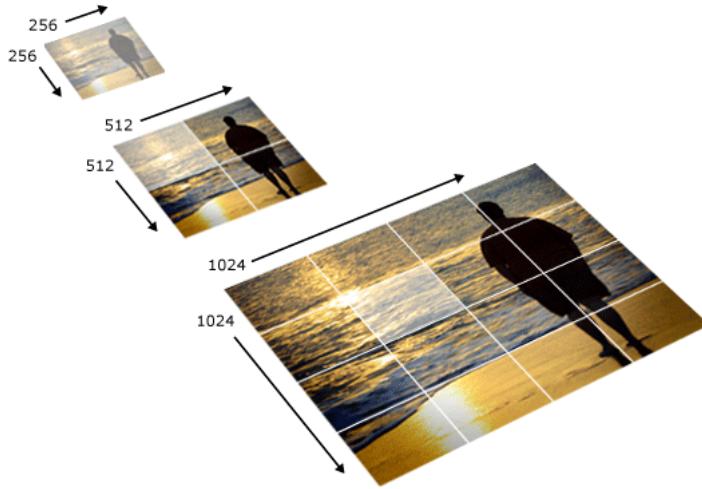
## Kapitola 4

# Formáty pro efektivní zobrazování velkých snímků na webu

V této kapitole bude popsána problematika toho, jak v rámci webu optimalizovaně zobrazovat snímky s vysokým rozlišením. Princip je následující, že je snímek oddálený, tak uživatel na svém monitoru stejně nevidí každý detail a ze serveru se zbytečně stahoval příliš detailní snímek. Přitom by stačilo, kdyby se vždy dostávala detailněji pouze ta část snímku, na kterou se uživatel zaměří. Přesně tento koncept využívají dlaždicové formáty, jež uchovávají snímky s různou mírou detailu. Většinou uživatele detailně zajímá pouze část obrázku a zbytek se stahuje úplně zbytečně. Další výhodou těchto formátů je kromě úspory přenosu dat bezpečnost autorských práv, jelikož snímek nelze tak jednoduše stáhnout. V případě stažení oddáleného snímku totiž bude výstup v nízké kvalitě. Tyto formáty se často využívají například u webových galerií nebo map, kde není žádoucí provést pouhou kompresi a poskytnout tak méně kvalitní verzi snímku. Kromě definice samotného formátu k němu existuje i podpora knihoven, které s daným formátem usnadní práci vývojářům.

### 4.1 Deep Zoom Image

Deep Zoom [8] je technologie od společnosti Microsoft určená pro efektivní zobrazování velkých snímků na webu. Technologie nahraje adekvátně kvalitní obrázek vzhledem k přiblížení a nabízí podporu jak pro jeden snímek, tak pro celou kolekci snímků. Snímek se při konverzi rozdělí na malé dlaždice a ty se uspořádají do pyramidového schématu. Při prvním načtení oddáleného obrázku se použije dlaždice z vrchu pyramidy, jež na malém rozlišení pokrývá celý snímek v nízké kvalitě. Jakmile uživatel začne přibližovat snímek, tak se vždy dostahuje detailnější snímek z nižší vrstvy pyramidy, který na stejném rozlišení vyobrazuje pouze část originálního snímku, takže celkový efekt spočívá v detailnějším snímku pro uživatele, a to bez stahování celého snímku ve velkém rozlišení.



Obrázek 4.1: Rozložení obrázku na dlaždice v rámci technologie Deep Zoom<sup>1</sup>

Informace o jednotlivých dlaždicích jsou zakódovány následující strukturou `[úroveň]/[slopec]_[řádek].[přípona souboru]` ve formátu XML. Na každé úrovni pyramidy je předešlá dlaždice rozložena na čtyři detailnější. Deep Zoom také podporuje řídké obrázky, kde různé části obrázku mohou mít různou úroveň detailu.

## 4.2 Zoomify

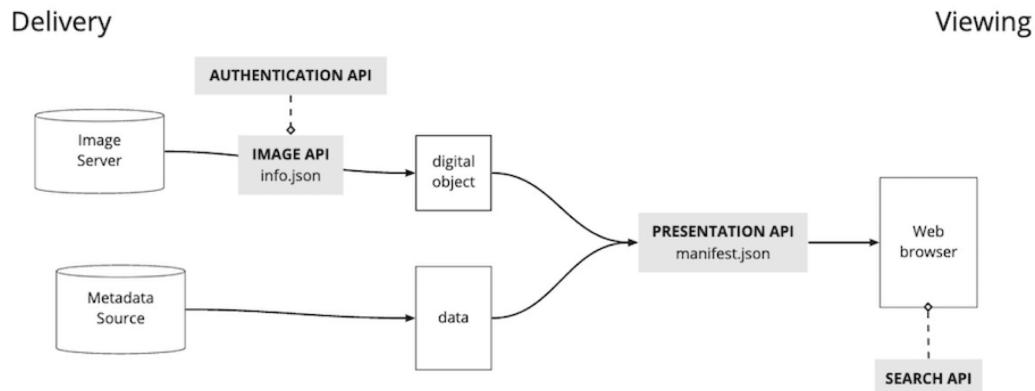
Zoomify [47] je konkurenční cross-platform opensource formát k Deep Zoom image, který vytvořila stejnojmenná společnost. Oba formáty mají mnoho společných rysů a pro popis jednotlivých dlaždic používají formát XML. Snímek je rozkládán do hierarchické pyramidy dlaždic o maximální velikosti  $256 \times 256$  pixelů. V další vrstvě se dlaždice agregují, takže jedna dlaždice obsahuje data ze čtyř dlaždic nižší úrovně. Dlaždice se tvoří zleva doprava a následně shora dolů. Indexování obrázků probíhá také obdobně jako u formátu Deep zoom `[[úroveň]]-[slopec]_[řádek].[přípona souboru]`. Indexování začíná od nuly, takže dlaždice, která obsahuje celý obrázek ve velikosti maximálně  $256 \times 256$  pixelů, má souřadnice `0-0-0.[přípona souboru]`.

## 4.3 International Image Interoperability Framework

IIIF [16] se na rozdíl od předchozích formátů nezaměřuje pouze na snímky, ale řeší poskytování veškerého audiovizuálního obsahu v rámci webu včetně metadat. Metadata jsou chápána jako data o datech a patří mezi ně například názvy, titulky a popisy. V rámci zobrazování snímků podporuje funkci přiblížování a načítání pouze části snímku stejně jako předchozí formáty. Tento aplikační rámec rozděluje svoji funkcionalitu na dvě části, doručování a zobrazování obsahu. Dále odděluje uložení dat a metadat, která spojuje na základě definice v souboru `manifest.json`. Aplikační rámec IIIF také umožnuje použít stejné standardizované principy, a to například pro poskytování autentizace a vyhledávání obsahu.

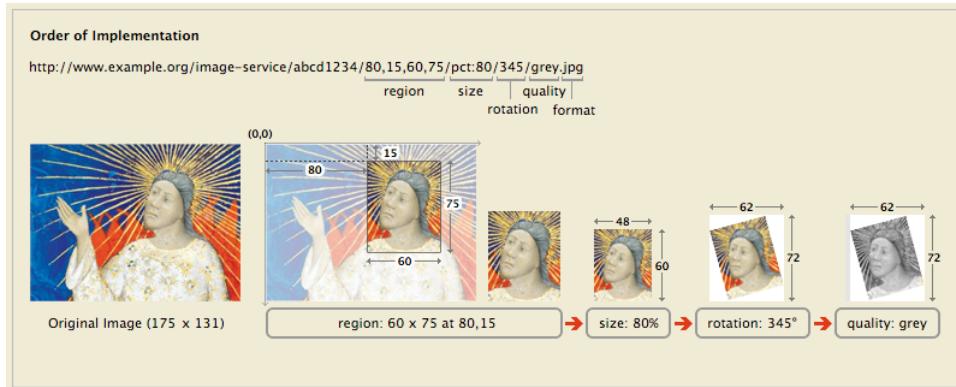
---

<sup>1</sup>Obrázek pochází ze stejného zdroje jako informace o Deep Zoom formátu.



Obrázek 4.2: Architektura pro použití IIIF<sup>2</sup>

V rámci podpory snímků umožňuje aplikační rámeček IIIF do URL volání daného API specifikovat parametry pro volbu výřezu snímku. Dále existují parametry pro nastavení velikostí, rotace a případně filtrování pro výsledný snímek. Oproti ostatním formátům je zde výřez specifikován pozici levého horního rohu a následně šírkou a výškou výřezu, což umožňuje přesně specifikovat část obrázku, která má být zobrazena.



Obrázek 4.3: Zpracování obrázku pomocí IIIF<sup>2</sup>

<sup>2</sup>Obrázek pochází ze stejného zdroje jako informace o IIIF formátu.

## Kapitola 5

# Webové technologie

V této části diplomové práce bude provedeno porovnání běžně dostupných technologií pro tvorbu webových aplikací a budou představeny základní návrhové vzory, které se při tvorbě webových aplikací využívají.

Nejjednodušší architektura [48] při stavbě webové aplikace je monolitická, kdy je aplikace vyvíjena a nasazena jako jedna aplikace (Tier), která může komunikovat s externími službami, například s databází. Při použití monolitické architektury se často využívá návrhový vzor MVC, v němž model uchovává stav aplikace a je zodpovědný za jeho aktualizaci a informování view o změně. View je oddělený od logiky aplikace a je zodpovědný za prezentaci dat uživateli. Controller má za úkol zpracování uživatelských vstupů a propojení modelu a view se zachováním nezávislosti mezi nimi.

Systematičtější přístup při vývoji monolitické architektury je strukturalizace aplikace do logických vrstev (layers), kde se standardně používají tři vrstvy. Nejnižší vrstva zapouzdruje práci s databází, druhá vrstva se využívá pro implementaci business logiky a třetí slouží pro prezentaci dat uživateli.

Monolitická architektura má hlavní výhodu v jednoduchosti a rychlosti vývoje, ovšem problém zde nastává v případě, kdy je potřeba aplikaci škálovat na vyšší zátěž. V tomto případě jsme převážně odkázáni na vertikální škálování.

Modernější přístup k vývoji webových aplikací je plně separovat klientskou část, jež je zodpovědná za prezentaci a aplikační logiku. Klientská část pak typicky bývá napsána v reaktivním JavaScriptovém frameworku a komunikuje se serverovou částí aplikace pomocí REST nebo GraphQL API. Serverová část zůstává trívrstvá, ale s tím rozdílem, že místo renderování view se vrací pouze data, a to prostřednictvím aplikačního rozhraní většinou ve formátu JSON nebo u starších systémů ve formátu XML.

V případě návrhu rozsáhlých aplikací, kde se očekává vysoká zátěž, lze využít návrh rozložení aplikace do jednotlivých mikroslužeb. Monolitická serverová část je rozdělena do více menších aplikací, kde každá aplikace může mít vlastní databázi a komunikovat s ostatními synchronně pomocí RPC nebo asynchronně pomocí message brokeru, jako je například RabbitMQ. Tato architektura je absolutně nejvolnější vzhledem ke škálování aplikace a umožňuje kromě vertikálního i horizontální škálování.

## 5.1 Serverové technologie

Serverové technologie slouží k tvorbě API, případně monolitických systémů. Víceméně každý modernější programovací jazyk má vlastní aplikační rámec pro tvorbu webových aplikací. Tyto aplikační rámce často sdílí podobné rysy a přechod na jiný aplikační rámec je pro zkušeného programátora velmi intuitivní a týká se pouze lehkých změn v syntaxi daného jazyka a jmenných konvencí, které se historicky v daném jazyce využívají. Jednotlivé programovací jazyky se liší v typování, kde jsou jazyky, co vyžadují statické typování, jiné zase datový typ odvozují, případně i dynamicky přetypovávají. Jako příklad lze uvést, že v rámci Javy se více využívá deklarativní programování, kdy některé konstrukce, jako například transakce, lze vyjádřit pomocí anotací. Většina těchto jazyků je dnes interpretována, což za cenu pomalejšího běhu dodává například možnosti modifikace kódu za běhu a přístup k objektům pomocí reflexe.

## 5.2 Technologie pro ukládání a poskytování dat

Většina těchto aplikací vyžaduje perzistentní uložení dat. Značnou část těchto webových aplikací tvoří OLTP a OLAP systémy. V rámci OLTP systémů je kladen velký důraz na konzistenci, jelikož modelují nějaký fyzický podsystém. Mezi modelem a skutečným fyzickým podsystémem musí existovat isomorfismus. K dosažení těchto cílů je často volena SQL databáze, jelikož plní podmínky ACID. Další možnosti pro implementaci úložiště pro OLTP systémy jsou NewSQL databáze, které jsou lépe připraveny do distribuovaného prostředí.

Dalším typem úložiště je NoSQL databáze, která neklade takový důraz na konzistenci a používá model BASE. Jednotlivé NoSQL databáze se od sebe velmi liší a každá z nich má specifický případ použití, takže je nutné v době návrhu zvolit vhodnou databázi. NoSQL databáze jsou v základu úložiště klíč-hodnota, kde klíče je možné ukládat v hashovacích tabulkách nebo ve stromové struktuře, což ovlivňuje možnost range scanu a rychlosť vyhledávání. V rámci NoSQL databází, jež se zaměřují na zvládání velkého množství zápisů, se používá datová struktura LSM tree. Jiné databáze se zase hodí pro optimalizované čtení, kde oproti SQL databázím se netrvá na normalizaci dat. Datová redundance je zaváděna záměrně, a to pro zrychlení dotazování.

## 5.3 Klientské technologie

Mezi klientské technologie se kromě standardního HTML, CSS a čistého JavaScriptu také řadí technologie používané u monolitických architektur, jako jsou Blazor, Java Server Pages a Java Server Faces. Nevýhoda tohoto řešení je, že při požadavku může docházet k přenačtení celého obsahu, což se dá vyřešit asynchronním voláním například pomocí technologie AJAX nebo v případě Blazoru může existovat spojení pomocí websocketů.

Při použití separátní aplikace pro klientskou část aplikace se v dnešní době standardně využívají reaktivní JavaScriptové aplikační rámce, které byly navrženy na tvorbu interaktivních webových aplikací, kde se veškeré volání provádí asynchronně, takže neblokuje aplikaci. Standardně se zde využívá vývoj pomocí komponent, kde každá komponenta může mít vnitřní stav. Ve chvíli kdy dojde ke změně vnitřního stavu, dojde následně i k překres-

lení výsledného pohledu. Mezi zástupce těchto technologií patří Vue.JS, Svelte, React nebo Angular.

# Kapitola 6

## Datová analýza

Cílem této práce je vytvořit datacentrickou aplikaci, jejímž vstupem mají být data nasbírána z různých archivů po celé České republice. Tématu sběru těchto dat se již v minulosti věnovali i jiní studenti a nyní proběhne pokus o jejich sjednocení a vytvoření ucelené datové sady použitelné pro web, který má tyto archiválie zobrazovat. Pro každou archiválii jsou potřeba její metadata, seznam URL skenů a obce, ze kterých data pocházejí. Informace o obcích musí obsahovat podrobnější informace o okresu, kraji a RÚIAN, aby bylo možné obec jednoznačně identifikovat z důvodu kolizí názvů obcí.

Pro vypracování datové analýzy byl zvolen programovací jazyk Python, který je vhodný pro datovou analýzu vzhledem k čitelné syntaxi a velké škále podpůrných knihoven pro datovou analýzu. Protokol byl vytvořen v interaktivním prostředí Jupyter notebook, jež kombinuje zobrazení zdrojového kódu a výstupů v podobě grafů. Jupyter notebook dále umožňuje výstup exportovat, a to například ve formátu PDF. Datové sady byly pro snazší práci nahrány do dataframů knihovny Pandas a grafy byly vizualizovány knihovnou Matplotlib. Dále proběhla interpretace výstupu datové analýzy. Tento výstup je dostupný v příloze A Výstup datové analýzy.

### 6.1 Datová sada Dominika Popa

Datová sada matrik od Dominika Popa [40] je distribuovaná ve formátu JSON a obsahuje detailnější informace pro území jednotlivých archivalií. Datová sada musela být před samotnou datovou analýzou upravena, jelikož využívala dynamický klíč názvu obcí, což po normalizaci vytvářelo velmi řídký dataset znemožňující další zpracování.

### 6.2 Datová sada Jakuba Sokolíka

Datová sada Jakuba Sokolíka [50] obsahuje data nejen pro matriky, ale i pro ostatní typy archivalií a obdobně jako předešlá datová sada je dostupná ve formátu JSON. Celá datová sada byla dodána v jednom souboru, který kvůli své velikosti nebyl dále použitelný, a proto byl manuálně rozdělen podle jednotlivých archivů. Před samotnou analýzou bylo nutné opravit překlepy v klíčích ohledně údajů RÚIAN. Datová sada ve všech grafech obsahuje nejvíce záznamů. Toto je zapříčiněno skutečností, že v případě, kdy archiválie obsahovala více obcí, tak se v datové sadě objeví záznam pro každou obec duplicitně. Tedy datová

sada nepracuje se seznamem lokalit, ale pouze s jednou lokalitou a v případě, kdy lokalita obsahuje více obcí, tak se záznam zduplicuje.

### 6.3 Datová sada Jana Valuška

Datová sada [51] je strukturována do jednotlivých JSON souborů podle archivu a typu archivního materiálu. Obdobně jako datová sada Jakuba Sokolíka obsahuje data pro matriky i ostatní typy archiválií.

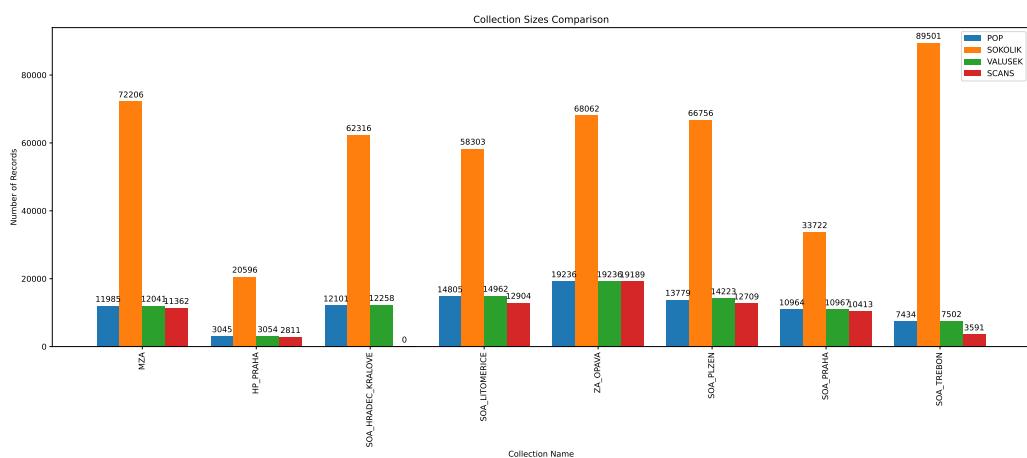
### 6.4 Datová sada obsahující URL snímky

Datová sada s URL adresami snímků je organizována podle archivů, odkud data pochází. Data jsou ve formátu JSON, a to až na matriky z Moravského zemského archivu, který jako jediný je ve formátu XML. Pro účely další analýzy byl vytvořen Python skript, který provedl transformaci na ekvivalentní JSON dokument. Formát dat není napříč celou datovou sadou jednotný, takže v příloze A je uvedena struktura, jež by vznikla při sjednocení souborů napříč datovou sadou. Při kontrole platnosti uvedených URL adres bylo zjištěno, že funkční odkazy jsou pouze pro Archiv hlavního města Prahy, SOA v Plzni, SOA v Litoměřicích a ZA v Opavě.

### 6.5 Porovnání datových sad

V této části práce bude zkoumán vztah jednotlivých datových sad a proběhne mapování napříč datovými sadami.

Jak naznačuje graf 6.1 porovnávající velikosti jednotlivých datových sad, tak zatímco datová sada Dominika Popa, Jana Valuška a datová sada s url adresami skenů obsahují přibližně totožný počet záznamů, tak datová sada od Jakuba Sokolíka obsahuje významně více záznamů, což je zapříčiněno tím, že neobsahuje data pouze pro matriky, ale i pro jiné typy archiválií, a v případě, že archiválie pokrývá více obcí, tak je zde záznam duplikován.



Obrázek 6.1: Porovnání velikosti jednotlivých datových sad

### **6.5.1 Moravský zemský archiv**

Jak je patrné z analýzy chybějících hodnot, tak každá datová sada obsahuje vždy jeden identifikátor. V rámci datové sady Jakuba Sokolíka, Jana Valuška a sady obsahující skeny se jedná o signaturu, ale v rámci datové sady Dominika Popa se jedná o inventární číslo. Toto může být zapříčiněno tím, že uživatelské rozhraní Actapublica od MZA používá označení Číslo knihy, což mohlo být jednotlivými autory vyloženo odlišně. Dále proběhlo párování záznamů přes vyplňené identifikátory a v případě, že se shodují, tak lze předpokládat, že se jedná o totožné záznamy. Napříč datovými sadami neexistuje záznam, který by neměl ani signaturu ani inventární číslo. Z kontroly duplicit vyplývá, že vyplňené identifikátory v rámci datových sad jsou jedinečné, a to až na datovou sadu Jakuba Sokolíka, která se potýká s problémem struktury uložení dat. Problém se strukturou dat byl nastíněn v popisu jeho datové sady, kdy v případě více vesnic v rámci jedné archiválie jsou vytvářeny duplicitní záznamy. Z Vennova diagramu lze vyčíst, že se povedlo namapovat většinu záznamů napříč všemi datovými sadami. Při kontrole konzistence bylo ověřeno, že data ohledně obcí a jejich RÚIANy se shodují ve většině případů. Porovnány byly datové sady Dominika Popa a Jakuba Sokolíka, jelikož jako jediné obsahují RÚIAN identifikátory. Více než 70 % vesnic se shoduje v názvu i identifikátoru. V opačném případě se jednalo převážně o problém, že se nenašel záznam v obou datových sadách. Neshoda mezi identifikátory nastala u 7 % záznamů.

### **6.5.2 Archiv hlavního města Praha**

Z analýzy chybějících hodnot vyplynulo, že zde se již autoři shodují v identifikátorech a bude tedy stačit porovnat shodu mezi signaturami archiválí. Z analýzy duplicitních hodnot vyplývá, že signatura je unikátní v rámci všech datových sad kromě Jakuba Sokolíka, kde duplicity jsou zde opět způsobeny nešťastným strukturováním dat. Při kontrole konzistence bylo zjištěno, že většina záznamů se nachází ve všech datových sadách. Problém nastává u shody obcí v rámci archiválí, kde se povedlo namapovat pouze 7 % záznamů. Při detailnějším přezkoumání bylo zjištěno, že Dominik Pop použil detailnější popis oblasti, což způsobuje nekonzistence napříč obcemi. Jako příklad lze uvést, že v rámci datové sady Jakuba Sokolíka je uvedena Praha a v datové sadě Dominika Popa je uvedeno Staré Město, což je část Prahy a jedná se o přesnější údaj. I přes fakt, že obě datové sady mají správné RÚIAN čísla, došlo ke špatnému výsledku mapování, a to kvůli rozdílné úrovni granularity. Všechny záznamy, které se povedlo namapovat, se shodují v údajích o RÚIANu. Přes 92 % záznamů se povedlo dohledat i v rámci datové sady se skeny, což značí, že tyto výstupy budou použitelné, jelikož pro tento archiv jsou dostupné URL adresy funkční.

### **6.5.3 Státní oblastní archiv v Hradci Králové**

Pro státní oblastní archiv v Hradci králové chybí datová sada se skeny a nejsou tak k dispozici nascrapované URL jednotlivých snímků. Lze provést pouze porovnání datových sad Jakuba Sokolíka, Jana Valuška a Dominika Popa, které obsahují informaci o signatuře i inventárním čísle. Podle analýzy chybějících hodnot bylo odhaleno, že několik záznamů postrádá informaci o inventárním čísle. Z analýzy duplicitních hodnot vyplývá, že datové sady Dominika Popa a Jana Valuška jsou konzistentní a lze tedy vůči nim porovnávat data z datové sady Jakuba Sokolíka. Z Vennova diagramu lze vyčíst, že dostupné datové sady se shodují u více jak 9 000 záznamů. Z výsledků vyplývá, že datová sada Dominika Popa se

úplně neshoduje s obcemi v druhé datové sadě, každopádně v případech, kdy daná obec je v obou datových sadách, tak se ve většině případech RÚIAN identifikátory shodují.

#### **6.5.4 Státní oblastní archiv v Litoměřicích**

Z analýzy chybějících hodnot lze vyčíst, že datové sady Jakuba Sokolíka a Jana Valuška neposkytují informaci o signatuře ani o inventárním čísle. Pro další analýzu je tedy nelze využít a budou porovnány pouze signatury ze dvou datových sad. Duplicity se v datové sadě Dominika Popa a další sadě skenů nevyskytují. Mapování datových sad proběhlo úspěšně u většiny záznamů napříč datovými sadami. Provádět kontrolu konzistence RÚIANů zde není možné, jelikož chybí druhá datová sada, která by tento údaj obsahovala.

#### **6.5.5 Zemský archiv v Opavě**

Z analýzy chybějících hodnot vyplývá, že se zde autoři shodují a všechny datové sady obsahují údaje o signatuře. Datová sada Jakuba Sokolíka opět obsahuje duplicity. Zbytek duplicit vyplývá z chybějících hodnot. Z Vennova diagramu lze vyčíst, že se datové sady z velké části shodují. Shoda nastává i mezi názvy obcí a RÚIAN identifikátory.

#### **6.5.6 Státní oblastní archiv v Plzni**

Podle chybějících hodnot zde proběhne mapování záznamů podle signatury, která, kromě již výše zmínované datové sady Jakuba Sokolíka, nemá duplicity. Celkem se povedlo namapovat přes 9 000 záznamů. Část obcí se povedlo úspěšně namapovat, nicméně několik záznamů vykazovalo chybu.

#### **6.5.7 Státní oblastní archiv v Praze**

Podle chybějících hodnot zde mapování proběhne na základě signatury. Datová sada Dominika Popa byla vyřazena z důvodu chybějících identifikátorů. Analýza duplicit naznačuje, že se jedná o jedinečné identifikátory. Vennův diagram poukazuje pouze na částečnou shodu mezi třemi datovými sadami a signifikantní shodu mezi sadou Jana Valuška a další sadou se skeny. Kontrola konzistence RÚIANů zde opět nemá smysl z důvodu absence druhé datové sady s informacemi o obcích.

#### **6.5.8 Státní oblastní archiv v Třeboni**

Všechny datové sady obsahují informaci o signatuře. Analýza duplicit odhalila anomálie, jelikož jsou zde duplicity i u datových sad, kde doposud žádný problém nebyl. Vennův diagram také naznačuje problém s mapováním, protože data jsou rozeseta napříč celým diagramem. Mezi datovými sadami byly nalezeny i různé struktury URL adres, což značí, že možná byla nasrapována různá data napříč sadami.

#### **6.5.9 Výsledky datové analýzy**

Většinu datových sad se povedlo namapovat a vypadá to, že jednotlivé scrapery jsou konzistentní. Standardně by stačilo sloučit datové sady, ale aplikace má podporovat automatické aktualizace, a proto to není možné. V rámci implementovaného systému bude muset být jeden ze scrapovacích systémů integrován.

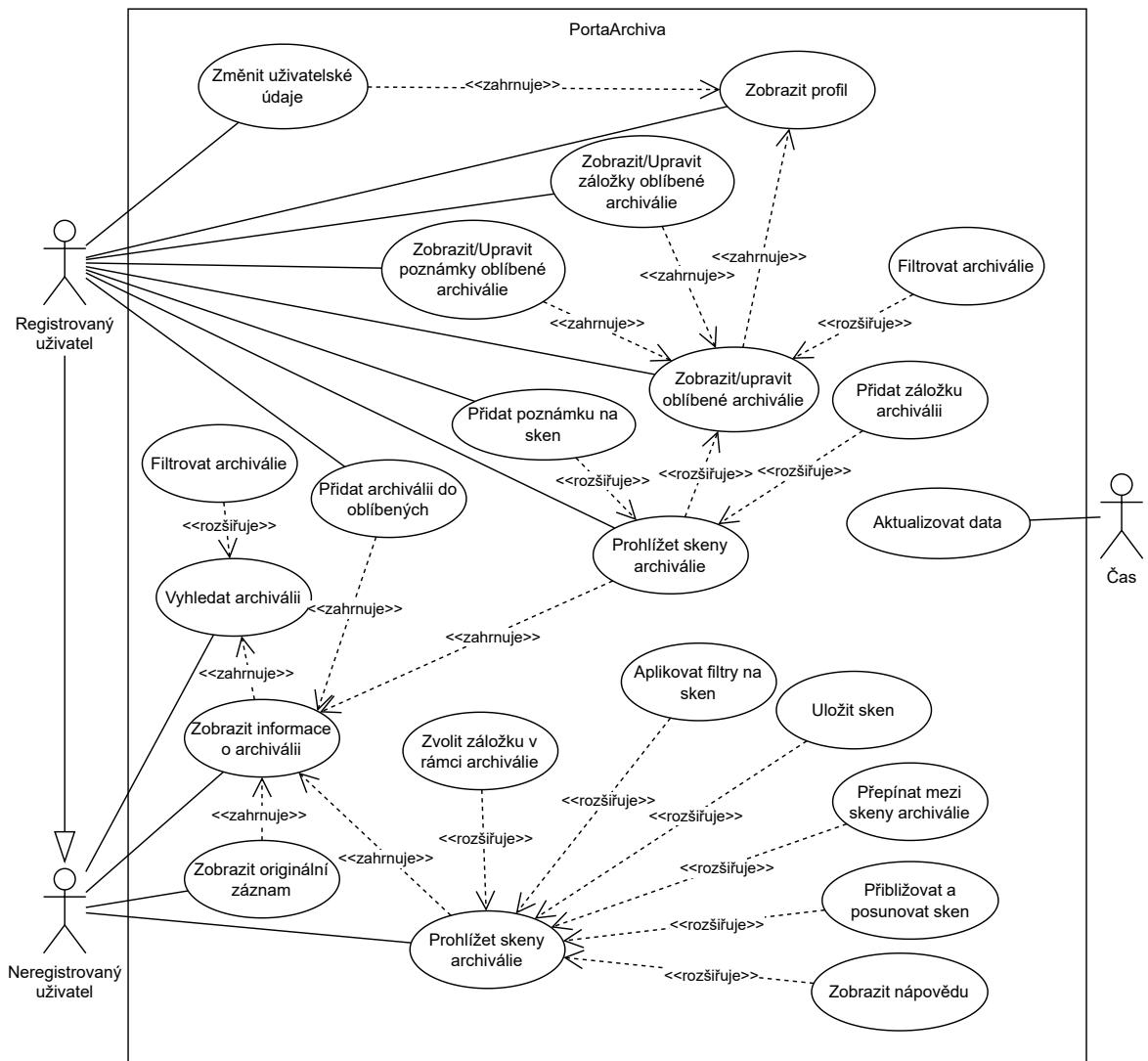
# Kapitola 7

## Analýza požadavků

Analýza požadavků je prvotní fáze návrhu softwaru a zaměřuje se na sběr požadavků, jejich pochopení a formalizaci. Dále zde může být zahrnuta studie proveditelnosti, formalizace požadavků a akceptační testování. Prvotní požadavky jsou často sbírány analýzou stávajícího řešení nebo řízeným rozhovorem se zadavatelem. V mém případě se jedná o analýzu stávajících řešení, kterou jsem provedl v rámci kapitoly 3 Webové aplikace pro zpřístupnění digitalizovaných archiválií.

### 7.1 Diagram případů užití

Diagram případů užití [43] je UML diagram chování zachycující funkcionality systému a aktéry, kteří v rámci systému mohou vystupovat. Z diagramu je dále patrný rozsah implementovaného systému. V tomto systému vystupuje pouze role uživatele, jenž v případě, že je přihlášen, má vyšší práva a může provádět více operací jako například přidávat poznámky a záložky k jednotlivých archivním skenům.

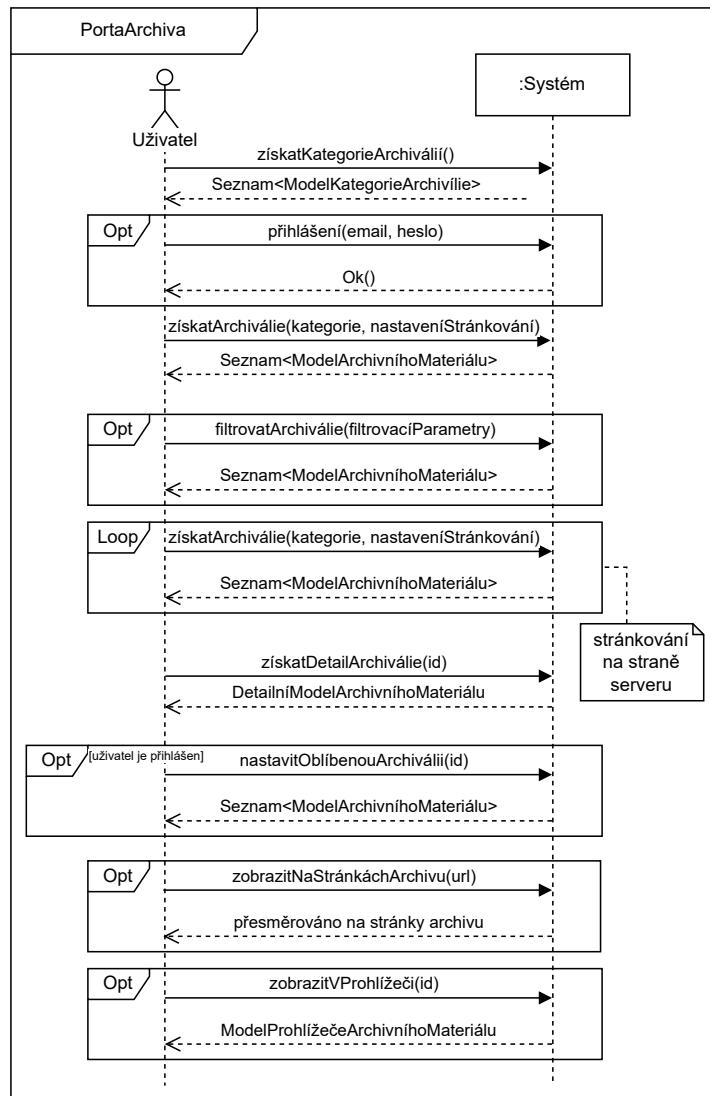


Obrázek 7.1: Diagram případů užití

## 7.2 Systémové diagramy sekvence

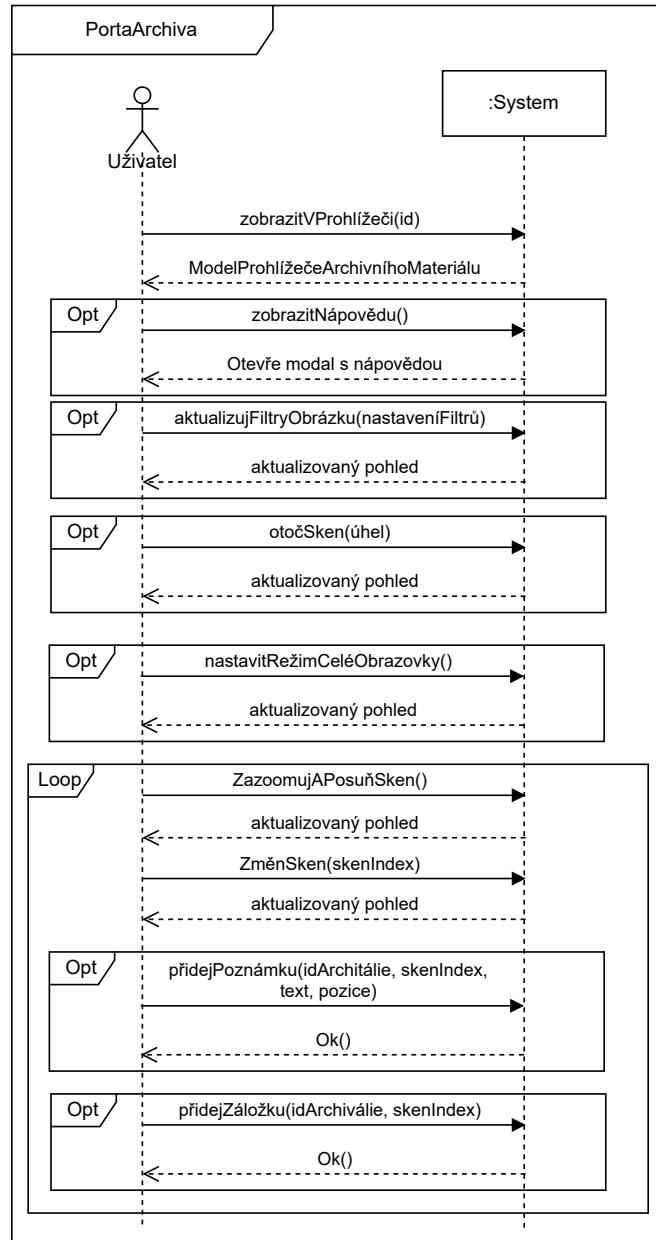
Systémový diagram sekvenční [45] slouží pro vizualizaci sledu interakcí uživatele se systémem. Jeho hlavním cílem je detailněji popsat a znázornit jednotlivé kroky, které uživatel vykonává při práci se systémem.

První diagram znázorňuje průchod systémem a vyhledání konkrétní archiválie v dané kategorii po specifikování filtrů. Při zobrazení detailu archiválie může přihlášený uživatel přidat danou archiválii do oblíbených. Dále si uživatel může zobrazit prohlížeč skenů nebo záznam na stránkách zdrojového archivního webu.



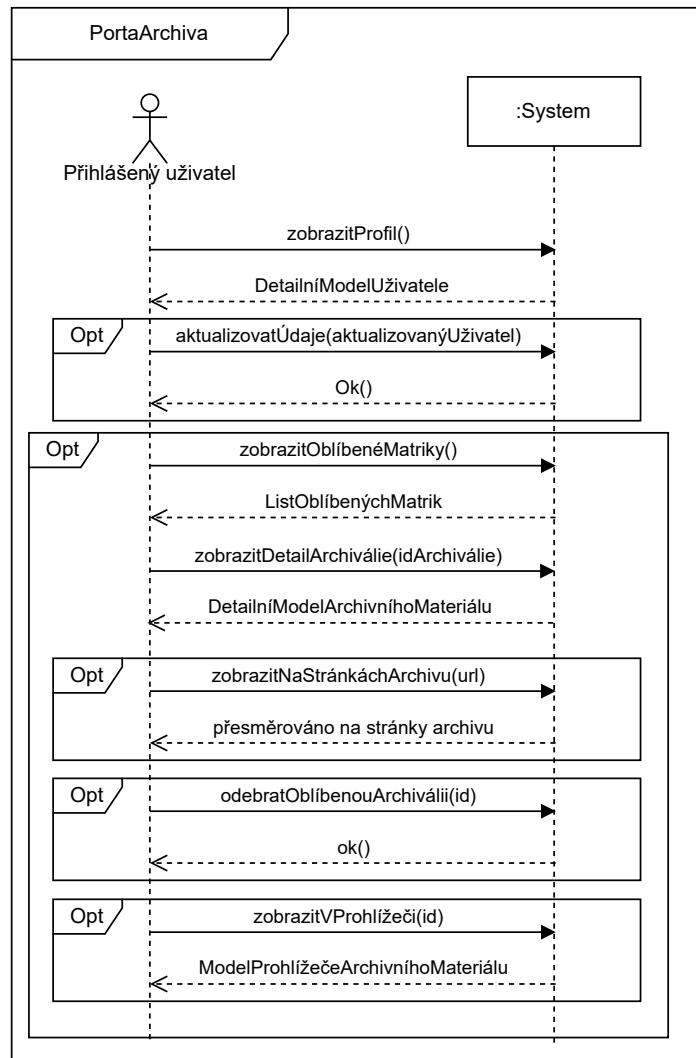
Obrázek 7.2: Systémové diagramy sekvence – vyhledání archiválie

Další diagram se zaměřuje na prohlížení archiválie, jelikož se jedná o primární funkcionalitu. Uživatel si může zobrazit nápovědu, upravit kontrast nebo jas pomocí filtrů a nastavit režim celé obrazovky. Následně už se opakovaně pohybuje po archiválii a přepíná mezi jednotlivými naskenovanými folii archiválie.



Obrázek 7.3: Systémové diagramy sekvence - prohlížení archiválie

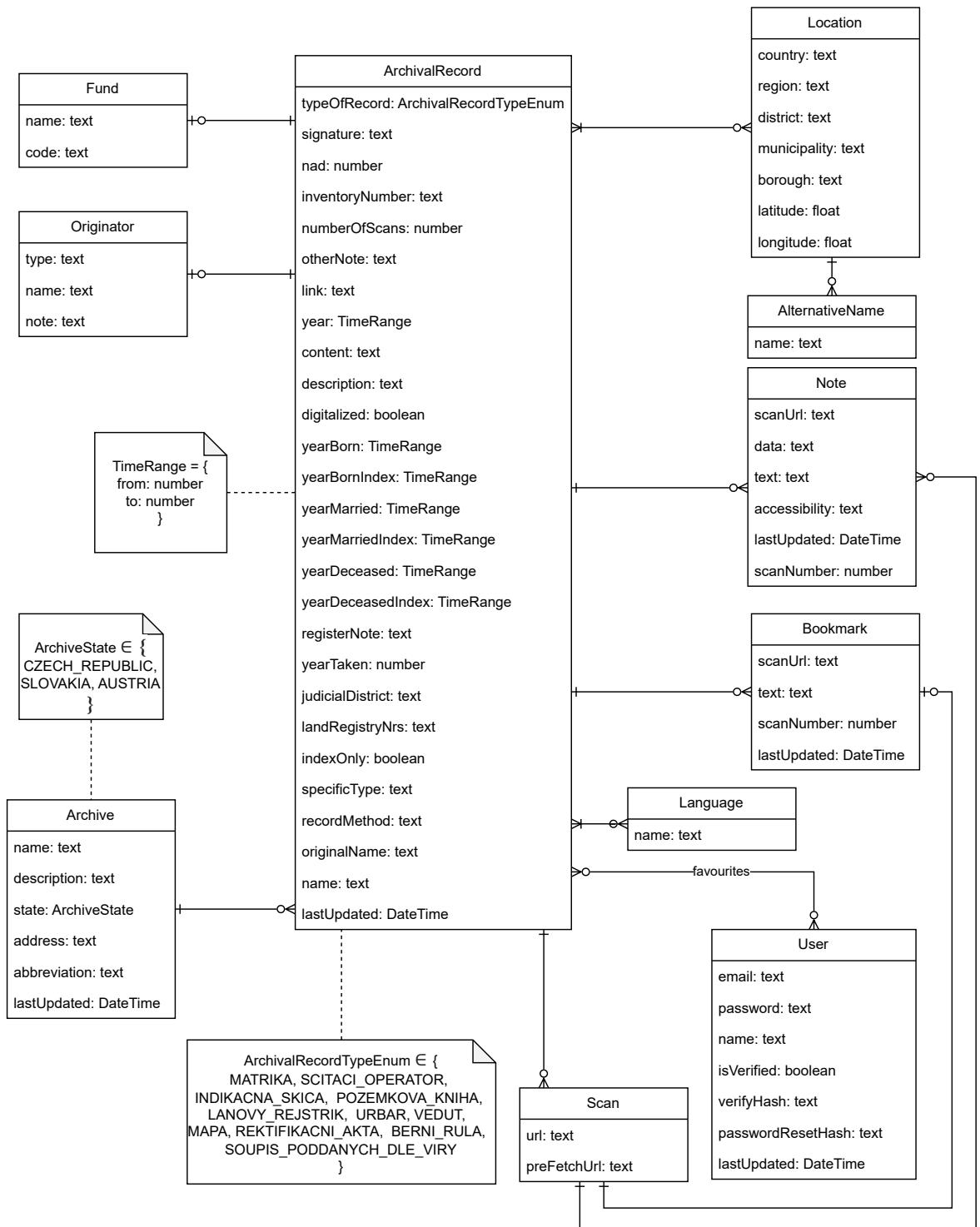
Poslední systémový diagram sekvence se zaměřuje na uživatelský profil, kde byla pro jednoduchost modelována práce pouze s oblíbenými matrikami. Uživatel může odebrat oblíbenou matriku a obdobně jako u detailního pohledu může přejít na stránky archivu, který danou archiválii zveřejnil, nebo do prohlížeče archiválií. V rámci profilu je dále možné zobrazit a editovat jednotlivé záložky nebo poznámky vytvořené daným uživatelem.



Obrázek 7.4: Systémové diagramy sekvence – uživatelský profil

### 7.3 Doménový model

Doménový model [46] se tvoří v rané fázi vývoje a slouží pro identifikaci entit, jež se v dané doméně vyskytují. Dále je potřeba určit jednotlivé vazby mezi entitami. Každá vazba má kardinalitu, kolikrát daná entita může vystupovat v konkrétním vztahu (například jeden bankovní účet má právě jednoho vlastníka, ale může mít více disponentů). V rámci doménového modelu se nutně nemusí vyskytovat datové typy. V případě, že jsou datové typy uváděny, tak by neměly být závislé na konkrétní technologii. V době tvorby tohoto modelu se pouze zkoumá, zda se jedná o textovou hodnotu, číslo, popřípadě o hodnotu z výčtu. Hlavní entitou je samotný archivní záznam. Každý archivní záznam může obsahovat informace o původci, fondu a skenech. Dále se ke každému záznamu vážou lokality, odkud daná archiválie pochází a jazyky, jakými je daná archiválie napsána. Uživatelé následně k jednotlivým skenům mohou přidávat poznámky a záložky.



Obrázek 7.5: Doménový model

## 7.4 Použitá architektura a technologie

U této aplikace se neočekává velká zátěž ze strany uživatelů, takže použití distribuované architektury pro aplikační server by zde bylo kontraproduktivní a pouze by vedlo ke zvýšení rezie a nákladů na provoz samotného systému. Jako vhodná se zde jeví monolitická architektura se třemi logickými vrstvami a separátní klientskou aplikací. Tato práce se primárně věnuje návrhu čisté klientské aplikaci. U aplikace se očekává vyšší míra čtení než zápisů. Je-likož je v plánu implementovat fulltextové vyhledávání, tak je zvolen návrhový vzor CQRS [3, 4], kde dochází k synchronizaci relační databáze s nerelační. Tato kombinace umožňuje využívat výhody obou typů databází. V rámci serverových technologií dojde k omezení na striktně typované, jelikož dynamické typování v rámci většího projektu je spíše kontraproduktivní a zatemňuje kód. Z analýzy aktuálně používaných systémů je patrné, že většina z nich běží na Javě. V tomto případě bude využit aplikační rámcem Spring boot [22], který je postaven na programovacím jazyku Java. Java splňuje všechny podmínky, mezi které patří silné typování, bohatý ekosystém, multiplatformita a opensource. Mezi technologiemi pro vývoj klientských aplikací nemá smysl uvažovat něco jiného než reaktivní JavaSriptový aplikační rámcem. Z jednotlivých aplikačních rámců byl zvolen React.JS [20] pro jeho vhodnou modulárnost vzhledem k velikosti projektu. JavaScriptový aplikační rámcem je rozšířen o Typescript [23], čímž je docíleno statického typování. Pro centralizované ukládání a přístup k datům je využit state management knihovna Redux [12]. Tato kombinace umožní vytvořit rozšířitelnou a snadno udržitelnou aplikaci, která bude ctít dobré programátorské principy jako SOLID [42] nebo pravidla čistého kódu.

Nyní budou představeny jednotlivé technologie a určité aspekty, na něž bude dále odkazováno v kapitole 9 Implementace.

### 7.4.1 Relační databáze

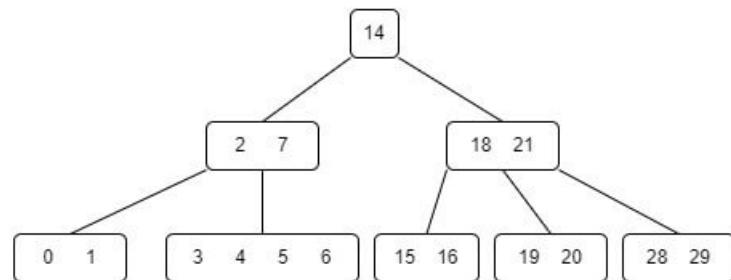
Relační databáze [25, 26] ukládá data do tabulek, mezi kterými mohou existovat vztahy pomocí primárních a cizích klíčů. Každá tabulka obsahuje množinu záznamů, jež mají stejnou strukturu. Každý jednotlivý záznam musí mít jednoznačný identifikátor. Identifikátory umožňují definovat vztahy mezi jednotlivými záznamy i napříč různými tabulkami. Relační databáze jsou velmi pesimistické a řádí se mezi silně konzistentní databáze. Jednotlivé operace jsou prováděny v rámci databázových transakcí, které se na konci potvrdí, případně zamítnou. Tímto se eliminuje případ, kdy by nastal výpadek uprostřed operace a například z jednoho bankovního účtu by odešly peníze, ale na druhý se již nepřipsaly. Relační databáze splňují ACID vlastnosti, mezi než patří atomicita, konzistence, izolace a trvanlivost.

### 7.4.2 Databázové indexy

Databázové indexy [17, 33, 41] jsou klíčovou součástí databází, bez nichž by nešlo provádět efektivní vyhledávání a dotazování nad velkými datovými sadami. Každý index má specifikovaný sloupec, podle kterého je seřazen, a následně se díky němu zvyšuje rychlosť vyhledání záznamů, jež splňují konkrétní podmínky. Implicitně databáze vytváří indexy pro primární klíče.

Existují dva hlavní typy databázových indexů, B-stromové indexy a hashovací indexy. B-strom je vyvážená datová struktura, kde uzly obsahují indexované klíče. Každý uzel má určitý počet klíčů. Celá struktura je seřazena, klíče s nižší hodnotou se nachází vlevo

a s vyšší vpravo. Použití seřazeného B-stromu posune vyhledávání z třídy lineární časové složitosti do třídy logaritmické, přičemž tato časová složitost je garantována díky vyvažování B-stromu.

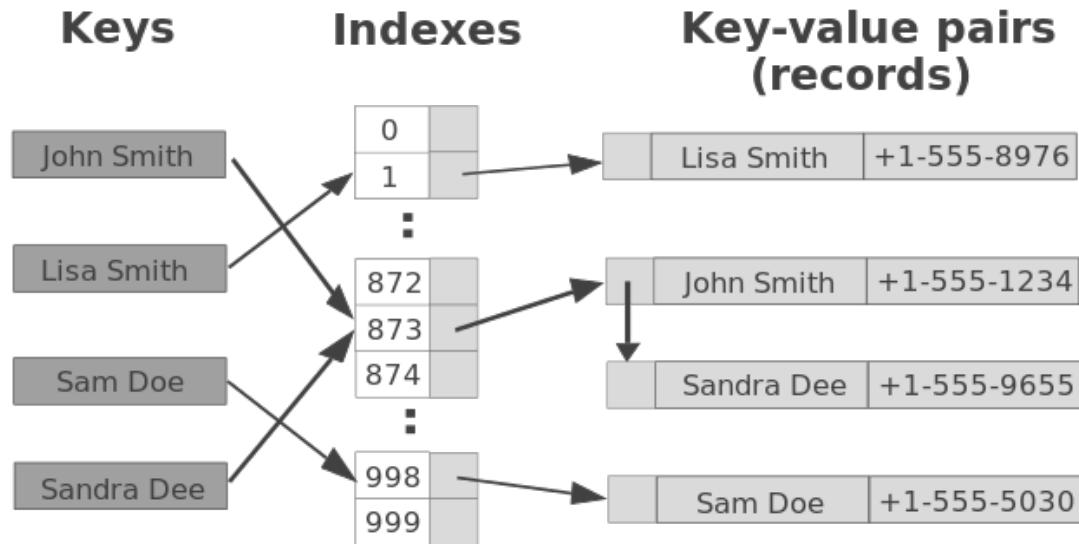


Obrázek 7.6: Vizualizace B-stromu<sup>1</sup>

---

<sup>1</sup>[https://www.tutorialspoint.com/data\\_structures\\_algorithms/images/b\\_trees.jpg](https://www.tutorialspoint.com/data_structures_algorithms/images/b_trees.jpg)

Alternativou k B-stromům jsou hashovací indexy [13], které využívají hashovací funkci. Funkce transformuje klíč na index v tabulce. Použitím tohoto přístupu se dosáhne konstantní časové složitosti, a to za předpokladu, že je zvolena vhodná hashovací funkce, jež klíče distribuuje rovnoměrně, čímž nedochází k častým kolizím klíčů.



Obrázek 7.7: Vizualizace hashovací tabulky<sup>2</sup>

Databázový index nemusí být pouze pro jeden sloupec, ale může se jednat o index složený, který lze využít v případě, že dotaz obsahuje filtrování na základě více klíčů současně.

Použití databázových indexů s sebou nese jistou režii, jelikož tyto indexační struktury zvyšují paměťové nároky a zpomalují operace modifikující data. Při každé operaci, která modifikuje obsah databáze, se musí upravit všechny indexy, jež jsou na daný záznam navázány. Databázové indexy by byly bezcenné v případě, že by nebyly synchronizovány s daty v databázi.

### Formální popis relačního datového modelu

Relační databáze vychází z relačního datového modelu, takže relační databázi lze formálně popsat. Relační datový model je založen na konceptu relační algebry a teorie množin. Každou databázi lze potom chápat jako množinu tabulek.

$$DB = \{T_1, T_2, \dots, T_n\}$$

Každá tabulka je definována pomocí matematické relace, která je podmnožinou kartézského součinu jednotlivých domén hodnot. Doména je množina možných hodnot pro daný atribut relace. Databázová tabulka je následně vizuální reprezentací dané relace.

$$R \subseteq D_1 \times D_2 \times \dots \times D_m$$

Následně je možné použít relační algebru, která nám poskytuje formální operace pro manipulaci s relacemi. Mezi základní operace patří selekce, projekce, sjednocení, rozdíl a kartézský součin.

<sup>2</sup><https://yourbasic.org/algorithms/hash-table.png>

### 7.4.3 Plné textové vyhledávání

Plné textové vyhledávání [28, 34] je proces hledání a získávání relevantních informací z textových dat. Oproti klasickému hledání podřetězce může zvládat hledání synonym, typografickou chybu v hledaném výrazu nebo hledání podle kořene slova.

Kritickou součástí, aby vyhledávání fungovalo podle očekávání, je správné nastavení analyzátorů. Analyzátory v rámci zpracování textu rozdělují textová data na podřetězce, jež jsou nazývány tokeny. Analyzátory mohou rozdělovat tokeny například pomocí bílých znaků a interpunkčních znamének. Dalším zajímavým příkladem tokenizeru je Ngram, který rozdělí text na podřetězce do určité délky, což umožňuje vytvořit index pro prefixové a infixové vyhledávání. Tokeny se následně normalizují, filtrují a případně se aplikují další transformace. Filtrování probíhá zpravidla na stop slovech, což jsou slova v daném jazyce mající vysokou frekvenci výskytu v textu a nejsou proto relevantní pro vyhledávání. Typickým příkladem stop slov jsou například předložky. Mezi typické příklady normalizérů patří převod na malá písmena, ASCII folding, stemmer a normalizér synonym. ASCII folding zahrnuje odstranění speciálních znaků, stemmer se stará o převod slova do základního tvaru a normalizér synonym převádí synonyma na jednotný tvar. Pro správné fungování musí být nastaven cílový jazyk.

### 7.4.4 Nerelační databáze Elasticsearch

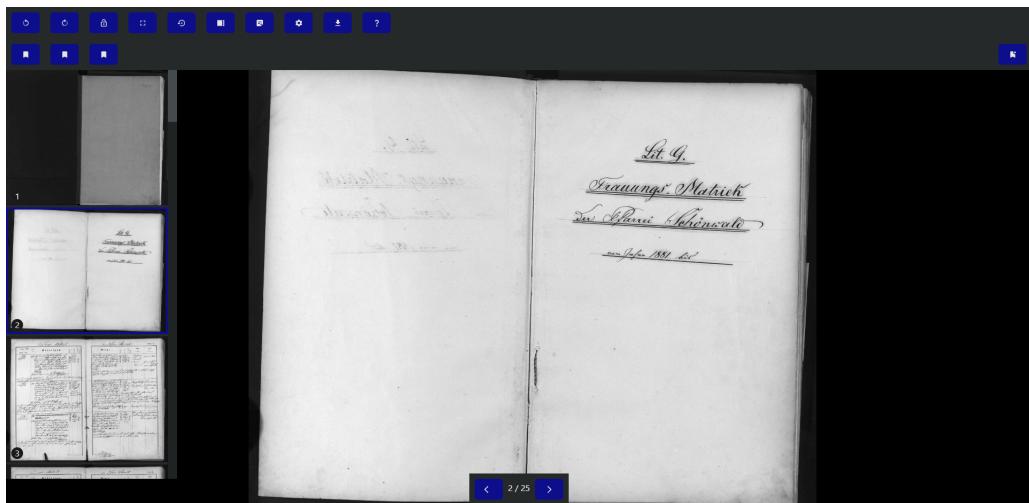
Elasticsearch [27] je distribuovaná, open-source, plné textová vyhledávací a analytická databáze. Databáze vychází z Apache Lucene a je navržena pro zpracovávání velkých dat. Data jsou namísto v tabulkách ukládána v kolekcích JSON dokumentů. Primárním důvodem, proč je tato databáze použita v tomto systému, je podpora plné textového vyhledávání. Databáze pro komunikaci se serverovou částí aplikace nabízí REST rozhraní.

V rámci Elasticsearch se rozlišují tři základní typy položek. Jedná se o Klíčové pole, Plné textové pole a Obecné pole. Klíčové pole slouží pro ukládání hodnot nevyžadujících analýzu, nebo v případě, kdy je požadováno porovnání na shodu. Klíčové pole je využito u parametrů, podle kterých jsou záznamy filtrovány. Plné textové pole je využito na položky jako popis archiválie, kde se očekává vyhledávání v textu. Obecné pole je kombinací dvou výše zmíněných a umožňuje filtrování na přesnou shodu i vyhledávání v textu.

Entity je potřeba obohatit o dodatečné anotace pro použití s Elasticsearch. Díky anotaci `@Indexed` je povoleno indexování dané entity do Elasticsearch databáze. Následně se musí specifikovat pro jednotlivé atributy, zda se jedná o klíčové slovo nebo plné textové vyhledávání. Pro složitější datové typy se musí vytvořit konvertory zabezpečující mapování mezi objekty v Javě a datovými typy v Elasticsearch. Pro každý atribut, který je anotován plným textovým vyhledáváním, se nastaví analyzátor a pro klíčové slovo normalizer. Je definován normalizer Czech, v rámci něhož je aplikován převod na malá písmena pomocí filtru lowercase a odstranění speciálních symbolů pomocí asciifoldingu. Stejnojmenný analyzátor využívá standardní tokenizér a následně aplikuje filtry: lowercase, czech\_stemmer, asciifolding, czech\_grammar\_normalizer. Token se tedy převede na malá písmena, slovo se analyzuje a převede se do kořenového tvaru. Následně se odstraní speciální znaky a aplikuje se normalizace asimilovaných hlásek. Hlásková asimilace [15] je jev ve fonetice spočívající ve výslovnostním přizpůsobováním hlásek. V českém jazyce se většinou jedná o asimilaci kontaktní, kdy se přizpůsobují hlásky sousedící.

## 7.5 Tvorba interaktivního prototypu

Před samotným návrhem aplikace a tvorbou grafického prototypu byl označen za nejkritičtější část systému samotný prohlížeč archiválů, tudíž byly zkoušeny možnosti zobrazení dlaždicových snímků, hledány vhodné knihovny, případně navržena vlastní řešení některých funkcionalit, aby bylo ověřeno, že systém je podle aktuálních požadavků možné realizovat. Prototyp byl realizován v technologiích, které byly zvoleny pro finální implementaci.



Obrázek 7.8: Screenshot z prototypu

Experimentálně bylo vyzkoušeno více různých řešení a vybráno bylo takové, které nejvíce vyhovovalo požadavkům. Jednalo se o knihovnu OpenSeaDragon podporující jak zobrazení standardních snímků, tak i snímků využívajících dlaždicové formáty. Knihovna dále obsahuje bohatou sadu funkcí, jež bude stačit rozšířit.

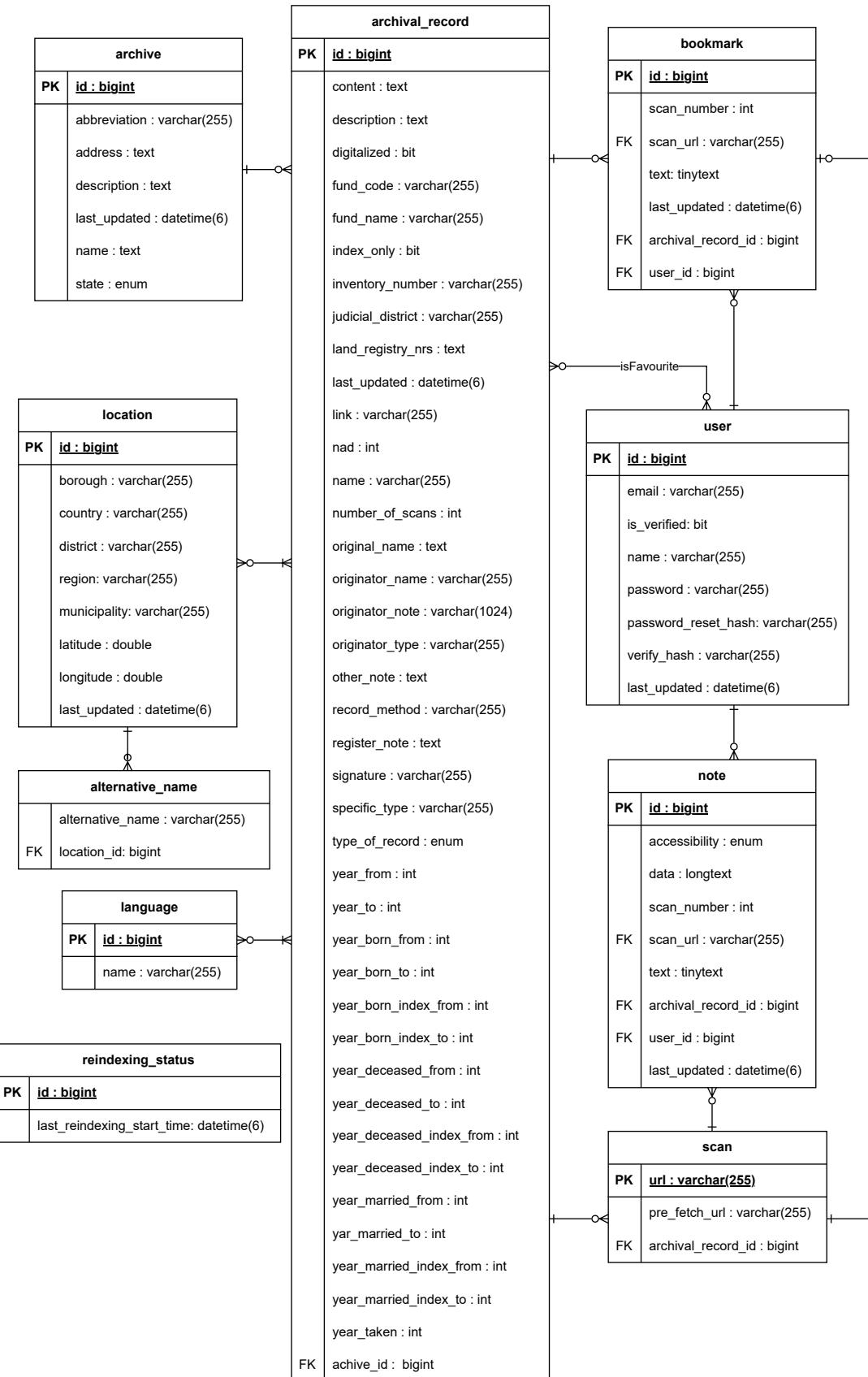
# Kapitola 8

## Návrh

Před samotnou implementací je třeba vyhotovit návrh systému. Behaviorální modely systému byly představeny již v kapitole 7 Analýza požadavků. Nyní je třeba systém navrhnut ze strukturálního pohledu. Součástí návrhu je dekompozice problému na jednotlivé služby, návrh architektur jednotlivých služeb a identifikace klíčových komponent. Návrh vychází z podkladů, které byly vytvořeny v kapitole 7 Analýza požadavků. Tvorba návrhu systému může být na první pohled chápána jako zbytečný dodatečný náklad na tvorbu systému, je-likož se tato rozhodnutí dají dělat při implementaci a není třeba ztrácat čas tvorbou UML diagramů. Při samotné implementaci je vývojář již ponoren do konkrétní funkce v dané technologii a chybí mu nadhled, jenž poskytuje model systému. Takto může programátor jednoduše něco přehlédnout, případně dojít do fáze, že bude třeba udělat výrazný zásah do architektury, který může značně převyšovat čas potřebný k tvorbě kvalitního návrhu. Návrh odstíňuje vývojáře od implementačních detailů, takže vzniká více prostoru soustředit se na pochopení uživatelských požadavků, optimalizaci výkonu a tvorbu robustní architektury. Změna v návrhu znamená změny v grafických modelech, což je daleko méně náročné než rozsáhlé změny na úrovni zdrojového kódu.

### 8.1 Entity-relationship diagram

ERD [32] je strukturální UML diagram, který se využívá při návrhu struktury uložení dat v rámci relační databáze. Reprezentuje jednotlivé tabulky, vztahy, primární a cizí klíče v relační databázi. Při tvorbě se vychází z doménového modelu, ovšem již jsou využity konkrétní datové typy pro konkrétní relační databázi. V rámci diagramu se již uvažují veškerá data, jež budou fyzicky uložena v databázi. Oproti doménovému modelu může obsahovat nové atributy a tabulky, které nutně nevychází ze specifikace, ale systém by bez nich nefungoval. Entity **Fund** a **Originator** jsou modelovány jako embedded v rámci archivního záznamu namísto separátní tabulky. Nově zde přibyla tabulka reindexace vážící se k informaci o posledním spuštění manuální reindexace dat v Elasticsearch. Vztahy s kardinalitou  $m \dots n$  v databázi obsahují pomocnou vazební tabulkou, která v tomto diagramu není explicitně modelována.



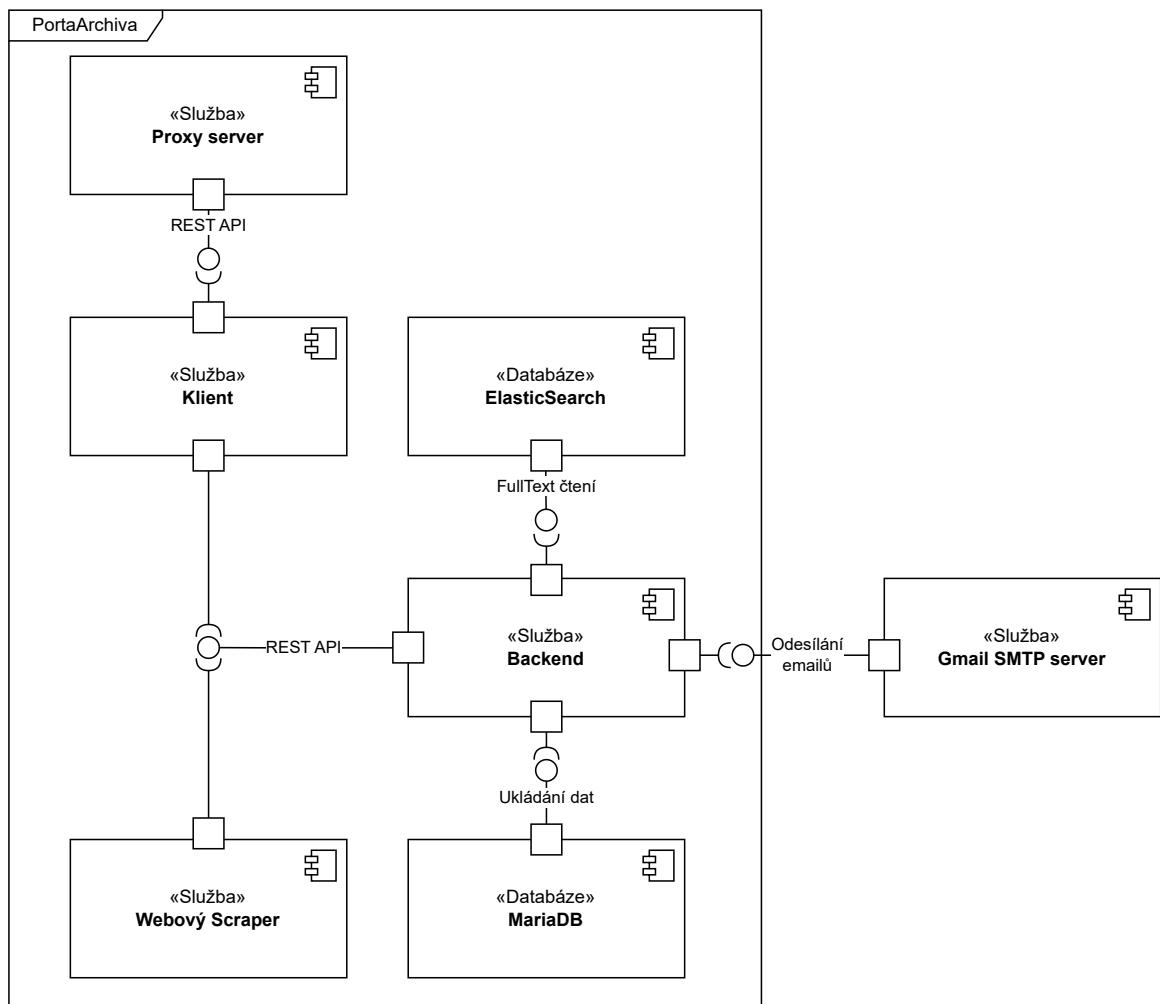
Obrázek 8.1: Entity-relationship diagram

## 8.2 Diagramy struktury

Strukturální UML diagramy pomáhají popsat architekturu aplikace na úrovni jednotlivých subsystémů, komponent a balíků. Oproti předchozím diagramům se již nezaměřují na pošpaní požadavků na nově vznikající systém, ale návrhář je využije při tvorbě návrhu architektury aplikace. Jedná se o velmi vhodné dokumentační diagramy, které usnadní pochopení vazeb mezi jednotlivými částmi systému bez nutnosti nahlízení do zdrojového kódu.

### 8.2.1 Vysokoúrovňový diagram komponent

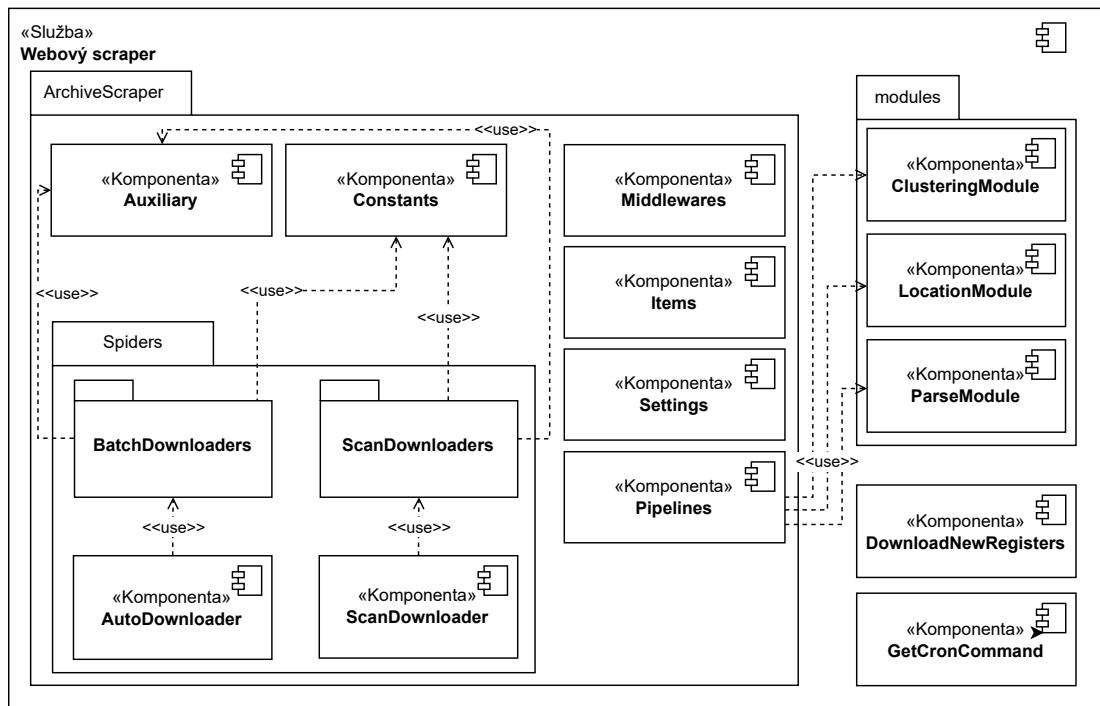
Úlohou diagramu komponent [44] na nejvyšší úrovni je rozdelení řešeného problému do jednotlivých služeb, které spolu budou kooperovat. V rámci této práce bude systém rozdelen do čtyř služeb, dvou databází a jedné externí služby pro odesílání e-mailových zpráv. Klasická architektura klient-server je rozšířena o další služby, jež se starají o stahování dat ze všech archivů a přístup k mediálním datům jednotlivých archiválií.



Obrázek 8.2: Vysokoúrovňový diagram komponent

### 8.2.2 Diagram komponent Webového scraperu

Úlohou Webového scraperu je stahování dat z webových stránek jednotlivých archivů. Systém používá aplikační rámec Scrapy a je napsán v Pythonu. Autorem systému je Jan Valušek [51], ovšem pro potřeby tohoto projektu bylo nutné stávající řešení adaptovat. Tento systém je vybrán z jednotlivých systémů na základě provedených kontrol konzistence datových sad a díky podpoře pro stahování mediálních dat jednotlivých archiválií. Jádrem aplikace jsou scripty pro jednotlivé archivy ve složce **Spiders**, které používají pomocné moduly jako **Auxiliary** a **Constants**. Moduly **Middlewares**, **Items**, **Settings** a **Pipelines** se věnují nastavení prostředí pro aplikační rámec Scrapy, zpracování záznamu a odeslání přes API na serverovou část aplikace. **LocationModule** se používá k vyhledávání jednotlivých lokalit pomocí API [OpenStreetMap](#). **LocationModule** pro každou hledanou lokalitu vrátí více možných výsledků a je tedy úlohou **ClusteringModule** vybrat tu nejpravděpodobnější správnou lokalitu. Princip algoritmu bude blíže popsán v kapitole 9 Implementace.

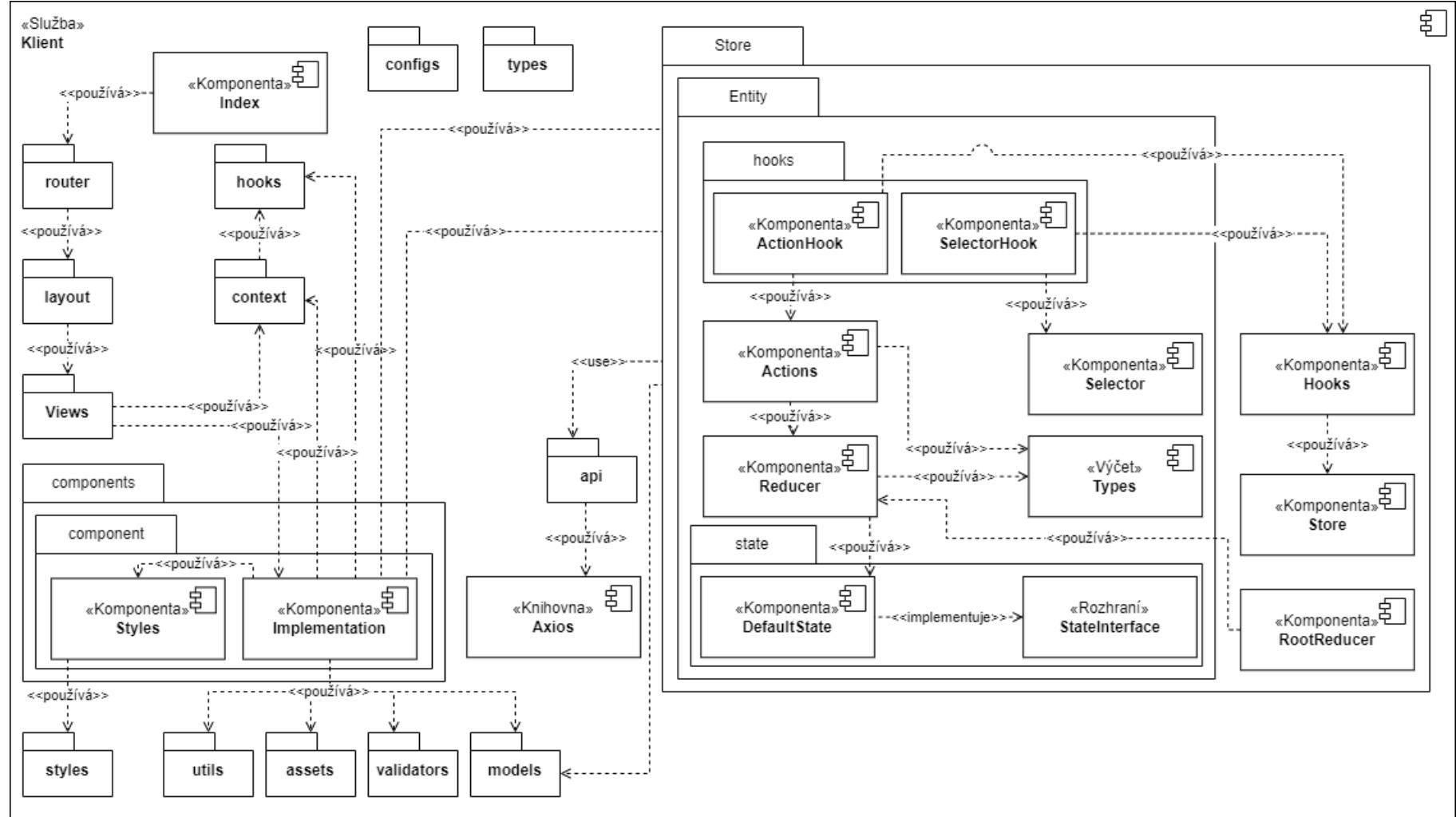


Obrázek 8.3: Diagram komponent webového scraperu

### 8.2.3 Diagram komponent klientské aplikace

Jelikož se jedná se o webovou aplikaci, tak i zde existuje startovní bod **Index**, v rámci kterého se volá **router**. **Router** se stará o mapování URL adresy na příslušné **view**. **Router** dále řeší zvolený **layout** aplikace, jenž se může lišit například na základě toho, zda je uživatel přihlášen. **View** reprezentuje jednu stránku naší aplikace jako například úvodní stránku, seznam archiválií, detail archiválie a další. Každý tento **view** se skládá z jednotlivých **komponent**.

**Komponenta** je znovupoužitelný kus kódu. Rozdelení systému do jednotlivých komponent umožňuje udržovat velikost jednotlivých zdrojových souborů v rozumném rozsahu, což zjednodušuje případné rozšíření a údržbu aplikace. Každá komponenta může mít svůj interní stav a pomocné funkce, které například reagují na specifickou událost. Hlavním úkolem komponenty je vykreslit část výsledného `view` a přidáním pomocných funkcí nebo stavových proměnných by byl porušen princip jedné zodpovědnosti. K tomu, aby došlo k oddělení komponenty od logiky, je nutné pomocné funkce (stavové proměnné a event handlery) vytknout do separátního `hooku`, jenž je následně využit v komponentě. Problém nastává při použití `hooku` ve více komponentách, kdy je nutné mezi nimi sdílet stav `hooku`. Sdílení stavu mezi komponentami lze dosáhnout pomocí kontextu umožňujícího sdílet stavové proměnné jednoho `hooku`. Ke každé komponentě se vztahuje její `Styles` modul, který obsahuje definici stylů pro konkrétní komponentu. Použití modulů způsobí, že za název použité `scss` třídy se vygeneruje hash, takže v případě, kdy je použit stejný název třídy ve dvou komponentech, nedojde ke kolizi těchto stylů. Komponenta kromě stavových proměnných potřebuje přístup ke statickým souborům, jež jsou uloženy v modulu `assets`. Další pomocné funkce jsou zabalené v modulu `utils` a v případě formulářů v modulu `validators`. Doposud byla rozebrána pouze problematika tvorby komponent, validace a zobrazení `view`. To, co doposud chybělo, byla komunikace s API. Pro uchovávání dat na klientovi je využita knihovna `Redux` sloužící jako centralizované úložiště, a její detailní popis je uveden v kapitole 9 Implementace. Pokud posílá komponenta požadavek na API, tak používá `ActionHook` a pro přístup ke staženým datům používá `SelectorHook`, kde jsou data neměnná. Na první pohled se může jednat o zbytečnou režii a nabízí se otázka, proč nelze rovnou upravovat a čist data z jedné proměnné jako například v aplikačním rámci `Vue.js`. V rámci malých projektů se vskutku jedná o zbytečnou režii, ale na větších projektech to pomáhá držet čistou architekturu a vzniká méně chyb v kódu, jelikož se striktně rozlišuje, kdy může být hodnota čtena a kdy modifikována.



Obrázek 8.4: Diagram komponent klienta

#### 8.2.4 Diagram komponent serverové části aplikace

Serverová část je standardní monolitická aplikace, kde je vhodné použít třívrstvou architekturu. Spodní vrstva je technického charakteru a převážně zodpovídá za připojení k datovým úložištím. Nad datovou vrstvou je aplikační logika, která je vždy závislá na konkrétní aplikaci, jelikož implementuje jednotlivé případy užití. Třetí vrstva zajišťuje, aby operace, které jsou definované v aplikační logice, byly poskytnuty ostatním službám přes aplikační rozhraní. V tomto případě bylo zvoleno konzervativní rozhraní REST.

Modul `core` obsahuje podpůrné nastavení. Modul `commandLineRunners` obsahuje funkce, jež se spouští při startu aplikace. Je zde například umístěna inicializace databáze. Dále je zde definice typů pro databázi Elasticsearch, nastavení pro JSON Web Token a definice jednotlivých výjimek, které aplikace vyvolává v případě chyby.

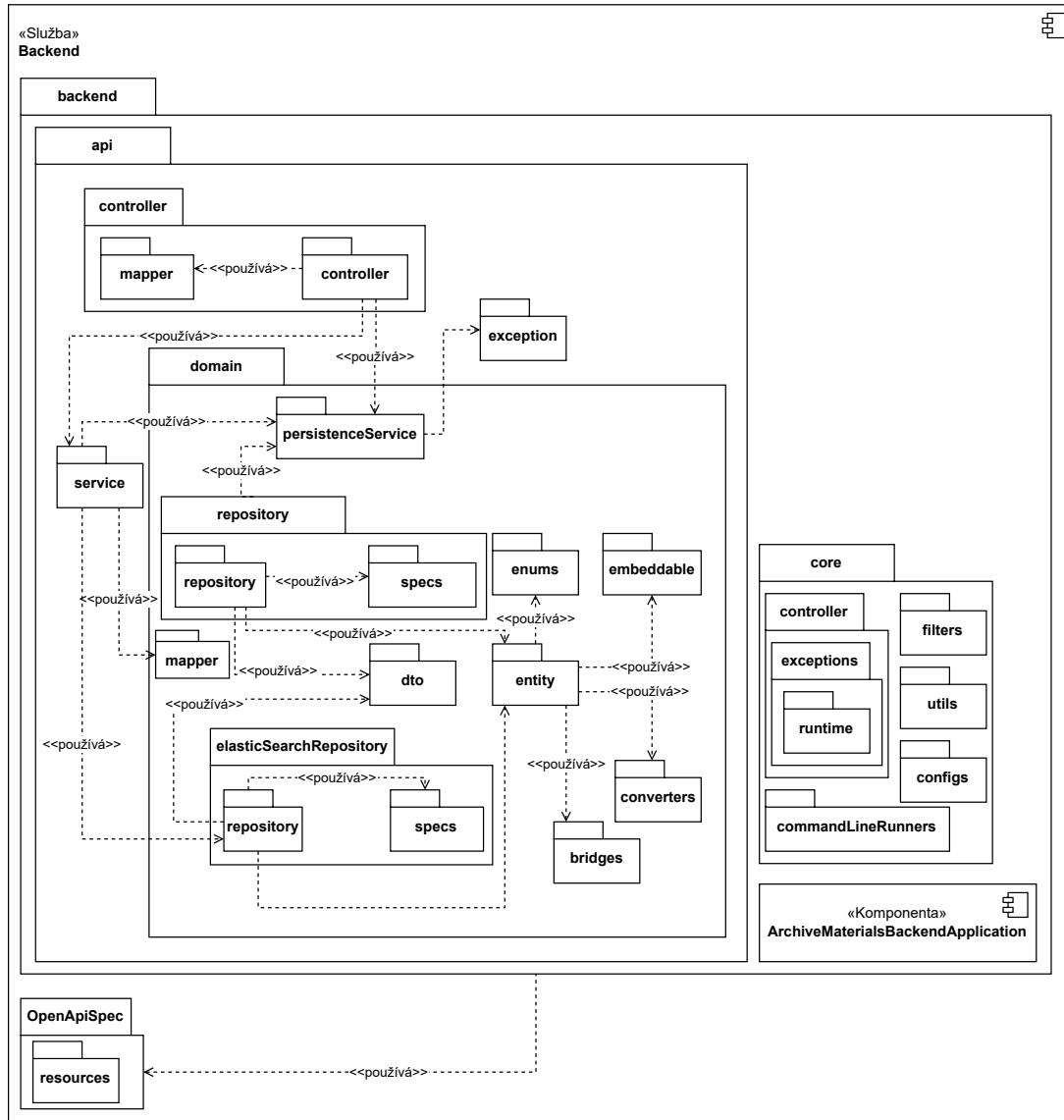
Nejnižší vrstva zodpovídá za přístup k databázím, definici entit a jedná se o základní stavební kámen serverové aplikace. V tomto případě se jedná o balík `domain`. V rámci balíku `domain` jsou definovány entity, jež vzhledem k návrhu ER diagramu vyžadují pomocné `embeddable` a `enums`. Vzhledem k tomu, že je použita jedna entita pro relační i nerelační databázi, tak pro použití Elasticsearch je potřeba definovat vlastní převodníky pro převod konkrétního datového typu do databáze a zpět. Tyto převodníky jsou definovány v balíku `converters` a `bridges`. Použití Java `persistence` API umožňuje definici repozitářů pouze pomocí rozhraní bez nutnosti implementovat veškeré základní operace jako například získání záznamu podle identifikátoru. Ne vždy nám bohužel stačí tato základní sada, která lze pomocí Java `persistence` API vygenerovat, a proto jsou tyto základní repozitáře zapouzdřeny do `persistenceService`. V rámci `persistenceService` jsou implementovány všechny chybějící dotazy na databázi. Pomocné znovupoužitelné predikáty jsou definovány v rámci modulu `specs`.

Separátní `persistenceService` a `specs` umožňují držet čitelnou aplikační logiku implementovanou v balíku `service`. Struktura entity musí odpovídat struktuře tabulků v databázi. Pokud operace nemá vracet všechna data z databáze, tak se využije `mapper`, jenž dokáže mapovat objekty různých datových typů. Vhodným mapováním se šetří čas, který by byl nutný pro přenos nepotřebných atributů skrze protokol HTTP na klienta. Pro mapování existuje více alternativních knihoven, jež se dělí podle toho, jak provádí mapování. Některé knihovny využívají reflexi, což je mocný nástroj interpretovaných jazyků, který je ovšem velmi nákladný na výpočetní zdroje. Reflexe umožňuje za běhu analyzovat a měnit strukturu chování, takže lze získat informace o třídách, metodách a polích, a také je volat a manipulovat s nimi, aniž by bylo nutné znát tyto detaily předem při komplikaci kódu. Alternativní přístup k mapování je analýza v době překladu, kde jsou jednotlivé třídy zodpovědné za mapování vygenerovány staticky a není tak nutné provádět analýzu za běhu.

V rámci nejvyšší vrstvy je řešeno poskytování aplikační logiky a interakce s okolím. Jelikož bylo zvoleno rozhraní REST, tak `controllery` obsahují definici jednotlivých koncových bodů, přes které může uživatel volat jednotlivé operace byznys logiky.

Pro definici aplikačního rozhraní dané aplikace byl zvolen dokumentační standard OpenAPI, jenž striktně definuje, jak popsat rozhraní aplikace, a to včetně typů parametrů, návratových hodnot a popisu jednotlivých koncových bodů. Popis je uložen v samostatném

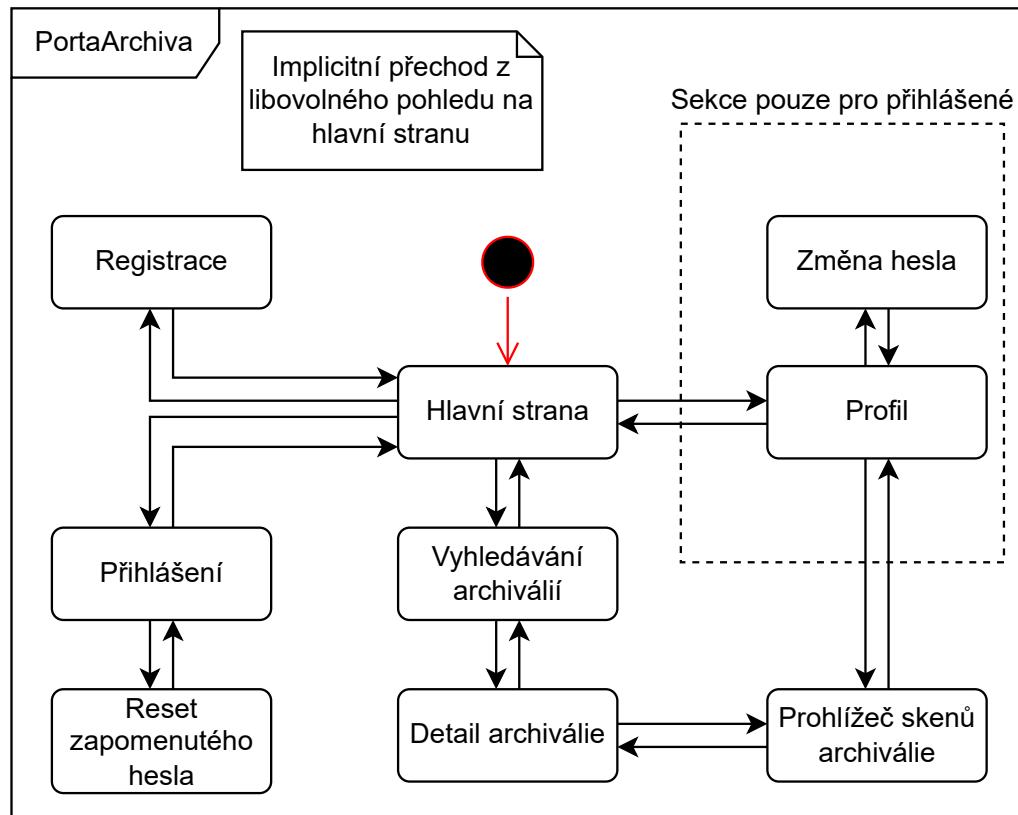
projektu OpenApiSpec a pro každý controller existuje separátní YAML soubor. Tuto specifikaci je možno importovat do nástroje Postman nebo ji interaktivně procházet pomocí nástroje Swagger.



Obrázek 8.5: Diagram komponent serverové části aplikace

### 8.3 Tvorba konečného automatu přechodů

Diagram reprezentuje jednotlivé pohledy v rámci systému jako stavy. Hrany značí možný přechod mezi pohledy. Z každého pohledu se lze implicitně přesunout na hlavní stránku, a to pomocí navigačního menu.

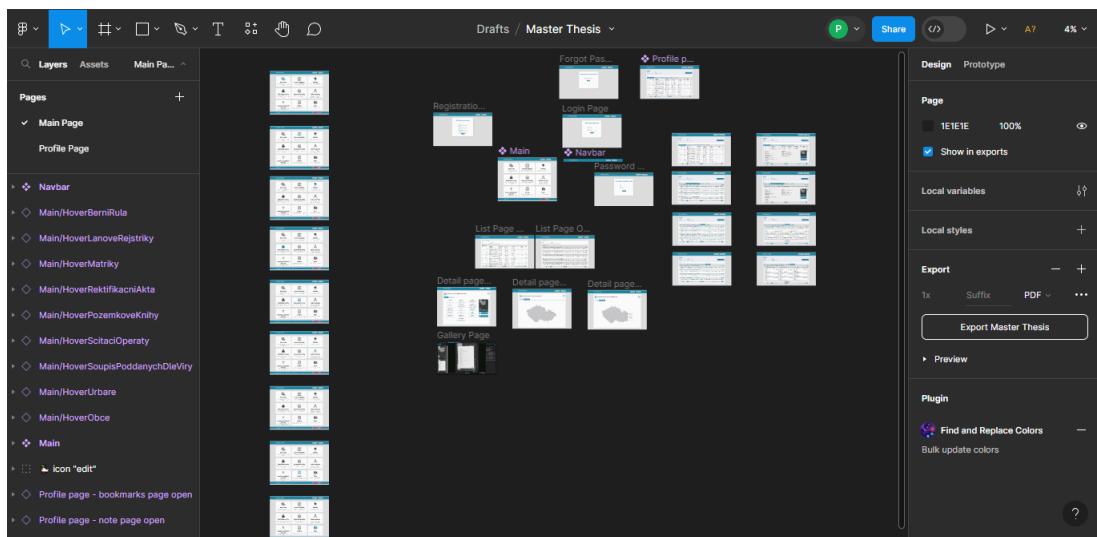


Obrázek 8.6: Konečný automat přechodů

## 8.4 Tvorba grafického návrhu

Pro tvorbu grafického návrhu byl zvolen nástroj [Figma](#) [36]. Jedná se o online kolaborativní nástroj pro tvorbu interaktivních prototypů. Nejedná se tedy o pouhé obrázky, ale uživatel může s prototypem interagovat a navigovat se napříč pohledy. Díky možnosti tvorby vlastních komponent urychluje jak samotnou tvorbu prototypu, tak i následně úpravu podle představ zadavatele. Do Figmy lze doinstalovat mnoho přídavných balíků, a to například pro kontrolu kontrastu jednotlivých komponent nebo sadu ikon.

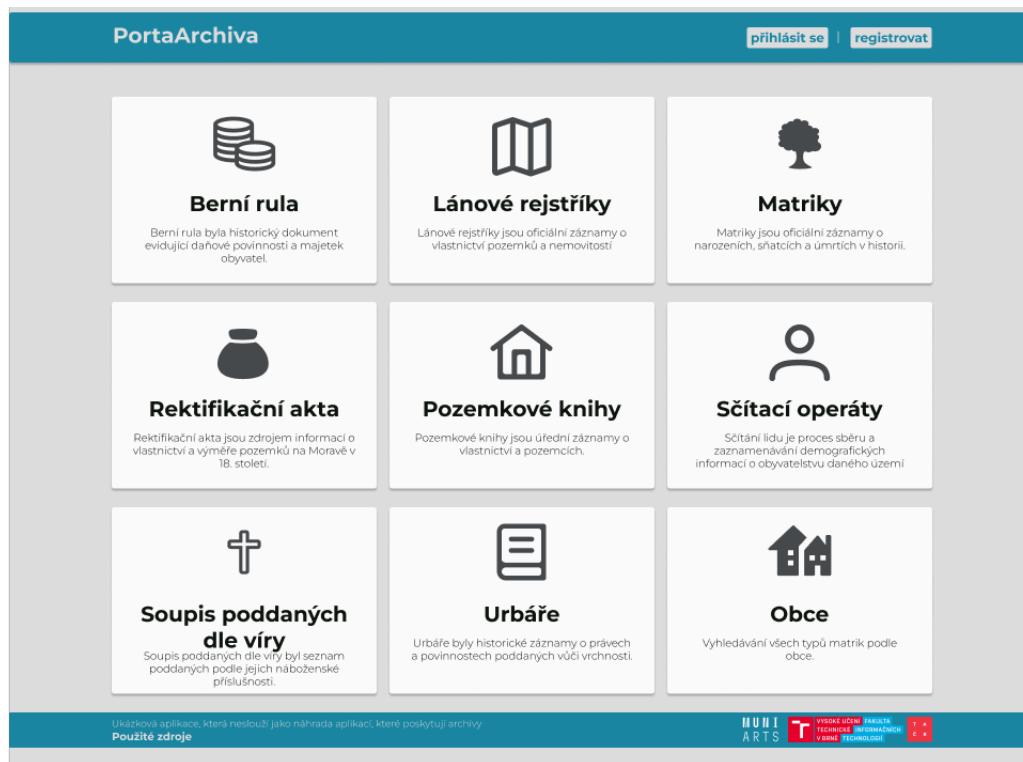
Interaktivní prototyp<sup>1</sup> se skládá z jednotlivých pohledů a různých překryvných prvků aplikujících se při spuštění určité události nad konkrétním prvkem. Dále jsou zde definovány přechody mezi pohledy, které byly navrženy při tvorbě konečného automatu přechodů v kapitole 8.3 Tvorba konečného automatu přechodů. Výsledný návrh vznikal iterativně a vždy byl konzultován s vedoucím práce, jenž sdělil konstruktivní kritiku a náměty na změny. V následujícím textu jsou uvedeny ukázky pohledů z návrhu, a to včetně popisu rozložení.



Obrázek 8.7: Návrhový režim ve Figmě

Při návrhu titulní strany byl kladen důraz na tvorbu rychlého rozcestníku pro uživatele, kde by si uživatel zvolil, v rámci jakých pramenů chce hledat, případně zda chce vyhledávat podle lokality. Každá kategorie je doplněna krátkým popisem sloužícím k rychlejší orientaci začínajícímu badateli.

<sup>1</sup><https://www.figma.com/proto/90Nm3ZEx48DYMICGL7u0MP/Master-Thesis?node-id=281-4650&t=cITZe8WZ8ckybige-1>



Obrázek 8.8: Návrh titulní strany ve Figmě

Seznam archiválií obsahuje plné textové vyhledávání i specifické vyhledání některých atributů v horní části pohledu. Seznam záznamů má formu tabulky, kde ovládání je umístěno v jejím záhlaví. Po kliknutí na záznam je uživatel přesměrován na detail daného záznamu. Na základě konzultace s vedoucím práce byl použit podobný formát z již existujících řešení, a to z toho důvodu, aby se usnadnil přechod badatelům z dosavadních systémů.

Seznam matrik obsahuje základní informace, kde odznak u obcí značí počet obcí v dané archiválii a po najetí kurzoru na odznak se zobrazí podrobný seznam obcí.

PortaArchiva

[přihlásit se](#)

[registrovat](#)

<input type="text"/> Zde vložte hledaný výraz															
Kraj		Okres													
<input type="text"/>															
<b>Matriky</b>															
Inv.č.	Sig.	Původce Typ původce	Narození od-do index narození	Oddání od-do index oddání	Zemřelé od-do index zemřelí	Obce	Počet snímků	Archiv							
55	A 34-5	Olešnice na Moravě (helvetské vyznání) českobratrská církve evangelické	1850 - 1850 1850 - 1850	1850 - 1850 1850 - 1850	1850 - 1850 1850 - 1850	(1)	14	MZA							
55	A 34-5	Olešnice na Moravě (helvetské vyznání) českobratrská církve evangelické	1850 - 1850 1850 - 1850	1850 - 1850 1850 - 1850	1850 - 1850 1850 - 1850	(1)	14	MZA							
55	A 34-5	Olešnice na Moravě (helvetské vyznání) českobratrská církve evangelické	1850 - 1850 1850 - 1850	1850 - 1850 1850 - 1850	1850 - 1850 1850 - 1850	(1)	14	MZA							
55	A 34-5	Olešnice na Moravě (helvetské vyznání) českobratrská církve evangelické	1850 - 1850 1850 - 1850	1850 - 1850 1850 - 1850	1850 - 1850 1850 - 1850	(1)	14	MZA							
55	A 34-5	Olešnice na Moravě (helvetské vyznání) českobratrská církve evangelické	1850 - 1850 1850 - 1850	1850 - 1850 1850 - 1850	1850 - 1850 1850 - 1850	(1)	14	MZA							
55	A 34-5	Olešnice na Moravě (helvetské vyznání) českobratrská církve evangelické	1850 - 1850 1850 - 1850	1850 - 1850 1850 - 1850	1850 - 1850 1850 - 1850	(1)	14	MZA							
55	A 34-5	Olešnice na Moravě (helvetské vyznání) českobratrská církve evangelické	1850 - 1850 1850 - 1850	1850 - 1850 1850 - 1850	1850 - 1850 1850 - 1850	(1)	14	MZA							
55	A 34-5	Olešnice na Moravě (helvetské vyznání) českobratrská církve evangelické	1850 - 1850 1850 - 1850	1850 - 1850 1850 - 1850	1850 - 1850 1850 - 1850	(1)	14	MZA							
55	A 34-5	Olešnice na Moravě (helvetské vyznání) českobratrská církve evangelické	1850 - 1850 1850 - 1850	1850 - 1850 1850 - 1850	1850 - 1850 1850 - 1850	(1)	14	MZA							
55	A 34-5	Olešnice na Moravě (helvetské vyznání) českobratrská církve evangelické	1850 - 1850 1850 - 1850	1850 - 1850 1850 - 1850	1850 - 1850 1850 - 1850	(1)	14	MZA							
55	A 34-5	Olešnice na Moravě (helvetské vyznání) českobratrská církve evangelické	1850 - 1850 1850 - 1850	1850 - 1850 1850 - 1850	1850 - 1850 1850 - 1850	(1)	14	MZA							
55	A 34-5	Olešnice na Moravě (helvetské vyznání) českobratrská církve evangelické	1850 - 1850 1850 - 1850	1850 - 1850 1850 - 1850	1850 - 1850 1850 - 1850	(1)	14	MZA							

Obrázek 8.9: Seznam matrik

Pro ostatní typy archiválií byl zvolen odlišný formát vizualizace záznamu tabulky, jelikož matriky obsahují specifické informace, které ostatní typy archiválií neobsahují.

PortaArchiva

[přihlásit se](#) | [registrovat](#)

<input type="text"/> Zde vložte hledaný výraz	
Kraj	
<input type="text"/>	
Okres	
<input type="text"/>	
 <b>Berní rula</b>	
<b>ABC investiční fond pro Slezsko, a. s. v likvidaci, Opava - NAD 2688</b>	
Fond sestává z podnikových směrnic, zápisů ze zasedání představenstva a výlých hromad společnosti, stanov a likvidační agenda, např. rozhodnutí o jmenování likvidátora, zpráv auditora, roční uzávěrky, zahajovací a roční účetní rozvahy, z výročních zpráv, zakladatelských smluv, listů členů investičního fondu, seznamu akcionářů, soupisu pohledávek a závazků aj.	<b>1993 - 1999</b>
<b>ABC investiční fond pro Slezsko, a. s. v likvidaci, Opava - NAD 2688</b>	
Fond sestává z podnikových směrnic, zápisů ze zasedání představenstva a výlých hromad společnosti, stanov a likvidační agenda, např. rozhodnutí o jmenování likvidátora, zpráv auditora, roční uzávěrky, zahajovací a roční účetní rozvahy, z výročních zpráv, zakladatelských smluv, listů členů investičního fondu, seznamu akcionářů, soupisu pohledávek a závazků aj.	<b>1993 - 1999</b>
<b>ABC investiční fond pro Slezsko, a. s. v likvidaci, Opava - NAD 2688</b>	
Fond sestává z podnikových směrnic, zápisů ze zasedání představenstva a výlých hromad společnosti, stanov a likvidační agenda, např. rozhodnutí o jmenování likvidátora, zpráv auditora, roční uzávěrky, zahajovací a roční účetní rozvahy, z výročních zpráv, zakladatelských smluv, listů členů investičního fondu, seznamu akcionářů, soupisu pohledávek a závazků aj.	<b>1993 - 1999</b>
<b>ABC investiční fond pro Slezsko, a. s. v likvidaci, Opava - NAD 2688</b>	
Fond sestává z podnikových směrnic, zápisů ze zasedání představenstva a výlých hromad společnosti, stanov a likvidační agenda, např. rozhodnutí o jmenování likvidátora, zpráv auditora, roční uzávěrky, zahajovací a roční účetní rozvahy, z výročních zpráv, zakladatelských smluv, listů členů investičního fondu, seznamu akcionářů, soupisu pohledávek a závazků aj.	<b>1993 - 1999</b>
<b>ABC investiční fond pro Slezsko, a. s. v likvidaci, Opava - NAD 2688</b>	
Fond sestává z podnikových směrnic, zápisů ze zasedání představenstva a výlých hromad společnosti, stanov a likvidační agenda, např. rozhodnutí o jmenování likvidátora, zpráv auditora, roční uzávěrky, zahajovací a roční účetní rozvahy, z výročních zpráv, zakladatelských smluv, listů členů investičního fondu, seznamu akcionářů, soupisu pohledávek a závazků aj.	<b>1993 - 1999</b>

Obrázek 8.10: Seznam archiválií pro ostatní typy archiválií

Po zvolení záznamu ze seznamu archiválií se zobrazí podrobnější informace o dané archiválii a v záložce územní rozsah si uživatel interaktivně zobrazí území na mapě, odkud daná archivália pochází. Uživatel může dále zobrazit digitalizované snímky archiválie, případně přejít na detail archiválie na stránkách archivu, odkud daný záznam pochází. Uživatel má možnost si uložit archiválii do oblíbených v případě, že je v systému přihlášený.

The screenshot shows a detailed view of an archival item in the PortaArchiva application. At the top, there is a header with the PortaArchiva logo, a login button ('přihlásit se'), a registration button ('registrovat'), and a heart icon for bookmarks.

The main content area has a title 'Kniha listin všeobecná kláštera Minoritů Opava' (Book of documents of the general chapter of the Minorites of Opava) with a house icon. Below the title, there are two tabs: 'Informace' (Information) and 'Územní rozsah' (Spatial scope). The 'Informace' tab is selected.

The information is presented in a grid format:

<b>Číslo NAD</b> 125	<b>Druh knihy</b> pozemková kniha	<b>Popis</b> 24 x 38 x 5 cm; polokož. vazba, záchr.
<b>Fond</b> Minorité Opava	<b>Původní název</b> Tropauer P.P. Minoriten Hypothekenbuch	<b>Množství</b> Folia celkem: 237; folia popsaná: 6; folia nepopsaná: 231; folia nepopsaná - výčet: 7-237
<b>Značka fondu</b> Minorité v Opavě	<b>Díl knihy</b> II.	<b>Způsob zápisu</b> chronologicky
<b>Inventární číslo</b> 55	<b>Pouze index</b> Ne	<b>Jazyk</b> Němčina
<b>Pořadové číslo</b> 6795	<b>Časový rozsah</b> 1850 - 1850	<b>Signatura</b> A 34-5
	<b>Místo uložení</b> Zemský archiv v Opavě	

On the right side of the grid, there is a thumbnail image of the book cover labeled '14 snímků' (14 scans). Below the grid, there are two buttons: 'Zobrazit v prohlížeči' (View in browser) with a camera icon and 'Zobrazit na stránkách archivu' (View on archive pages) with a document icon.

Obrázek 8.11: Detailní pohled na archiválii

V případě, že je uživatel přihlášený, tak má přístup ke svému profilu. V profilu aplikace může kromě změny osobních a přihlašovacích údajů také spravovat svoje oblíbené archiválie, přidané záložky a osobní poznámky v rámci systému.

Obrázek 8.12: Detailní pohled na archiválii

Samotné jádro aplikace je prohlížeč digitalizovaných snímků archiválií, který vychází z rozložení použitého při tvorbě interaktivního prototypu, jenž byl představen v kapitole 7.5 Tvorba interaktivního prototypu. Hlavní ovládací panel se nachází v horní části obrazovky a je doplněn o základní informace aktuálně prohlížené archiválie.

Obrázek 8.13: Prohlížeč snímků archiválie

# Kapitola 9

# Implementace

V této části je popsána implementace celého systému. Vzhledem k faktu, že většina rozhodnutí byla již učiněna ve fázi návrhu, tak zde budou pouze nastíněny problémy, které se při implementaci vyskytly, a jak byly řešeny.

## 9.1 Stahování a aktualizace dat

Vzhledem k tomu, že se jedná o data centrickou aplikaci, tak je tento subsystém v rámci celé aplikace velmi kritický. Většinu scrapovacích pavouků se povedlo rozšířit o získávání URL adres snímků, ale u některých to vedlo na více HTTP dotazů pro získání jednoho záznamu. V rámci některých archivů je nutné provést dodatečný dotaz předtím, než je možné zobrazit snímek, proto v doménovém modelu entita snímku obsahuje položku `prefetchUrl`. Jedním z velmi omezujičích požadavků na systém bylo, že má uchovávat pouze odkazy na původní datové úložiště, jelikož data sesbíraná z více archivů by kladla velké požadavky na kapacitu úložiště. Ve výsledku se jednalo o největší problém v rámci řešení daného systému, jelikož jednotlivé archivy používají různé formáty a typy uložení, což vyžadovalo mnoho práce na straně jak samotných scraperů, tak i na straně klientské aplikace, která se musela postarat o bezproblémové zobrazení různorodých formátů.

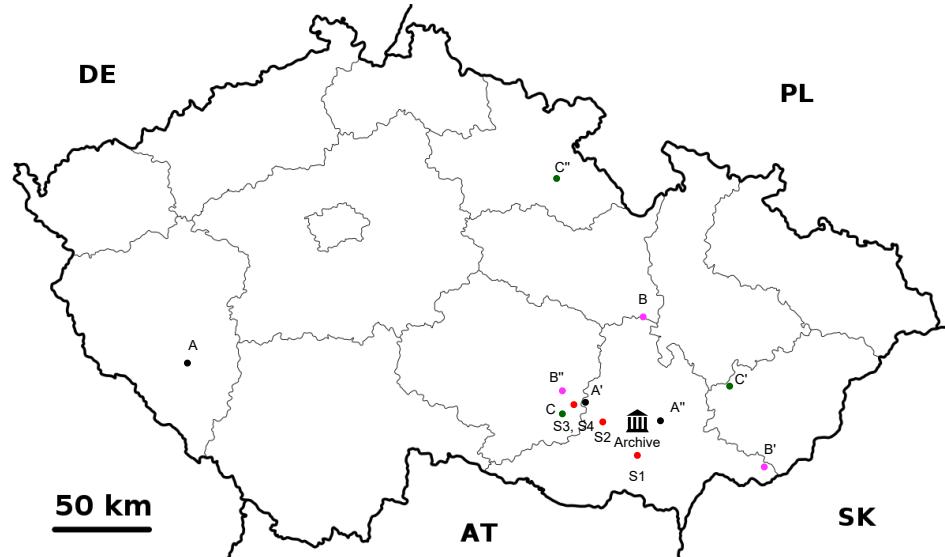
### 9.1.1 Lokality

Každý archivní materiál je navázán na jednotlivé obce, ale většina archivních systémů poskytuje informaci pouze o názvu obce. Samotný název obce bohužel nestačí k přesné identifikaci obce, jelikož názvy obcí nejsou unikátní. Prvotní myšlenka, jež byla nastíněna vedoucím práce, spočívala ve využití jedinečných identifikátorů RÚIAN. Tento přístup narázел na problém, že daný systém identifikátorů se používá pouze na území České republiky a mnoho archiválií, které se nacházejí v pohraničních archivech, obsahuje města i v zahraničí. Další problém tohoto přístupu spočíval v tom, že identifikátor RÚIAN je sice unikátní, ale tím, že ho archivní weby neuvádějí, tak je nezbytné RÚIAN vyhledat podle nascrapovaných dat. V případě získání více shod na název obce, není možné jednoznačně určit jaký záznam zvolit. Odpověď nelze ohodnotit podle korektnosti shody a následně porovnat jednotlivé shody. Vhodným ukazatelem by mohl být kraj nebo okres, kde se daná obec nachází. Problém je, že kraje a okresy se v průběhu času formovaly a posouvaly se jednotlivé hranice, takže není zaručeno, že Moravský zemský archiv obsahuje záznamy pouze z Jihomoravského kraje. Na základě tohoto zjištění bylo nutné od identifikace pomocí RÚIAN ustoupit a hledat alternativní přístup. Jako alternativní přístup bylo zvoleno použití zeměpisných souřadnic

jednotlivých obcí a porovnání vzdálenosti od archivu, který danou archiválii uchovává.

V první řadě tedy bylo potřeba z informací dostupných pomocí scrapování získat jednotlivé obce. Proces přeměny adresy na zeměpisné souřadnice se nazývá geokódování a existuje několik poskytovatelů, kteří tuto službu nabízí. V rámci České republiky se jedná například o [Mapy.cz](#), které poskytují pravděpodobně nejdetailnější informace o oblastech na našem území, a to včetně již zaniklých obcí. Použití geolokačního API od [Mapy.cz](#) by se jevilo jako nejlepší možná volba, kdyby se od určitého vytížení za službu nemuselo platit. Za tímto účelem bylo využito geolokační API od neziskové organizace [OpenStreetMap.org](#), které je plně zdarma. Mezi hlavní problémy tohoto poskytovatele patří fakt, že jej plní lidé, tudíž se zde vyskytují často chybné či neúplné informace. Další limitace tohoto poskytovatele je, že v rámci podmínek vkládání nových záznamů zakazuje vkládat historické názvy a jeho snahou je mapovat pouze aktuální lokality, což vzhledem k našemu řešenému problému představuje velkou komplikaci. Geokódovací API funguje podobně jako vyhledávač, takže v případě, že je vložena adresa, je získáno více možných výsledků. Vzhledem k tomu, že se jedná o volání další externí služby, jež vytváří velkou prodlevu, tak jsou výsledky memorizovány a v případě, kdy je zavolán dotaz na shodnou adresu, tak se použije hodnota, kterou vrátila služba naposledy. Pro zvýšení přesnosti se v rámci dotazu posílá poloha archivu jako středový bod, poblíž kterého se mají výsledky hledat.

Dalším krokem je tedy algoritmicky rozhodnout, jaký výsledek je správný, a to se snahou o minimalizaci chybovosti. Navržený algoritmus vychází z faktu, že obce jsou často poblíž daného archivu, kde jsou archiválie uloženy, a zároveň v případě, kdy se k archivnímu záznamu vztahuje více obcí, tak tyto obce tvoří shluk. Tímto způsobem lze heuristicky vybrat správnou obec v případě, kdy existuje více obcí se stejným názvem. Tento problém bohužel naráží na limitace využitého Geolokačního API, jelikož existují případy, kdy se vesnice jmenovala stejně jako dnešní větší město, takže API naleze nesprávně dnešní město. Na následujícím snímku lze vidět ilustrační příklad, kdy daný archivní záznam se vztahuje ke třem obcím A, B a C. Pro každou obec nám API vrátilo více možných výsledků. Tento problém je v informatice obecně znám pod pojmem hledání shluků. Žádoucí tedy je vybrat nejbližší body a přepočítat nový střed shluku. Tento postup je nutné opakovat, a to do té doby, dokud se střed neustálí a nejsou vybrány nejbližší body z každé skupiny. Jelikož obce by měly být blízko archivu, tak do bodů, ze kterých se střed počítá, je přidána i lokace archivu. Body S1, S2, S3 a S4 značí postupný vývoj středu shluku.



Obrázek 9.1: Vizualizace clusterizačního algoritmu<sup>1</sup>

Je patrné, že se tedy jedná o algoritmus z třídy **fixed-point** a tělo algoritmu tvorí primární smyčka, která čeká, dokud se neustálí střed. Jakmile dvě po sobě jdoucí iterace vrátí stejný výsledek, tak je zaručeno, že algoritmus může skončit. Jelikož vstupní parametry jsou fixní a jediný volatilní parametr je střed, tak jakmile se nezmění střed, tak neexistuje způsob, jak by algoritmus mohl vrátit jinou hodnotu, a to i v případě, kdy by algoritmus iteroval do nekonečna. V hlavní smyčce je tedy z každé kategorie vybírána nejbližší bod. K výsledné množině se přidá aktuální střed, jenž je inicializován na bod archivu a přepočítá se nový střed. Na konci algoritmus vybere nejbližší bod z každé kategorie pro ustálený střed a danou množinu vrátí.

---

#### Algoritmus 1 Hledání shluku lokalit

---

- 1: **Funkce** NAJDI SHLUK LOKALIT( $S, C$ )
  - 2:     **Vstup:** Množina množin potenciálních oblastí  $S$ , počáteční střed shluku  $C$
  - 3:     **Výstup:** Množina výsledných oblastí  $V$
  - 4:     **Opakuj**
  - 5:          $P \leftarrow \{p \in \mathbb{R} \times \mathbb{R} \mid \exists s \in S : |s| > 0 \wedge p = \text{nejblížšíBod}(C, s)\} \cup \{C\}$
  - 6:          $C' \leftarrow C$
  - 7:          $C \leftarrow \text{spoctiStred}(P)$
  - 8:     **Dokud**  $C \neq C'$
  - 9:          $V \leftarrow \{v \in \mathbb{R} \times \mathbb{R} \mid \exists s \in S : |s| > 0 \wedge v = \text{nejblížšíBod}(C, s)\}$
  - 10:    **Vrat**  $V$
  - 11: **Konec Funkce**
- 

V rámci shlukovacího algoritmu je využita metoda pro nalezení nejbližšího bodu z množiny k aktuálnímu středu. Tento algoritmus spočítá vzdálenost od středu pro každý bod. Funkce vrací nejmenší prvek ze spočtené množiny  $D$ .

---

<sup>1</sup>Podkladová mapa dostupná z: [https://upload.wikimedia.org/wikipedia/commons/thumb/4/48/Czechia\\_outline\\_map.svg/1200px-Czechia\\_-\\_outline\\_map.svg.png](https://upload.wikimedia.org/wikipedia/commons/thumb/4/48/Czechia_outline_map.svg/1200px-Czechia_-_outline_map.svg.png)

---

**Algoritmus 2** Hledání nejbližšího bodu

---

- 1: **Funkce** NEJBLIZSI(BOD( $C, P$ )

---
- 2:   **Vstup:** Středový bod  $C$ , množina bodů  $P$
- 3:   **Výstup:** Nejbližší bod  $V$  k bodu  $C$
- 4:    $D \leftarrow \{d \in \mathbb{R} \times \mathbb{R} \mid \exists p \in P : d = \text{spoctiVzdalenost}(p, C)\}$
- 5:    $V \leftarrow \arg \min_{d \in D} d$
- 6:   **Vrat V**
- 7: **Konec Funkce**

---

Střed se spočítá jako aritmetický průměr zeměpisné šířky a výšky.

---

**Algoritmus 3** Výpočet středu bodů

---

- 1: **Funkce** SPOCTISTRÉD( $P$ )

---
- 2:   **Vstup:** Množina bodů  $P$
- 3:   **Výstup:** Středový bod  $C$
- 4:    $n \leftarrow |P|$
- 5:    $\text{sumLat} \leftarrow \sum_{p \in P} p.\text{latitude}$
- 6:    $\text{sumLon} \leftarrow \sum_{p \in P} p.\text{longitude}$
- 7:    $C \leftarrow \left( \frac{\text{sumLat}}{n}, \frac{\text{sumLon}}{n} \right)$
- 8:   **Vrat C**
- 9: **Konec Funkce**

---

Pro výpočet nejbližšího bodu ke středu je potřeba spočítat vzdálenost dvou bodů. Standardně se vzdálenost dvou bodů počítá pomocí Eukleidovské vzdálenosti. V tomto případě se jedná o souřadnice na planetě Zemi, takže je potřeba vzít v potaz, že planeta Země není rovina. V případě použití Eukleidovské vzdálenosti na zeměpisné souřadnice by výsledky nebyly korektní. Pro výpočet bude použit vzorec ze sférické trigonometrie, který se využívá například i v oblasti navigace. Vzdálenost dvou bodů na kouli se vypočítá pomocí Harvesinova vzorce [14], jenž počítá se zakřivením povrchu.

---

**Algoritmus 4** Výpočet vzdálenosti mezi dvěma body na Zemi pomocí Harvesinova vzorce

---

- 1: **Funkce** SPOCTIVZDALENOST( $U, V$ )

---
- 2:   **Vstup:** Bod  $U$  s zeměpisnou šířkou  $U_{lat}$  a délou  $U_{lon}$
- 3:   **Vstup:** Bod  $V$  s zeměpisnou šířkou  $V_{lat}$  a délou  $V_{lon}$
- 4:   **Výstup:** Vzdálenost  $d$  mezi dvěma body
- 5:    $R \leftarrow 6371$  ▷ Přibližný poloměr Země v kilometrech
- 6:    $\Delta\text{lat} \leftarrow V_{lat} - U_{lat}$
- 7:    $\Delta\text{lon} \leftarrow V_{lon} - U_{lon}$
- 8:    $a \leftarrow \sin^2\left(\frac{\Delta\text{lat}}{2}\right) + \cos(U_{lat}) \cdot \cos(V_{lat}) \cdot \sin^2\left(\frac{\Delta\text{lon}}{2}\right)$
- 9:    $d \leftarrow 2 \times R \times \arcsin(\sqrt{a})$
- 10:   **Vrat d**
- 11: **Konec Funkce**

---

## 9.2 Implementace serverové části aplikace

Hlavní úlohou serverové části je zpracování dat, komunikace s databází a poskytování dat pomocí aplikačního rozhraní. V tomto případě je aplikace napsána v rámci aplikačního rámce Spring boot a stará se o komunikaci s relační i nerelační databází, komunikaci s e-mailovým klientem a poskytování dat pomocí REST rozhraní. Jak bylo zmíněno v návrhu, tak aplikace je monolitická a rozdělena na tři základní vrstvy.

### 9.2.1 Ukládání dat

Data nesplňují parametry velkých dat, a dokonce mají i pevnou strukturu, takže se jeví jako nejlepší nápad uložit data do standardní relační databáze. Relační databáze poskytuje konzistenci a eliminuje redundanci dat. Problém nastává v oblasti čtení z databáze, kdy složitější dotazy vedou na join tabulek na základě cizích klíčů, což je náročná operace na zdroje. Výkon relační databáze se dá velmi dobře podpořit tvorbou vhodných indexovacích stromů, nicméně plně textové vyhledávání není operace, pro niž byly tyto databázové systémy vytvořeny. Tento úkol je jako stvořený pro nerelační databázi ElasticSearch, která je vhodná pro analýzu velkých dat a vyhledávání v textu, ovšem použití NoSQL databáze by nezaručovalo konzistenci dat a záznamy by obsahovaly mnoho redundantních informací. Existuje návrhový vzor CQRS, jenž si s tímto dokáže poradit a počítá v jedné aplikaci s použitím obou databází a následnou synchronizací. Aplikaci využije tento návrhový vzor a poskytne uživateli maximální komfort v oblasti vyhledávání a zároveň si udrží konzistentní data v relační databázi.

#### Command Query Responsibility Segregation

Command Query Responsibility Segregation [3, 4] je architektonický vzor oddělující operace na příkazy (commands) a dotazy (queries).

Příkazy jsou operace, jež modifikují stav systému, tedy přidávají či upravují data. Tato data se zpracují a uloží do relační databáze. Následně se poše zpráva, že byla provedena změna s konkrétními daty a na straně nerelační databáze se provede synchronizace. Je potřeba mít na paměti, že použití takto oddělených databází již nesplňuje podmínky ACID, ale BASE. Toto v praxi znamená, že může nastat okamžik, kdy data mezi databázemi nejsou konzistentní, ale v budoucnu se vždy sesynchronizují.

Dotazy zpracovávají požadavky, které nemodifikují stav systému a pouze čtou z databáze. Tyto operace jsou prováděny nad nerelační databází, jež je v našem případě lépe optimalizovaná na plně textové vyhledávání a redukuje nutnost použití operací spojování tabulek, a to díky vhodně nastavené denormalizaci.

Toto řešení dokáže výrazně navýšit propustnost systému a v případě, kdy jsou nasazeny dvě samostatné aplikace, jedna pro zápis a druhá pro čtení, tak se aplikace bude do budoucna i lépe horizontálně škálovat.

### 9.2.2 Databázové indexy

Jak již bylo zmíněno dříve, tak správné nastavení indexů je klíčové pro rychlé vyhledávání v rámci relační databáze. Dále budou rozepsány indexy pro jednotlivé entity, které byly odvozeny z operací v rámci repozitářů. Java Persistence API ve výchozím nastavení využívá B-strom jako typ indexu.

Pro archivní záznam vznikl pouze jeden databázový index nad sloupcem link, jenž se využívá při upsertování záznamů ze scraperů a to ve chvíli, kdy se ověřuje, zda již daný záznam v databázi neexistuje.

Název indexu	Indexované sloupce	Unikátní
idx_link	link	✓

Tabulka 9.1: Indexy pro archivní záznam

Tabulka pro archivy obsahuje indexy nad položkami názvu a zkratky, podle nichž se archiv vždy hledá. Toto umožňuje ze scraperů posílat pouze název archivu a nemusí se tak znát přesný identifikátor daného archivu.

Název indexu	Indexované sloupce	Unikátní
idx_abbreviation	abbreviation	✓
idx_name	name	✓

Tabulka 9.2: Indexy pro archivy

Záložky obsahují dva databázové indexy. Index `idx_scan_url` slouží pro upsert, kdy se kontroluje, zda daná záložka již neexistuje. Index `idx_archival_record_id` akceleruje získávání všech záložek pro danou archiválii.

Název indexu	Indexované sloupce	Unikátní
idx_scan_url	scan_url	✓
idx_archival_record_id	archival_record_id	✗

Tabulka 9.3: Indexy pro záložky

Jazyky obsahují pouze jeden index, a to podle názvu daného jazyka. Název by mohl být použit i jako primární klíč, bohužel při použití složených databázových indexů existuje horní limit délky a použití primárního klíče jako textového řetězce značně omezuje možnosti použití v těchto indexech.

Název indexu	Indexované sloupce	Unikátní
idx_name	name	✓

Tabulka 9.4: Indexy pro jazyky

U lokalit existuje složený index pro kontrolu, zda se daná lokalita již v databázi nenačází. Indexy pro jednotlivé části adresy jsou určeny pro efektivní hledání adres, které dále slouží pro naplnění dropdown boxů na straně uživatelské aplikace.

Název indexu	Indexované sloupce	Unikátní
idx_location	country, region, district, municipality, borough	✗
idx_region	region	✗
idx_country	country	✗
idx_district	district	✗

Tabulka 9.5: Indexy pro lokality

Tabulka určená pro ukládání obsahuje tři databázové indexy. Index `idx_search` slouží pro efektivní vyhledávání poznámek pro danou archiválii, a to na základě aktuálně přihlášeného uživatele. Index `idx_search_2` je určen pro upsert, kde se kontroluje, zda daná poznámka již neexistuje.

Název indexu	Indexované sloupce	Unikátní
idx_archival_record_id	archival_record_id	✗
idx_search	archival_record_id, accessibility, user_id	✗
idx_search_2	scan_url, user_id, accessibility	✓

Tabulka 9.6: Indexy pro poznámky

Indexy pro uživatele jsou nastaveny tak, aby aplikace mohla efektivně vyhledávat uživatele při ověření JWT tokenu. Dále jsou vytvořeny indexy pro jednotlivé hashe, v případě že si chce uživatel ověřit účet, tak se na základě tohoto hashe dohledává účet.

Název indexu	Indexované sloupce	Unikátní
idx_email	email	✓
idx_is_verified_and_email	email, isVerified	✓
idx_verify_hash	verify_hash	✓
idx_password_reset_hash	password_reset_hash	✓

Tabulka 9.7: Indexy pro uživatele

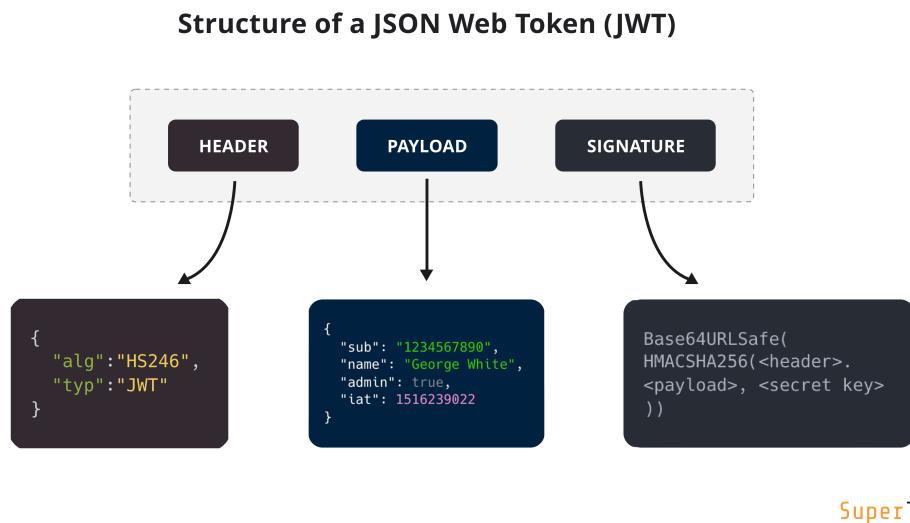
### Synchronizace databází

Relační i nerelační databáze je nastavená, nyní je třeba zařídit jejich vzájemnou synchronizaci. Při implementaci čistého CQRS návrhového vzoru se uvažují dvě separátní služby, které spolu komunikují zasíláním zpráv. Tento typ implementace by za cenu lepší škálovatelnosti přinášel značnou režii navíc. Vzhledem k tomu, že se pro mapování objektů mezi relační databází a objekty v Javě využívá aplikační rámec Hibernate, tak je použita nadstavba Hibernate-search [35]. Hibernate musí sledovat změny na objektech a v případě ukončení transakce propíše změny do databáze. To, že sleduje změny, je v tomto případě klíčové, jelikož to je přesně to, co je potřeba pro synchronizaci daných databází. Hibernate-search po správné konfiguraci propisuje změny do indexů v Elasticseach databázi. Všechna data

jsou uložena v relační databázi, sloužící jako zdroj faktů, a v rámci Elasticsearch se udržuje pouze indexační stromy bez samotných dat. Dotazy tedy najdou identifikátory v nerelační databázi a následně je Hibernate vyhledá v relační databázi. Tento přístup je sice pomalejší než duplikování dat do nerelační databáze, ale má nižší paměťové nároky. V případě, že by se data duplikovala do nerelační databáze, tak by musely být nastaveny atributy pro projekci.

### 9.2.3 Bezpečnost

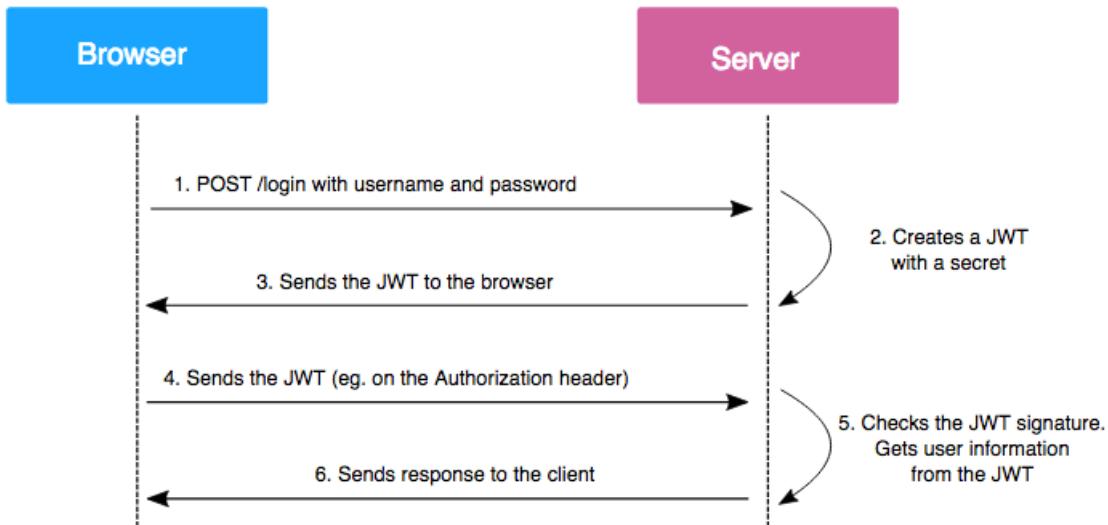
V návrhu se vyskytuje pojem přihlášený uživatel, a proto je na místě řešit bezpečnost a jak bude probíhat autentizace uživatelů. Při použití architektury klient-server se může použít HTTP Basic autentizace. Problém tohoto přístupu je, že se heslo posílá v čitelné podobě, a jelikož protokol HTTP je bezstavový, tak se musí posílat při každém požadavku na server. Tento problém může být částečně vyřešen použitím šifrovaného spojení pomocí HTTPS, ovšem stále je nutné řešit problém uchování hesla na straně klienta, jelikož ho klient musí přiložit ke každému požadavku. Z hlediska bezpečnosti se jeví daleko praktičtější použití token-based autentizace. Token-based autentizace vyžaduje poslání citlivých údajů pouze jednou a po provedení autentizace se vygeneruje token, jenž bude uživatel používat dál pro autorizaci. Dalším přístupem je použití OAuth, který pro autentizaci používá služby třetích stran. Tato služba je použita v mnoha systémech, kdy se uživatelé přihlašují pomocí Facebook nebo Google účtu bez registrace v rámci dané aplikace. Pro požadavky této aplikace se jeví jako nejlepší řešení použít **token-based** autentizaci, konkrétně JSON Web Token [18], který je definován standardem RFC 7519. Token se skládá z hlavičky, těla a podpisu. Hlavička obsahuje informace o typu tokenu a použitém algoritmu, jímž byl token podepsán. Tělo obsahuje uživatelská data ve formátu JSON. Třetí část je podpis dat tokenu a slouží k ověření autenticity a integrity tokenu.



Obrázek 9.2: Struktura JWT tokenu<sup>2</sup>

<sup>2</sup>Diagram dostupný z: <https://supertokens.com/static/b0172cabcd583dd4ed222bdb83fc51a/9af93/jwt-structure.png>

Na obrázku 9.3 lze vidět ukázku použití tokenu v komunikaci mezi klientem a serverem. Uživatel provede přihlášení, server ověří údaje a vygeneruje token. Tento token si klient uchová a použije při dalších požadavcích vyžadujících autorizaci. Klient si sice musí informaci pamatovat stejně jako u HTTP basic, ale v případě, že se útočník dostane k tokenu, tak ten zpravidla platí po omezenou dobu a bez znalosti hesla není útočník schopný vygenerovat nový token, a tím prodloužit platnost.



Obrázek 9.3: Ukázka použití JWT tokenu<sup>3</sup>

#### 9.2.4 Odesílání e-mailů

Požadavky jako ověřování uživatelů a resetování hesla vyžadují implementovat podporu pro komunikaci se SMTP serverem. V rámci produkční verze je využit SMTP server od Gmailu, ovšem v době implementace není vhodné zasílat e-maily na skutečné adresy z důvodu ladění a možného nechtěného spamu. Za tímto účelem byl použit nástroj Mailhog, který se tváří jako SMTP server, ovšem místo skutečného odesílání e-mailů je pouze zachytává a zobrazuje skrze webové rozhraní. Pro samotnou komunikaci se serverem bylo využito JavaMail API. Veškeré nastavení e-mailového serveru je skrze proměnné prostředí a konfiguraci je tedy možné provést bez zásahu do zdrojového kódu.

### 9.3 Dokumentace

V době, kdy programátor tvoří nějaký program nebo algoritmus, tak mu je úplně jasné, jak daný kus kódu funguje. Problém nastává v případě, že má do projektu začít přispívat nový kolega nebo se k projektu vývojář vrací po delší době. Kvůli těmto aspektům je dokumentace velmi důležitou součástí zdrojových kódů. Pro Javu existuje dokumentační nástroj Javadoc, který na úkor striktního formátu dokumentačních komentářů vygeneruje celou programovou dokumentaci ve formě webové stránky. Dokumentaci je možné vygenerovat pomocí Maven pluginu Javadoc.

<sup>3</sup>[https://www.vaadata.com/blog/wp-content/uploads/2016/12/JWT\\_tokens\\_EN.png](https://www.vaadata.com/blog/wp-content/uploads/2016/12/JWT_tokens_EN.png)

## 9.4 Implementace klientské aplikace

Klientská aplikace je napsána v reaktivním JavaScriptovém aplikačním rámci React.JS [20], který se primárně stará o vizualizaci dat a poskytuje grafické uživatelské rozhraní pro naše aplikační rozhraní. React.JS využívá deklarativní a komponentově orientovaný přístup, což znamená, že celý pohled systému se dekomponuje na jednotlivé znovupoužitelné komponenty. V rámci komponent se deklaruje jejich chování a rozložení na základě stavu. Každá komponenta může používat další komponenty, což umožňuje hierarchicky budovat nové rozhraní. Komponenta má dále stav a vlastnosti, jež si udržuje v průběhu celého svého životního cyklu. Životní cyklus komponenty zahrnuje více fází, mezi něž patří inicializace, aktualizace a odstranění. Na tyto fáze životního cyklu je možné navázat obslužné metody, které například při inicializaci mohou provést stažení požadovaných dat a naopak u odstranění po sobě uklidit. Ke komponentě může náležet soubor s definicí stylů. V případě, že má komponenta obsahovat logiku, tak je tato logika vytáhnutá do separátního hooku, díky čemuž je dodržen princip jedné zodpovědnosti. React pracuje nad virtuálním DOM stromem, při každé změně stavu aplikace se vygeneruje nový virtuální DOM strom, který se porovná s aktuálním, a nahradí se pouze změněné části. Toto řešení má pozitivní dopad na výkon aplikace.

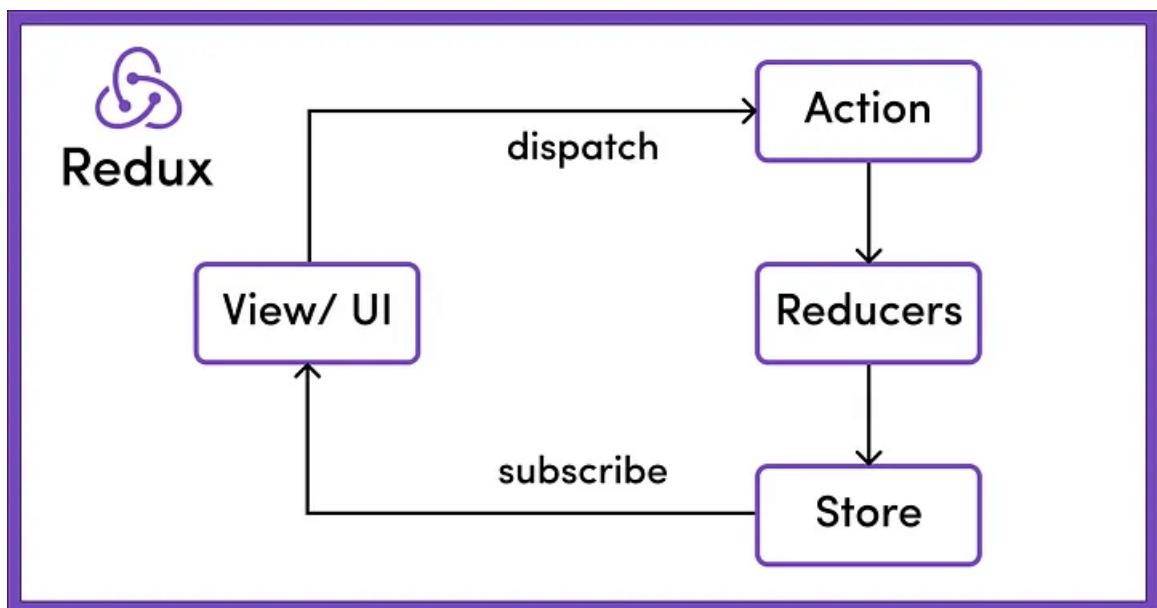
JavaScript je dynamicky typovaný, což znamená, že proměnné mohou měnit svůj typ za běhu programu. Pro větší aplikace je lepší použít typový systém umožňující psát robustnější kód a snižovat pravděpodobnost chyb. TypeScript je opensource nadstavba od Microsoftu pro jazyk JavaScript. Tato nadstavba přidává statické typování. Největší nevýhoda použití TypeScriptu spočívá v použití knihoven, které nemají vytvořené definice typů pro TypeScript. V tomto případě existují tři možnosti: najít alternativní knihovnu s podporou TypeScriptu, vytvořit definici typů nebo komponentu, v níž má být knihovna použita přepsat do JavaScriptu. První volba je nejrozumnější, pokud daná alternativa existuje. Problém u poslední alternativy nastává v případě použití ESLint, který v konfiguraci může zakazovat použití JavaScriptových komponent, a proto je potřeba tuto konfiguraci upravit.

Pro kontrolu kvality kódu byl zvolen nástroj ESLint, který provádí statickou analýzu TypeScriptového kódu. Nástroj se soustředí na nalezení a odstranění chyb, konvenční nedostatky a eliminaci anti návrhových vzorů. ESLint funguje na základě předdefinovaných pravidel, a v tomto projektu byla zvolena konfigurace, která se využívá ve společnosti Google.

Uživatelské rozhraní by mělo být konzistentní napříč celým systémem a mělo by používat jednotný vzhled u jednotlivých komponent. Cílem tvorby nových systémů není znova objevovat kolo. Existují různé sady předdefinovaných komponent, které se dají použít při tvorbě nových pohledů v systému. Pro tento projekt byla zvolena knihovna [PrimeReact](#) nabízející dostatečnou paletu komponent. Vzhled výsledné aplikace vychází z návrhu v nástroji Figma a ukázky uživatelského rozhraní jsou v rámci přílohy C Ukázka grafického uživatelského rozhraní výsledné aplikace.

Pro komunikaci s API pomocí protokolu HTTP se používá knihovna Axios. Samotné volání Axios je zapouzdřeno a komponenta k němu přistupuje skrze Redux hooky. Jak bylo zmíněno v popisu aplikačního rámce React.JS, tak každá komponenta má interní stav, na základě kterého se provádí překreslení. Problém nastává, když více komponent má sdílet stejný stav. Toho se dá docílit předáváním hodnot skrze parametry komponent, což vyža-

duje, aby definice tohoto stavu byla v komponentě, jež je kořenem podstromu DOM, v němž jsou všechny komponenty s touto závislostí. Definování proměnné v komponentě, která danou závislost nevyžaduje, je nepřehledné a nepřenositelné. Lepší alternativou je použít React konstruktu **kontext**, kde se definuje rozsah platnosti daného kontextu a následně si komponenty skrze něj mohou sdílet data. Robustním řešením, které kombinuje možnost sdílet data napříč komponentami a zároveň zapouzdruje komunikaci s API, je Redux. Redux definuje centrální úložiště dat **store** a striktně odděluje operace pro čtení a zápis. V případě, že má dojít k úpravě dat ve **storu**, tak se volá přes **dispatch** hook konkrétní akce. Akce může obsahovat volání API a následně vytvoří zprávu pro **reducer**. Zpráva pro **reducer** obsahuje data a typ. Typ akce musí být unikátní pro celou aplikaci. Následně Redux prochází jednotlivé **reducery**, jež jsou implementovány formou switch konstrukce. Jakmile je konkrétní obslužná funkce nalezena, tak se provede změna stavu v **store**. V případě, že komponenta chce číst hodnotu ze **store**, tak použije **selector** hook, který začne odebírat data z centrálního úložiště, a v případě změny závislých dat ve **storu** je daná komponenta překreslena. Tato základní architektura je rozšířena o **selectorHook** a **actionHook**, které zapouzdrují jednotlivé akce a selectory a eliminují nutnost použití Redux hooků přímo v komponentech. Pomocí této abstrakce lze přímo volat v komponentech metody a tím dochází k plnému odstínění od architektury Reduxu.



Obrázek 9.4: Architektura Redux<sup>4</sup>

Jak bylo popsáno v úvodu práce, tak archivy poskytují naskenované archiválie v různých formátech, a to včetně dlaždicových. Pro vizualizaci těchto dlaždicových snímků je využita opensource knihovna OpenSeaDragon [19], která má nativní podporu pro zobrazování dlaždicových formátů. Kromě samotného zobrazení obsahuje knihovna možnosti pro navigaci, přibližování a posouvání po snímků. Základní funkcionalita OpenSeaDragon lze rozšířit díky zásuvným modulům. V této práci je využit zásuvný modul mající podporu pro Fabric.JS vrstvu. Fabric.JS [11] je knihovna pro práci s kreslením a manipulací s grafickými prvky

<sup>4</sup>[https://miro.medium.com/v2/resize:fit:720/format:webp/1\\*EV1KQifr5KwiupJ9q7tdxA.png](https://miro.medium.com/v2/resize:fit:720/format:webp/1*EV1KQifr5KwiupJ9q7tdxA.png)

v rámci HTML5 canvasu. Bohužel použití těchto knihoven hlásilo chyby a bylo nutné referenční knihovnu forknout<sup>5</sup> na githubu upravit a následně publikovat<sup>6</sup> na platformě npm.

The image shows two pages of a historical baptismal register from 1862. The left page is titled 'Narození a pokřtění' (Birth and Baptism) and the right page is titled 'pro rok 1862' (for the year 1862). Both pages have columns for 'Strana' (Page), 'Datum' (Date), 'Křest' (Baptism), 'Jméno' (Name), 'Vira' (Father), 'Otec' (Mother), and 'Jméno, stav a přibuzenství' (Name, status and kinship). Handwritten annotations in green highlight specific details: on the left, the name 'Josef' is highlighted; on the right, 'Františka' and 'Franz Špidla' are highlighted. Other annotations include 'Coop ??', 'hab Katharina?', 'Hofinek aus Boskowic', 'N 167', 'Františka, tochter des Anton Sicher, zimmermanns in Boskowic, und Vincencia geb. Franz Pach, schuster hier.', 'Schüler in Boskowic', 'Mariana dessen gattin?', and 'číslo domu ?'. The handwriting is cursive and varies in color (black, blue, green).

Obrázek 9.5: Ukázka archiválie s anotacemi

## 9.5 Implementace proxy pro snímky archiválií

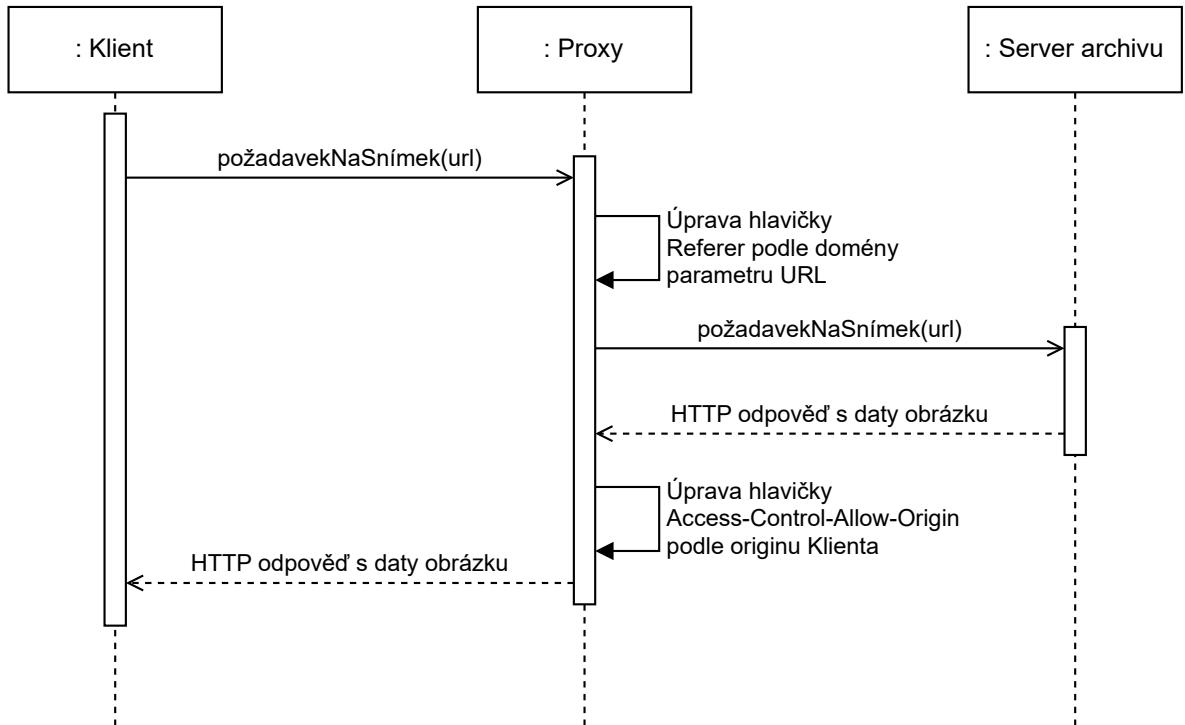
Pokud by byl použit externí odkaz na snímek přímo z webové aplikace a ta by se jej pokusila stáhnout, tak by požadavek skončil s HTTP stavovým kódem 403 – zamítnuto. Na vině je bezpečnostní mechanismus CORS [5], který je implementován ve webovém prohlížeči. Bezpečnostní mechanismus CORS se spouští v případě, kdy načtená webová stránka z jedné domény (origin) požádá o zdroje z jiné domény. Klient v požadavku tedy zasílá hlavičku `origin` a server následně musí k odpovědi přidat hlavičku `Access-Control-Allow-Origin`. Prohlížeč následně porovná, že se tyto hodnoty rovnají a v opačném případě zamítne přístup k datům.

S touto mechanikou je potřeba počítat i při implementaci klientské aplikace, jež komunikuje se separátní serverovou aplikací. Zde je problém lehce řešitelný, jelikož na straně serveru stačí správně nastavit odesílání hlaviček, které se ke CORS vztahují. Problém nastává v případě, kdy je třeba přistoupit k cizímu API. V případě, že má cizí API omezený seznam povolených klientů, tak toto chování nelze změnit bez zásahu do kódu.

Jednoduchým řešením je tvorba samostatné webové aplikace, která se bude chovat jako proxy. Proxy je takový prostředníček, jenž vezme požadavek, přepošle ho a následně vrátí odpověď. Celé řešení je napsáno pomocí JavaScriptového aplikačního rámce pro tvorbu aplikačních rozhraní Express.JS.

<sup>5</sup><https://github.com/sestakp/OpenseadragonFabricjsOverlay>

<sup>6</sup><https://www.npmjs.com/package/@sestakp/openseadragon-fabricjs-overlay>



Obrázek 9.6: Ukázka komunikace pomocí proxy

## 9.6 Kontejnerizace

Celý systém se skládá z několika separátních služeb, kde každá služba má svůj seznam závislostí, které je potřeba nainstalovat a udržovat. U každé závislosti se mohou vyskytnout problémy s nekompatibilním nastavením prostředí a verzemi jednotlivých knihoven. V neposlední řadě je zde otázka bezpečnosti, jelikož aplikace jsou spuštěny v izolovaném prostředí a pro potencionální nebezpečnou aplikaci je tedy složitější se dostat do hostujícího stroje. Kontejnerizace je lehčí varianta virtualizace, jež díky využívání hostitelského operačního systému není tak náročná na výpočetní výkon. V našem případě využijeme platformu Docker.

### 9.6.1 Docker

Docker [10] je open source platforma pro vývoj, doručení a provozování aplikací. Pomocí Dockeru lze každou aplikaci zabalit a spustit v separátním kontejneru. Na jednom stroji je následně možné provozovat více kontejnerů současně. Pro každou aplikaci nebo službu, která má být nasazena přes Docker, je třeba vytvořit **Dockerfile**. V rámci každého **Dockerfile** je nutné specifikovat Docker Image, což je základní balík obsahující závislosti vždy pro konkrétní aplikaci, jako je například Python nebo běhové prostředí pro Java. Následuje sekvence příkazů, která většinou provádí nakopírování zdrojových souborů, stažení závislostí, překlad a spuštění aplikace. Většinou se po takto nasazené službě vyžaduje interakce, proto se vytvoří tunel portu z daného kontejneru. Tuto situaci si lze představit následovně – běží-li v kontejneru webový server na portu 80, tak lze port vytunelovat na libovolný neobsazený hostitelský port a přistoupit tak k aplikaci, která je nasazená v kontejneru. Pokud je potřeba, aby kontejner nepřišel o svoje data v případě restartu, tak lze na daný

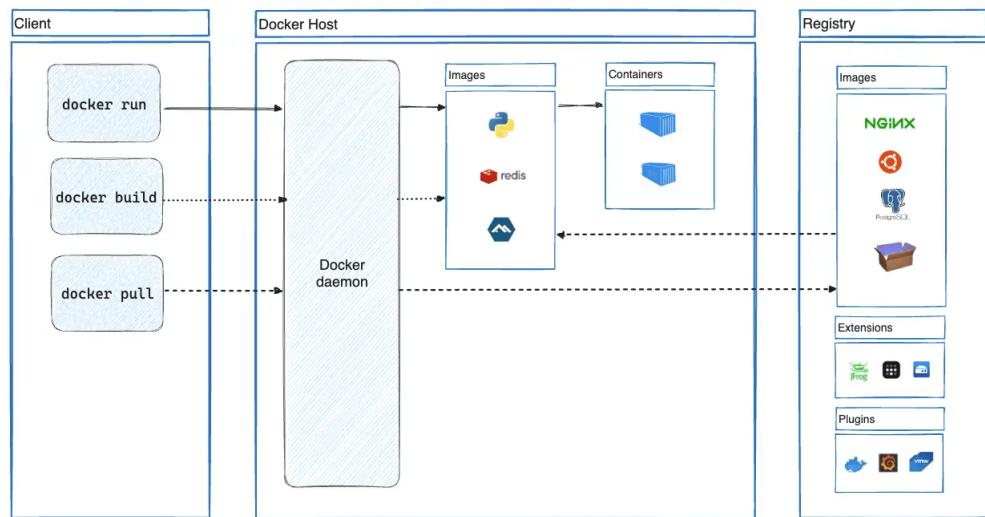
kontejner napojit volume. Jak bylo zmíněno výše, tak jednotlivé kontejnery jsou vzájemně izolované, což v případě distribuovaných systémů znemožňuje komunikaci mezi jednotlivými službami. Za tímto účelem lze v rámci Dockeru vytvořit síť obdobně jako volume. Následně tak všechny kontejnery ve stejné síti mohou mezi sebou komunikovat.

### 9.6.2 Docker compose

Použití jednotlivých `Dockerfile` sice odstíňuje od instalování závislostí, ale pořád osoba, jež systém instaluje, musí mít povědomí o architektuře aplikace. Je třeba vytvořit jednotlivé volumes, případně sít, a znát konkrétní porty jednotlivých kontejnerů, které je nutné vytunetoval. Pro úspěšné nasazení nezasvěcenou osobou by bylo nutné vytvořit metodiku, jež daný proces popisuje, ale stále zde hrozí, že osoba metodiku špatně interpretuje nebo udělá chybu. Pro úplné odstínění problémů s nasazením aplikace je použit konfigurační YAML soubor `docker-compose`, kde jsou specifikovány jednotlivé kontejnery a jejich `Dockerfile`. Dále je zde nakonfigurováno všechno přidružené nastavení, jako jsou zmíněné porty, volumes a sít. V případě, kdy je celý systém popsán v jednom `Docker compose` konfiguračním souboru, pak danou síť není třeba tvořit, jelikož pro každý `Docker compose` existuje síť implicitně. Následně stačí systém jako celek nasadit jedním příkazem `docker compose up`, který může být doplněn o přepínač `-d`, což daný příkaz spustí na pozadí, a `-build` pro znovu přeložení aplikace v případě, že nahráváme novou verzi.

### 9.6.3 Architektura Docker

Docker používá architekturu klient-server, kde klient s daemonem komunikuje pomocí UNIX socketů, síťového rozhraní nebo REST API. Klient se může připojit na lokálního i vzdáleného daemona. Jak bylo zmíněno, tak v každém `Dockerfile` je specifikován Docker image, který se stahuje ze vzdálených Registry. Jakmile se daný Docker image stáhne, tak se uloží na disk a je dostupný pro opakování použití bez nutnosti opětovného stahování. Jakmile daemon provede stažení Docker image, tak se paralelně spustí `Dockerfile` jednotlivých služeb a následně se vytvoří kontejnery.



Obrázek 9.7: Architektura Dockeru <sup>7</sup>

---

<sup>7</sup>Obrázek pochází ze stejného zdroje jako informace o Dockeru.

# Kapitola 10

## Testování

Testování softwaru [31] se zaměřuje na ověření funkčnosti, spolehlivosti a rychlosti odezvy výsledného systému. Po testu se požaduje, aby byl deterministický a nezávislý na okolním prostředí. Test se zpravidla skládá ze tří částí. První část slouží pro přípravu prostředí pro test, nahrání testovacích dat a vytvoření SUT (System under test). V druhé části se volá testovaná metoda nad SUT. Třetí fáze porovnává skutečné výsledky s těmi očekávanými. Každá technologie poskytuje vlastní knihovnu pro testování, ale tyto rysy mají společné. Existence automatických testů je nutným předpokladem pro případný budoucí refactoring. Je třeba mít na paměti, že testování nikdy nemůže sloužit jako důkaz korektnosti obdobně jako matematický důkaz.

V rámci testování jsou rozlišovány různé typy testů, mezi které patří akceptační, unit, integrační a systémové testy.

Akceptační testování vzniká ve fázi analýzy požadavků, kde jsou podle výsledné specifikace navrženy testy zaměřující se na požadavky zákazníka. Testují se jednotlivé funkcionality a v rámci popisu testu je zahrnut postup a očekávaný výstup. Na základě výstupu akceptačního testování zákazník přebírá výsledný produkt.

Unit testy píší autoři jednotlivých komponent, kde se testuje očekávané chování v izolovaném prostředí. Při použití techniky test driven development testy vznikají před samotným kódem. V případě, že testovaná komponenta má externí závislosti, tak závislosti dostane v podobě namockovaných tříd. Mock je zástupná třída, která neobsahuje skutečnou implementaci, pouze si pamatuje, která metoda byla zavolána s jakými parametry a následně lze tyto vlastnosti ověřovat ve třetí fázi testu. K tomu, aby bylo možné zaměnit skutečné třídy za jejich mocky, je třeba aby závislosti třídy byly předávány skrze parametry. Tato praktika se nazývá inversion of control a bez ní nelze závislosti nahradit.

Integrační testování je podobné unit testům, ovšem místo mockování závislostí jsou zde již použity skutečné implementace závislostí a je testováno, zda fungují dohromady. Testuje se rozhraní mezi jednotlivými komponentami.

Systémové testování je testování systému jako celku se všemi závislostmi. V rámci API se testuje volání aplikačního rozhraní, návratové kódy a vrácená data.

Samotnou kapitolou je zátěžové testování. Zátěžové testování je metoda, která simuluje

reálné podmínky a zátěž na aplikaci za účelem zhodnocení její výkonnosti, odolnosti a stability.

## 10.1 Testování modelu v rámci programu Figma

Cílem tvorby prototypu bylo odhalit chyby v oblasti uživatelské zkušenosti již ve fázi návrhu, a to bez nutnosti přepisovat zdrojový kód. Iveta Brabcová společně s vedoucím práce poskytli cennou zpětnou vazbu, která byla reflektována v následujícím návrhu.

Samotný prototyp byl dále testován na čitelnost pro lidi s očními vadami. Výsledek testování je uveden v příloze B Simulace očních vad pohledů prototypu v nástroji Figma a z výsledků testování je patrné, že aplikace je použitelná ve všech testovaných scénářích. Testování proběhlo na pohledu hlavní strany a prohlížeče archiválií.

## 10.2 Testování serverové části

Serverová část aplikace byla pokryta automatickými testy a dodatečně testována pomocí softwaru Postman.

V první řadě byly implementovány unit testy, které pomocí knihovny **Mockito** vytvořily zástupné závislosti. Pro jednotlivé namockované závislosti se dále definuje jejich návratová hodnota pro specifické parametry. Ve fázi ověřování výsledků se kontroluje, zda byl mock zavolán s konkrétními parametry nebo se testuje počet zavolání dané metody.

Integrační testy již vyžadují skutečné závislosti, a to včetně databází. Mockito je zde použito pouze pro službu na odesílání e-mailů. Tato služba není předmětem testování. Spuštění testů vyžaduje již nasazené databáze, ovšem není žádoucí zasahovat do produkční databáze v rámci Dockeru. Systém Docker lze ovšem využít a pomocí knihovny **TestContainer** vytvořit požadované služby pro každý test a následně je po ukončení testu smazat. Alternativní řešení je použití relační databáze H2, která běží v operační paměti. Toto řešení naráží na dva problémy. Navrhovaná aplikace obsahuje specifické dotazy, jež nejsou přenositelné z MariaDB do H2 a pro databázi ElasticSearch nebyla nalezena ekvivalentní in-memory databáze. Použití Docker kontejnerů značně prodlužuje dobu běhu testů a není vhodné testy spouštět před každým přeložením aplikace. Pro testování se využívá alternativní konfigurace, která se nastavuje pomocí anotace **TestPropertySource**.

Systémové testy již testují aplikaci jako celek. Obdobně jako u integračních testů je zde využito knihovny **TestContainer**. K aplikaci se přistupuje pomocí třídy MockMvc, jež umožňuje emulovat zaslání HTTP požadavků na API bez nutnosti běhu aplikačního serveru. Ve Výsledku volání ověřujeme stavový kód, typ a obsah dat.

Celkem bylo na serverové části implementováno 93 testovacích scénářů ověřujících korektní chování aplikace.

Tests passed: 92 of 92 tests		
✓ backend (cz.vut.fit.archiveMaterials)	7 sec 944 ms	ckerjava.zerodep.shaded.
> ✓ ArchivalRecordControllerTest	827 ms	13:17:31.084 [main] DEBU
> ✓ ArchivalRecordPersistenceServiceTest	346 ms	13:17:31.084 [main] DEBU
> ✓ ArchivalRecordServiceTest	658 ms	13:17:31.084 [main] DEBU
> ✓ ArchivalRecordServiceTest	232 ms	13:17:31.084 [main] DEBU
> ✓ ArchiveControllerTest	221 ms	13:17:31.084 [main] DEBU
> ✓ ArchivePersistenceServiceTest	84 ms	13:17:31.084 [main] DEBU
> ✓ ArchiveServiceTest	167 ms	13:17:31.084 [main] DEBU
> ✓ ArchiveServiceTest	57 ms	13:17:31.084 [main] DEBU
> ✓ BookmarkControllerTest	825 ms	13:17:31.084 [main] DEBU
> ✓ BookmarkPersistenceServiceTest	141 ms	13:17:31.084 [main] DEBU
> ✓ BookmarkServiceTest	161 ms	13:17:31.084 [main] DEBU
> ✓ NoteControllerTest	1 sec 54 ms	13:17:31.084 [main] DEBU
> ✓ NotePersistenceServiceTest	1 sec 450 ms	13:17:31.084 [main] DEBU
> ✓ NoteServiceTest	153 ms	13:17:31.084 [main] DEBU
> ✓ ScanPersistenceServiceTest	152 ms	13:17:31.084 [main] DEBU
> ✓ UserControllerTest	863 ms	13:17:31.084 [main] DEBU
> ✓ UserPersistenceServiceTest	134 ms	13:17:31.084 [main] DEBU
> ✓ UserServiceTest	419 ms	13:17:31.084 [main] DEBU

Obrázek 10.1: Výsledky automatických testů

Postman umožňuje posílat HTTP požadavky včetně nastavení hlaviček. Této skutečnosti bylo využito i u analýzy jednotlivých systémů, kde bylo nutné rozklíčovat správné URL na mediální soubory, aby byly následně funkční v rámci knihovny OpenSeaDragon. Jednotlivé koncové body serverové části byly vyexportovány pomocí nástroje Swagger z OpenApi specifikace.

### 10.3 Zátěžové testování

K tomu, aby nebylo nutné nabírat desítky dobrovolníků, kteří podle metodiky budou obsluhovat systém a bude se tak testovat zátěž, bude použit specializovaný software, který se zaměřuje na vytváření zátěžových testů. Za tímto účelem bude použit open-source nástroj JMeter, jenž má ve své paletě podporu i pro protokol HTTP, a bude tak možné otestovat tuto aplikaci.

V rámci aplikace se vytvoří testovací plán a skupina vláken. V rámci skupiny vláken se nastaví počet vláken (uživatelů) a počet cyklů. V tomto testovacím případě je zvoleno 100 uživatelů a 50 opakování scénáře. Kdyby test trval příliš krátkou dobu, tak by výsledky nemusely být přesné.

JMeter má podporu pro nahrávání sekvencí dotazů, takže stačí zapnout JMeter, provádět operace v testovaném systému a testovací plán se vytvoří automaticky. K tomu, aby software mohl vytvořit tento plán, tak musí být v rámci webového prohlížeče nastavena proxy skrze server JMeteru, přes který zaznamenává dotazy na konkrétní doménu. Fun-

gování proxy bylo popsáno v kapitole 9 Implementace. Vytvořený testovací plán obsahuje 49 volání API a provádí se standardní úkony jako procházení stránek vyhledávání, filtrování, zobrazování detailů záznamů a samotných snímků. JMeter je kromě samotného měření schopen získané výsledky i vizualizovat.

### 10.3.1 Zátěžové testování na serveru Perun

Perun běží v rámci školní infrastruktury a jedná se o první server, na který byla aplikace nasazena.

Server Perun je osazen šestijádrovým (dvanáctivláknovým) procesorem Intel® Xeon® E5-2630 v2. Procesor disponuje základní taktovací frekvencí 2.60 GHz a v turbu dosahuje taktu až 3.10 GHz. Systém je dále vybaven 16GB operační pamětí a 4GB na oddílu swap. O spojení s internetem se stará rozhraní od Intelu s přenosovou rychlostí 1Gbit/s.

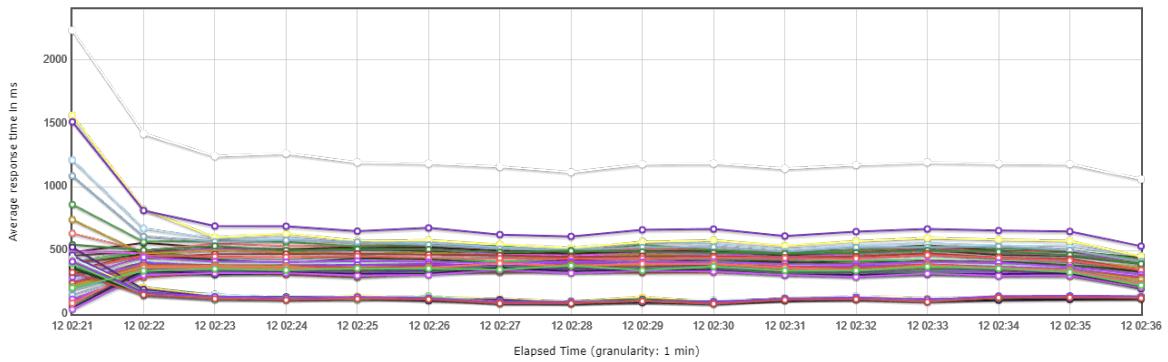
V průběhu testování bylo kontrolováno zatížení serveru, které oscilovalo mezi 80-90 %. Zatížení serveru bylo kontrolováno pomocí programu Htop a podle vytížení jader aplikace rovnoměrně distribuuje úlohy.



Obrázek 10.2: Zatížení serveru Perun v průběhu testu

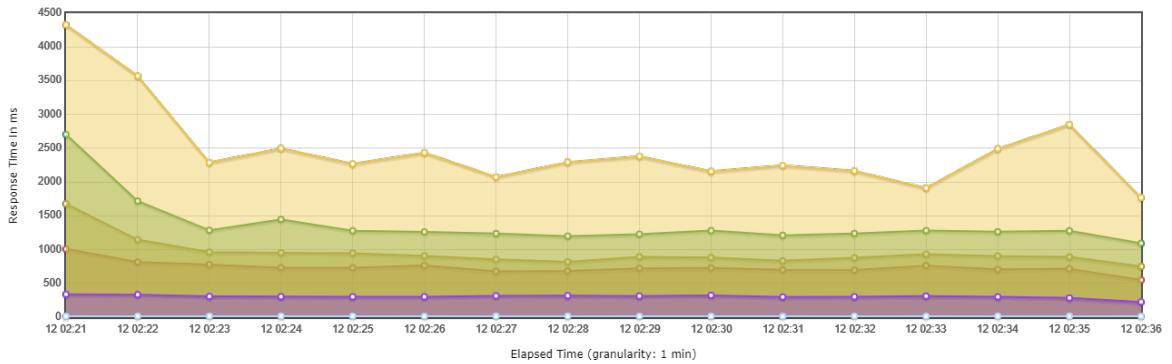
V průběhu testu, který trval 15 minut, bylo zasláno 245 000 požadavků. Z tohoto necelého čtvrt milionu požadavků žádný neskončil chybou, což značí, že systém tuto zátěž ustál.

Jak je vidět na grafu 10.3, tak průměrná doba odezvy se pohybovala okolo 500 ms. Akceptovatelná mez u informačních systémů se pohybuje do 2s [49].



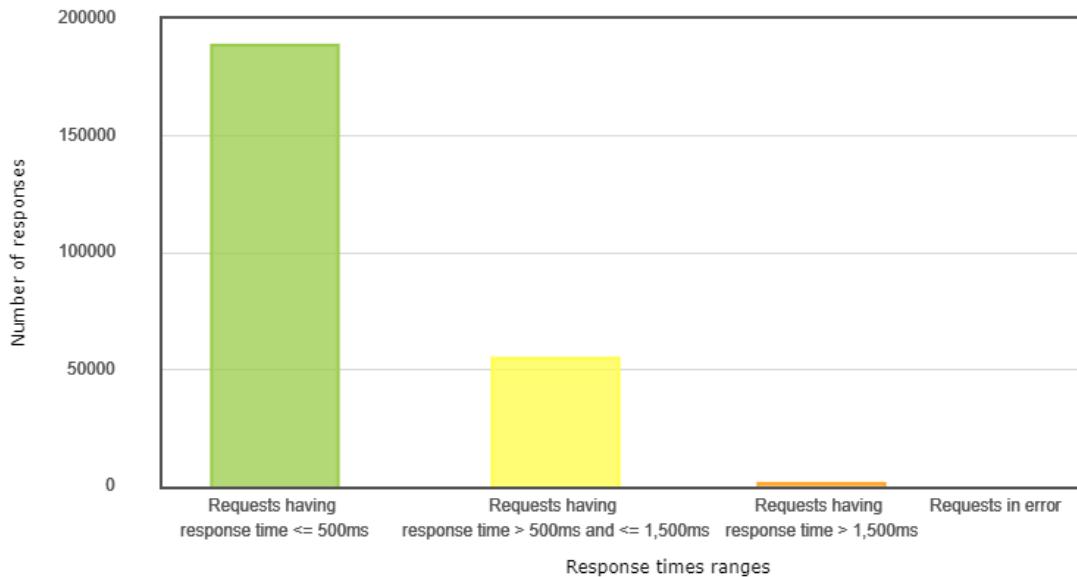
Obrázek 10.3: Průměrná doba odpovědi

Na grafu 10.4 jsou vidět jednotlivé percentily, kde oranžová horní spojnice značí maximum a fialová spojnice medián. Zajímavější údaj než maximum je zelená spojnice, která značí percentil 99. Žlutá spojnice dále reprezentuje percentil 95 a dolní červená spojnice percentil 90. To znamená, že již u percentilu 99 je většina dotazů pod 2 vteřiny.



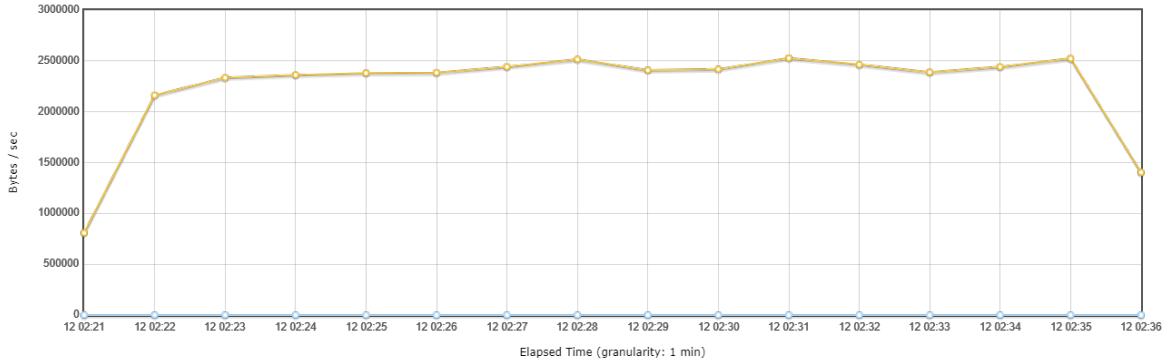
Obrázek 10.4: Percentil průměrné doby odpovědi

Zde je porovnání odezvy rozděleno do tří kategorií, přičemž dle zvoleného kritéria jsou přípustné zelené a žluté hodnoty.



Obrázek 10.5: Přehled latence

V průběhu testu bylo odesláno ze serveru více než 2MB dat každou vteřinu.



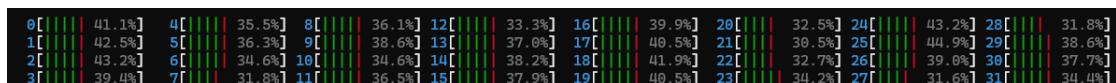
Obrázek 10.6: Datová propustnost

### 10.3.2 Zátěžové testování na serveru Radegast

Radegast běží v rámci školní infrastruktury a jedná se o server, na který má být aplikace v budoucnu přesunuta. Za tímto účelem byl vznesen požadavek vedoucím práce na porovnání výkonosti serverů Radegast a Perun.

Server Radegast je osazen modernějším šestnáctijádrovým (třicetidvoukláknovým) procesorem EPYC 7313 od konkurenční značky AMD. Procesor za cenu skoro dvojnásobného TDP nabízí základní takt 3.0GHz a v turbo režimu dokonce 3.7GHz. Velikost operační paměti zde činí 66GB, což je čtyřnásobný nárůst oproti serveru Perun. Server dále disponuje dvěma síťovými kartami. Pomalejší karta disponuje přenosovou rychlostí 1Gbit/s a druhá rychlosťí 10Gbit/s.

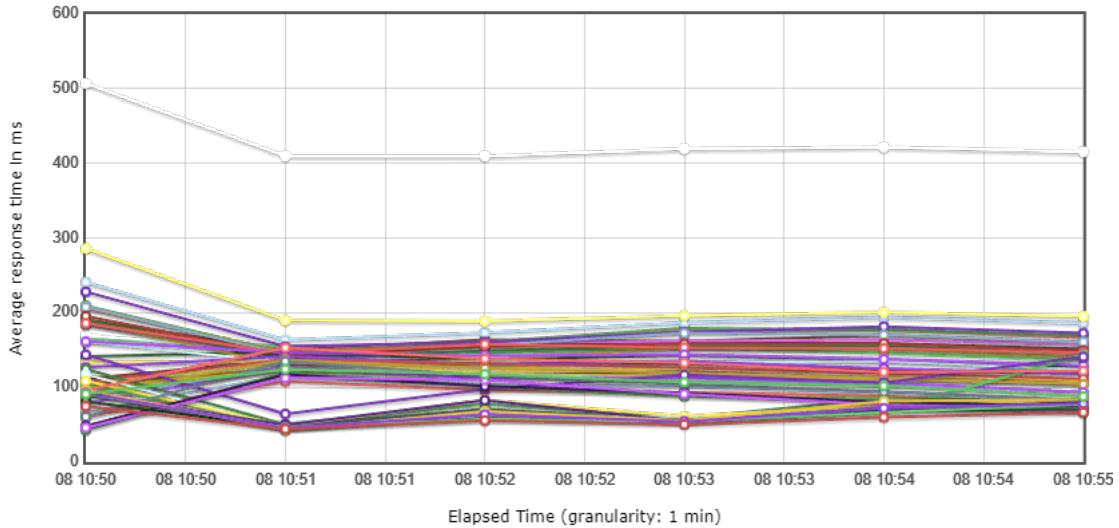
V průběhu testování bylo kontrolováno zatížení serveru, které oscilovalo okolo 40 %, což je výrazné zlepšení oproti serveru Perun. Zatížení serveru bylo kontrolováno pomocí programu Htop a podle vytížení jader aplikace rovnoměrně distribuuje úlohy.



Obrázek 10.7: Zatížení serveru Radegast v průběhu testu

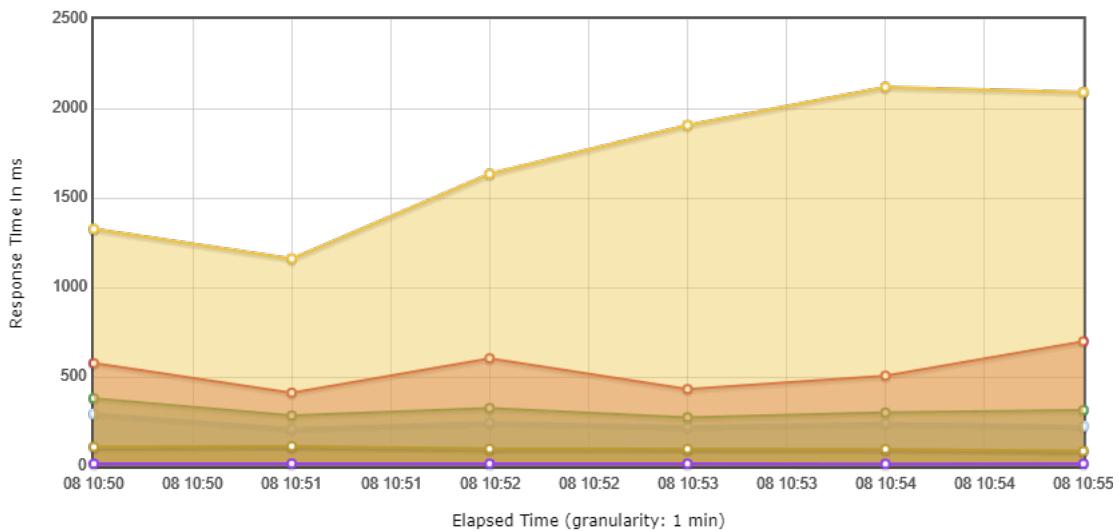
Test i jeho parametry jsou shodné s testováním serveru Perun. Test trval kratší dobu a v průběhu se vyskytlo několik chyb. Všechny chyby vznikly na proxy serveru, takže s velkou pravděpodobností archivní web nevrátil včas odpověď.

Jak je vidět na grafu 10.8, tak průměrná doba odezvy se pohybovala okolo 150 ms, což je výrazné zlepšení oproti předchozímu serveru.



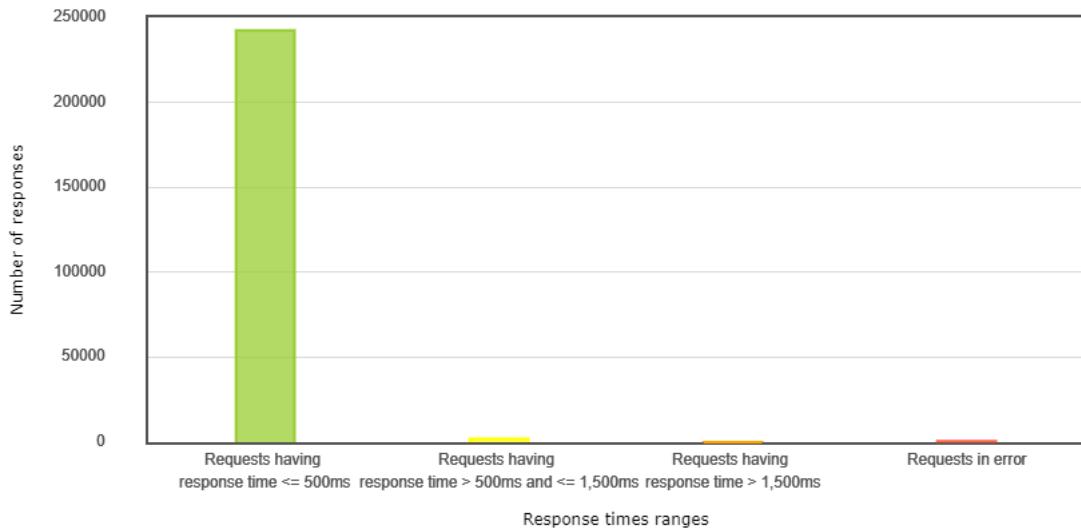
Obrázek 10.8: Průměrná doba odpovědi

Na grafu 10.9 jsou vidět jednotlivé percentily, kde oranžová horní spojnice značí maximum a fialová spojnice medián. Zajímavější údaj než maximum je zelená spojnice značící percentil 99. Žlutá spojnice dále reprezentuje percentil 95 a dolní červená spojnice percentil 90. To znamená, že již u percentilu 99 jsou dotazy okolo 500 ms, což je hluboko pod hranicí 2 vteřin.



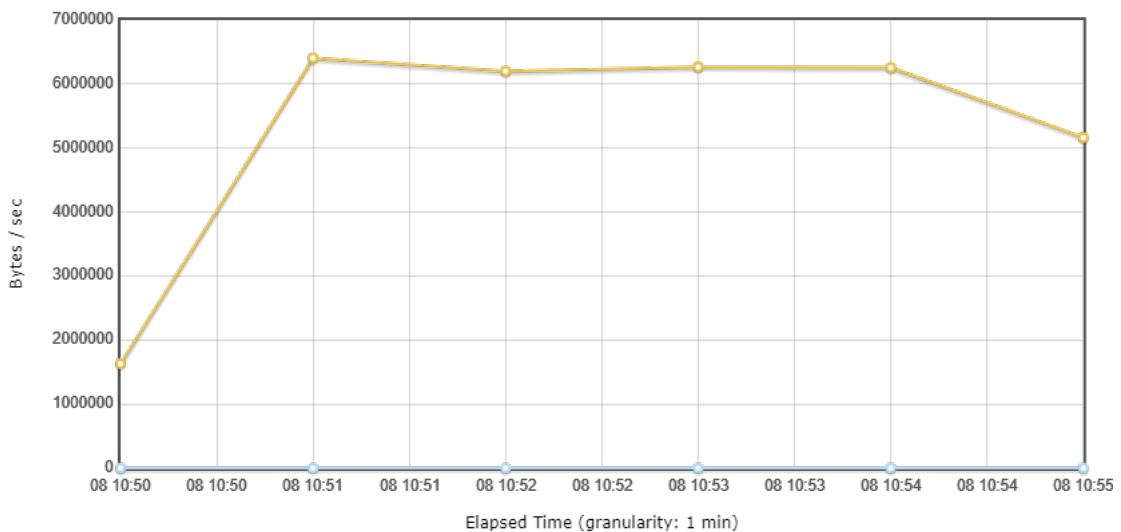
Obrázek 10.9: Percentil průměrné doby odpovědi

Zde je porovnání odezvy rozděleno do tří kategorií, přičemž dle zvoleného kritéria jsou přípustné zelené a žluté hodnoty.



Obrázek 10.10: Přehled latence

V průběhu testu bylo odesláno ze serveru okolo 6MB dat každou vteřinu.



Obrázek 10.11: Datová propustnost

### 10.3.3 Výsledky porovnání

Ze zmíněných grafů je patrné, že server Radegast je výrazně rychlejší než server Perun, což je očekávatelné vzhledem k novějšímu hardwaru, kde například procesor je o osm let mladší než u serveru Perun. Průměrná doba odezvy spadla z 500 ms na 150 ms, což se kladně projeví při simultánním přístupu více uživatelů.

## **10.4 Testování odbornou veřejností**

Systém byl po nasazení prezentován odborné veřejnosti skrze soukromou skupinu Genealogie CZ+SK na sociální síti Facebook. Příspěvek nasbíral přes 80 reakcí a 60 komentářů. Mnoho komentářů se věnovalo problematice rozšíření datových sad aplikace. Několik uživatelů se podělilo o uživatelskou zkušenosť s nově vyvýjeným systémem. Došlo přibližně k patnácti úpravám v oblasti uživatelské zkušenosťi na základě zpětné vazby od uživatelů.

# Kapitola 11

## Závěr

Cílem této práce bylo vytvořit webový systém pro zobrazování digitalizovaných archiválií z celé České republiky. Před samotnou implementací bylo nutné analyzovat stávající systémy a pochopit souvislosti v řešené doméně. U analýzy stávajících systémů byla snaha identifikovat jejich silné, ale i stinné stránky. Systém vychází ze stávajících archivních systémů, aby se badatelé nemuseli seznamovat s novým uživatelským rozhraním. Zároveň byla snaha vyhnout se chybám, které byly v aktuálních systémech nalezeny ve fázi analýzy stávajícího řešení. Z analýzy dosavadních systému vzešla myšlenková mapa s funkcionalitami, jež byly následně implementovány.

Před samotným návrhem byla provedena kontrola výstupů u čtyř scrapovacích systémů, které psali předešlí absolventi Fakulty informačních technologií VUT v Brně. Výstupy dvou systémů byly spárovány s RÚIAN identifikátory a jeden obsahoval adresy na naskenované folia archiválií. Bohužel bylo zjištěno, že rozhraní některých archivů se změnilo a odkazy na folia nebyly použitelné. Porovnání těchto scrapovacích systémů pomohlo s pochopením jednotlivých entit a vztahů mezi nimi. Z porovnání jednotlivých scrapovacích systémů byl vybrán a integrován systém od Jana Valuška. Scrapovací systém byl rozšířen o vyhledávání obcí včetně souřadnic a odesílání dat na serverovou část aplikace.

Tvorba detailního návrhu splnila svůj účel a fáze implementace byla přímočará, a to díky zpracovanému návrhu.

Aktuálně je systém nasazen v rámci školní infrastruktury a je veřejně dostupný. V systému se aktuálně nachází archivní materiál z osmi archivů. Mezi archivy patří Zemský archiv v Opavě, Státní oblastní archiv v Praze, Moravský zemský archiv, Státní oblastní archiv v Plzni, Archiv hlavního města Prahy, Státní oblastní archiv v Litoměřicích, Státní oblastní archiv v Třeboni a Státní oblastní archiv v Hradci Králové. V systému je aktuálně přes 149 000 archivních záznamů, mezi nimiž může uživatel vyhledávat. Jedná se o lánové rejstříky, matriky, rektifikační akta, pozemkové knihy, sčítací operáty a urbáře. Bohužel berní rula a soupis poddaných dle víry doposud vyšly pouze knižně, takže v systému nejsou zahrnuty. Kromě státního oblastního archivu v Třeboni bylo pro výše zmíněné archivy zprovozněno prohlížení digitalizovaných skenů archiválií.

Vývoj systému doprovázelo několik problémů, které bylo potřeba řešit. V rané fázi vývoje byl kladen požadavek na použití čistě relační databáze. Byla snaha tvořit indexy ručně v relační databázi a následně podle nich vyhledávat. Vlastní implementace indexování v rámci relační databáze byla asi  $10\times$  pomalejší při benchmarku než při vyhledávání v ElasticSearch. Dále je potřeba zmínit, že vlastní implementace indexování nebyla technicky tak propracovaná jako osvědčené řešení nabízené databází ElasticSearch.

Samotné scrapování dat a dotazování na externí úložiště dat vytvořilo v průběhu implementace mnoho výzev. Funkční zobrazování naskenovaných folií z externích zdrojů vyžaduje vlastní proxy server a alternativní konfigurace pro jednotlivé formáty v rámci OpenSeaDragon knihovny.

Součástí scrapování dat je doplnění zeměpisných souřadnic pro lokality. Při získávání zeměpisných souřadnic se vyskytlo více problémů a tato funkcionality zahrnuje použití geolokačního API a shlukovacího algoritmu. Prvním zdrojem nepřesnosti jsou samotná data archivů, kde je často v rámci názvu obce přiložen komentář. Tento problém je částečně řešen filtrováním frekventovaných klíčových slov. Druhým zdrojem nepřesnosti je geolokační API, jelikož OpenStreetMap ve svých podmínkách zakazuje přidávání již zaniklých obcí a použití jiných geolokačních API je zpoplatněno. Posledním zdrojem potencionální nepřesnosti je samotný shlukovací algoritmus. V případě, že archiválie obsahuje více obcí, tak výsledky jsou korektní. Nejhorší případ pro navržený algoritmus je takový, kdy archiválie obsahuje pouze jednu obec a existuje více stejnojmenných obcí v rámci stejného okresu. V tomto případě algoritmus zvolí obec, která je vzdáleností blíže archivnímu městu.

Tvorba uživatelských poznámek vyžadovala před samotný prohlížeč archiválií vykreslit plátno, které zvládá kreslení, interakci s prvky a jejich přemisťování. Pro implementaci této funkcionality byla zvolena knihovna Fabric.JS. Problém nastal při integraci s OpenSeaDragon, kde bylo nutné přepsat stávající knihovnu a publikovat vlastní verzi skrze npm.

Obdobně jako každý softwarový produkt i tento by se dal dále rozširovat. V první řadě by se dalo rozšířit portfolio podporovaných archivů. Dále by bylo vhodné začít ukládat mediální data na lokální server, což by vedlo ke sjednocení formátu a případné další optimalizace. Z důvodu, že systém spoléhá na uložení dat v rámci jednotlivých archivů, tak tyto optimalizace nejsou možné a případná změna v archivním systému povede k nutnému zásahu do příslušného scraperu. Anotování archiválií by se dalo rozšířit o kooperativní režim, kde by uživatelé mohli živě spolupracovat na tvorbě poznámek pro konkrétní folio.

# Literatura

- [1] *1651 / Soupis poddaných podle víry* [online]. Český statistický úřad [cit. 2023-11-11]. Dostupné z: <https://www.statistikaam.cz/pribeh-statistiky/1651-soupis-poddanych-podle-viry/>.
- [2] *Berní rula* [online]. Vodní mlýny [cit. 2024-05-09]. Dostupné z: <https://www.vodnimlyny.cz/no/berni-rula/>.
- [3] *CQRS pattern* [online]. Microsoft [cit. 2024-12-12]. Dostupné z: <https://learn.microsoft.com/en-us/azure/architecture/patterns/cqrs>.
- [4] *CQRS pattern* [online]. Amazon [cit. 2024-12-12]. Dostupné z: <https://docs.aws.amazon.com/prescriptive-guidance/latest/modernization-data-persistence/cqrs-pattern.html>.
- [5] *Cross-Origin Resource Sharing (CORS)* [online]. Mozilla [cit. 2024-12-14]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>.
- [6] *D1 - Lánové rejstříky* [online]. Moravský zemský archiv v Brně [cit. 2024-05-09]. Dostupné z: <http://www.mza.cz/a8web/a8apps1/d1/a8s14dd2bad3D1.htm>.
- [7] *D2 - Rektifikační akta* [online]. Moravský zemský archiv v Brně [cit. 2024-05-09]. Dostupné z: <http://www.mza.cz/a8web/a8apps1/D2/A8SL4DD2Bad3D2.htm>.
- [8] *Deep Zoom File Format Overview* [online]. Microsoft [cit. 2023-11-11]. Dostupné z: [https://learn.microsoft.com/en-us/previous-versions/windows/silverlight/dotnet-windows-silverlight/cc645077\(v=vs.95\)](https://learn.microsoft.com/en-us/previous-versions/windows/silverlight/dotnet-windows-silverlight/cc645077(v=vs.95)).
- [9] *Digitalizace v archivech* [online]. Česká genealogická a heraldická společnost v Praze [cit. 2024-05-09]. Dostupné z: <http://www.genealogie.cz/aktivity/digitalizace/>.
- [10] *Docker overview* [online]. [cit. 2024-04-03]. Dostupné z: <https://docs.docker.com/get-started/overview/>.
- [11] *Fabric.JS Docs* [online]. [cit. 2024-12-14]. Dostupné z: <http://fabricjs.com/docs/>.
- [12] *Getting Started with Redux* [online]. [cit. 2024-12-13]. Dostupné z: <https://redux.js.org/introduction/getting-started>.
- [13] *Hash tables explained* [online]. Your basic [cit. 2024-12-14]. Dostupné z: <https://yourbasic.org/algorithms/hash-tables-explained/>.
- [14] *Haversiniův vzorec* [online]. [cit. 2024-04-04]. Dostupné z: [https://cs.frwiki.wiki/wiki/Formule\\_de\\_haversine](https://cs.frwiki.wiki/wiki/Formule_de_haversine).

- [15] *Hlásková spodoba* [online]. [cit. 2024-04-30]. Dostupné z:  
[https://cs.wikipedia.org/wiki/Hl%C3%A1skov%C3%A1\\_spodoba#:~:text=V%C3%BDrazem%20asimilace%20\(%C4%8Desky%20t%C3%A9m%C5%BE%20spodoba,hl%C3%A1sky%20jin%C3%A9%20h1%C3%A1sce%2C%20v%C4%9Bt%C5%A1inou%20sousedn%C3%AD.](https://cs.wikipedia.org/wiki/Hl%C3%A1skov%C3%A1_spodoba#:~:text=V%C3%BDrazem%20asimilace%20(%C4%8Desky%20t%C3%A9m%C5%BE%20spodoba,hl%C3%A1sky%20jin%C3%A9%20h1%C3%A1sce%2C%20v%C4%9Bt%C5%A1inou%20sousedn%C3%AD.)
- [16] *How It Works* [online]. International Image Interoperability Framework [cit. 2023-11-13]. Dostupné z: <https://iiif.io/get-started/how-iiif-works/>.
- [17] *IBM Database Relational Indexes* [online]. [cit. 2024-12-13]. Dostupné z:  
<https://www.ibm.com/docs/en/db2/11.5?topic=indexes-database-relational>.
- [18] *Introduction to JSON Web Tokens* [online]. JWT [cit. 2024-12-14]. Dostupné z:  
<https://jwt.io/introduction>.
- [19] *OpenSeaDragon Docs* [online]. [cit. 2024-12-14]. Dostupné z:  
<https://openseadragon.github.io/docs/>.
- [20] *React* [online]. [cit. 2024-12-13]. Dostupné z: <https://react.dev/>.
- [21] *Registr územní identifikace, adres a nemovitostí* [online]. Správa základních registrů [cit. 2023-11-23]. Dostupné z:  
<https://www.szrcr.cz/cs/registr-uzemni-identifikace-adres-a-nemovitosti>.
- [22] *Spring Boot* [online]. [cit. 2024-12-13]. Dostupné z:  
<https://spring.io/projects/spring-boot>.
- [23] *TypeScript* [online]. [cit. 2024-12-13]. Dostupné z:  
<https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html>.
- [24] *Urbáře* [online]. Vodní mlýny [cit. 2024-05-09]. Dostupné z:  
<https://www.vodnimlyny.cz/no/urbare/>.
- [25] *What is a relational database?* [online]. IBM [cit. 2024-12-13]. Dostupné z:  
<https://www.ibm.com/topics/relational-databases>.
- [26] *What is a Relational Database (RDBMS)?* [online]. Oracle [cit. 2024-12-13]. Dostupné z: <https://www.oracle.com/cz/database/what-is-a-relational-database>.
- [27] *What is Elasticsearch?* [online]. ElasticSearch [cit. 2024-12-14]. Dostupné z:  
<https://www.elastic.co/guide/en/elasticsearch/reference/current/elasticsearch-intro.html>.
- [28] *What is Full-Text Search and How Does it Work?* [online]. MongoDB [cit. 2024-12-14]. Dostupné z: <https://www.mongodb.com/basics/full-text-search/>.
- [29] *Za bohatstvím našich předků* [online]. Hledání předků [cit. 2024-05-09]. Dostupné z:  
<https://www.hledanipredku.cz/gruntovni-pozemkove-knihy/>.
- [30] BRUMBY, J. *World report on vision*. Robert “Bob” E. Corlew, Martin Dinham et al. World Health Organization, 2019. Dostupné z:  
<https://www.who.int/publications/i/item/9789241516570>.

- [31] DYTRYCH, J. *Testing, Maven and JAX* [Slidy magisterského kurzu]. Grafická uživatelská rozhraní v Javě, Fakulta informačních technologií Vysokého učení technického v Brně, Slide 3-19. Dostupné z: [https://moodle.vut.cz/pluginfile.php/709326/mod\\_folder/content/0/GJA\\_02-Testing\\_Maven\\_JAX.pdf](https://moodle.vut.cz/pluginfile.php/709326/mod_folder/content/0/GJA_02-Testing_Maven_JAX.pdf).
- [32] EASTERBROOK, S. *Class Diagrams and Entity Relationship Diagrams (ERD)* [Slidy kurzu]. [cit. 2024-04-30]. Requirements Engineering, University of Toronto. Dostupné z: [https://www.cs.toronto.edu/~sme/CSC340F/2005/slides/tutorial-classes\\_ERDs.pdf](https://www.cs.toronto.edu/~sme/CSC340F/2005/slides/tutorial-classes_ERDs.pdf).
- [33] FARILLO, J. *The Basics of Database Indexes For Relational Databases* [online]. [cit. 2024-12-13]. Dostupné z: <https://medium.com/@jimmyfarillo/the-basics-of-database-indexes-for-relational-databases-bfc634d6bb37>.
- [34] GRAHAM, J. *Full Text Search Explained* [online]. [cit. 2024-12-14]. Dostupné z: <https://www.joshgraham.com/full-text-search-explained/>.
- [35] GULDEN, M. *Introduction to Hibernate Search* [online]. [cit. 2024-12-14]. Dostupné z: <https://www.baeldung.com/hibernate-search>.
- [36] JWEL, D. *What is figma design and how important is it?* [online]. [cit. 2024-12-14]. Dostupné z: <https://www.linkedin.com/pulse/what-figma-design-how-important-designer-jwel-fm77c/>.
- [37] LEDNICKÁ, B. *Sestavte si rodokmen, pátráme po svých předcích*. 1. vyd. Grada, 2012. ISBN 978-80-247-4069-0.
- [38] MALÍK, O. *Web pro zobrazování archiválií* [online]. Brno, 2024. [cit. 2024-04-30]. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií, Ústav inteligentních systémů. Vedoucí práce Jaroslav Rozman. Dostupné z: <https://www.vut.cz/studenti/zav-prace/detail/156406>.
- [39] NESLÁDKOVÁ, L. *Sčítací operáty z moderního sčítání obyvatel*. [cit. 2024-05-09]. Dostupné z: [https://is.muni.cz/el/phil/jaro2020/ETBB111/99696730/Scitaci\\_operaty.pdf?kod=PV084;lang=en](https://is.muni.cz/el/phil/jaro2020/ETBB111/99696730/Scitaci_operaty.pdf?kod=PV084;lang=en).
- [40] POP, D. *Podpora vyhledávání matričních událostí*. Brno, CZ, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: <https://www.vut.cz/studenti/zav-prace/detail/146877>.
- [41] RIMONTAWADROUS. *Introduction To Database Indexing* [online]. [cit. 2024-12-13]. Dostupné z: <https://medium.com/@rtawadrous/introduction-to-database-indexes-9b488e243cc1>.
- [42] RYCHLÝ, M. *The Principles of Object-Oriented Design* [Slidy magisterského kurzu]. Analýza a návrh informačních systémů, Fakulta informačních technologií Vysokého učení technického v Brně, Slide 7. Dostupné z: [https://rychly-edu.gitlab.io/lectures/ais/AIS10-oop-principles\\_print.pdf](https://rychly-edu.gitlab.io/lectures/ais/AIS10-oop-principles_print.pdf).
- [43] RYCHLÝ, M. *Unified Modelling Language (UML)* [Slidy magisterského kurzu]. Analýza a návrh informačních systémů, Fakulta informačních technologií Vysokého učení technického v Brně, Slide 21. Dostupné z: [https://rychly-edu.gitlab.io/lectures/ais/AIS03-uml\\_print.pdf](https://rychly-edu.gitlab.io/lectures/ais/AIS03-uml_print.pdf).

- [44] RYCHLÝ, M. *Unified Modelling Language (UML)* [Slidy magisterského kurzu]. Analýza a návrh informačních systémů, Fakulta informačních technologií Vysokého učení technického v Brně, Slide 85. Dostupné z:  
[https://rychlly-edu.gitlab.io/lectures/ais/AIS03-uml\\_print.pdf](https://rychlly-edu.gitlab.io/lectures/ais/AIS03-uml_print.pdf).
- [45] RYCHLÝ, M. *Unified Process – The Elaboration Phase* [Slidy magisterského kurzu]. Analýza a návrh informačních systémů, Fakulta informačních technologií Vysokého učení technického v Brně, Slide 49. Dostupné z:  
[https://rychlly-edu.gitlab.io/lectures/ais/AIS06-up-elaboration\\_print.pdf](https://rychlly-edu.gitlab.io/lectures/ais/AIS06-up-elaboration_print.pdf).
- [46] RYCHLÝ, M. *Unified Process – The Elaboration Phase* [Slidy magisterského kurzu]. Analýza a návrh informačních systémů, Fakulta informačních technologií Vysokého učení technického v Brně, Slide 37. Dostupné z:  
[https://rychlly-edu.gitlab.io/lectures/ais/AIS06-up-elaboration\\_print.pdf](https://rychlly-edu.gitlab.io/lectures/ais/AIS06-up-elaboration_print.pdf).
- [47] SMITH, A. Introducing Zoomify Image. *Information Technology and Libraries* [online]. Březen 2007, sv. 26, s. 48–51, [cit. 2023-11-13]. DOI: 10.6017/ital.v26i1.3288. Dostupné z:  
[https://www.researchgate.net/publication/293184550\\_Introducing\\_Zoomify\\_Image](https://www.researchgate.net/publication/293184550_Introducing_Zoomify_Image).
- [48] SMITH, S. ardalis. *Architecting Modern Web Applications with ASP.NET Core and Microsoft Azure*. V7.0. Microsoft, 2023 [cit. 2023-11-23]. Dostupné z:  
<https://dotnet.microsoft.com/en-us/download/e-book/aspnet/pdf>.
- [49] SMRČKA, A. *Výkonnostní testování* [Slidy magisterského kurzu]. [cit. 2024-04-15]. Automatizované testování a dynamická analýza, Fakulta informačních technologií Vysokého učení technického v Brně, Slide 5. Dostupné z: [https://moodle.vut.cz/pluginfile.php/567041/mod\\_folder/content/0/8-performance-testing.pdf](https://moodle.vut.cz/pluginfile.php/567041/mod_folder/content/0/8-performance-testing.pdf).
- [50] SOKOLÍK, J. *Web pro přepisování genealogických pramenů*. Brno, CZ, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: <https://www.vut.cz/studenti/zav-prace/detail/146882>.
- [51] VALUŠEK, J. *Aktualizace dat z webových stránek*. Brno, CZ, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z:  
<https://www.vut.cz/studenti/zav-prace/detail/144933>.
- [52] ŠTOURAČOVÁ, J. *Archivnictví*. 1. vyd. Masarykova univerzita, 2013 [cit. 2023-11-10]. ISBN 978-80-210-6512-3. Dostupné z:  
[https://digilib.phil.muni.cz/\\_flysystem/fedora/monography/128576-monography.pdf](https://digilib.phil.muni.cz/_flysystem/fedora/monography/128576-monography.pdf).

## Příloha A

# Výstupy datové analýzy

V této příloze jsou zahrnuty výstupy datové analýzy jako jsou formáty jednotlivých datových sad a grafy, které se snaží zmapovat stav jednotlivých datových sad a vztahy mezi nimi.

### A.1 Formáty datových sad

---

```
1 {
2   "url": string,
3   "puvodce": string,
4   "signatura": string,
5   "invCislo": string,
6   "typ": string,
7   "jazyk": string,
8   "rozsah": string,
9   "invCislo": string,
10  "obsah": string[],
11  "uzemi": {
12    [nazev_obce]: {
13      "typ": decimal,
14      "ruian": decimal,
15      "latitude": float,
16      "longitude": float,
17      "obec": string,
18      "okres": string,
19      "variancy": string[]
20    },
21    ...
22  },
23  "okresy": string[]
24 }
```

---

Výpis A.1: Formát datové sady Dominika Popa

---

```
1 {
2   "id_registra": string,
3   "signature": string,
4   "inv_id": string,
5   "type": string,
6   "scan_count": string,
7   "archive": string,
8   "fond": string,
9   "obec": string,
10  "obec_ruian": string,
11  "cast_obce": string,
12  "cast_obce_ruian": string
13 }
```

---

Výpis A.2: Formát datové sady Jakuba Sokolíka

---

```
1  {
2    "type_of_record": string,
3    "archive": string,
4    "languages": string[],
5    "signature": string,
6    "inventory_number": string,
7    "originator_name": string,
8    "originator_type": string,
9    "originator_note": string,
10   "signature": string,
11   "y_deceased_from": number,
12   "y_deceased_to": number,
13   "covered_area": [
14     {
15       "country": string,
16       "region": string,
17       "district": string,
18       "municipality": string,
19       "borough": string,
20       "german_name": string,
21       "alternative_names": string[],
22     }
23   ],
24   "number_of_scans": number,
25   "register_note": string,
26   "link": string,
27   "content": string,
28   "other_note": string,
29   "y_born_from": string,
30   "y_born_to": string,
31   "y_index_born_from": string,
32   "y_index_born_to": string,
33   "y_married_from": string,
34   "y_married_to": string,
35   "y_index_married_from": string,
36   "y_index_married_to": string,
37   "y_deceased_from": string,
38   "y_deceased_to": string,
39   "y_index_deceased_from": string,
40   "y_index_deceased_to": string,
41   "content": string,
42   "description": string,
43   "fund": string,
44   "index_only": string,
45   "judicial_district": string,
46   "land_registry_nrs": string,
47   "nad": string,
48   "name": string,
49   "original_name": string,
50   "persons_counted": string,
51   "record_method": string,
52   "register_note": string,
53   "specific_type": string,
```

```
54  "y_from": string,  
55  "y_taken": string,  
56  "y_to": string,  
57 }
```

---

Výpis A.3: Formát datové sady Jana Valuška

---

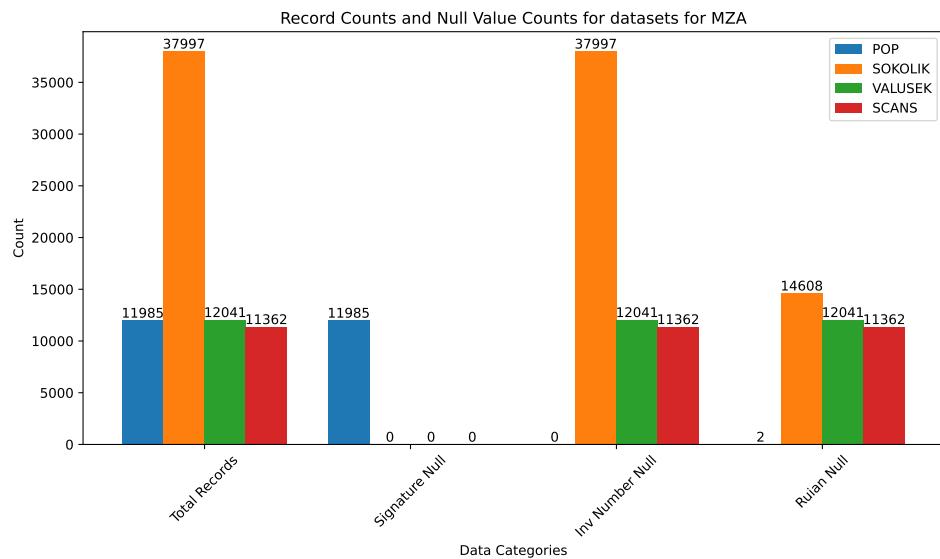
```
1  {
2      "id": string,
3      "signatura": string,
4      "inv. cislo": string,
5      "puvodce": string,
6      "misto_ulozeni": string,
7      "obsah": {
8          "Narození": {
9              "od": string,
10             "do": string
11         },
12         "INDEX Narozených": {
13             "od": string,
14             "do": string
15         },
16         "Oddání": {
17             "od": string,
18             "do": string
19         },
20         "INDEX Oddaných": {
21             "od": string,
22             "do": string
23         },
24         "Zemřelí": {
25             "od": string,
26             "do": string
27         },
28         "INDEX Zemřelých": {
29             "od": string,
30             "do": string
31         },
32         "poznamka":string
33     },
34     "obce_seznam": string[], //In some files of dataset just string
35     "obce": [{ //In some files of dataset just string[] or string[][][]
36         "od": string,
37         "do": string,
38         "umistení": {
39             "stat": string,
40             "kraj": string,
41             "okres": string,
42             "obec": string,
43             "cast_obce": string,
44             "samota": string
45         }
46     }],
47     "jazyky": string[],
48     "snimky": {
49         "jmeno": string,
50         "url": string[]
51     }
52 }
```

---

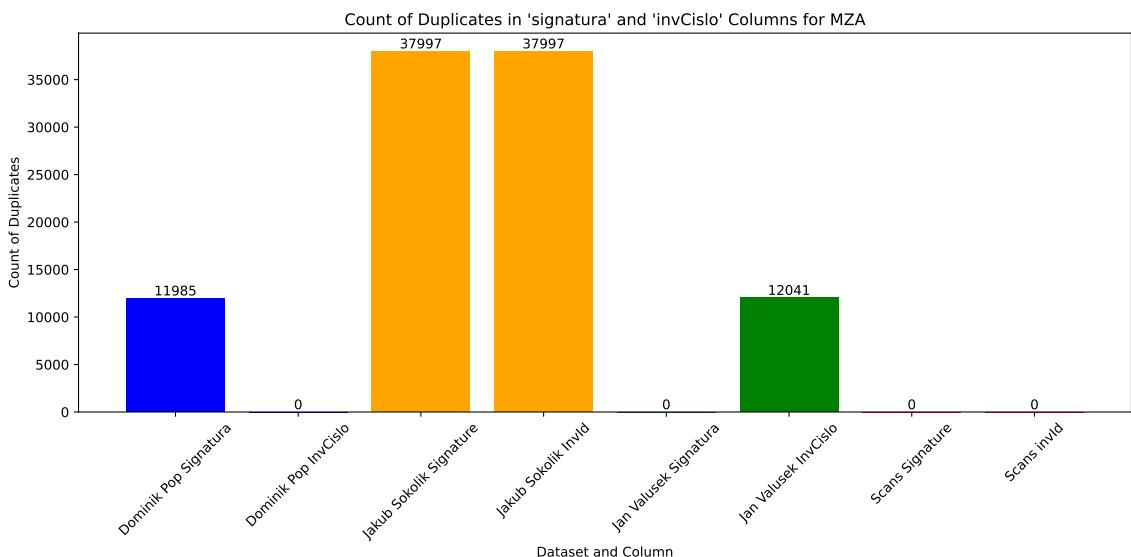
Výpis A.4: Formát datové sady se skeny

## A.2 Grafy pro jednotlivé archivy

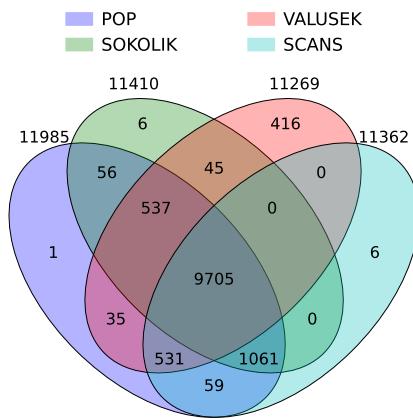
### Moravský zemský archiv



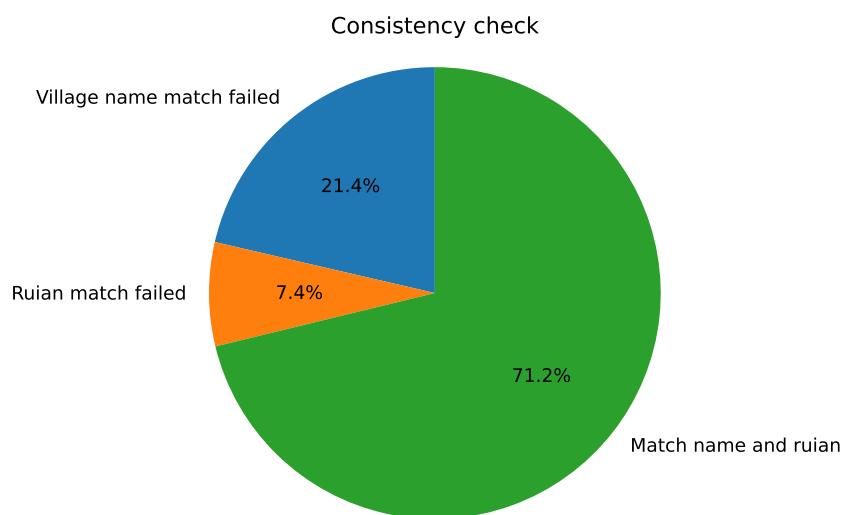
Obrázek A.1: Analýza chybějících hodnot pro MZA



Obrázek A.2: Analýza duplicitních hodnot pro MZA

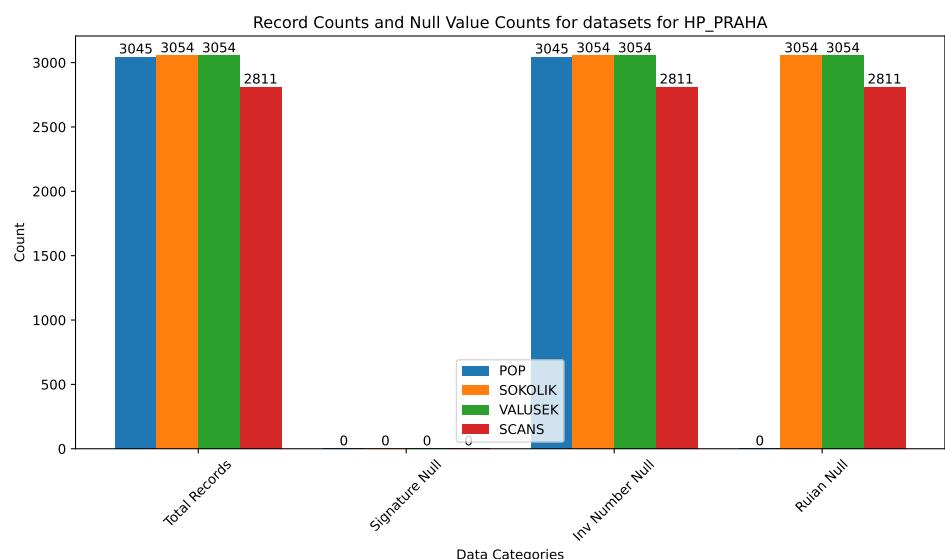


Obrázek A.3: Vennův diagram pro MZA

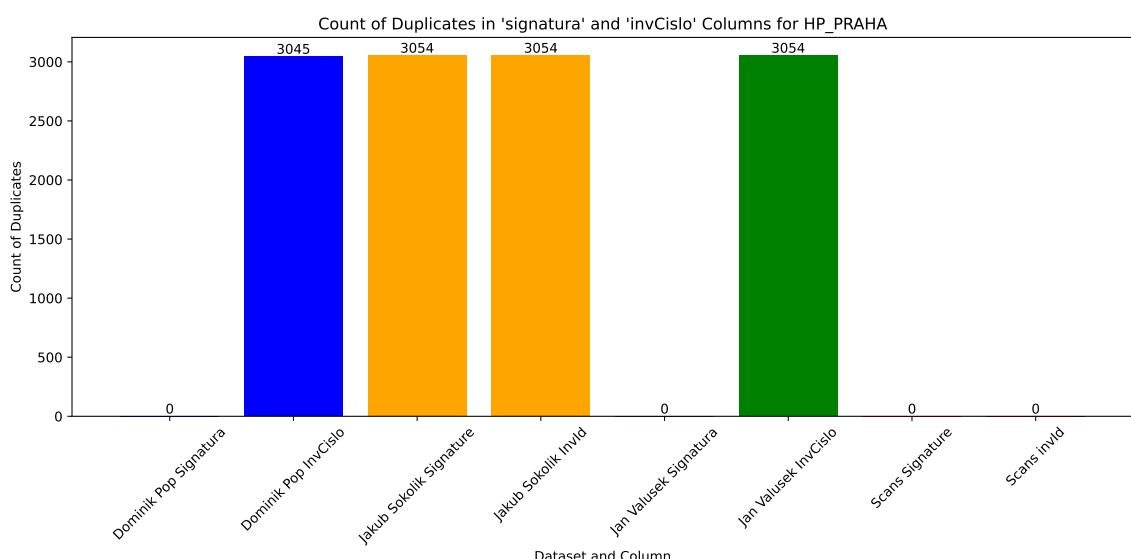


Obrázek A.4: Kontrola konzistence pro MZA

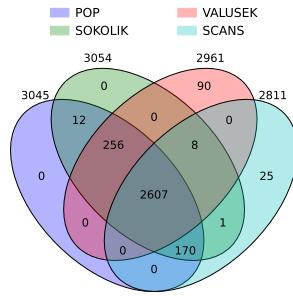
## Archiv hlavní města Praha



Obrázek A.5: Analýza chybějících hodnot pro Národní archiv

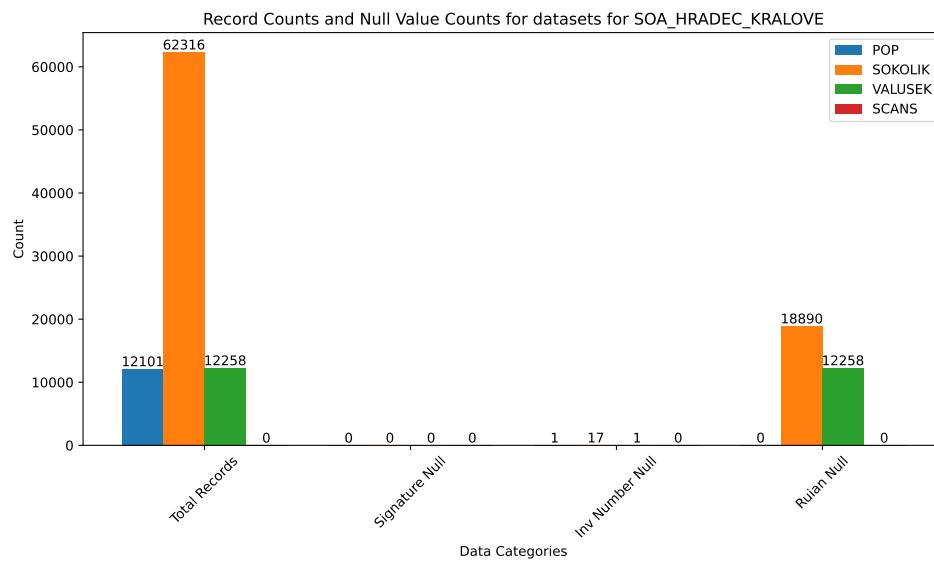


Obrázek A.6: Analýza duplicitních hodnot pro Národní archiv

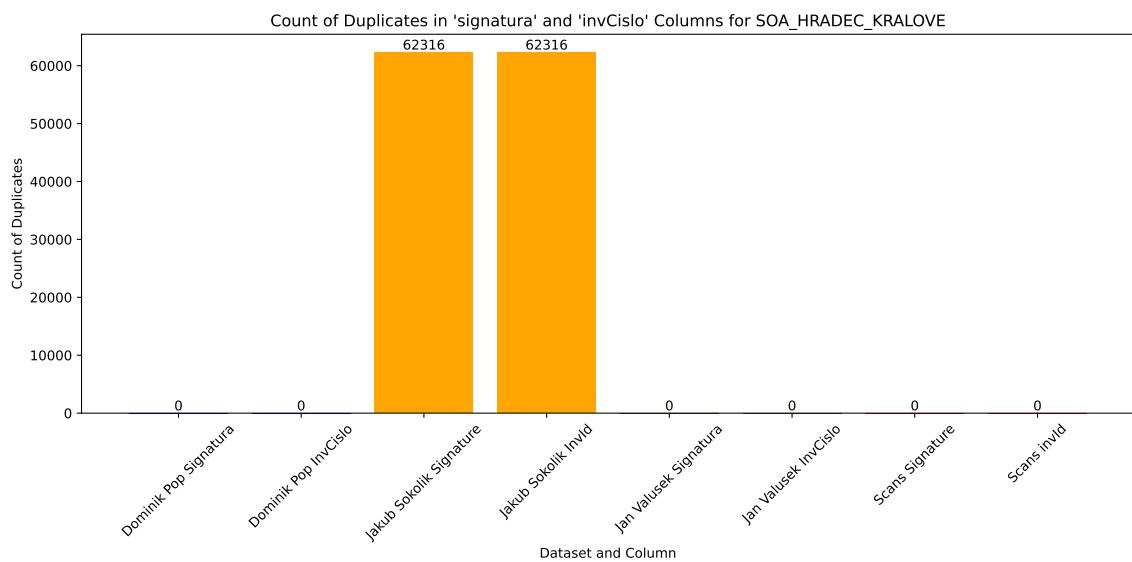


Obrázek A.7: Kontrola konzistence pro Národní archiv

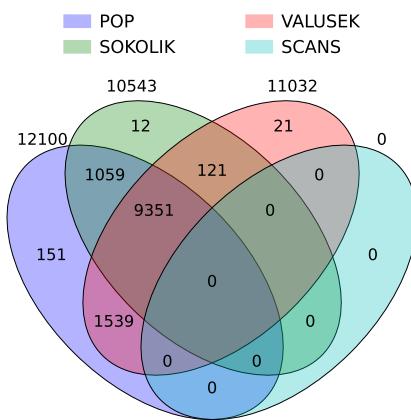
### Státní oblastní archiv v Hradci Králové



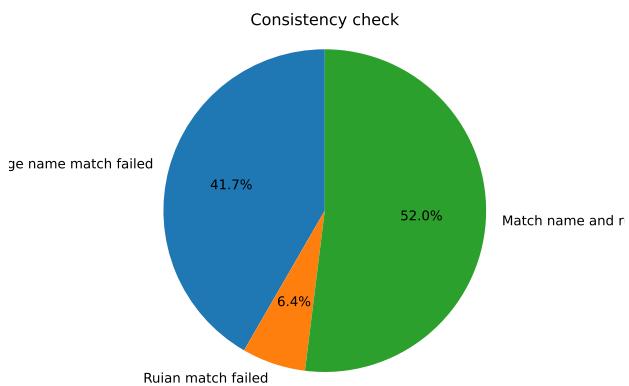
Obrázek A.8: Analýza chybějících hodnot pro SOA v Hradci Králové



Obrázek A.9: Analýza duplicitních hodnot pro SOA v Hradci Králové

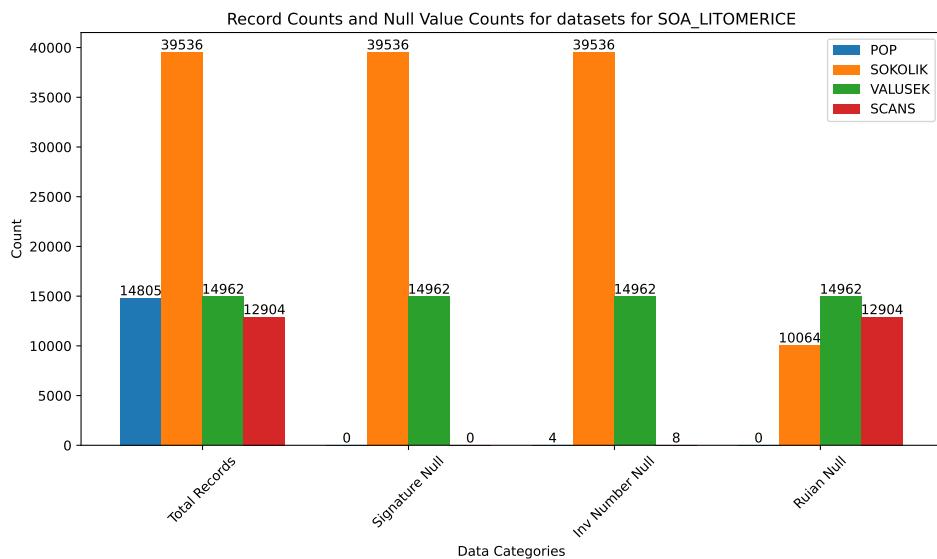


Obrázek A.10: Vennuv diagram pro SOA v Hradci Králové

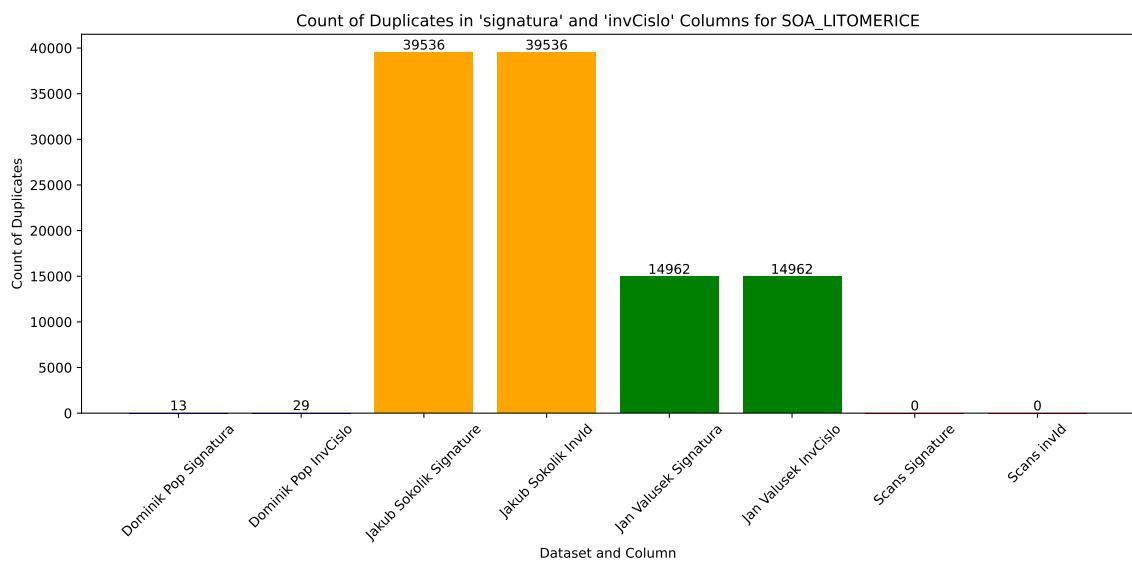


Obrázek A.11: Kontrola konzistence pro SOA v Hradci Králové

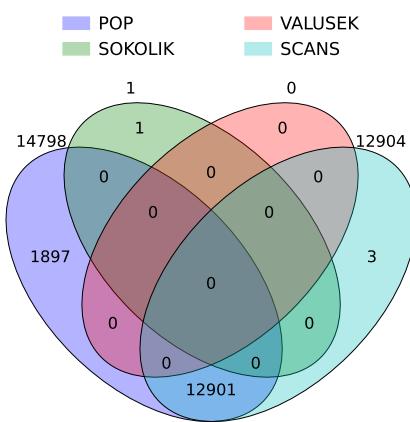
### Státní oblastní archiv v Litoměřicích



Obrázek A.12: Analýza chybějících hodnot pro SOA v Litoměřicích

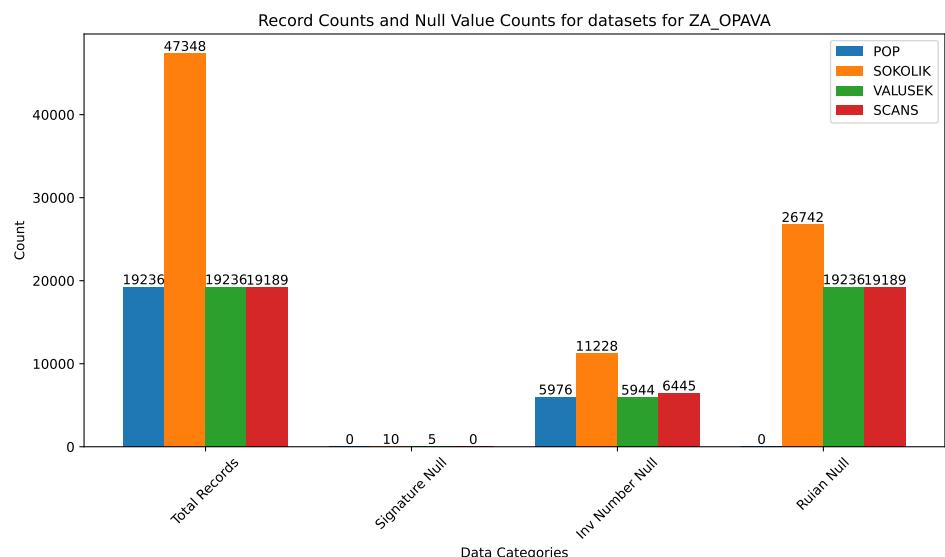


Obrázek A.13: Analýza duplicit pro SOA v Litoměřicích

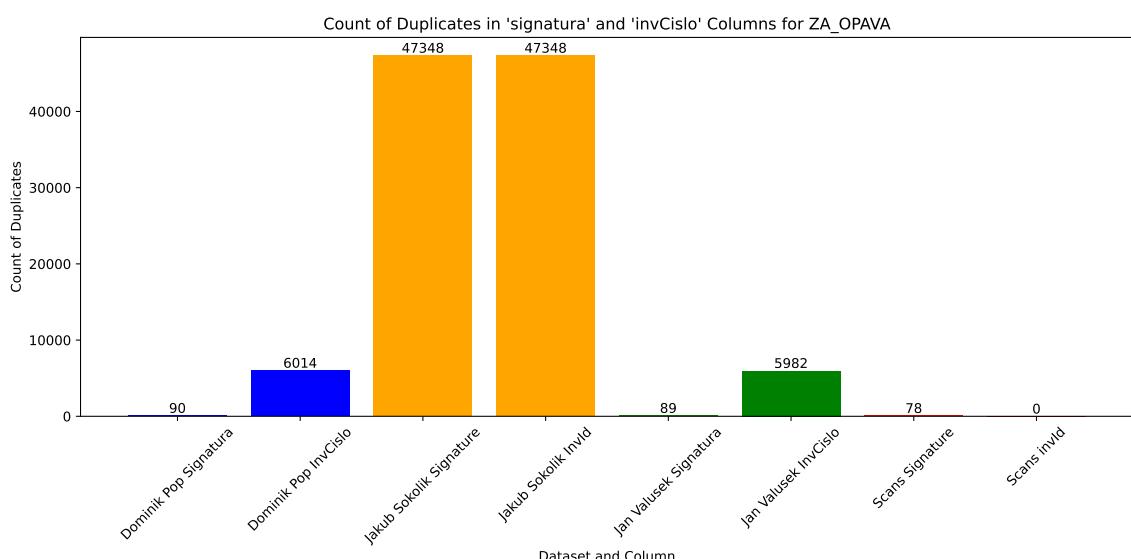


Obrázek A.14: Vennuv diagram pro SOA v Litoměřicích

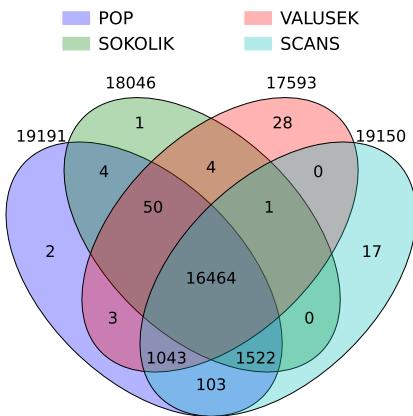
## Zemský archiv v Opavě



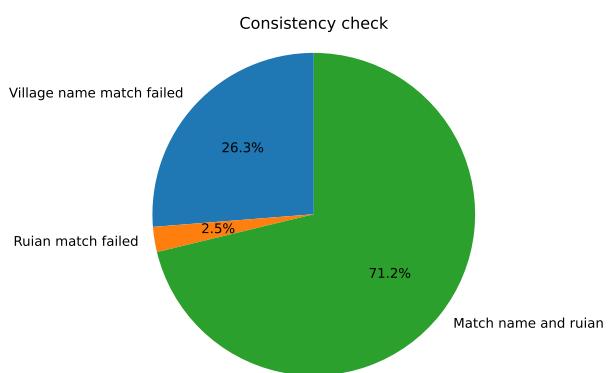
Obrázek A.15: Analýza chybějících hodnot pro ZA v Opavě



Obrázek A.16: Analýza duplicit pro ZA v Opavě

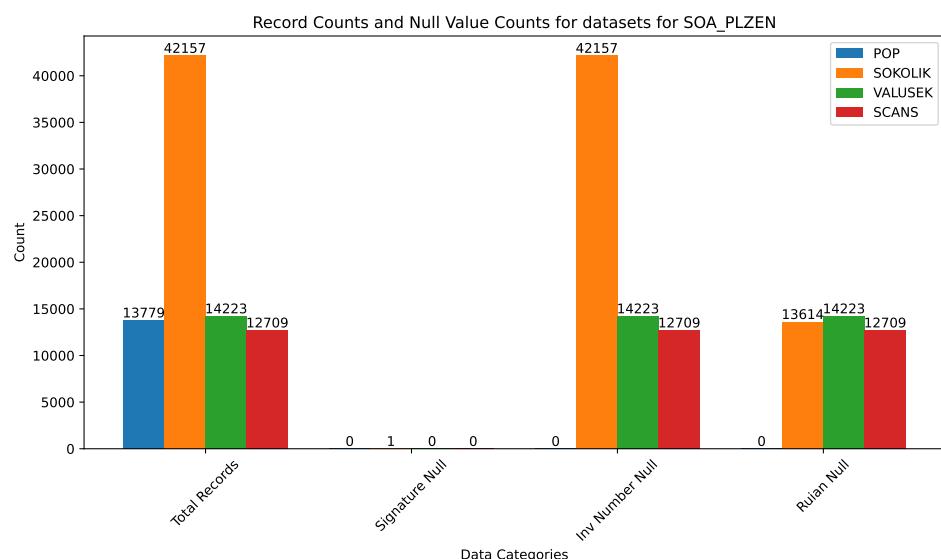


Obrázek A.17: Vennuv diagram pro ZA v Opavě

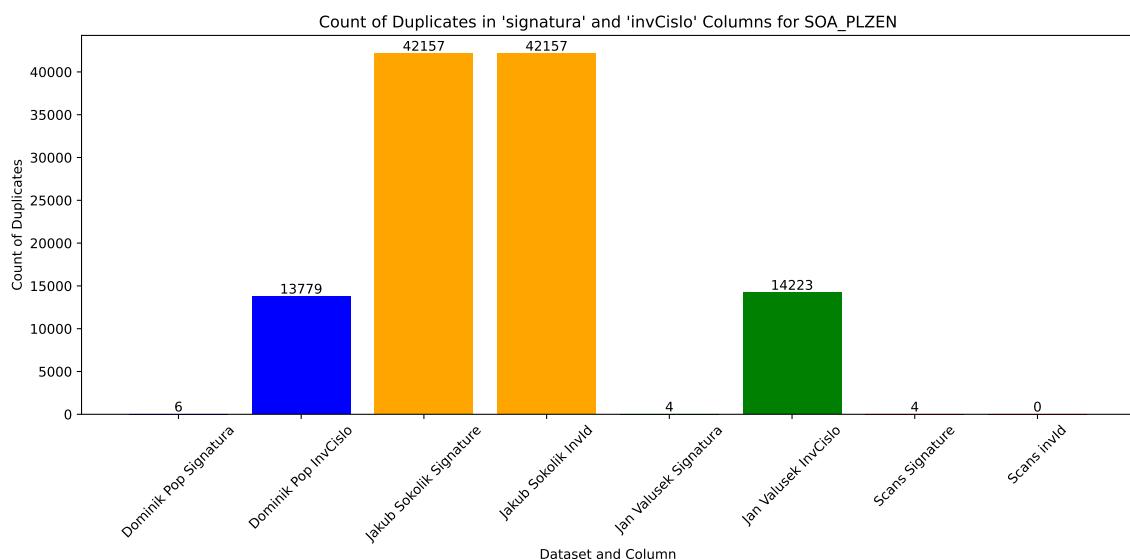


Obrázek A.18: Kontrola konzistence pro ZA v Opavě

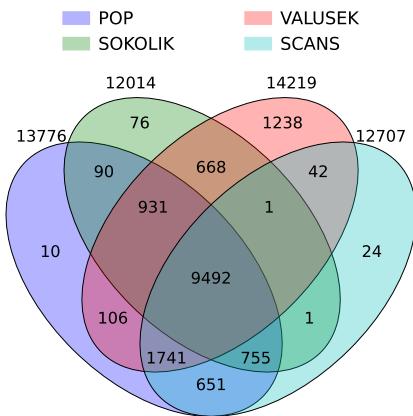
## Státní oblastní archiv v Plzni



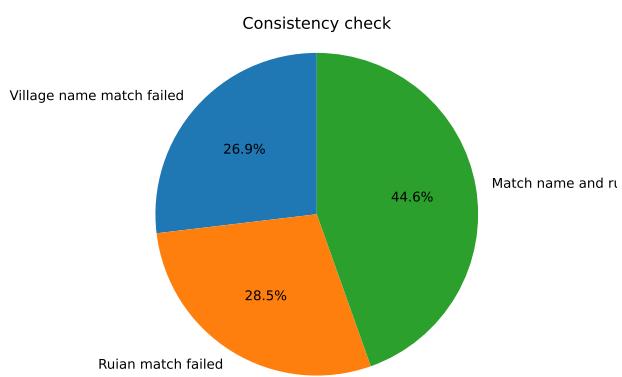
Obrázek A.19: Analýza chybějících hodnot pro SOA v Plzni



Obrázek A.20: Analýza duplicit pro SOA v Plzni

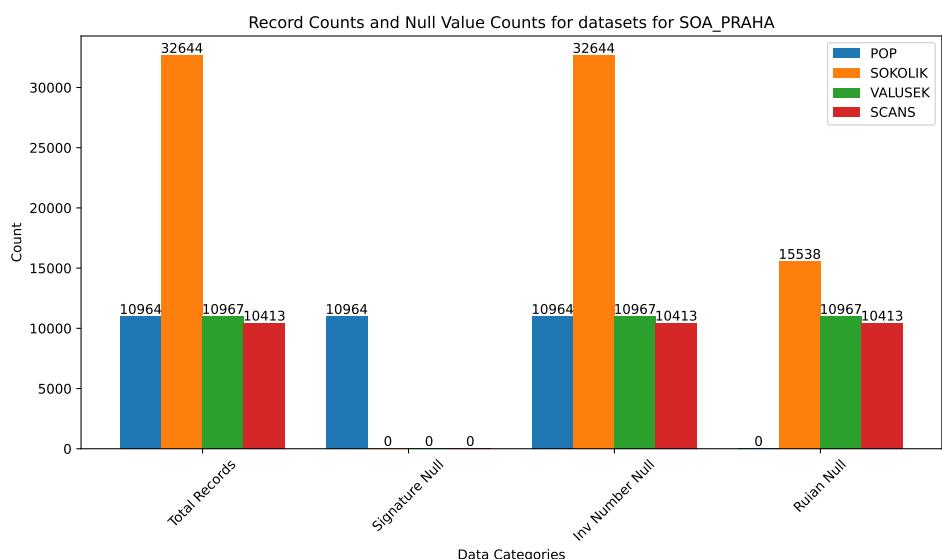


Obrázek A.21: Vennuv diagram pro SOA v Plzni

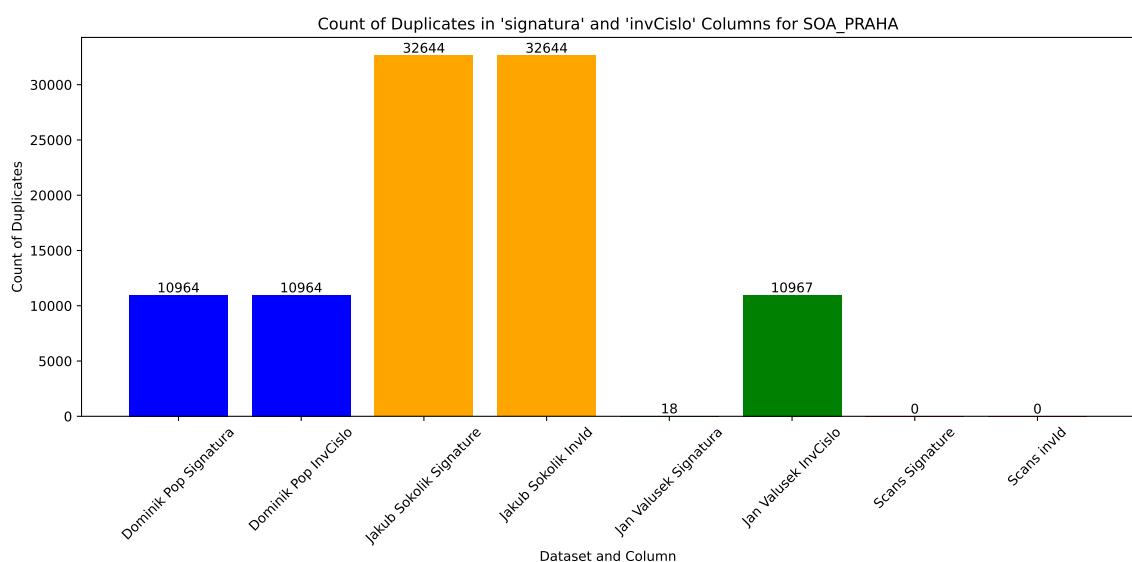


Obrázek A.22: Kontrola konzistence pro SOA v Plzni

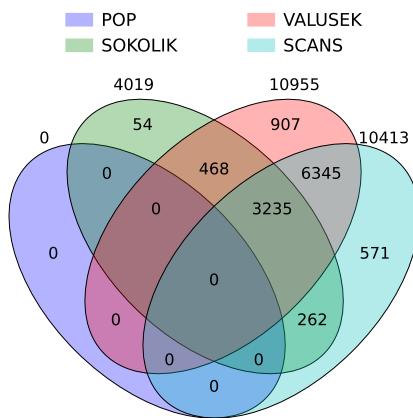
## Státní oblastní archiv v Praze



Obrázek A.23: Analýza chybějících hodnot pro SOA v Praze

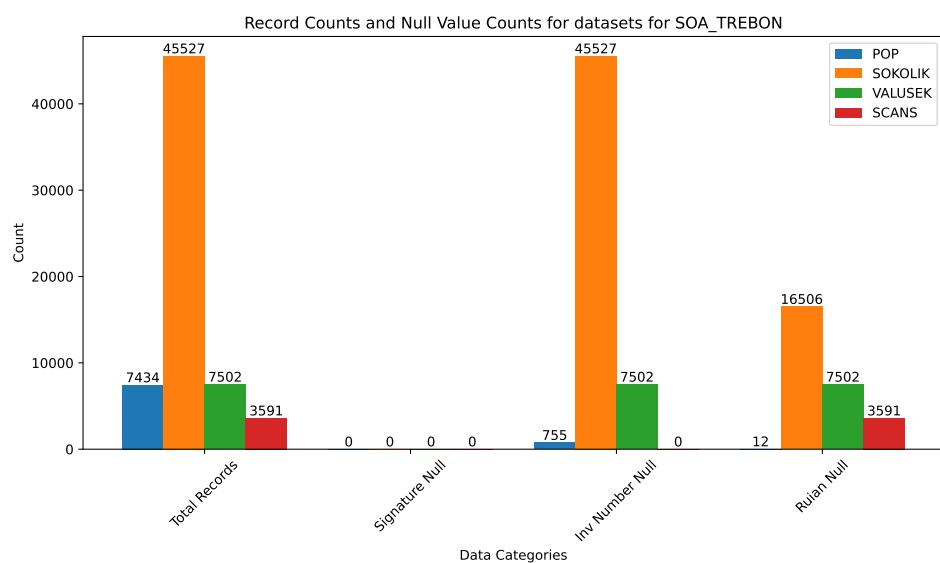


Obrázek A.24: Analýza duplicit pro SOA v Praze

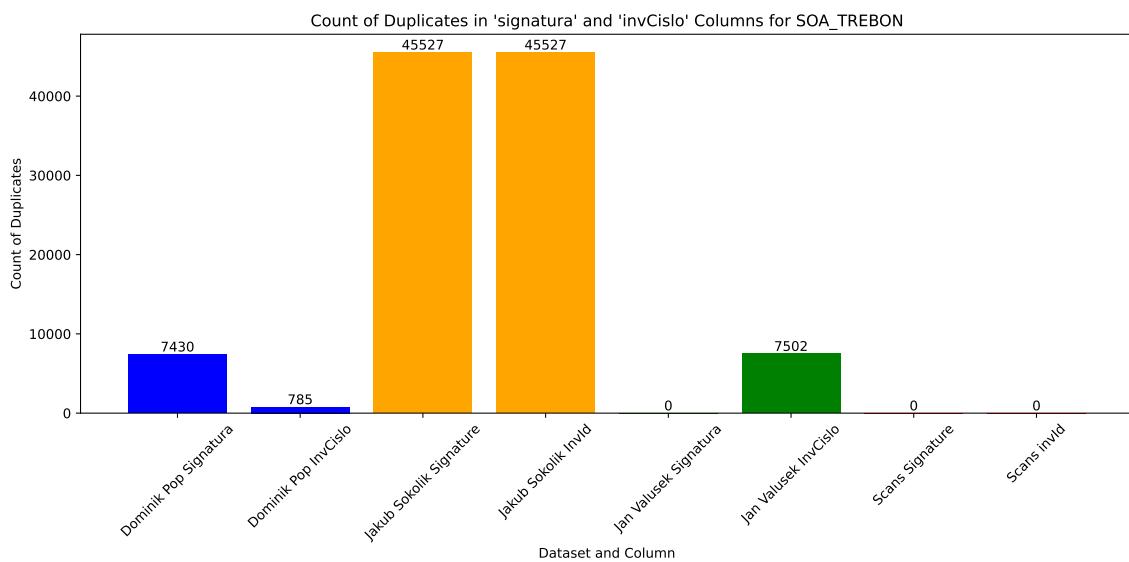


Obrázek A.25: Vennuv diagram pro SOA v Praze

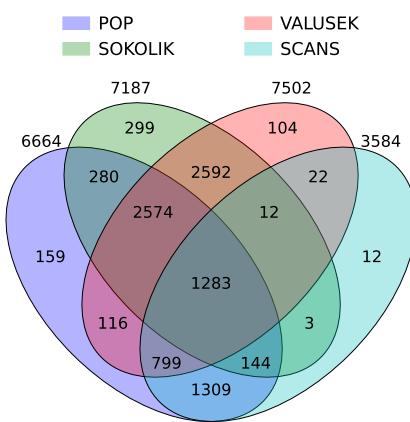
### Státní oblastní archiv v Třeboni



Obrázek A.26: Analýza chybějících hodnot pro SOA v Třeboni



Obrázek A.27: Analýza duplicit pro SOA v Třeboni



Obrázek A.28: Vennuv diagram pro SOA v Třeboni

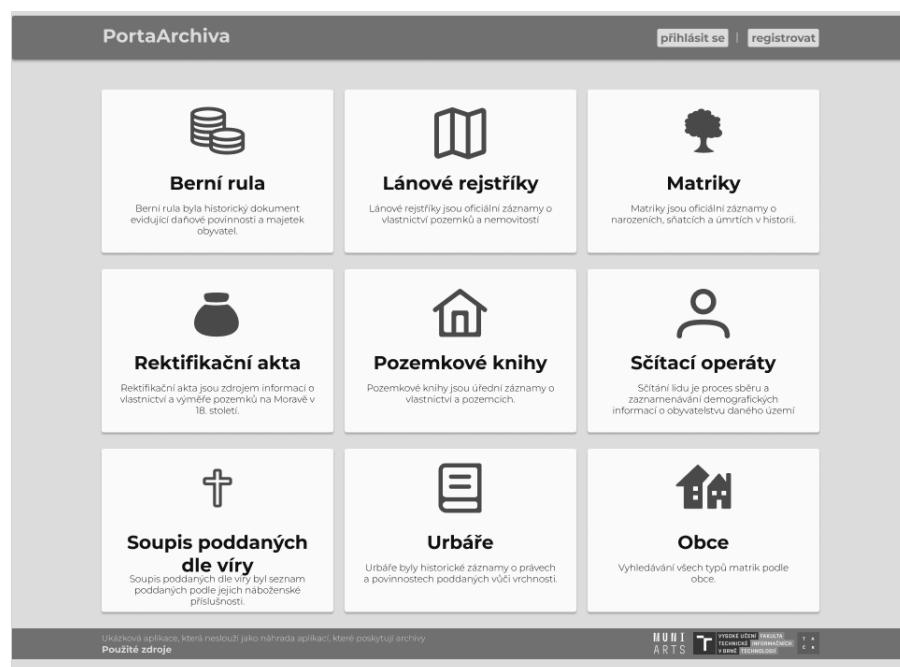
## Příloha B

# Simulace očních vad pohledů prototypu v nástroji Figma

Na světě v současné době s poruchami zraku bojuje minimálně 2,2 miliardy lidí, [30] což není malá skupina a proto je důležité se zaměřit na téma, zda je výsledný produkt použitelný pro lidi, kteří mají problémy s vnímáním barev. Může se stát, že se jednotlivé barvy nebudou mít dostatečný kontrastní poměr a pro člověka s poruchou vnímání barev se tyto barvy mohou jevit jako totožné, což zapříčiní, že daná aplikace pro ně bude nepoužitelná.

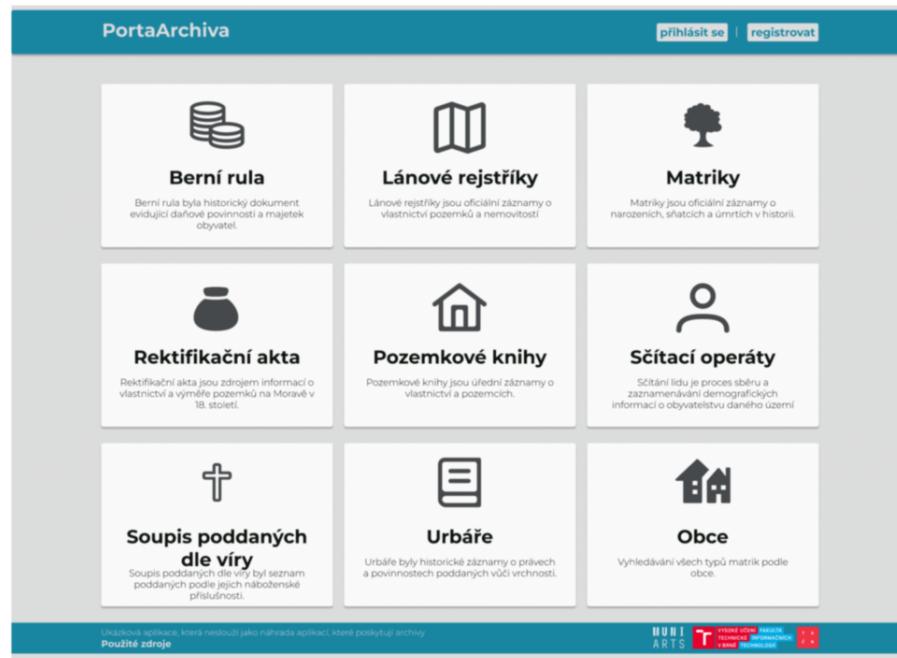
### B.1 Simulace očních vad pro hlavní stranu

Achromatopsie znamená úplnou barvoslepotu, kdy oko nedokáže vnímat žádné barvy, což vede k tomu, že vidění je omezeno pouze na černou a bílou s odstíny šedi.



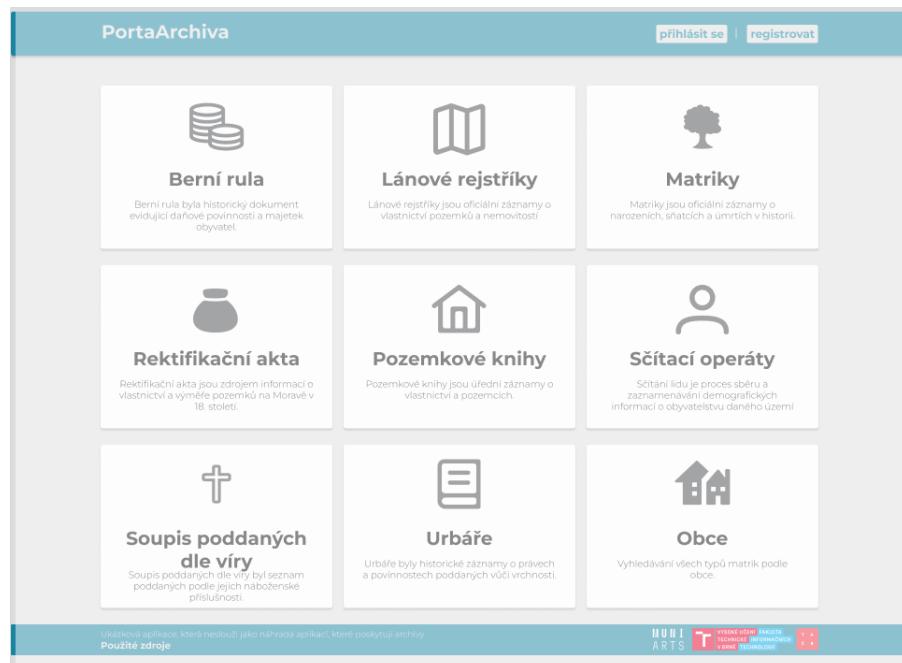
Obrázek B.1: Hlavní strana - simulace achromatopsie

Oční vada, při které se paprsky světla, které procházejí čočkou lidského oka, sbíhají před sítnicí. Tím není na sítnici vytvořen ostrý obraz.



Obrázek B.2: Hlavní strana - simulace krátkozrakosti

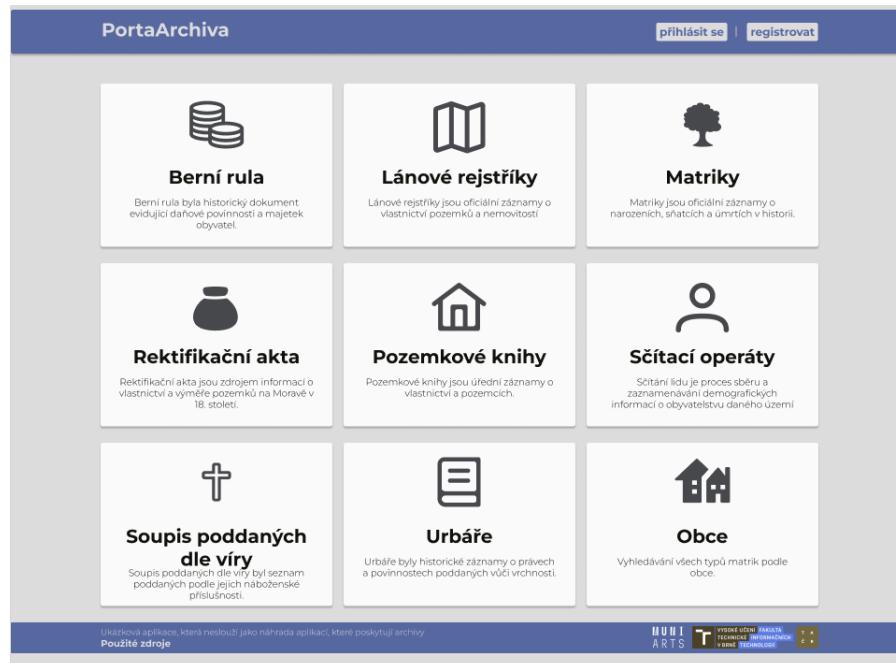
Světloplachost (fotofobie) se projevuje pocitem oslnění u pacientů i při pohledu za běžných světelných podmínek.



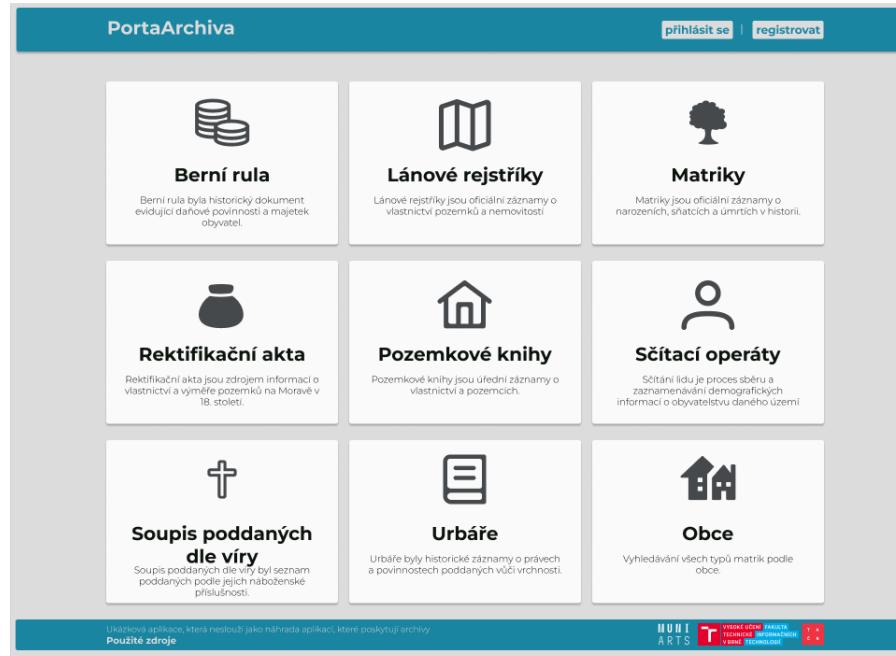
Obrázek B.3: Hlavní strana - simulace fotofobie

Deutanopie je forma poruchy barvocitu, při níž pacient není schopen vnímat zelenou barvu.

Diplopie je porucha, při níž pacient vnímá dva obrazy jednoho předmětu současně. Toto onemocnění se může také projevovat krátkodobým rozmazeným viděním.

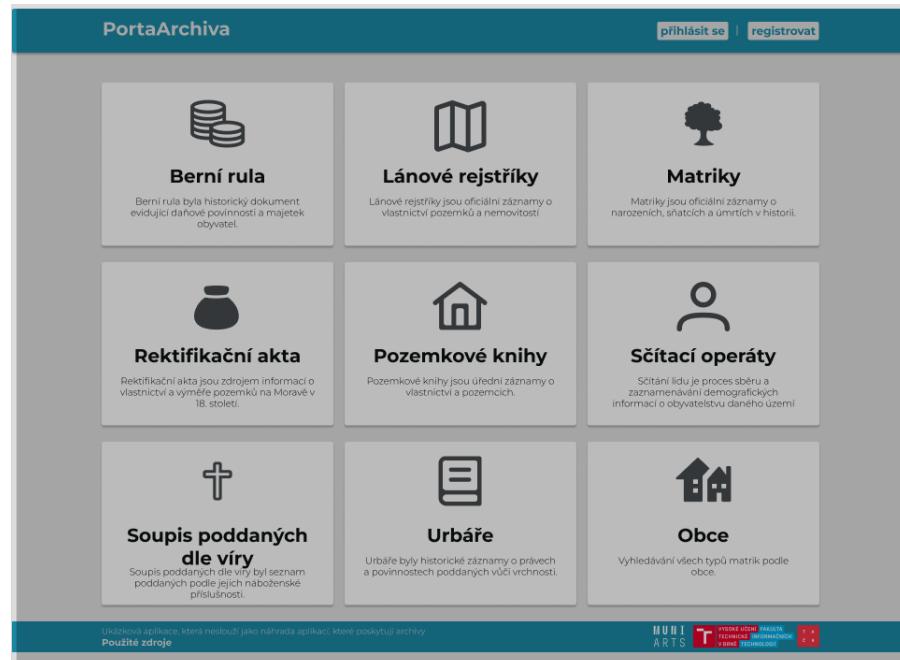


Obrázek B.4: Hlavní strana - simulace Deuteranopie



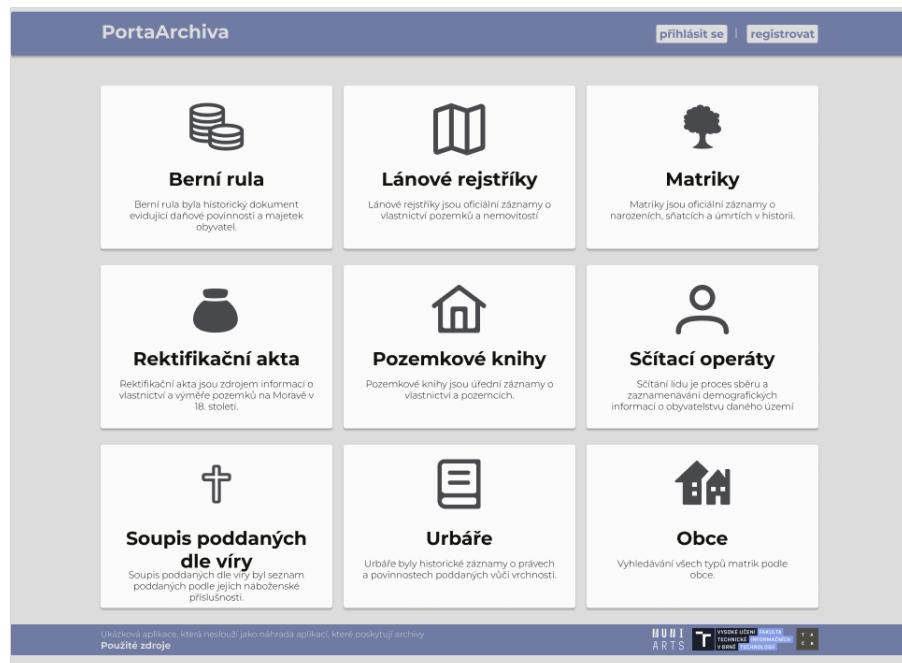
Obrázek B.5: Hlavní strana - simulace Diplopie

Zelený zákal, lékařsky označovaný jako glaukom, je onemocnění, které způsobuje trvalé poškození zrakového nervu.



Obrázek B.6: Hlavní strana - simulace Zeleného zákalu

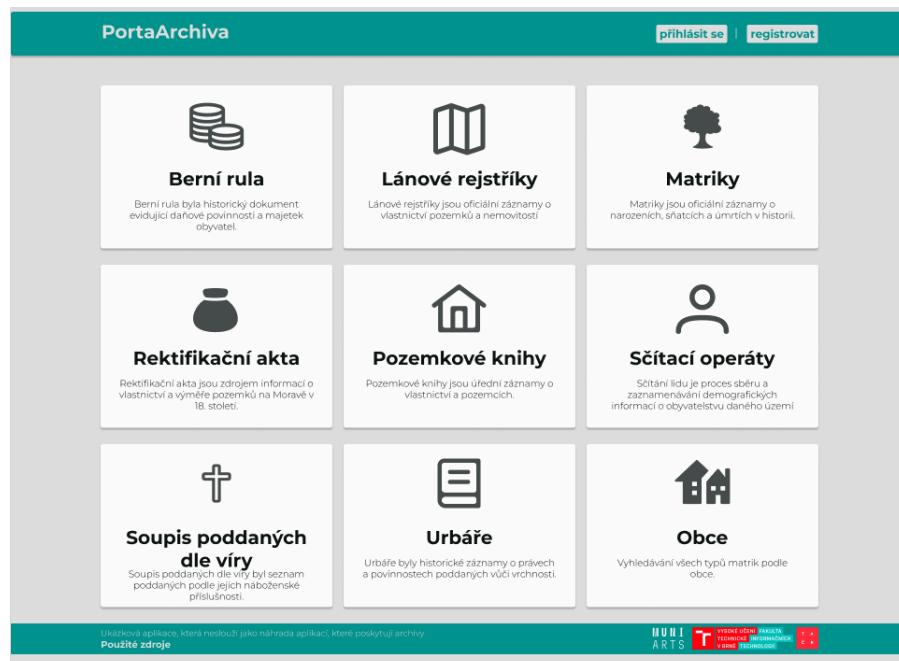
Deutanopie je forma poruchy barvocitu, při níž pacient není schopen vnímat červenou barvu.



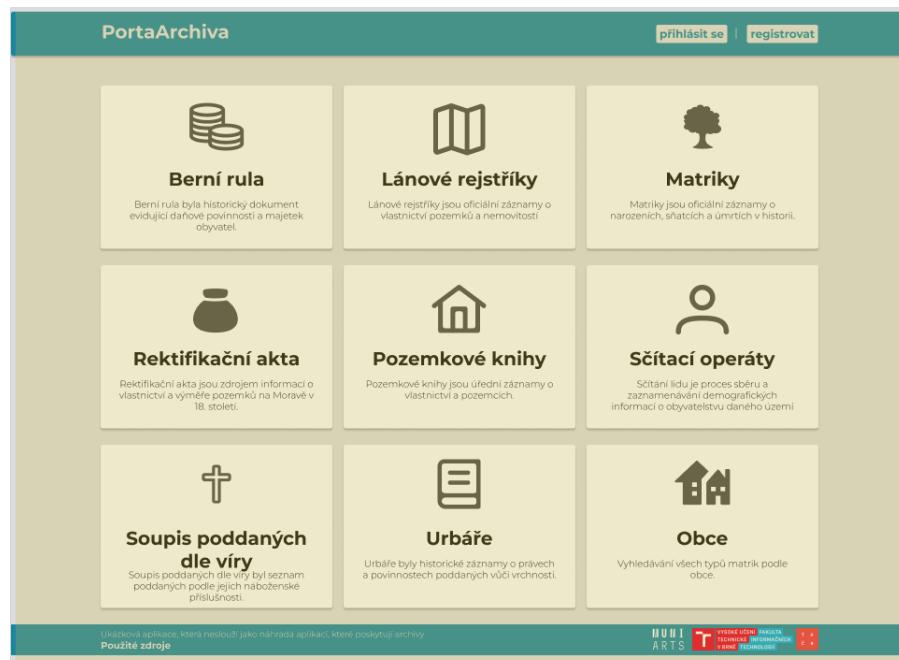
Obrázek B.7: Hlavní strana - simulace Protanopie

Deuteranopie je forma poruchy barvocitu, při níž pacient není schopen vnímat modrou barvu.

Sedý zákal, nazývaný též katarakta, je onemocnění charakterizované zakalením čočky oka a způsobuje postupné zhoršování zrakové ostrosti.



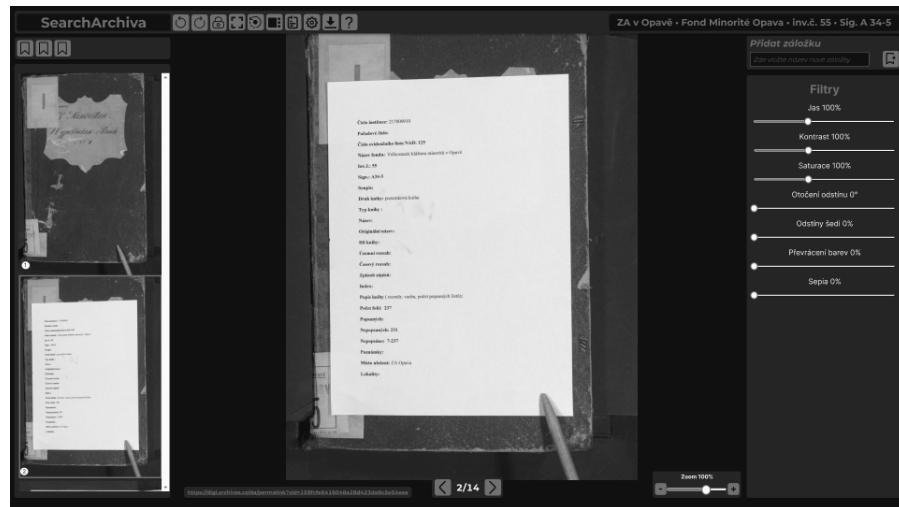
Obrázek B.8: Hlavní strana - simulace Tritanopie



Obrázek B.9: Hlavní strana - simulace Šedého zákalu

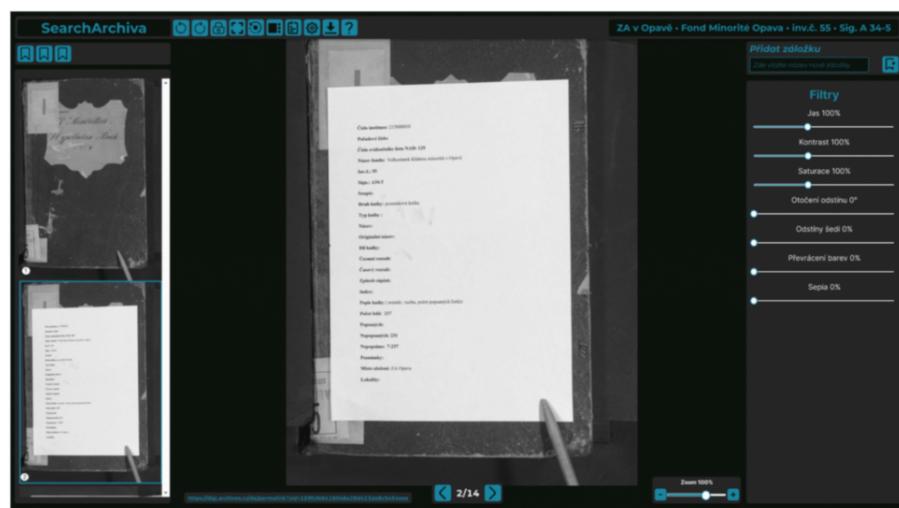
## B.2 Simulace očních vad pro prohlížeč archiválií

Achromatopsie znamená úplnou barvoslepost, kdy oko nedokáže vnímat žádné barvy, což vede k tomu, že vidění je omezeno na černou a bílou s odstíny šedi.



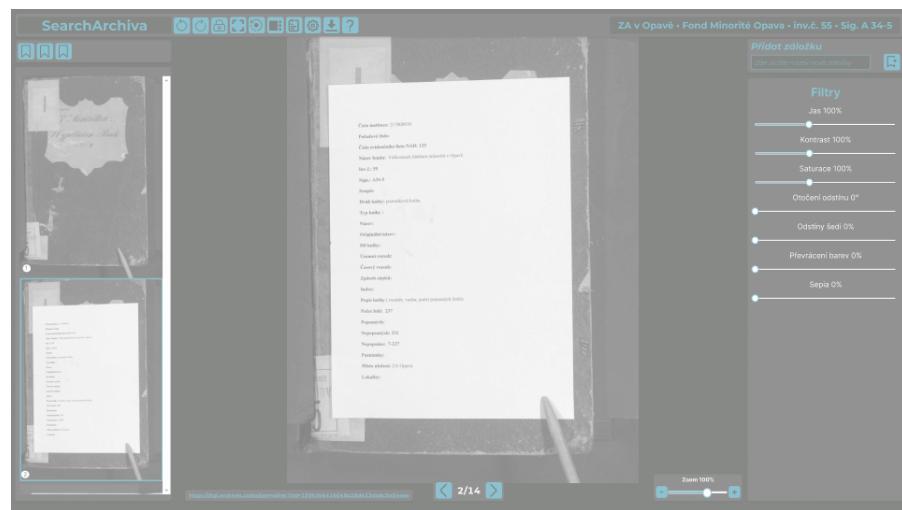
Obrázek B.10: Prohlížeč archiválií - simulace achromatopsie

Oční vada, při které se paprsky světla, které procházejí čočkou lidského oka, sbíhají před sítnicí. Tím není na sítnici vytvořen ostrý obraz.



Obrázek B.11: Prohlížeč archiválií - simulace krátkozrakosti

Světloplachost (fotofobie) se projevuje pocitem oslnění u pacientů i při pohledu za běžných světelních podmínek.



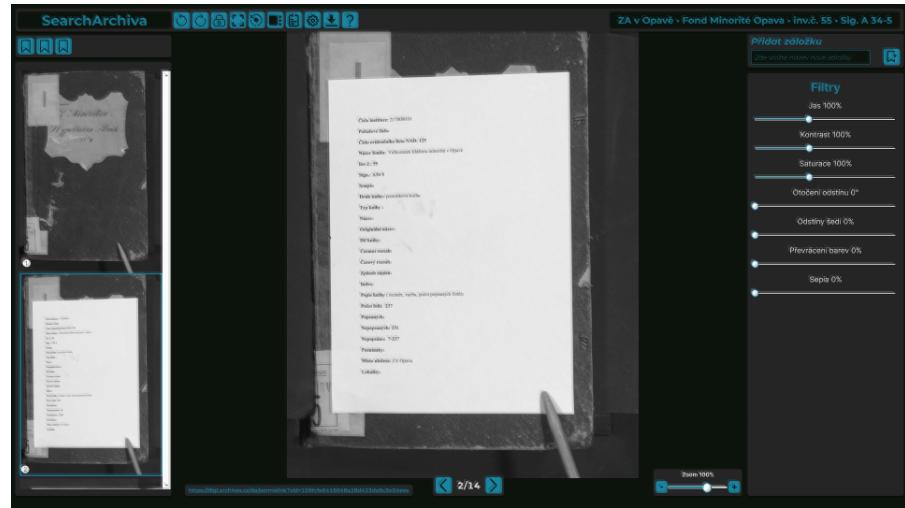
Obrázek B.12: Prohlížeč archiválií - simulace fotofobie

Deuteranopie je forma poruchy barvocitu, při níž pacient není schopen vnímat zelenou barvu.



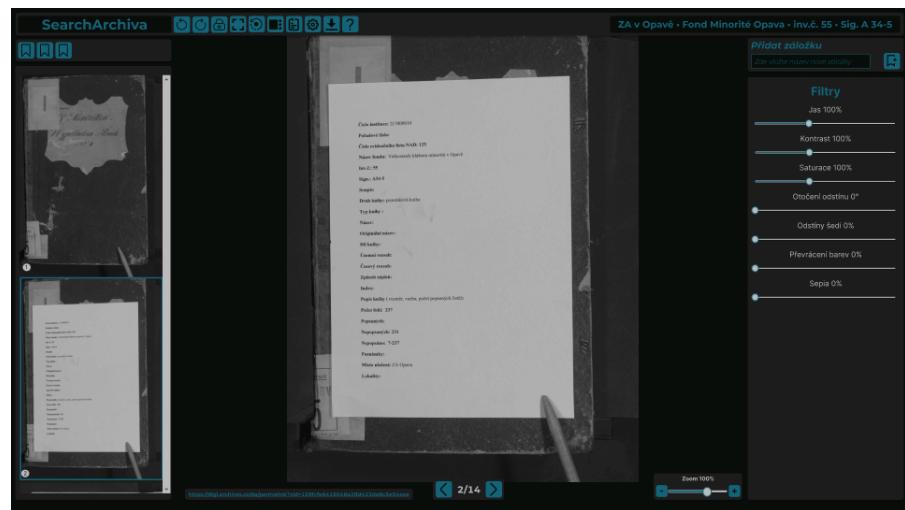
Obrázek B.13: Prohlížeč archiválií - simulace Deuteranopie

Diplopie je porucha, při níž pacient vnímá dva obrazy jednoho předmětu současně. Toto onemocnění se může také projevovat krátkodobým rozmazeným viděním.



Obrázek B.14: Prohlížeč archiválií - simulace Diplopie

Zelený zákal, lékařsky označovaný jako glaukom, je onemocnění, které způsobuje trvalé poškození zrakového nervu.



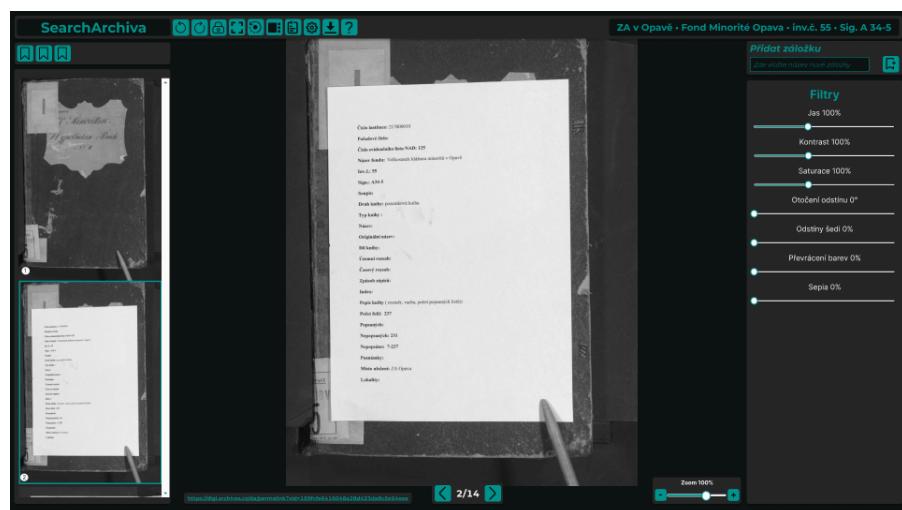
Obrázek B.15: Prohlížeč archiválií - simulace Zeleného zákalu

Protanopie je forma poruchy barvocitu, při níž pacient není schopen vnímat červenou barvu.



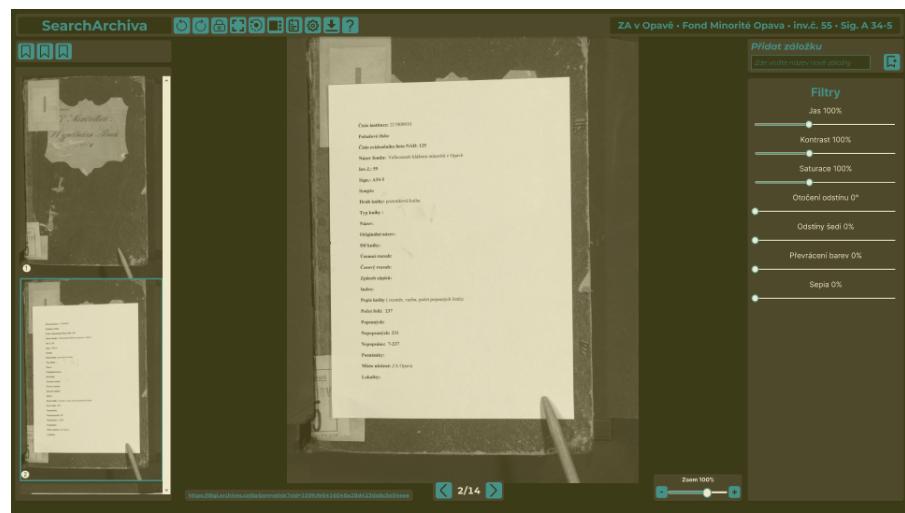
Obrázek B.16: Prohlížeč archiválií - simulace Protanopie

Tritanopie je forma poruchy barvocitu, při níž pacient není schopen vnímat modrou barvu.



Obrázek B.17: Prohlížeč archiválií - simulace Tritanopie

Šedý zákal, nazývaný též katarakta, je onemocnění charakterizované zakalením čočky oka a způsobuje postupné zhoršování zrakové ostrosti.

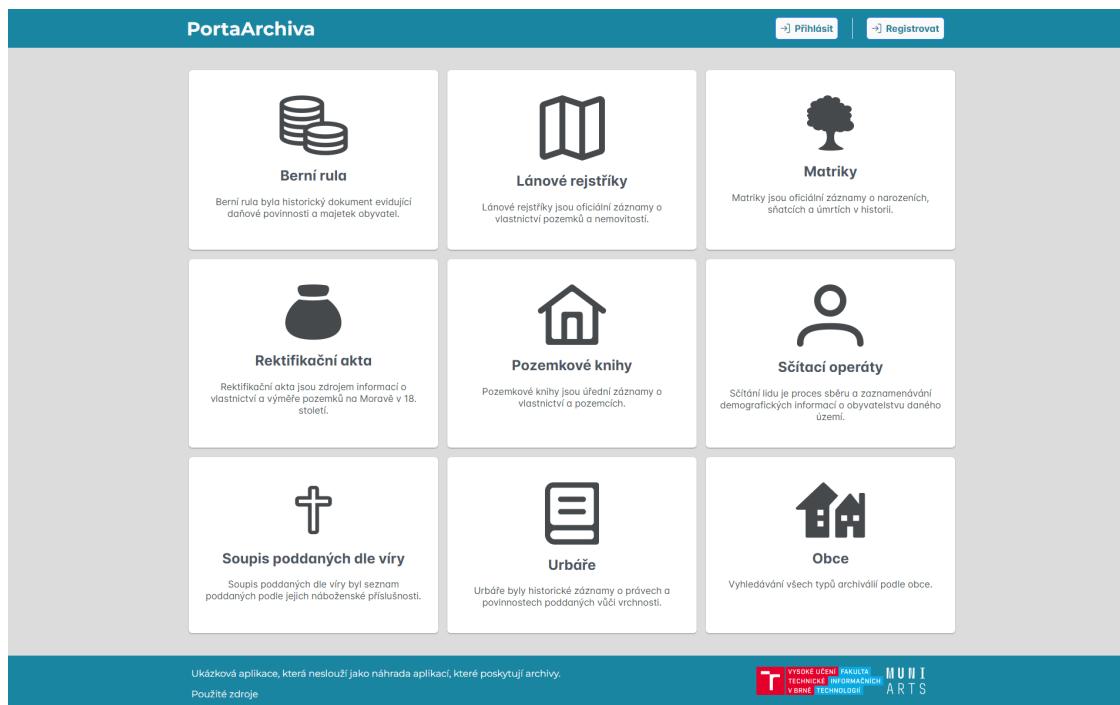


Obrázek B.18: Prohlížeč archiválií - simulace Šedého zákalu

## Příloha C

# Ukázka grafického uživatelského rozhraní výsledné aplikace

Tato příloha obsahuje snímky výsledné aplikace. Grafický návrh byl vytvořen ve fázi návrhu a následně dle něho byl systém implementován. Mezi návrhem a implementací vznikly drobné odchylky. Jednotlivé pohledy byly popsány ve fázi návrhu.



Obrázek C.1: Titulní strana

**PortaArchiva**

Přihlásit | Registrat

Matriky

Zde vložte hledaný výraz

Kraj: Okres: Obec: Zobrazit pouze digitalizované

Od roku: Do roku: Archiv:

Celkem nalezeno 12525 záznamů

Inv.č.	Sig.	Původce Typ původce	Narození od-do Index narození	Oddání od-do Index oddání	Zemření od-do Index zemřeli	Obce	Počet snímků	Archiv
2575		Dolní Věstonice římskokatolická církev	1579 - 1614			(3)	142	Moravský zemský archiv v Brně
14896		Znojmo - Louka	1580 - 1618	1580 - 1618	1580 - 1618	(10)	198	Moravský zemský archiv v Brně
16850		Brno - sv. Jakub římskokatolická církev	1587 - 1599	1587 - 1596		(1)	128	Moravský zemský archiv v Brně
6400		Jihlava-sv. Jakub římskokatolická církev	1599 - 1625	1599 - 1625		(18)	513	Moravský zemský archiv v Brně
16851		Brno - sv. Jakub římskokatolická církev	1600 - 1607			(2)	186	Moravský zemský archiv v Brně

<< < 1 2 3 4 5 > >> 1 z 2505 stránek nožit další Počet záznamů na stránce: 5

Ukázková aplikace, která neslouží jako náhrada aplikací, které poskytují archivy.  
Použité zdroje

VYSOKÉ ŠKOLY FAKULTA MUNI ARTS

Obrázek C.2: Seznam archiválií

**PortaArchiva**

Přihlásit | Registrat

Matriky > Sig 14896

Informace Územní rozsah

Archiv Název: Moravský zemský archiv v Brně	Signatura 14896	Jazyk latina němčina	Narození 1580 - 1618
Index narozených 1580 - 1618	Oddání 1580 - 1618	Index oddaných 1580 - 1618	Zemřeli 1580 - 1618
Index zemřelých 1580 - 1618	Původce Název: Znojmo - Louka Typ: římskokatolická církev		

  
198 snímků

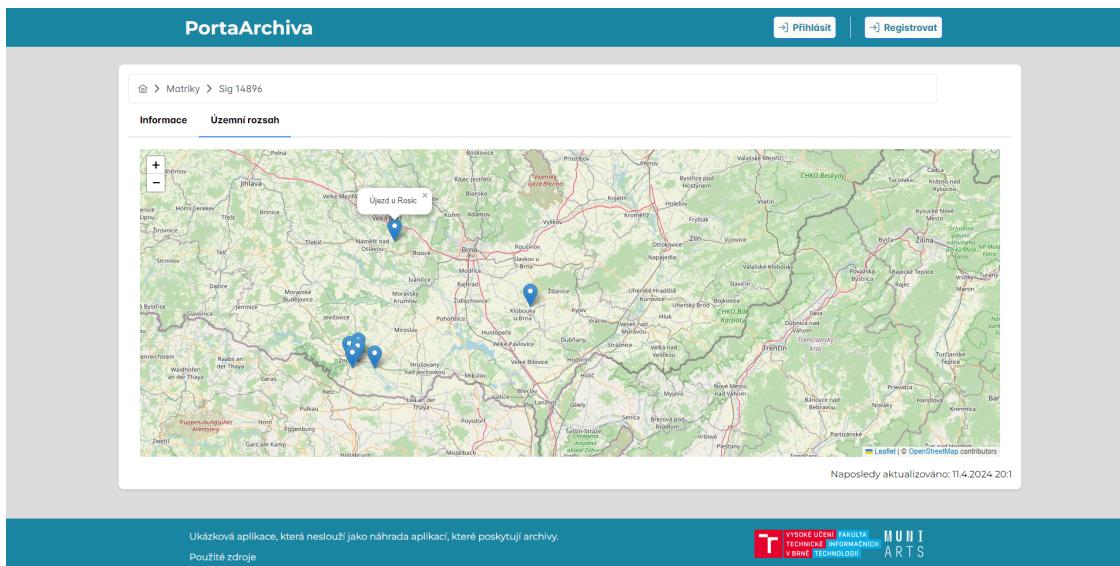
Zobrazit v prohlížeči  
Zobrazit na stránkách archivu

Naposledy aktualizováno: 11.4.2024 20:1

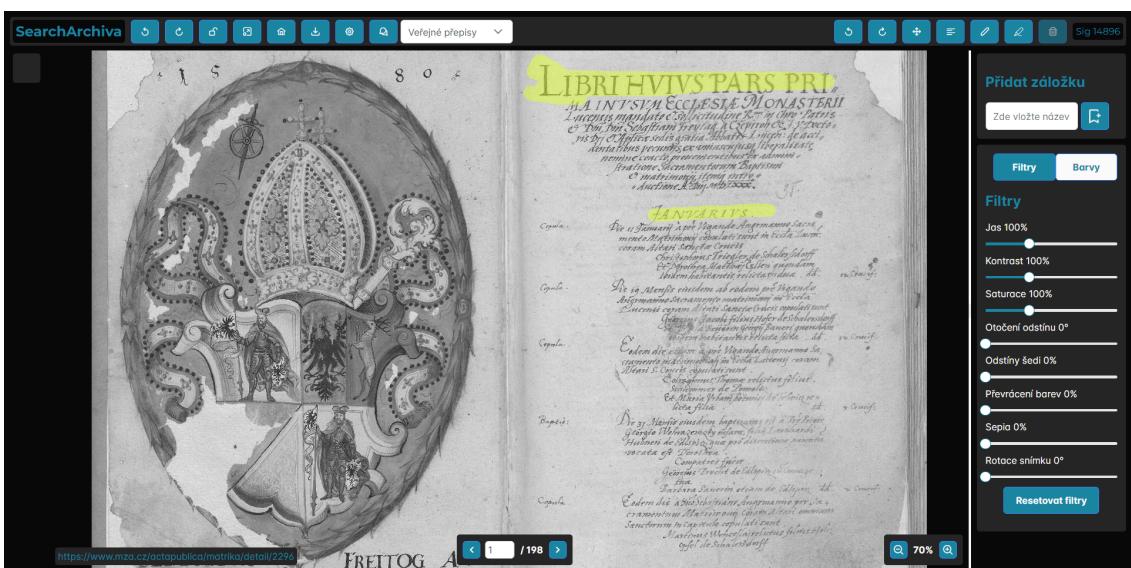
Ukázková aplikace, která neslouží jako náhrada aplikací, které poskytují archivy.  
Použité zdroje

VYSOKÉ ŠKOLY FAKULTA MUNI ARTS

Obrázek C.3: Detailní pohled na archiválii



Obrázek C.4: Zobrazení lokalit dané archiválie



Obrázek C.5: Prohlížeč zdigitalizovaných archiválií

Uživatelské jméno\*  
alepir

Email\*  
pavel.sestak10@gmail.com

Heslo\*  
\*\*\*\*\*

Heslo znova\*  
\*\*\*\*\*

**Registrovat**

Obrázek C.6: Formulář

Jméno Email  
alepir / pavel.sestak10@gmail.com /

Zde vložte hledaný výraz

Oblibené matrky Oblibený ostatní archivní materiál Poznámky Záložky

Inv.č.	Sig.	Původce typ původce	Narození od-do Index narození	Oddání od-do Index oddání	Zemřelí od-do Index zemřeli	Obce	Počet snímků	Archiv
▼	16442	Velké Mezíříčí římskokatolická cirkev	1622 - 1669	1622 - 1669	1657 - 1665	(14)	352	Moravský zemský archiv v Brně

Informace Územní rozsah

Archiv Název: Moravský zemský archiv v Brně  
Signature 16442  
Language českina, latinka  
Birthplace 1622 - 1669

Oddání 1622 - 1669  
Zemřelí 1657 - 1665  
Původce Název: Velké Mezíříčí, Typ: Římskokatolická cirkev

352 snímků

Zobrazit v prohlížeči

Zobrazit na stránkách archivu

Naposledy aktualizováno: 11.4.2024 19:58

Počet záznamů na stránce: 5

Ukázková aplikace Použité zdroje

Obrázek C.7: Uživatelský profil

PortaArchiva

[Přihlásit](#) | [Registrovat](#)

> Obce

Státy

Kraje

Okresy

Obec

Zobrazit pouze digitalizované

Od roku

Do roku

Archiv

[Resetovat filtry](#)

> Lánové rejstříky (362)

> Matriky (21727)

Matriky
Celkem nalezeno 21727 záznamů

Inv. č.	↑↓	Sig. ↑↓	Původce ↑↓	Narození od-do	Oddání od-do	Zemřelí od-do	Obce	Počet snímků	Archiv ↑↓
			Typ původce	Index narození	Index oddání	Index zemřeli			
581	Br IX 10	Razová, římskokat. f. ú. evangelická		1571 - 1669	1572 - 1669	(1)	104	Zemský archiv v Opavě	
5495	118/1	Most - děkanství, římskokat. f. ú., Most - děkanství katalická		1574 - 1617 1574 - 1617		(1)	374	Státní oblastní archiv v Litoměřicích	
107	3/1	Beněšov nad Ploučnicí, římskokat. f. ú., Beněšov nad Ploučnicí katalická		1580 - 1624	1580 - 1624	1580 - 1624	(6)	506	Státní oblastní archiv v Litoměřicích
594	16/1	Březno, římskokat. f. ú., Březno katalická		1580 - 1653	1580 - 1653	1580 - 1653	(1)	150	Státní oblastní archiv v Litoměřicích
3517	L67/1	Kamenický Šenov, římskokat. f. ú., Kamenický Šenov katalická		1580 - 1670	1653 - 1670	1580 - 1670	(2)	120	Státní oblastní archiv v Litoměřicích

<< <
1
2
3
4
5
> >>
1 z 4346 stránek
načíst další
Počet záznamů na stránce:
5

> Rektifikační aktá (221)

> Pozemkové knihy (13644)

> Sčítací operáty (8882)

> Urbáře (2649)

Ukázková aplikace, která neslouží jako náhrada aplikací, které poskytují archivy.

Použité zdroje | Ostatní projekty

VYSOKÉ UČENÍ FAKULTA  
TECHNICKÉ INFORMAČNÍ  
V BRNĚ TECHNOLOGIÍ

MUUNI  
ARTS

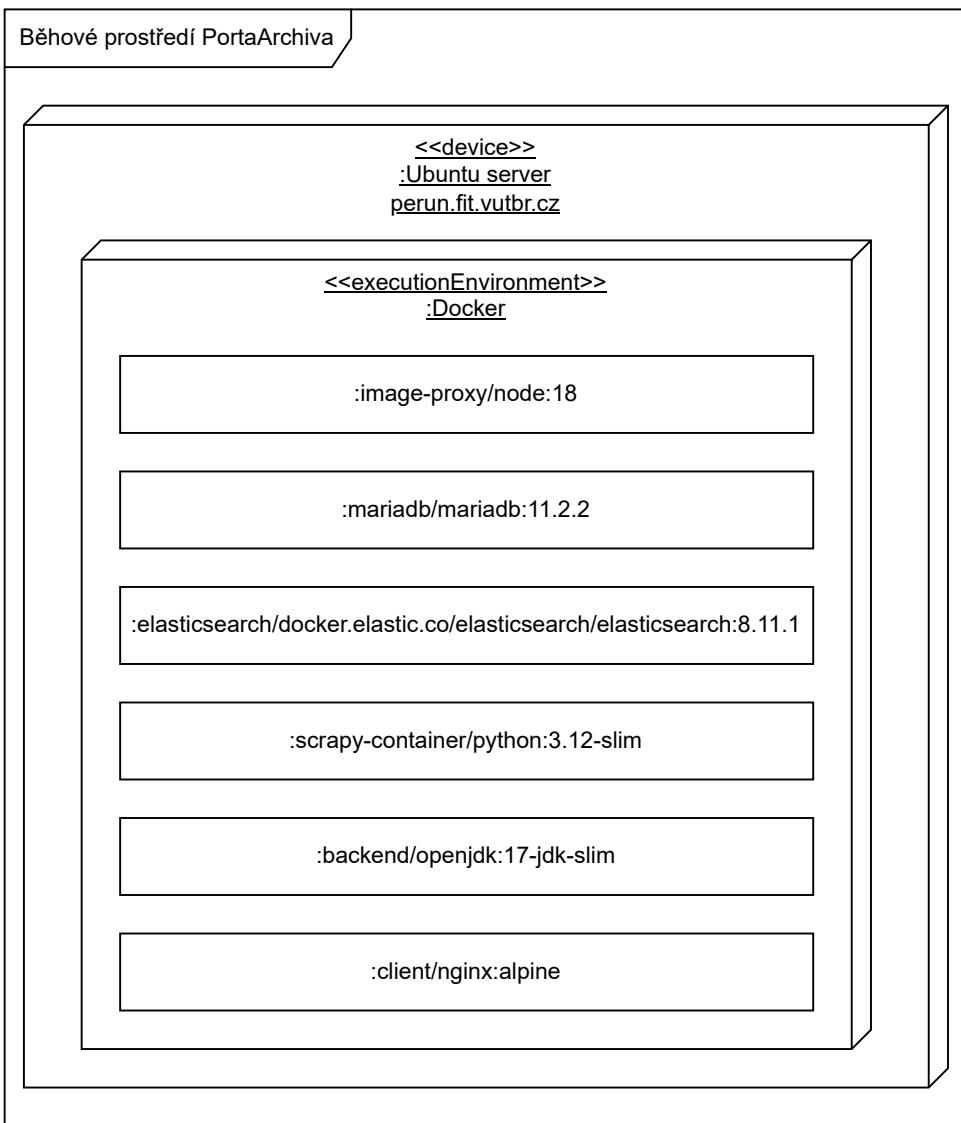
Obrázek C.8: Vyhledávání obcí

127

## Příloha D

### Nasazení

Produkční server běží v rámci školní infrastruktury. Pro úspěšné nasazení stačí vzít zdrojové soubory, nastavit soubor `.env` podle souboru `.env.example` a spustit příkaz `docker compose up -d` jak pro serverovou, tak pro klientskou aplikaci. Po nasazení systému se doporučuje pro každý scraper spustit příkaz `ssh <username>@perun.fit.vutbr.cz "docker exec -d <scraper container id> nohup scrapy crawl <scraper name> &"`. Jednotlivé scrapery mohou běžet paralelně, ale je vhodné rozložit scrapery pro více archivů, aby nedocházelo ke zbytečnému přetěžování konkrétního archivního systému. O následnou aktualizaci dat se stará úloha cron, která periodicky stahuje datové sady a aktualizuje záznamy v databázi.



Obrázek D.1: Diagram nasazení