UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

# DIPLOMOVÁ PRÁCE

2024 Bc. Petr Bednář

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

# Informační systém pro uchování a správu multimediálních metadat

Diplomová práce

2024 Bc. Petr Bednář

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2023/2024

# ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

| | |
|---|---|
| Jméno a příjmení: | **Bc. Petr Bednář** |
| Osobní číslo: | **I22153** |
| Studijní program: | **N0613A140007 Informační technologie** |
| Téma práce: | **Informační systém pro uchování a správu multimediálních metadat** |
| Zadávající katedra: | **Katedra softwarových technologií** |

## Zásady pro vypracování

Cílem diplomové práce je navrhnout a implementovat webový informační systém s primárním zaměřením na uchování a správu multimediálních metadat, přičemž cílem je zpřístupnit metadata uživatelům, kteří je mohou dále spravovat (např. hodnotit obsah apod.).
Důležitou částí diplomové práce bude také synchronizace mezi desktop verzí aplikace a webové aplikace, což poskytne uživatelský komfort pro oba způsoby práce s metadaty.
Uživatelské profily bude možné sdílet s ostatními uživateli (také v podobě přístupného API) pro zpřístupnění uživatelských dat, což umožní vzájemnou interakci mezi uživateli.

Rozsah pracovní zprávy: **cca 60 stran**
Rozsah grafických prací:
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. ARLOW, Jim a NEUSTADT, Ila. UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky. 2., aktualiz. a dopl. vyd. Brno: Computer Press, 2007. ISBN 978-80-251--1503-9.
2. LEONARD, Anghel. Spring Boot persistence best practices: optimize Java persistencce performance in Spring Boot Applications. [New York, NY]: Apress, [2020]. ISBN 978-1-4842-5625-1.

Vedoucí diplomové práce: **doc. Ing. Michael Bažant, Ph.D.**
Katedra softwarových technologií

Datum zadání diplomové práce: **8. listopadu 2023**
Termín odevzdání diplomové práce: **17. května 2024**

L.S.

**Ing. Zdeněk Němec, Ph.D.** v.r.
děkan

**prof. Ing. Antonín Kavička, Ph.D.** v.r.
vedoucí katedry

V Pardubicích dne 30. listopadu 2023

Prohlašuji:

Práci s názvem Informační systém pro uchování a správu multimediálních metadat jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

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

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

V Pardubicích dne 11. 5. 2024

Bc. Petr Bednář

# PODĚKOVÁNÍ

Chtěl bych poděkovat doc. Ing. Michaeli Bažantovi, Ph.D. za odborné vedení mé diplomové práce, cenné rady spojené s tvorbou analytických modelů, trpělivost a vstřícnost při mnohých konzultacích. V neposlední řadě bych chtěl poděkovat své rodině a blízkým přátelům, jejichž čas a zpětná vazba prospěla k úspěšnému dokončení této práce.

# ANOTACE

Diplomová práce se zaměřuje na návrh a implementaci informačního systému pro správu, uchování a sdílení multimediálních metadat. Systém poskytne snadné šíření uživatelských profilů a, podpoří tak jejich sociální interakci. S vyvíjeným systémem souvisí také integrace synchronizační funkcionality do již vytvořené desktopové aplikace.

Hlavním cílem této práce je tedy vytvoření uceleného hodnotitelského systému, který umožní robustní správu hodnocení napříč zařízeními. Teoretická část prozkoumává použité technologie spojené s vývojem webových aplikací a zahrnuje průzkum již existujících řešení na trhu. Praktická část se věnuje návrhu, implementaci a nasazení celého systému. Sem patří prohlídka FE webové aplikace, náhled do implementace BE, využití systému společně s doprovodnou desktopovou aplikací a důkladný postup automatizace vývoje.

# KLÍČOVÁ SLOVA

Pallidium.cz, Java, Spring, React.js, PostgreSQL, Docker, Nginx, CI/CD, hodnocení, synchronizace, uchování, šíření

# TITLE

Information system for storing and managing multimedia metadata.

# ANNOTATION

The diploma thesis focuses on the design and implementation of an information system for the management, storage, and sharing of multimedia metadata. The system will facilitate easy sharing of user profiles, thereby supporting their social interaction. The developed system is also associated with the integration of synchronization functionality into an existing desktop application.

The main goal of this work is to create a comprehensive rating system that enables robust rating management across devices. The theoretical part explores the technologies used in web application development and include an examination of existing solutions in the market. The practical part focuses on the design and description of individual components of the implemented system. This includes an overview of the web application's front-end, insight into the backend implementation, utilization of the system in conjunction with the accompanying desktop application, and a thorough process of development automation.

# KEYWORDS

Pallidium.cz, Java, Spring, React.js, PostgreSQL, Docker, Nginx, CI/CD, rating, synchronization, storage, distribution

# OBSAH

**SEZNAM ILUSTRACÍ** ..... **13**

**SEZNAM TABULEK** ..... **14**

**SEZNAM ZDROJOVÝCH KÓDŮ** ..... **15**

**SEZNAM ZKRATEK A ZNAČEK** ..... **17**

**TERMINOLOGIE** ..... **19**

**ÚVOD** ..... **20**

**1 REŠERŠE SOUVISEJÍCÍCH SYSTÉMŮ** ..... **22**

1.1 Pallidium ..... 22

1.2 Informační systémy pro shromažďování hodnocení ..... 25

1.2.1 Česko-Slovenská filmová databáze (ČSFD) ..... 25

1.2.2 Internet Movie Database (IMDb) ..... 27

1.2.3 The Movie Database (TMDB) ..... 28

**2 ANALÝZA** ..... **29**

2.1 Webová aplikace ..... 30

2.1.1 Funkční požadavky ..... 30

2.1.2 Nefunkční požadavky ..... 31

2.2 Rozšíření desktopové aplikace ..... 32

2.2.1 Funkční požadavky ..... 32

2.2.2 Nefunkční požadavky ..... 32

2.3 Modely případů užití ..... 33

2.3.1 Základní model přístupu ..... 33

2.3.2 Modely správy knihoven ..... 34

2.3.3 Model sdílení knihoven ..... 35

2.4 Sekvenční diagramy ..... 35

2.4.1 Registrace uživatelského účtu ..... 36

2.4.2 Aktivace uživatelského účtu ..... 37

2.4.3 Přihlášení registrovaného uživatele ..... 38

2.4.4 Autentizace uživatele pomocí JWT ..... 39

2.5 Diagramy aktivit ..... 40

2.5.1 Změna hodnocení filmu ..... 40

2.5.2 Změna hodnocení seriálu ..... 41
2.6 Diagram infrastruktury ..... 42
2.7 Diagram nasazení ..... 43
2.8 Balíčkový diagram ..... 44
2.8.1 Backend ..... 44
2.8.2 Frontend ..... 44
**3 POUŽITÉ TECHNOLOGIE ..... 45**
3.1 Backend ..... 45
3.1.1 Java ..... 45
3.1.2 Spring Framework ..... 47
3.1.3 Spring Core ..... 47
3.1.4 Spring Boot ..... 48
3.1.5 Spring Data JPA ..... 48
3.1.6 AspectJ ..... 49
3.1.7 Gradle ..... 50
3.1.8 REST ..... 50
3.1.9 YAML ..... 52
3.1.10 Swagger API ..... 52
3.1.11 PostgreSQL ..... 53
3.2 Frontend ..... 54
3.2.1 Node.js ..... 54
3.2.2 Vite ..... 54
3.2.3 TypeScript ..... 55
3.2.4 React.js ..... 55
3.2.5 Axios ..... 56
3.2.6 MUI ..... 56
3.2.7 I18next ..... 56
3.2.8 MobX ..... 56
3.3 Aplikace ..... 57
3.3.1 JavaFX / OpenJFX ..... 57
3.3.2 SQLite ..... 57
3.4 Systém pro verzování ..... 57
3.4.1 Git ..... 57

3.5 Nasazení ..... 59
3.5.1 Docker ..... 59
3.5.2 Nginx ..... 60
3.6 Bezpečnost ..... 60
3.6.1 OpenVPN ..... 60
3.6.2 Let's Encrypt ..... 61
**4 IMPLEMENTACE ..... 62**
4.1 Databázový systém ..... 62
4.1.1 Segment uživatelského účtu ..... 63
4.1.2 Segment filmové cache ..... 64
4.1.3 Segment filmů ..... 65
4.1.4 Segment seriálové cache ..... 66
4.1.5 Segment seriálů ..... 67
4.2 Backend ..... 68
4.2.1 Zabezpečení ..... 69
4.2.2 Controller ..... 71
4.2.3 Service ..... 72
4.2.4 Repository ..... 72
4.2.5 Domain ..... 73
4.3 Frontend ..... 75
4.3.1 Architektura ..... 75
4.3.2 Aplikace ..... 76
4.3.3 Komponenty ..... 77
4.3.4 Design aplikace ..... 77
4.3.5 Propojení s desktop aplikací ..... 78
4.4 Nginx server ..... 78
4.4.1 Globální nastavení ..... 78
4.4.2 HTTP server ..... 79
4.4.3 HTTPS servery pro hlavní portál ..... 80
4.4.4 Reverse proxy pro backend ..... 81
4.4.5 SSL ..... 82
4.5 Desktopová aplikace ..... 83
4.5.1 Architektura ..... 83

4.5.2 Implementované změny ..... 84
4.6 Synchronizace ..... 84
4.6.1 Změny hodnocení ..... 84
4.6.2 Synchronizace z aplikace do webu ..... 84
4.6.3 Synchronizace z webu do aplikace ..... 85
4.6.4 Shrnutí ..... 86
4.7 Sociální síť ..... 86
4.7.1 Sdílení knihoven ..... 86
4.7.2 Přátelé ..... 87
4.7.3 Uživatelský profil ..... 87
4.8 Testování ..... 87
4.9 Profilování ..... 89
**5 POUŽITÍ SYSTÉMU ..... 90**
5.1 Webová aplikace ..... 90
5.2 Desktopová aplikace ..... 94
**6 INTEGRACE A NASAZENÍ ..... 95**
6.1 Kontinuální integrace (CI) ..... 95
6.2 Kontinuální dodávání (CD) ..... 95
6.3 Kontinuální nasazení (CD) ..... 96
6.4 Doménové jméno ..... 96
6.4.1 Registrace doménového jména ..... 97
6.4.2 Problém veřejných dat ..... 97
6.4.3 Nastavení domény ..... 97
6.5 Virtuální privátní server (VPS) ..... 98
6.5.1 Prvotní konfigurace ..... 99
6.5.2 Konfigurace SSH ..... 99
6.5.3 Nastavení firewallu ..... 100
6.6 Docker kontejnery ..... 101
6.6.1 Příprava na kontejnerizaci ..... 103
6.6.2 Dockerfile pro backend ..... 103
6.6.3 Dockerfile pro frontend ..... 104

6.7 Workflow kontinuální integrace a dodávky ..... 105
6.7.1 Backend ..... 106
6.7.2 Frontend ..... 108
6.8 Workflow nasazení na VPS ..... 108
6.8.1 Docker Compose pro databázi ..... 109
6.8.2 Docker Compose pro backend ..... 110
6.8.3 Docker Compose pro frontend ..... 110
6.8.4 Workflow pro backend ..... 111
6.8.5 Workflow pro frontend ..... 112
6.9 OpenVPN ..... 113
**ZÁVĚR ..... 114**
**POUŽITÁ LITERATURA ..... 116**
**SEZNAM PŘÍLOH ..... 125**

# SEZNAM ILUSTRACÍ

Obrázek 1: Výběr záznamu pro stažení metadat v aplikaci Pallidium ....24
Obrázek 2: Přehled záznamů ve filmové knihovně aplikace Pallidium ....24
Obrázek 3: Hlavní webová stránka ČSFD ....25
Obrázek 4: Uživatelský profil na webu ČSFD ....26
Obrázek 5: Filmový profil na webu ČSFD ....26
Obrázek 6: Hlavní webová stránka IMDb ....27
Obrázek 7: Hlavní webová stránka TMDB ....28
Obrázek 8: UC model základního přístupu ....33
Obrázek 9: UC model správy knihoven ve webové aplikaci ....34
Obrázek 10: UC model správy knihoven v desktopové aplikaci ....34
Obrázek 11: UC model sdílení knihoven ....35
Obrázek 12: Sekvenční diagram uživatelské registrace ....36
Obrázek 13: Sekvenční diagram aktivace uživatelského účtu ....37
Obrázek 14: Sekvenční diagram přihlášení registrovaného uživatele ....38
Obrázek 15: Sekvenční diagram autentizace uživatele pomocí JWT ....39
Obrázek 16: Diagram aktivity změny hodnocení filmu ....40
Obrázek 17: Diagram aktivity změny hodnocení seriálu ....41
Obrázek 18: Diagram infrastruktury ....42
Obrázek 19: Diagram nasazení ....43
Obrázek 20: Úvodní strana prohlížeče dokumentace OpenAPI ve SwaggerUI ....53
Obrázek 21: Porovnání rychlosti nástrojů sestavení FE [43] ....54
Obrázek 22: Databázové schéma uživatelského segmentu ....63
Obrázek 23: Databázové schéma segmentu filmové cache ....64
Obrázek 24: Databázové schéma segmentu filmů ....65
Obrázek 25: Databázové schéma segmentu seriálové cache ....66
Obrázek 26: Databázové schéma segmentu seriálů ....67
Obrázek 27: Diagram interakce komponent BE ....68
Obrázek 28: Diagram interakce BE s ostatními systémy ....69
Obrázek 29: Diagram interakce komponent FE ....75
Obrázek 30: Odpověď personálu TMDB v reakci na X-Forwarded-For ....78
Obrázek 31: Diagram interakce desktopové aplikace ....83
Obrázek 32: Vstupní stránka ....90
Obrázek 33: Email pro aktivaci uživatelského účtu ....90
Obrázek 34: Hlavní stránka a přehled nedávných hodnocení ....91
Obrázek 35: Správa aktivních relací ....91
Obrázek 36: Uživatelský profil ....92
Obrázek 37: Přehled knihoven a posledních úprav ....93
Obrázek 38: Nastavení sdílení a synchronizace knihovny ....93
Obrázek 39: Nastavení synchronizace knihovny ....94
Obrázek 40: Výběr režimu synchronizace ....94
Obrázek 41: Graf využití Docker kontejnerů [92] ....102
Obrázek 42: Graf nejpoužívanějšího software v Docker kontejnerech [92] ....102

# SEZNAM TABULEK

Tabulka 1: Verze použitých technologií BE....45
Tabulka 2: Verze použitých technologií FE....54
Tabulka 3: Verze použitých technologií desktopové aplikace....57
Tabulka 4: Synchronizace dat z desktopové aplikace do webové aplikace....86
Tabulka 5: Synchronizace dat z webové aplikace do desktopové aplikace....86
Tabulka 6: Nastavení DNS domény pallidium.cz....98
Tabulka 7: Rozpis podporovaných tarifů VPS poskytovatele Hukot.net [87]....99
Tabulka 8: Nastavení povoleného provozu pro VPS firewall....101
Tabulka 9: Nastavení zakázaného provozu pro VPS firewall....101

# SEZNAM ZDROJOVÝCH KÓDŮ

Zdrojový kód 1: Instalace Vite pomocí NPM a Yarn ..... 55
Zdrojový kód 2: Nastavení privátních GitHub balíčků pro Gradle ..... 58
Zdrojový kód 3: Nastavení privátních GitHub balíčků pro Maven, část 1. ..... 59
Zdrojový kód 4: Nastavení privátních GitHub balíčků pro Maven, část 2. ..... 59
Zdrojový kód 5: Nastavení security filter chain ..... 70
Zdrojový kód 6: Ukázka zápisu komponenty controller ..... 71
Zdrojový kód 7: Ukázka zápisu komponenty service ..... 72
Zdrojový kód 8: Ukázka zápisu komponenty repository ..... 72
Zdrojový kód 9: Ukázka zápisu komponenty domain, část 1. ..... 73
Zdrojový kód 10: Ukázka zápisu komponenty domain, část 2. ..... 74
Zdrojový kód 11: Nastavení kořenového scriptu FE ..... 76
Zdrojový kód 12: Ukázka navigačního směrovače React ..... 76
Zdrojový kód 13: Funkce pro tvorbu dynamických odkazů ..... 76
Zdrojový kód 14: Funkce pro komunikaci s BE ..... 77
Zdrojový kód 15: Mapa výjimek subdomén ..... 79
Zdrojový kód 16: Nastavení HTTP serveru ..... 79
Zdrojový kód 17: Nastavení HTTPS serveru pro přesměrování ..... 80
Zdrojový kód 18: Nastavení hlavního HTTPS serveru ..... 80
Zdrojový kód 19: Dodatečné nastavení Nginx serveru ..... 81
Zdrojový kód 20: Nastavení reverzního proxy serveru ..... 81
Zdrojový kód 21: Příkaz pro odebrání speciálních komentářů ..... 82
Zdrojový kód 22: Anotace s právy uživatele a administrátora ..... 88
Zdrojový kód 23: Ukázka přístupových testů ..... 88
Zdrojový kód 24: Anotace pro profilování ..... 89
Zdrojový kód 25: Ukázka profilování koncových bodů ..... 89
Zdrojový kód 26: Konfigurace SSH na VPS ..... 100
Zdrojový kód 27: Dockerfile pro backend, část 1. ..... 103
Zdrojový kód 28: Dockerfile pro backend, část 2. ..... 103
Zdrojový kód 29: Dockerfile pro backend, část 3. ..... 103
Zdrojový kód 30: Dockerfile pro backend, část 4. ..... 104
Zdrojový kód 31: Dockerfile pro backend, část 5. ..... 104
Zdrojový kód 32: Dockerfile pro frontend, část 1. ..... 104
Zdrojový kód 33: Dockerfile pro frontend, část 2. ..... 104
Zdrojový kód 34: Dockerfile pro frontend, část 3. ..... 104
Zdrojový kód 35: Dockerfile pro frontend, část 4. ..... 105
Zdrojový kód 36: Specifikace workflow, část 1. ..... 105
Zdrojový kód 37: Specifikace workflow, část 2. ..... 105
Zdrojový kód 38: Workflow CI pro BE, část 1. ..... 106
Zdrojový kód 39: Workflow CI pro BE, část 2. ..... 106
Zdrojový kód 40: Workflow CI pro BE, část 3. ..... 107
Zdrojový kód 41: Workflow CI pro FE ..... 108
Zdrojový kód 42: Nastavení sítě v docker compose ..... 109
Zdrojový kód 43: Docker compose pro databázi ..... 109
Zdrojový kód 44: Import hesel v docker compose ..... 110

Zdrojový kód 45: Docker compose pro backend.....110
Zdrojový kód 46: Docker compose pro frontend.....110
Zdrojový kód 47: Workflow CD pro backend, část 1.....111
Zdrojový kód 48: Workflow CD pro backend, část 2.....111
Zdrojový kód 49: Workflow CD pro backend, část 3.....111
Zdrojový kód 50: Workflow CD pro backend, část 4.....112
Zdrojový kód 51: Workflow CD pro frontend s obnovou SSL.....112
Zdrojový kód 52: Workflow CD pro frontend.....113

# SEZNAM ZKRATEK A ZNAČEK

| | |
|---|---|
| ČSFD | Česko-Slovenská filmová databáze |
| IMDb | Internet Movie Database |
| TMDB | The Movie Database |
| VOD | Video On Demand |
| DVD | Digital Video Disc |
| WWW | World Wide Web |
| FDb | Filmová databáze |
| FR | Functional Requirement |
| NFR | Non Functional Requirement |
| UC | Use Case |
| BE | Back End / Backend |
| FE | Front End / Frontend |
| IT | Information Technology |
| AI | Artificial Intelligence |
| IoT | Internet of Things |
| API | Application Programming Interface |
| JVM | Java Virtual Machine |
| JDK | Java Development Kit |
| IDE | Integrated Development Environment |
| AOP | Aspect Oriented Programming |
| DI | Dependency Injection |
| POJO | Plain Old Java Object |
| RT | Real Time |
| EJB | Enterprise Java Beans |
| JSF | JavaServer Faces |
| MVC | Model View Controller |
| IoC | Inversion of Control |
| DAO | Data Access Object |
| ORM | Object Relational Mapping |
| JDBC | Java Database Connectivity |
| JPA | Java Persistence API |
| HTTP | Hypertext Transfer Protocol |
| HTTPS | Hypertext Transfer Protocol Secure |
| OOP | Object Oriented Programming |
| REST | Representational State Transfer |
| XML | Extensible Markup Language |
| RPC | Remote Procedure Call |
| SOAP | Simple Object Access Protocol |
| URI | Uniform Resource Identifier |
| URL | Uniform Resource Locator |
| UI | User Interface |
| JSON | JavaScript Object Notation |
| IS | Information System |
| YAML | YAML Ain’t Markup Language |
| POX | Plain Old XML |
| SQL | Structured Query Language |
| RDBMS | Relational Database Management System |
| ACID | Atomicity Consistency Isolation Durability |

| | |
|---|---|
| NPM | Node Package Manager |
| HTML | Hypertext Markup Language |
| ES | ECMAScript |
| JSX | JavaScript XML |
| SPA | Single Page Application |
| MUI | Material UI |
| JFX | JavaFX |
| FXML | FX Markup Language |
| IDEA | Integrated Development Environment |
| CI | Continuous Integration |
| CD | Continuous Delivery / Continuous Deployment |
| PaaS | Platform as a Service |
| IBM | International Business Machines Corporation |
| SSL | Secure Sockets Layer |
| VPN | Virtual Private Network |
| VPS | Virtual Private Server |
| CA | Certificate Authority |
| ISRG | Internet Security Research Group |
| TLS | Transport Layer Security |
| ACME | Automated Certificate Management Environment |
| CORS | Cross-Origin Resource Sharing |
| JWT | JSON Web Token |
| JPQL | Java Persistence Query Language |
| DTO | Data Transfer Object |
| IP / IPv4 / IPv6 | Internet Protocol, Internet Protocol version 4/6 |
| DPH | Daň z přidané hodnoty |
| Kč | Korun českých |
| ID | Identifier |
| DNS | Domain Name System |
| DNSSEC | Domain Name System Security Extensions |
| TXT | Text |
| MX | Mail Exchange |
| DKIM | DomainKeys Identified Mail |
| SPF | Sender Policy Framework |
| DMARC | Domain-based Message Authentication |
| TTL | Time To Live |
| NS | Name Server |
| SLA | Service-Level Agreement |
| LTS | Long-Term Support |
| GB | GigaByte |
| ISO | Identical Storage Image of Optical Media |
| SSH | Secure Shell |
| RSA | Rivest-Shamir-Adleman encryption |
| TCP | Transmission Control Protocol |
| UDP | User Datagram Protocol |
| ICMP | Internet Control Message Protocol |
| IN | Inside |
| IPAM | IP Address Management |
| EOF | End Of File |

# TERMINOLOGIE

| | |
|---|---|
| Frontend | Část aplikace starající se o uživatelské rozhraní |
| Backend | Obslužná část aplikace zpracovávající příchozí požadavky |
| Setter | Nastavuje nebo aktualizuje hodnotu |
| Controller | Správa koncových bodů REST a rozhraní do logiky systému |
| Service | Implementace logiky systému |
| Repository | Obsluha úložiště |
| Query | Dotaz SQL |

# ÚVOD

S postupem času se zvyšuje počet dostupných metod sledování filmů a seriálů. Podle dostupných statistik alespoň 70 % uživatelů preferuje sledování v pohodlí domova [1]. Někteří nechtějí být limitováni přenosovou rychlostí prohlížečových streamů, a tak filmy či seriály stahují na svá zařízení. Takto stažené soubory multimédií následně mohou analyzovat pomocí automatizovaných programů, které je ucelí v přehledné knihovny a dodají i jejich informace z webových portálů. Uživatelé tak mohou snadno, a především rychle, přehrávat i nejnáročnější multimédia z hlediska přenosové rychlosti. Tento přístup však přináší jisté limitace, kterými se tato práce zabývá a usiluje o jejich vyřešení.

Cílem této diplomové práce je návrh a implementace informačního systému, který usnadní sdílení uživatelských hodnocení a knihoven multimédií mezi ostatními uživateli. Každý uživatel bude mít možnost navázat kontakt se svými nebo novými přáteli a společně tak vytvořit komunitu sledujících dnešní nabité tvorby. Dále budou mít dostupné statistiky ve formě grafů ukazujících postup sledování filmů a seriálů. Důležitou metrikou bude také správa jejich hodnocení. Díky rozšíření v doprovodné desktopové aplikaci bude uživatel schopen synchronizovat svá hodnocení a knihovny napříč zařízeními. Funkcionalita synchronizace bude tedy jedním z primárních milníků vytváření informačního systému. Uživatelská data tak budou bezpečně uložena na cloudu, který předejde jejich ztrátě a dovolí jejich sdílení.

První část práce je zaměřena na průzkum existujících informačních systémů věnujících se problematice správy a shromažďování multimediálních metadat. Přesněji se jedná o systémy poskytující dostupné informace o multimediálních záznamech a umožňujících správu jejich hodnocení.

Druhá část probírá návrh vývoje rozsáhlého informačního systému tvořeného z několika vzájemně komunikujících částí pro poskytnutí cílené funkčnosti. V této části jsou zpracovány funkční a nefunkční požadavky vývoje. Dále jsou předvedeny diagramy různých úseků návrhu.

Třetí část podrobněji seznámí s použitými technologiemi vývoje softwaru, kde jsou zmíněny i nástroje používané pro zvýšení bezpečnosti a usnadnění procesu nasazování produkčních verzí systémů.

Čtvrtá část se věnuje samotné implementaci navrhovaného informačního systému a všech jeho komponent. Implementace objasňuje způsob testování a profilování systému, jejichž cílem je nalézt kritické úseky před celkovým nasazením systému. Kapitoly také tvoří tři hlavní milníky dosažené při implementaci.

První milníkem se rozumí správa ukládaných dat a jejich význam v rámci celého systému. Další blíže seznámí s postupem synchronizace, která pracuje na základě různých režimů, kde každý z režimů obsahuje doplňkové možnosti. Posledním milníkem je pokryt princip sociální interakce mezi uživateli a jeho dopad na využití systému.

Pátá část provede čtenáře využitím vytvořeného informačního systému. Bude tak ukázán životní cyklus uživatele systémem ve skutečných případech na reálných datech.

Závěrečná kapitola je zaměřená na průběžnou integraci systému a nasazení produkční verze. Obsahem nasazení je rozsáhlý postup nasazení systému od obdržení vlastního doménového jména až po automatizaci nasazování v prostředí kontejnerů na virtuálním serveru.

Tato diplomová práce nejvíce přispěje vývojářům na projektech s využitím podobných technologií, neboť obsahuje podrobné informace z hlediska návrhu a implementace. Práci také ocení vývojáři při tvorbě workflow pro průběžnou integraci a nasazování. Výsledná webová aplikace je především cílena na uživatele se zájmem v hodnocení filmů a seriálů. Doprovodná desktopová aplikace slouží zručnějším jedincům z ohledu manipulace se softwarovými nástroji a nadšence do správy vlastních multimediálních dat.

Správou multimediálních dat se již zabývala vlastní bakalářská práce „Správa multimediálních metadat“, jejíž součástí byla tvorba desktopové aplikace pro centralizovanou správu souborových multimediálních metadat. Aplikace nabízela možnost identifikace filmů, seriálů a jiných dat. Vybrané záznamy následně obohacuje o data získaná z internetových databází. Další vlastností byla také přenositelnost, malá režie ze strany uživatele a úsporné ukládání dat. Aplikace je v této práci využito obohacením již zavedené funkčnosti o novou, a tak vytvořením spolupracujícího informačního ekosystému multimediální správy.

# 1 REŠERŠE SOUVISEJÍCÍCH SYSTÉMŮ

Pod slovy rešerše rozumíme prohledání dostupných informačních zdrojů, díky kterým je literatura shromážděna, prostudována a informativní podobou přednesena. Pomocí rešerše dokážeme utřídit různé poznatky a získat tak čistější vizi na aktuálně řešenou problematiku. Tato sekce se věnuje popisu souvisejících informačních systémů a aplikací. [2]

Jedinou způsobilou aplikací pro navrhovaný informační systém je již vytvořená aplikace v rámci vlastní bakalářské práce „Správa multimediálních souborů“. Aplikace nese název Pallidium, který je stejný s navrhovaným systémem, pro dodržení konzistence názvosloví vytvářeného ekosystému aplikací.

Informačním systémům je v této sekci dáván větší důraz díky přímé interakci s navrhovaným systémem. Podobně jako u aplikací byly zástupci nalezeny pomocí internetových zdrojů. Hlavním cílem všech nalezených systémů je udržovat aktuální informace o multimediálních záznamech. Systémy se od sebe primárně odlišují jejich zaměřením na daný typ záznamů. Většina také obsahuje API pro přístup k datům nebo provádění vzdálených akcí s uživatelským účtem. Přístup je však omezen pouze na registrované uživatele a za pomocí přístupového klíče. Postup získání může být, jak jednoduchý vytvořením pomocí jednoho tlačítka, tak složitý kvůli formulářům a žádostem.

## 1.1 Pallidium

Pallidium je aplikace pro správu a organizaci souborových metadat. Práce na aplikaci započala v roce 2020 a byla dokončena o dva roky později v roce 2022. Aplikace byla původně vyvíjena pro účely bakalářské práce „Správa multimediálních souborů“ a již při jejím vývoji byly vytvořeny návrhy na její zasazení do většího ekosystému spolupracujících aplikací. [3]

Změny provedené na aplikaci od jejího dokončení:

- opravy kritických chyb,
- migraci na Java 11,
- migraci z Ant na Maven,
- migraci na Java 19 s OpenJFX 19,
- oddělení od knihovny pro TMDB API verze 3.

Hlavním účelem aplikace je centralizovaná správa souborových metadat. Souborová metadata jsou získávána ze souborů pomocí nástroje FFmpeg, který provede analýzu požadovaného souboru a vrátí všechny informace v čitelném formátu JSON. Taková data obsahují informace

o stopách videa, zvuku a titulků. Data jsou následně zpracována a aplikací uložena v souborové databázi SQLite. Takto uložené soubory s databází poskytují jednoduchou přenositelnost a transparentnost dat. Databáze tak může být nechána přímo v místě spravovaných souborů.

Aplikace shlukuje metadata stejného typu v knihovnách. Každá vytvořená knihovna má svůj soubor s databází, ve kterém databáze udržuje nastavení knihovny a její data. Aplikace nabízí tvorbu knihoven typu: filmy, seriály, hudba, obrázky a knihy. Některé z knihovních typů obsahují alternativy s jiným pojmenováním, ale v jádru stejnou funkčností. Již vytvořené knihovny lze načíst z jakéhokoliv umístění v počítači nebo na vyměnitelných discích. V případě odpojení takového disku není knihovna přístupná a bude opět zpřístupněna po jeho připojení.

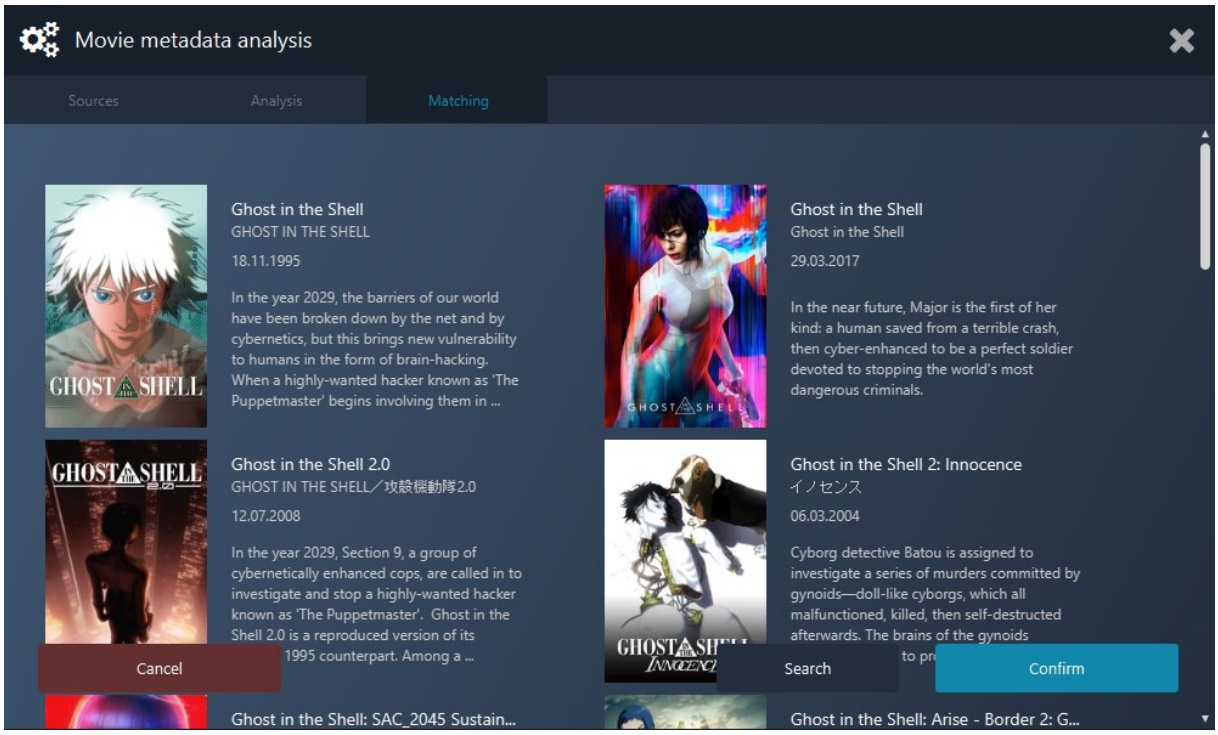
Po vytvoření vybrané knihovny může uživatel upravit její nastavení a přidat umístění v úložišti, kde by měla aplikace provést souborovou analýzu. Taková analýza je spuštěna tlačítkem „Refresh“ v ovládacím panelu příslušné knihovny. Zadaná paměťová úmístění jsou prohledána, nalezené soubory zanalyzovány a získaná metadata uložena do databáze knihovny. Vytvořené záznamy nesou jméno svého souboru, ze kterého vznikly. Knihovna tak zobrazí pouze názvy a poskytne detailní informace o původním souboru.

Dále následuje získávání dodatečných metadat z internetových sborníků. Aplikace využívá přímého spojení s TMDB API pomocí přístupového klíče. Přístup k TMDB je využit pro získání dodatečných metadat popisujících nikoliv soubor, ale multimediální dílo uložené v takovém souboru.

Dle dostupných informací od administrátorů TMDB, je přístup limitován pouze na danou IP adresu a ne na klíč jako takový. V původní práci pro vytvoření aplikace bylo milně usouzeno, že nemůže být volně přístupný uživatelům cílové aplikace. Předpoklad byl takový, že se jedná o pojistku proti zneužití daného klíče, který by mohl kdokoliv volně používat, a nepřiměřeně tak zvyšovat používání mimo cílený systém. Statistická data pro takovou aplikaci by byla zavádějící a mohla vyústit v zablokování přístupového klíče. [4]

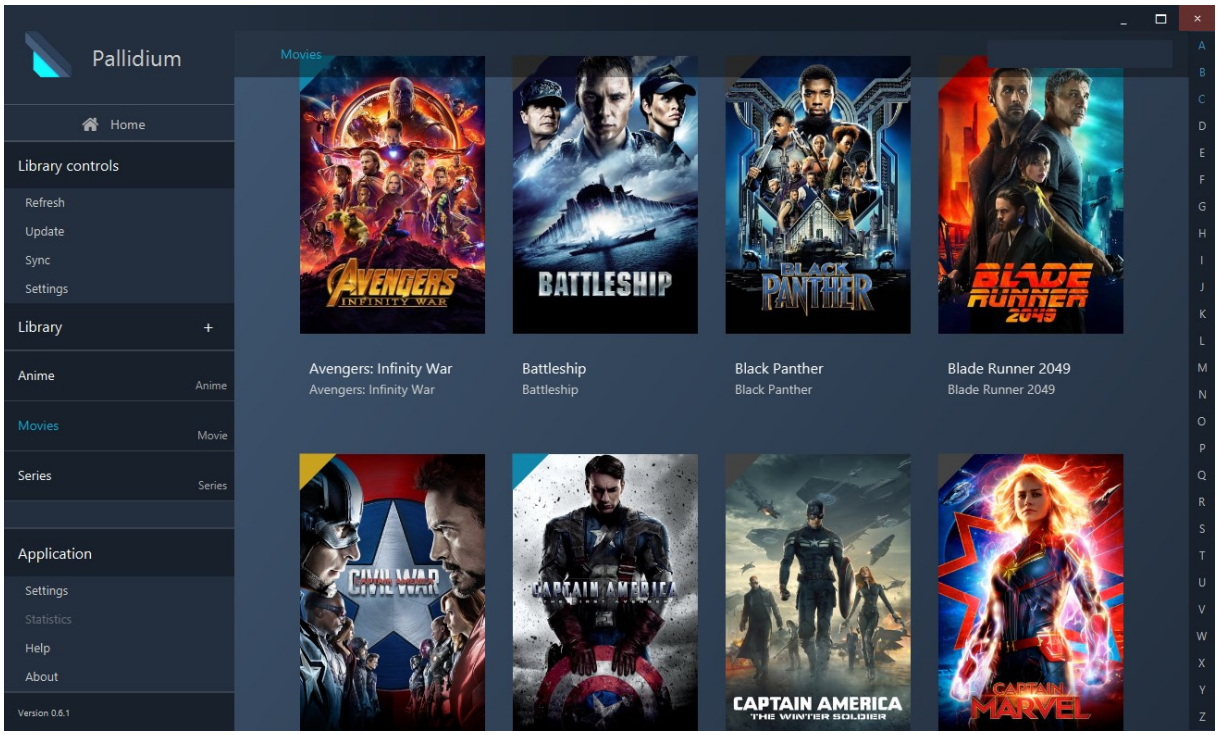
Analýza obsahuje dva režimy:

- **Automatický režim** – vyhledá záznam pomocí TMDB podle názvu souboru a vybere záznam s nejbližší shodou. V případě správného a výstižného pojemnování souborů se jedná o dobrou a rychlou možnost. Záporem jsou situace, kdy první možnost není chtěnný záznam nebo záznam není vůbec nalezen. V takovém případě je celý režim spíše nepříjemností a není doporučován.
- **Manuální režim** – poskytuje volbu z nabízených výsledů, které byly získány hledáním. Každý záznam obsahuje obrázek a popis, které jsou užitečné pro jeho správnou identifikaci. Chtěnný záznam stačí jednoduše vybrat a jeho data budou stažena.




*Obrázek 1: Výběr záznamu pro stažení metadat v aplikaci Pallidium*

V souhrnu aplikace nabízí: tvorbu knihoven, načtení existujících knihoven, správu knihoven, souborovou analýzu, metadatovou analýzu, přehled posledních změn v knihovnách, zobrazení záznamů v knihovně, detail záznamu knihovny s podrobnými informace o souboru, možnost spuštění souboru přímo z aplikace, pokročilé nastavení chtěnných metadat z TMDB, základní podporu označení záznamů za shlédnuté a další.




*Obrázek 2: Přehled záznamů ve filmové knihovně aplikace Pallidium*

## 1.2 Informační systémy pro shromažďování hodnocení

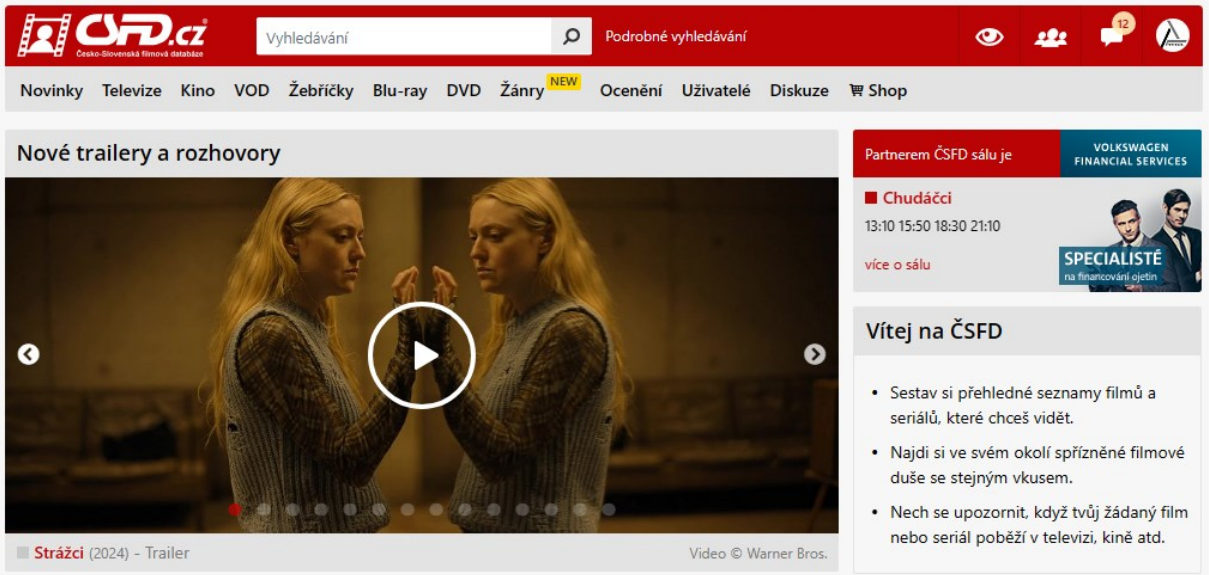
### 1.2.1 Česko-Slovenská filmová databáze (ČSFD)

ČSFD je globální databáze poskytující informace o různých multimediálních záznamech a služby hodnotitelského fóra. Po registraci uživatelského účtu je možné přidávat hodnocení filmů nebo seriálů a sdílet své názory pomocí diskusí. Projekt byl spuštěn slovenským novinářem Martinem Pomothym v roce 2001. Databáze neobsahuje pouze české a slovenské záznamy, jak napovídá název, ale poskytuje informace o všeobecné globální tvorbě. Každý záznam filmů a seriálů obsahuje procentuální hodnocení získané jako průměr ze všech hodnocení uživatelů. Kromě detailnějších informací o záznamu jsou obsahem podrobné informace o jednotlivých osobách podílejících se na tvorbě a uživatelské komentáře v diskusích. [5]

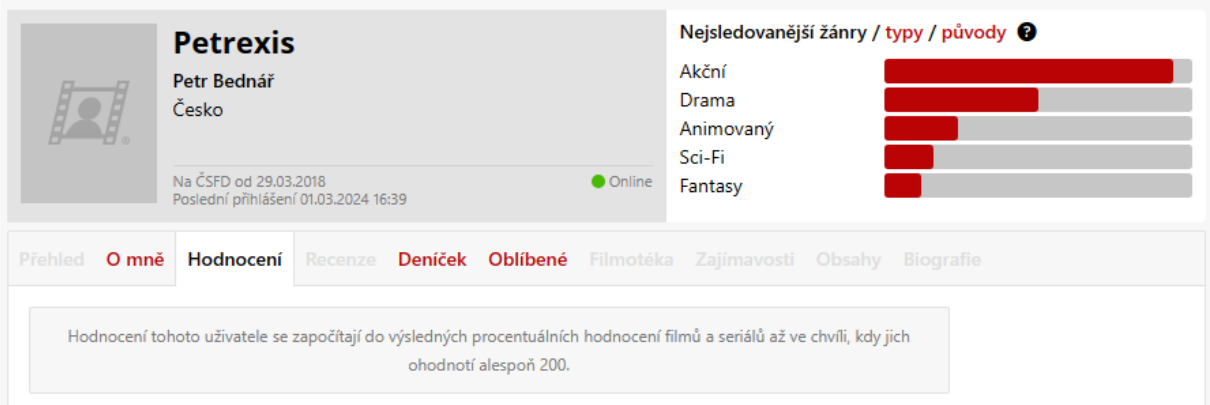
ČSFD poskytuje informace z kategorií:

- filmy,
- seriály,
- žánry,
- ocenění,
- kino programy,
- VOD produkce,
- žebříčky,
- vydání na blu-ray,
- vydání na DVD,
- tvůrci,
- jiné. [6]

Webová stránka si zakládá na přehlednosti a korektnosti poskytovaných informací. Její obsah je komunitně získávaný, aktualizovaný a překládaný. Většina informací má přidaný zdroj a je podrobena kontrole týmem specializovaných editorů. O hlavní vytváření obsahu se dle dostupných informací stará tým redaktorů ČSFD složený z 10 osob. Komunitně přidaný obsah je schvalován 8 lidmi. Hlavní stránka je složená z nejnovějších a nejoblíbenějších informacích z různých úseků poskytovaných informací.




*Obrázek 3: Hlavní webová stránka ČSFD*




*Obrázek 4: Uživatelský profil na webu ČSFD*

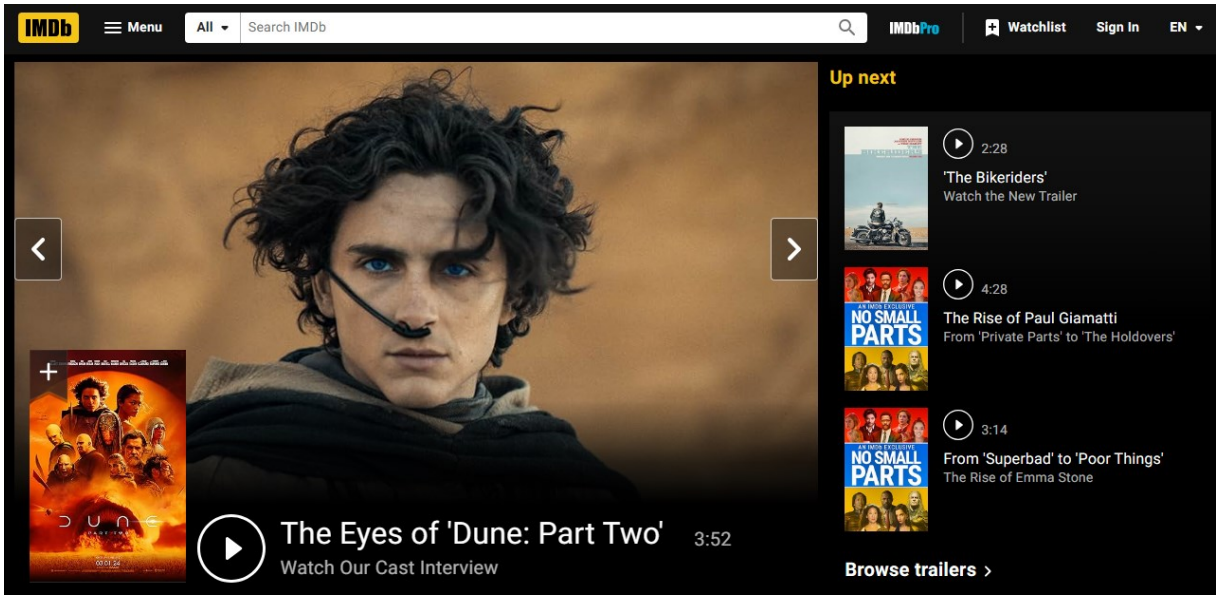
Uživatelský profil obsahuje pouze základní graf nejsledovanějších žánrů a výpis hodnocení jednotlivých záznamů. Systém hodnocení je realizován jednoduchým způsobem 6 stupňů hodnocení, kde nejhorší je tzv. „odpad“ vizuálně označený jako palec dolů a dalších pět stupňů jsou hvězdičky od jedné do pěti. Systém tak automaticky vede ohodnocený záznam jako shlédnutý. Záznamy lze přidat do seznamu „Chci vidět“ nebo osobní filmotéky. Filmotékou je myšleno vytvoření prodeje/shánění fyzické nebo digitální nahrávky záznamu. ČSFD neobsahuje systém přátelství uživatelů, ale nabízí přidání cíleného uživatele do oblíbených. U označených uživatelů se následně zobrazuje jejich nedávná aktivita. Kromě uživatelských notifikací jsou zobrazovány i informace o sledovaných záznamech a aktuality ČSFD. Uživatelské nastavení dovoluje sdílení stavu aktivity, osobních informací, jazykové lokalizace, avataru, notifikací a aktivních sezení.


*Obrázek 5: Filmový profil na webu ČSFD*

Filmy a seriály obsahují informace o zúčastněncýh tvůrcích a přehled obsahu z různých kategorií. Seriály mají na úvodní kartě rychlý přehled všech sezón a poté detailnější přehled epizod v detailu sezóny.

### 1.2.2 Internet Movie Database (IMDb)

IMDb je největší filmovou databází na světe s vytvořením před více než 30 lety. Projekt vznikl v roce 1990 sloučením dvou seznamů (seznam hereček, seznam hodnocení filmů) a vytvořením tak globální databáze, která mimo jiné poskytovala informace o dalších zúčastněných osobách. V roce 1993 byla stránka přesunuta na WWW a k dnešnímu datu je největší databází filmů a seriálů na světě. Českou alternativou jsou databáze ČSFD, Kinobox nebo FDb. [7]



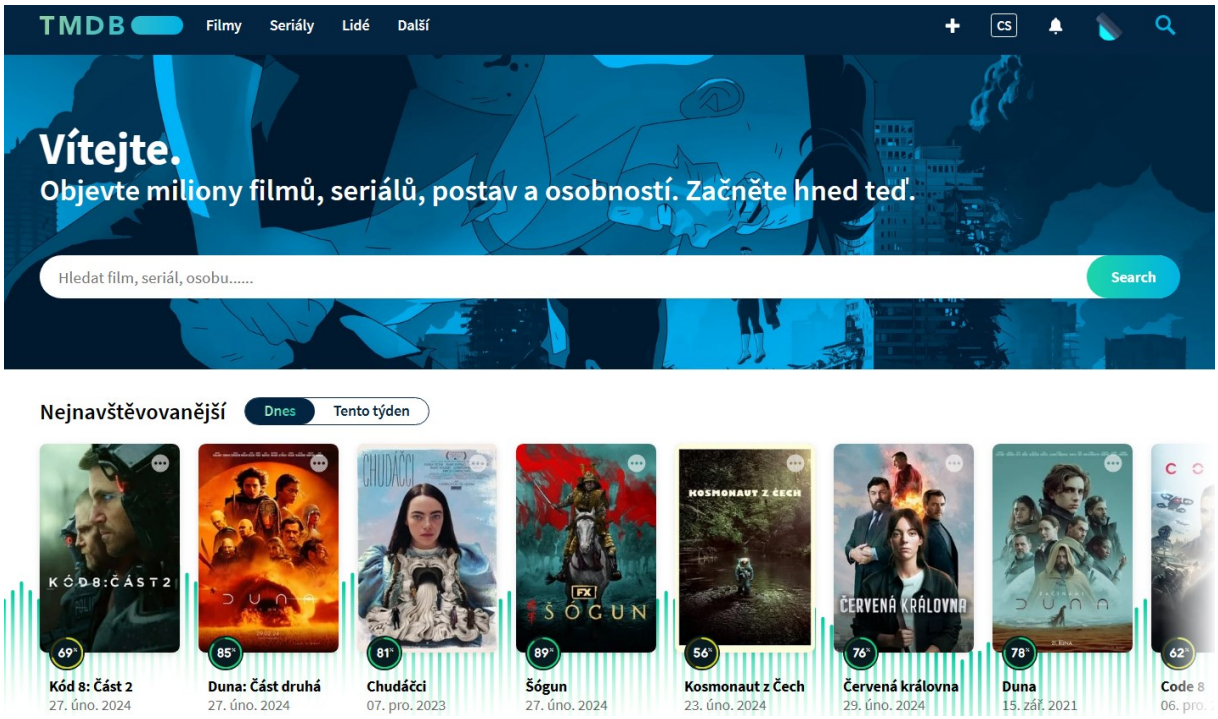

*Obrázek 6: Hlavní webová stránka IMDb*

Díky své globální popularitě bylo při tvorbě práce důležité poskytnout průstup do takové databáze. Tvořená aplikace tak poskytuje odkaz na záznam z této databáze.

IMDb poskytuje API pro získávání metadat. Zkušební verze trvá jeden měsíc a nelze ji obnovit. Plný placený přístup k API stojí 150000 amerických dolarů, což je dle ceny ze dne 1. 3. 2024 přibližně 3500000 Kč. Tato cena je pouze platbou předem a neobsahuje platby za využitý datový limit. Za každý využitý GB dat se platí dodatečných 9,3 amerických dolarů neboli přibližně 220 Kč. [8]

### 1.2.3 The Movie Database (TMDB)

TMDB je populární komunitní databáze pro filmy a seriály. Od svého spuštění v roce 2008 je aktivně aktualizována a obsahuje důvěryhodné informace o požadovaných záznamech. Databáze obsahuje záznamy z globální tvorby a lokalizace těchto záznamů v různých jazycích.




*Obrázek 7: Hlavní webová stránka TMDB*

TMDB poskytuje bezplatnou API pro přístup k metadatům a hostovaným obrázkům osob i plakátů. Dle dostupných oficiálních informací podporuje 39 lokalizací dat. Data TMDB jsou používána ve více než 180 státech a jejich API měsíčně obslouží přes 3 miliardy dotazů. API poskytuje detailní informace nejen filmů a seriálů, ale i tvůrců. Na rozdíl od již zmíněných databází (ČSFD, IMDb) není API zpoplatněna. ČSFD aktuálně nepodporuje žádnou API pro získávání informací o záznamech. IMDb je z hlediska obrovských cen vhodná pouze pro velké společnosti s vysokou monetizací. TMDB je tedy vynikající API pro různorodé projekty bez nutnosti placení. [9]

# 2 ANALÝZA

Cílem softwarové analýzy je zodpovědět klíčové otázky a prozkoumat technologické aspekty před samotným zahájením vývoje softwaru. Tímto způsobem se minimalizuje vznik rizik a umožňuje tak včasnou identifikaci problémů. Během analýzy je klíčová efektivní komunikace se zadavatelem pro zajištění, že navrhovaný software plně odpovídá jeho očekáváním. Prvním krokem návrhu je analýza požadavků. Cílem analýzy je specifikace funkčních a nefunkčních požadavků, které určují, co bude navrhovaný systém umět a jak dobře/rychle to musí zvládnout. Prací analytika je udržet požadavky srozumitelné, nesporné, obsáhlé, kompletní a stručné. Účelem požadavků je určit hlavní cíl práce, a tak usnadnit práci při jeho návrhu. Návrh je posléze ověřen na úplnost vůči požadavkům a zda splňuje nastavená omezení. [10]

Požadavky dělíme na:

- **Funkční požadavky** – vyjadřují chování navrhovaného systému určením vstupních a výstupních podmínek.
- **Nefunkční/Doplňující požadavky** – vykazují hlavní jakostní znaky:
    - **Použitelnost** – se zabývá lidskými faktory, jako je estetika, snadné učení, snadné použití a podobně.
    - **Spolehlivost** – adresuje četnost a závažnost možných selhání, obnovitelnost systému a přesnost.
    - **Výkon** – popisuje rychlost systému, dobu odezvy, vytížení systému a další.
    - **Udržitelnost** – řeší, jak těžké je udržet systém a další vlastnosti spojené s jeho budoucím provozem. [11]

## 2.1 Webová aplikace

V této sekci budou vypsány navržené funkční a nefunkční požadavky hlavního systému. Do požadavků jsou zahrnuty, jak požadavky na frontend, tak požadavky na backend. Bylo tak učiněno z důvodu přehlednosti a ovlivňování navrhovaných částí systému podobným způsobem.

### 2.1.1 Funkční požadavky

FR01: Systém umožní vytváření uživatelských účtů.
FR02: Systém umožní ověření účtu nově registrovaných uživatelů.
FR03: Systém umožní přihlášení a odhlášení uživatelů.
FR04: Systém umožní obnovu zapomenutého hesla účtu.
FR05: Systém umožní správu uživatelských informací.
FR06: Systém umožní změnu přihlašovacího hesla.
FR07: Systém umožní správu aktivních relací uživatele.
FR08: Systém bude zobrazovat přehled hodnocení uživatele.
FR09: Systém bude také zobrazovat původ hodnocení.
FR10: Systém bude zobrazovat přehled posledních hodnocení uživatele.
FR11: Systém bude zobrazovat přehled vytvořených knihoven multimédií.
FR12: Systém bude zobrazovat přehled nedávno přidaných záznamů knihoven.
FR13: Systém bude zobrazovat obsah knihoven multimédií.
FR14: Systém umožní vytvoření a odebrání knihovny.
FR15: Systém umožní změnu názvu knihovny.
FR16: Systém umožní změnu lokalizace knihovny.
FR17: Systém umožní sdílet přístup ke knihovně přátelům.
FR18: Systém umožní změnu oprávnění přístupu ke knihovně.
FR19: Systém umožní vytvoření přístupového klíče k vybrané knihovně.
FR20: Systém umožní vzdálenou správu knihovny dle přístupového klíče.
FR21: Systém umožní vzdálenou správu hodnocení pomocí přístupového klíče.
FR22: Systém bude zobrazovat informace:
- FR22.1: filmů,
- FR22.2: seriálů,
- FR22.3: sezón seriálů,
- FR22.4: epizod seriálů.

FR23: Systém umožní změnu hodnocení:
- FR23.1: filmů,
- FR23.2: sezón seriálů,
- FR23.3: epizod seriálů.

FR24: Systém umožní vyhledání uživatelů systému.
FR25: Systém bude zobrazovat profily uživatelů.
FR26: Systém bude zobrazovat informace o spřátelených uživatelích.
FR27: Systém bude zobrazovat informace o příchozích a odchozích žádostech o přátelství.
FR28: Systém umožní odeslání a zrušení žádostí o přátelství.
FR29: Systém umožní odebrání spřátelených účtů.
FR30: Systém bude zobrazovat aktuální stav připojení doprovodné aplikace.
FR31: Systém umožní změnu jazyka prostředí.

### 2.1.2 Nefunkční požadavky

NFR01: Systém bude implementován jako webová aplikace s využitím architektury REST.
NFR02: Backend systému bude naprogramován za použití jazyka Java s využitím frameworku Spring Boot.
NFR03: Frontend systému bude naprogramován za použití jazyka TypeScript s využitím knihovny React.
NFR04: Frontend bude nasazen za pomoci Nginx serveru.
NFR05: Pro ukládání dat bude využito databázového systému PostgreSQL.
NFR06: Struktura dat databázového systému bude optimalizována z hlediska eliminace duplicit.
NFR07: Metadata filmů budou ukládána v jazyce:
- NFR07.1: angličtina,
- NFR07.2: čeština.

NFR08: Metadata seriálů budou ukládána v jazyce:
- NFR08.1: angličtina,
- NFR08.2: čeština.

NFR09: Metadata sezón seriálů budou ukládána v jazyce:
- NFR09.1: angličtina,
- NFR09.2: čeština.

NFR10: Metadata epizod seriálů budou ukládána v jazyce:
- NFR10.1: angličtina,
- NFR10.2: čeština.

NFR11: Autentizace v rámci relace uživatele bude fungovat na základě JWT.
NFR12: Koncové body systému budou dokumentovány promocí Swagger UI.
NFR13: Systém povolí přístup pouze k povoleným koncovým bodům.
NFR14: Koncové body budou zabezpečeny ověřením konkrétního uživatele a jeho přístupových práv.
NFR15: Metadata filmů a seriálů budou získána pomocí TMDB API.
NFR16: Uživatelské prostředí systému bude podporovat jazyky:
- NFR16.1: angličtina,
- NFR16.2: čeština.

## 2.2 Rozšíření desktopové aplikace

Kapitola obsahuje funkční a nefunkční požadavky na rozšíření již existující aplikace správy souborových metadat. Požadavky tedy popisují pouze doplňující vlastnosti nutné ke správné činnosti navrhovaného systému.

### 2.2.1 Funkční požadavky

FR01: Aplikace umožní změnu hodnocení:
- FR01.1: filmů,
- FR01.2: sezón seriálů,
- FR01.3: epizod seriálů.

FR02: Aplikace umožní přidání synchronizačních klíčů knihoven.
FR03: Aplikace umožní ověření platnosti synchronizačních klíčů.
FR04: Aplikace bude zobrazovat cílovou knihovnu pro synchronizaci.
FR05: Aplikace umožní synchronizaci hodnocení:
- FR05.1: filmů,
- FR05.2: sezón seriálů,
- FR05.3: epizod sezón.

FR06: Synchronizace bude podporovat různé režimy.
FR07: Aplikace bude schopna synchronizovat hodnocení z aplikace do webového systému.
FR08: Aplikace bude schopna synchronizovat hodnocení z webového systému do aplikace.

### 2.2.2 Nefunkční požadavky

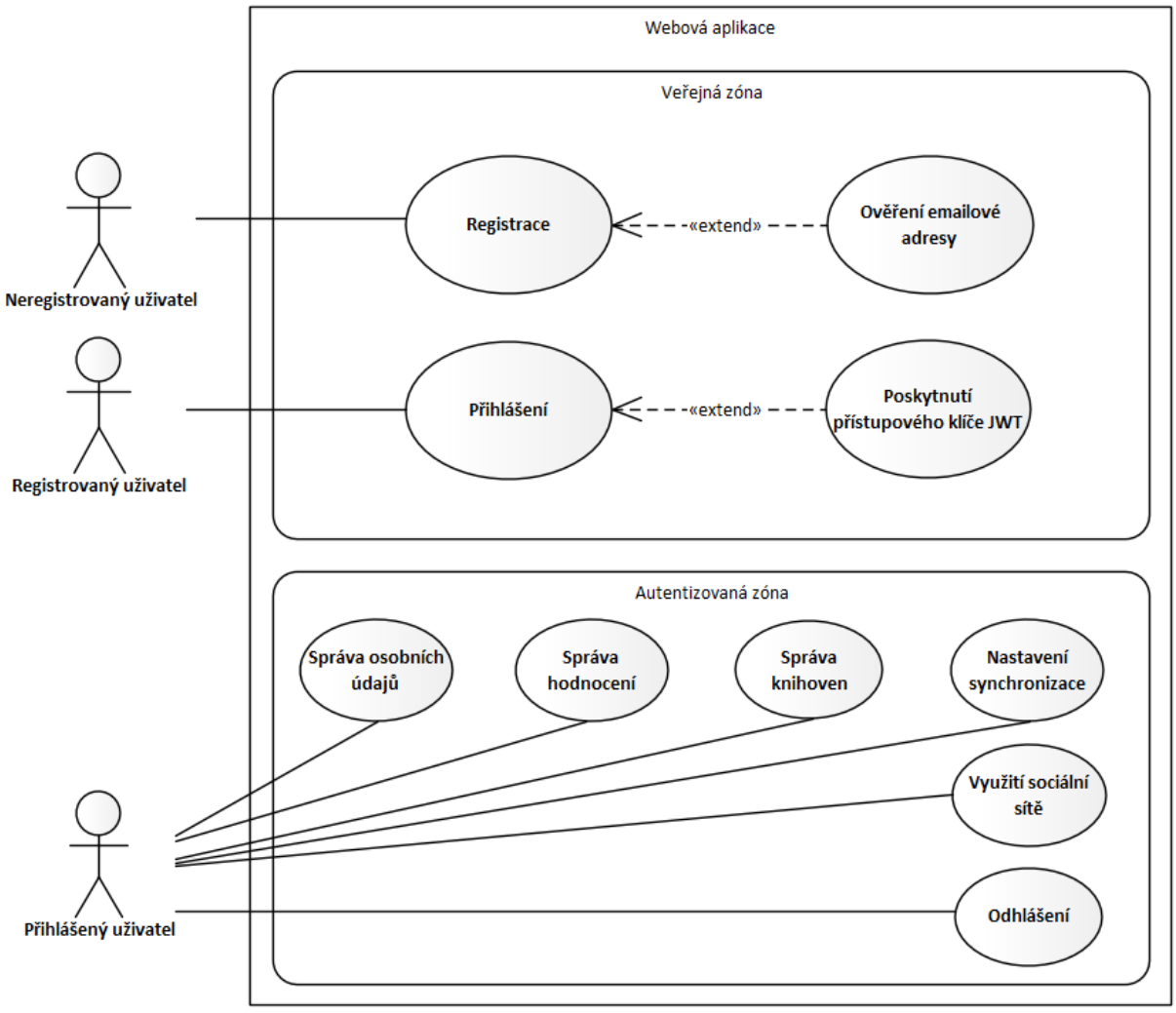
NFR01: Funkčnost bude přidána jako rozšíření již existující aplikace Palladium.
NFR02: Aplikace bude spustitelná na operačních systémech Windows 10 a Windows 11.
NFR03: Pro vývoj aplikace bude použit programovací jazyk Java.
NFR04: Při vývoji grafického rozhraní bude použit programovací jazyk JavaFX.
NFR05: Pro ukládání dat aplikace bude využita databáze SQLite.
NFR06: Synchronizace bude možná pouze pro záznamy s přiřazeným TMDB identifikátorem.

## 2.3 Modely případů užití

Případy užití jsou funkce poskytované systémem, které mohou být využity jedním nebo více aktéry. Aktéři interagují se systémem z vnější a požadují vykonání požadované funkčnosti. Pro modely platí pravidla taková, že případ užití je vždy iniciován aktérem a je napsán z jeho pohledu. Vyobrazené postavy v modelu symbolizují aktéry a jejich napojení představují interakci s funkcemi systému. Vazba „extend“ rozšiřuje vybraný případ o další chování a můžeme ho chápat jako takový portál, který obsahuje místa pro přechod do jiných případů. [12]

### 2.3.1 Základní model přístupu

Model popisuje základní užití systému z pohledu aktérů: neregistrovaný uživatel, registrovaný uživatel a přihlášený uživatel. Prostředí navrhovaného systému je opticky rozděleno na části veřejné a autentizované zóny, do které se uživatel dostane po přihlášení. Model je maximálně zjednodušen a samotný přístup do systému obsahuje pouze případy využití systému související s hlavními cíli práce.

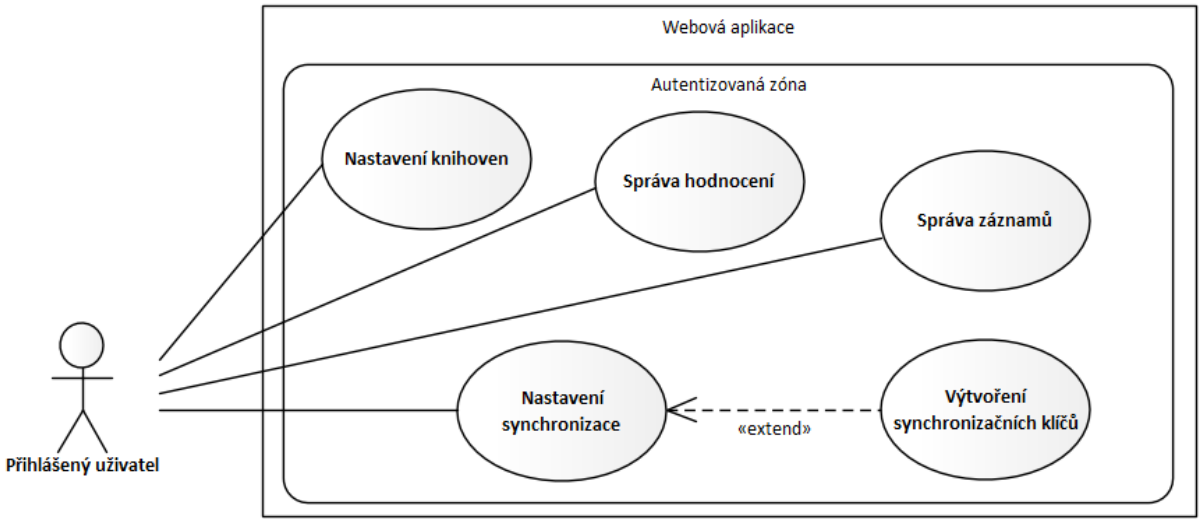



*Obrázek 8: UC model základního přístupu*

### 2.3.2 Modely správy knihoven

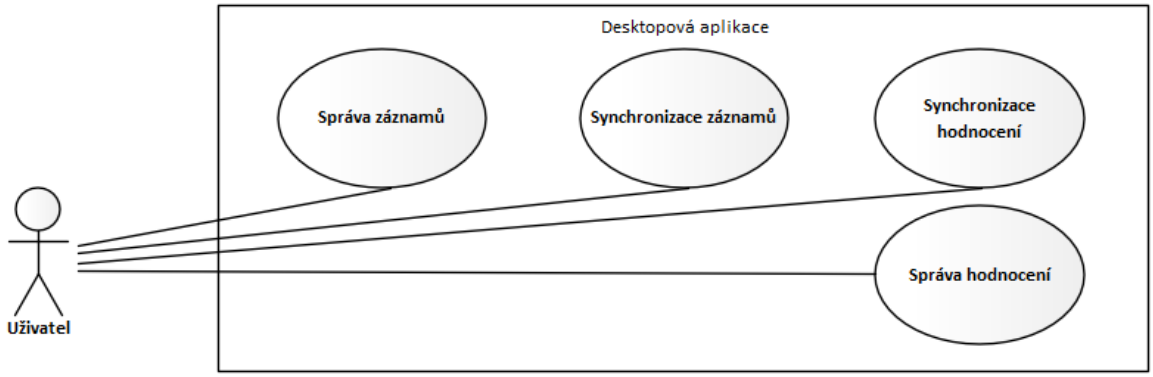
Následující modely popisují případy užití z pohledu přihlášeného uživatele webové aplikace a uživatele využívajícího desktopovou aplikaci. Popisované případy přímo odráží požadovanou funkčnost ve stanovených požadavcích a ukazují společné vlastnosti mezi aplikacemi. Veřejná zóna je na tomto modelu návrhem pro volně přístupné API využívané desktopovou aplikací pro účely synchronizace obsahu. Autentizovaná zóna dále obsahuje tvorbu synchronizačních klíčů, které jsou potřebné pro úspěšné dokončení synchronizace z aplikace.

#### 2.3.2.1 Webová aplikace




*Obrázek 9: UC model správy knihoven ve webové aplikaci*

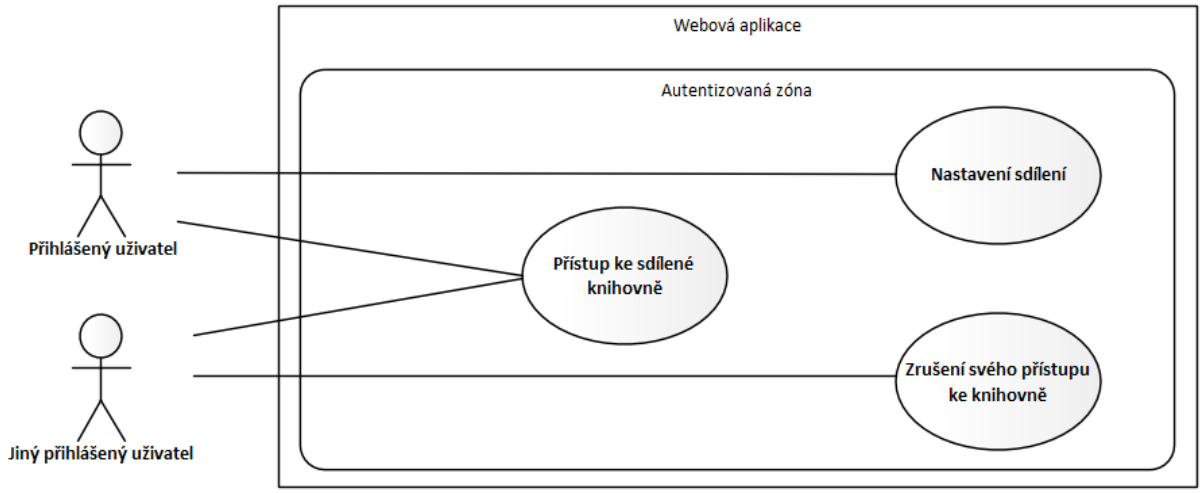
#### 2.3.2.2 Desktopová aplikace




*Obrázek 10: UC model správy knihoven v desktopové aplikaci*

### 2.3.3 Model sdílení knihoven

Model nastiňuje možnost sdílení vytvořených knihoven ostatním spřáteleným uživatelům. Přihlášený uživatel, v tomto případě i vlastník knihovny, má absolutní moc nad správou své knihovny. Jeho účet má tak přístup ke sdílení knihovny svému příteli a správu nad takovým přístupem. Po nasdílení knihovny je dán cílovému příteli přístup a ke knihovně tak může volně přistupovat. Přístup ke knihovně může být kdykoliv odebrán vlastníkem knihovny nebo se od knihovny může sám odpojit.



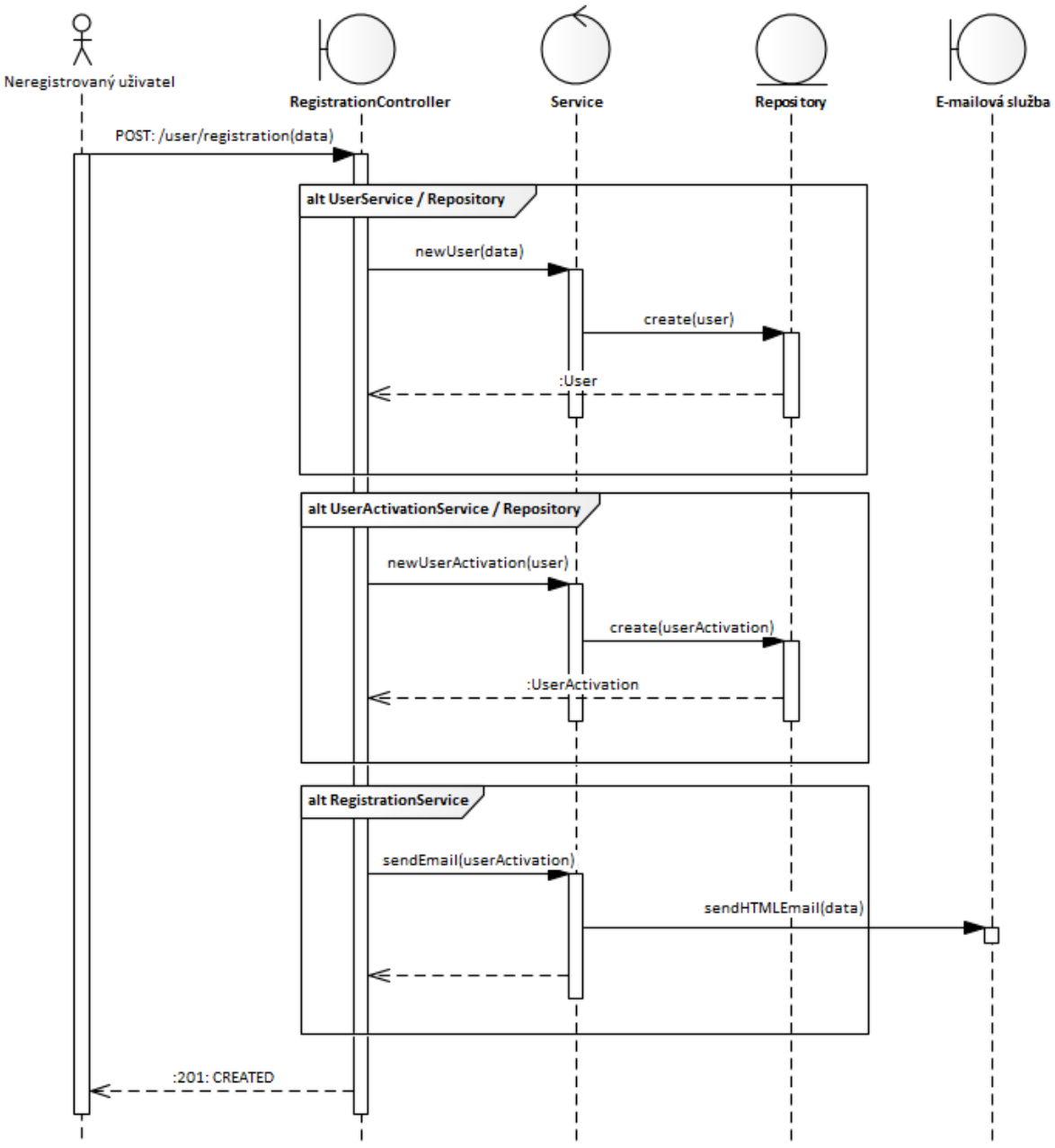

*Obrázek 11: UC model sdílení knihoven*

## 2.4 Sekvenční diagramy

Sekvenční diagramy znázorňují interakce životního cyklu jednotlivých komponent a aktérů napříč časem. Jedná se o nejbohatší a nejpružnější formu diagramů interakce. Interakční diagramy nejsou přímým přepisem případu užití a pouze ilustrují jejich chování v systému nad analytickými třídami. Interakce většinou začíná požadavkem uživatele, který je následně zpracován okrajovými komponentami. Svislá čára modelu se nazývá „čára života“ a symbolizuje pokrok v čase. Horizontální čáry ukazují volání napříč systémem, kde čáry vpravo zasílají nějaký požadavek a čáry vlevo vrací odpověď na takový požadavek. Cílení čar na sebe představuje zpracování potřebných dat přímo v místě obsloužení. Popsané obdélníky separují prostor a označují související interakce. [12]

### 2.4.1 Registrace uživatelského účtu

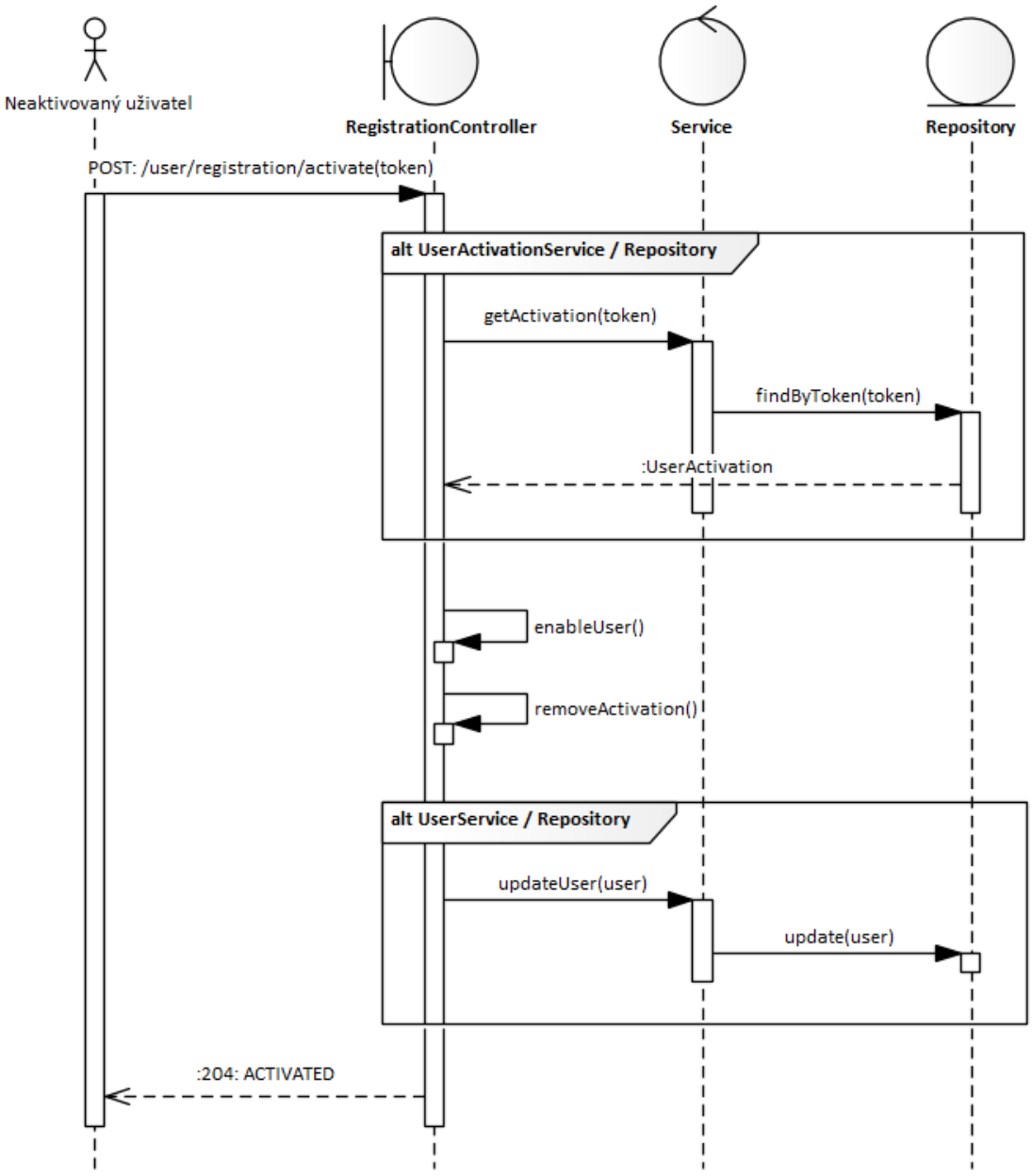
Diagram popisuje zpracování dotazu o registraci nového účtu. Registrace začíná dotazem na určený koncový bod registrace a zasláním informací pro nově vytvářený účet. Uživatelská služba se postará o vytvoření nového uživatele, čímž je zahájen aktivační proces. Aktivace se skládá z vytvořeného záznamu v databázi obsahujícím ověřovací klíč účtu a zasláním ověřovacího emailu na adresu registrovaného uživatele. Pomocí obsaženého odkazu si uživatel může ověřit a aktivovat svůj účet.




*Obrázek 12: Sekvenční diagram uživatelské registrace*

### 2.4.2 Aktivace uživatelského účtu

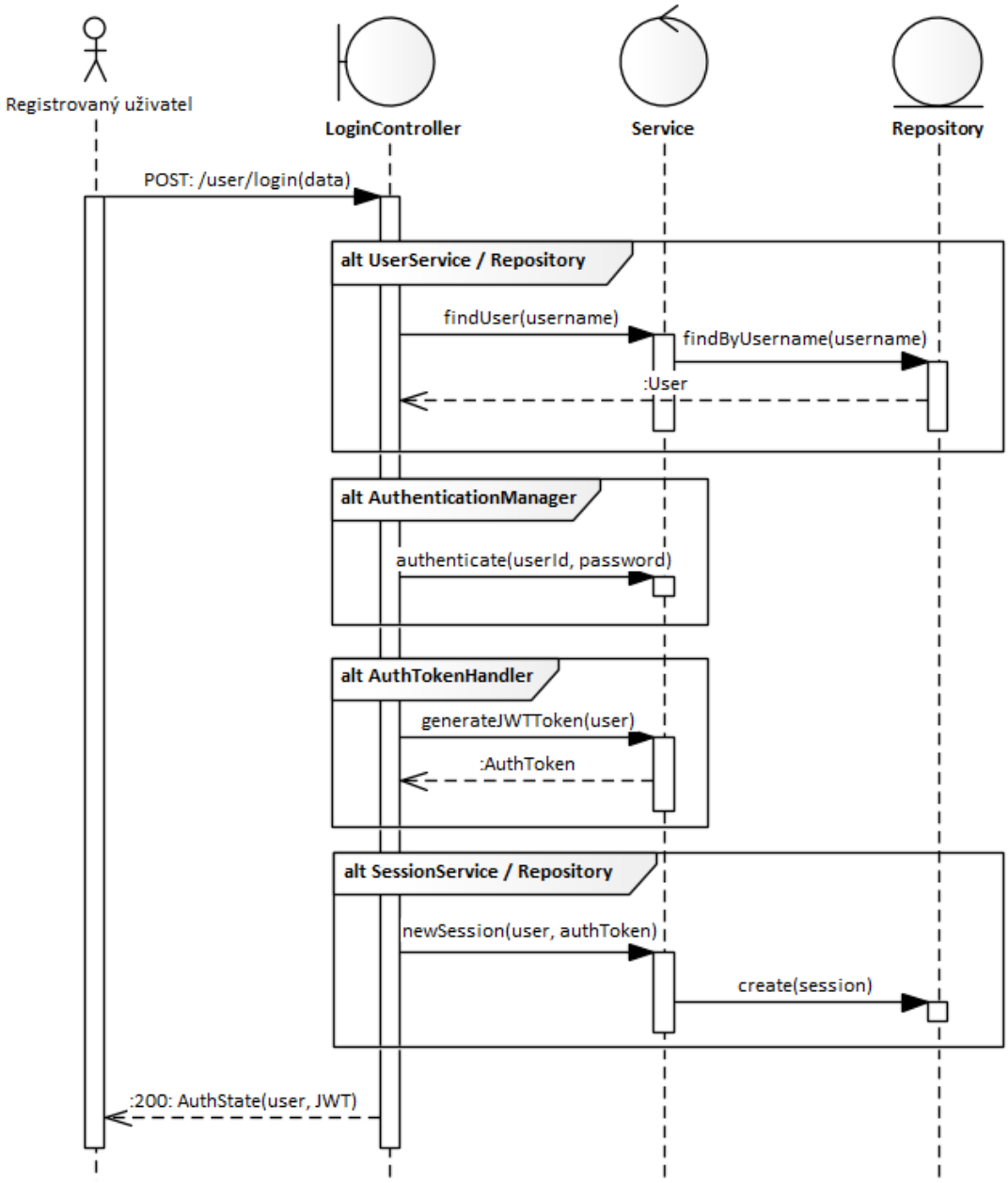
Aktivace účtu začíná obdržením emailu s aktivačním kódem. Přesněji se jedná o odkaz na frontend webové aplikace odkud je zavolán dotaz pro zpracování. V systému je nalezena příslušná aktivace uživatele pod kódem a uživatel je aktivován. S dokončením aktivace je kód z databáze odebrán.




*Obrázek 13: Sekvenční diagram aktivace uživatelského účtu*

### 2.4.3 Přihlášení registrovaného uživatele

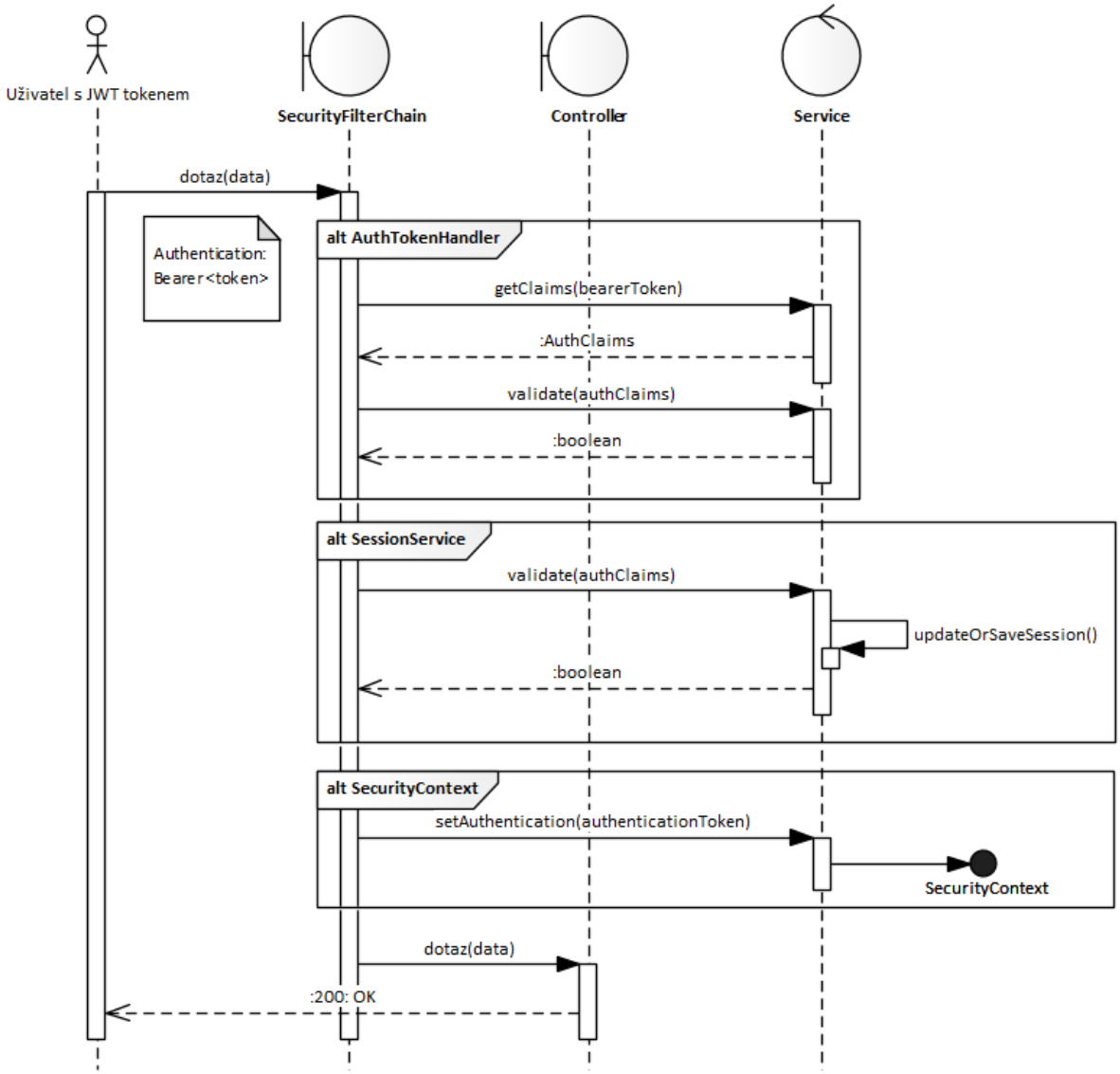
Pro zajištění přístupu k zabezpečením musí uživatel získat autentizační JWT token. Model níže popisuje funkčnost přihlášení pomocí zadaných přihlašovacích údajů, které jsou složeny z uživatelského jména a hesla. V prvním kroku je nalezen příslušný uživatelský účet pomocí poskytnutého přihlašovacího jména a následně jsou získané informace využity k autentizaci uživatele. Při úspěšné autentizaci je vytvořen nový přístupový JWT token a sezení uživatele. Vytvořené sezení je provázáno na vytvořený JWT token a slouží k monitorování nebo správě aktivních sezení.




*Obrázek 14: Sekvenční diagram přihlášení registrovaného uživatele*

### 2.4.4 Autentizace uživatele pomocí JWT

Autentizace probíhá při každém připojení ke koncovým bodům autentizované zóny. Každý dotaz nejdříve prochází řetězem filtrů, kde je ověřena identita vlastníka JWT tokenu a uživatel je přihlášen do systému pomocí bezpečnostní kontextu. Napříč systémem je následně přístupný přes účet uživatele ve funkcích @PreAuthorize nebo pomocí objektu Authentication. Prvním krokem autentizace je přečtení JWT tokenu a získání potřebných informací o připojovaném uživateli. Dále je token ověřen, zda již nevypršel. O další část ověření se stará třída service, která aktualizuje informace o přístupu v rámci sezení nebo vytvoří nové sezení. V tomto kroku jsou také automaticky čištěna propadlá sezení. Po dokončení procesu autentizace je uživatelská identita uložena v bezpečnostním a dochází k provedení požadovaného dotazu.



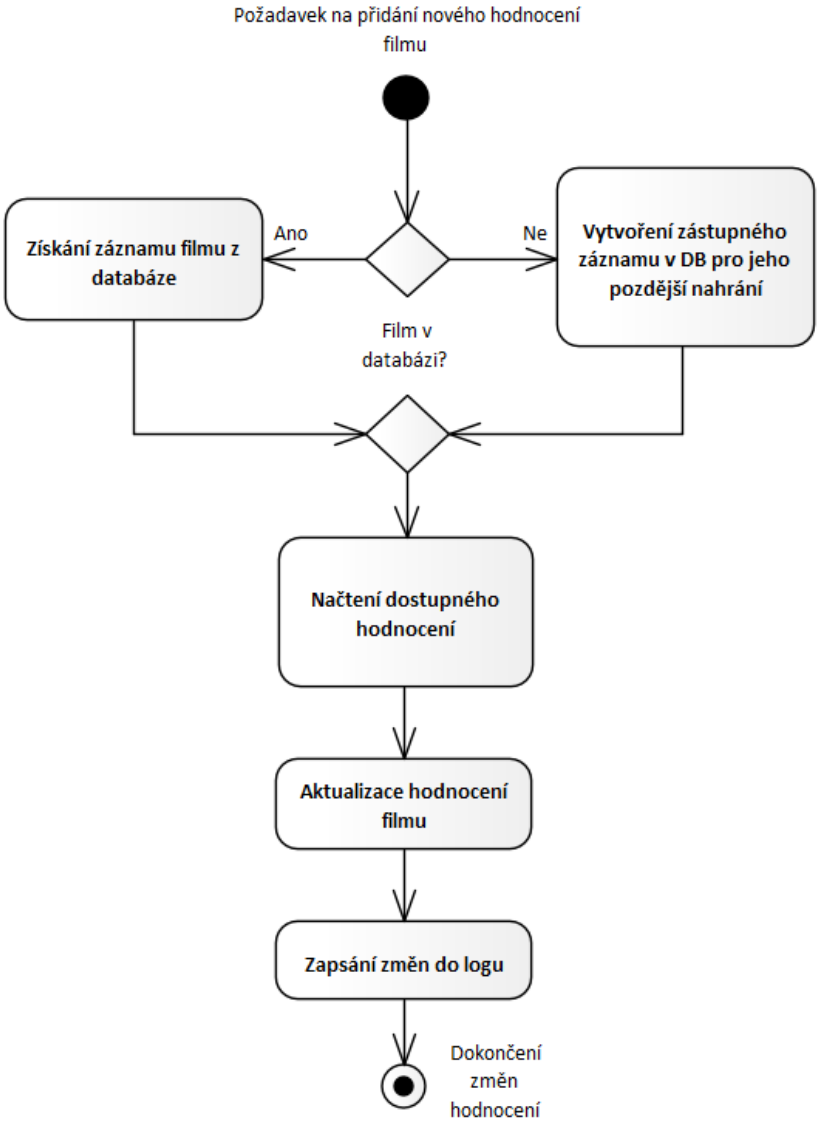

*Obrázek 15: Sekvenční diagram autentizace uživatele pomocí JWT*

## 2.5 Diagramy aktivit

Diagram aktivit je objektově orientovaný vývojový diagram. Vybraný proces systému je modelován pomocí aktivity, která se skládá z kolekce uzlů spojených hranami. Hrany představují vstupní a výstupní tok dat mezi uzly. Akční uzel volání můžeme chápat jako vykonávanou funkci, která má jeden vstup a jeden výstup. Použité diagramy obsahují dělení aktivity pomocí uzle rozhodnutí a následného spojení uzlem sloučení. [12]

### 2.5.1 Změna hodnocení filmu

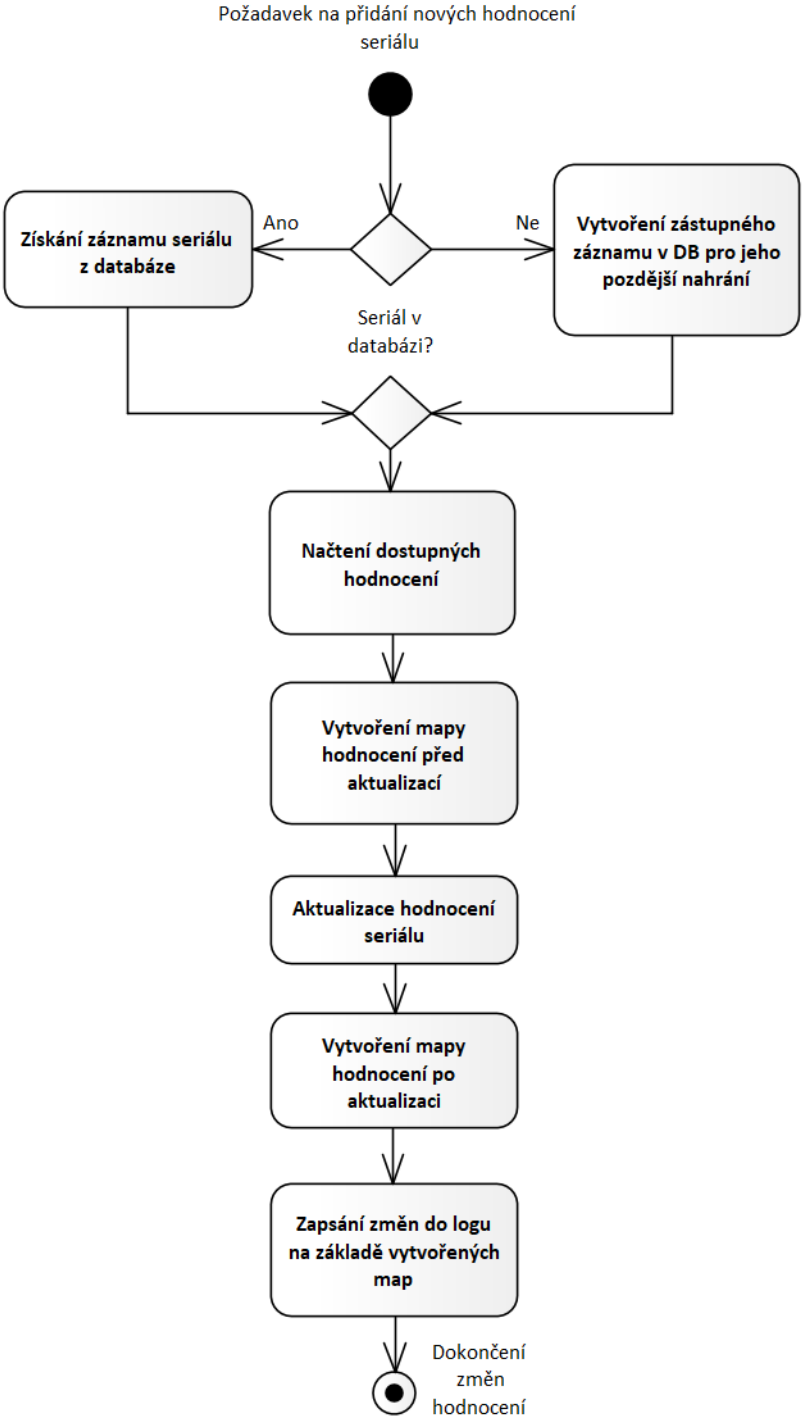
Diagram níže zjednodušeně vizualizuje postup změny hodnocení filmu. Proces změny je jednoduchý a začíná načtením dostupných informací k danému filmu. Již vytvořené hodnocení bude novými hodnotami přepsáno. Změna končí zápisem změn do logu hodnocení filmů.




*Obrázek 16: Diagram aktivity změny hodnocení filmu*

### 2.5.2 Změna hodnocení seriálu

Změny hodnocení seriálu jsou mnohem složitější než ty u filmů. Celý princip je v rámci modelu mimořádně zjednodušen a neobsahuje žádné techniky optimalizace ukládání. Pro začátek jsou získány veškeré informace o celém seriálu včetně dostupných hodnocení. Před zahájením změny je vytvořen záchytný bod, kde jsou hodnocení strukturovaně uložena. Následně je zahájen proces změny hodnocení. Dále je vytvořen další záchytný bod s uloženými hodnoceními. Poslední krok určí, jaká hodnocení různých epizod či sezón byla upravena a zavede změny do logu hodnocení seriálů.



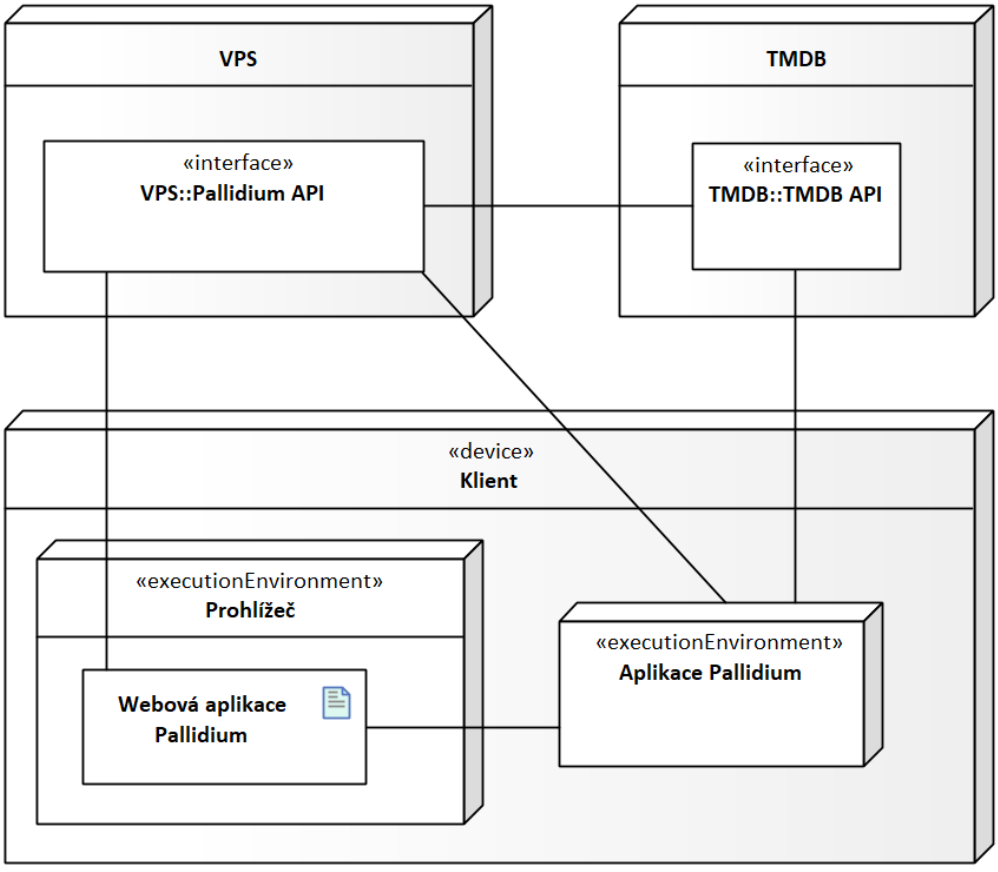

*Obrázek 17: Diagram aktivity změny hodnocení seriálu*

## 2.6 Diagram infrastruktury

Diagram představuje vytvářený ekosystém aplikací pod názvem Pallidium. Původní desktopová aplikace měla možnost získávat metadata pouze přímým spojením s TMDB API a díky tomu vyžadovala přístupový klíč. Systém tak musel dostat změnu, aby byl více uživatelsky přívětivý. Dle modelu je novým přístupem přímé spojení se serverem Pallidium, který přeposílá požadavky bez nutnosti přístupového klíče. Aplikace bude i nadále podporovat využití vlastního klíče pro účely vlastního monitorování nebo nezávislosti s aktivitou webového serveru Pallidium.

Webová aplikace využije připojení k TMDB pro účely ukládání multimediálních metadat. Uložená data budou po jejich stažení k dispozici uživatelskému rozhraní v prohlížeči. Webový server Pallidium také obsahuje přístupové body pro zpracování požadavků na synchronizaci záznamů knihoven a hodnocení.

Prohlížečová webová aplikace dokáže komunikovat s desktopovou aplikací na úrovni lokální sítě počítače. Komunikace bude použita k odesílání pokynů aplikaci a získávání potřebných souborových metadat.

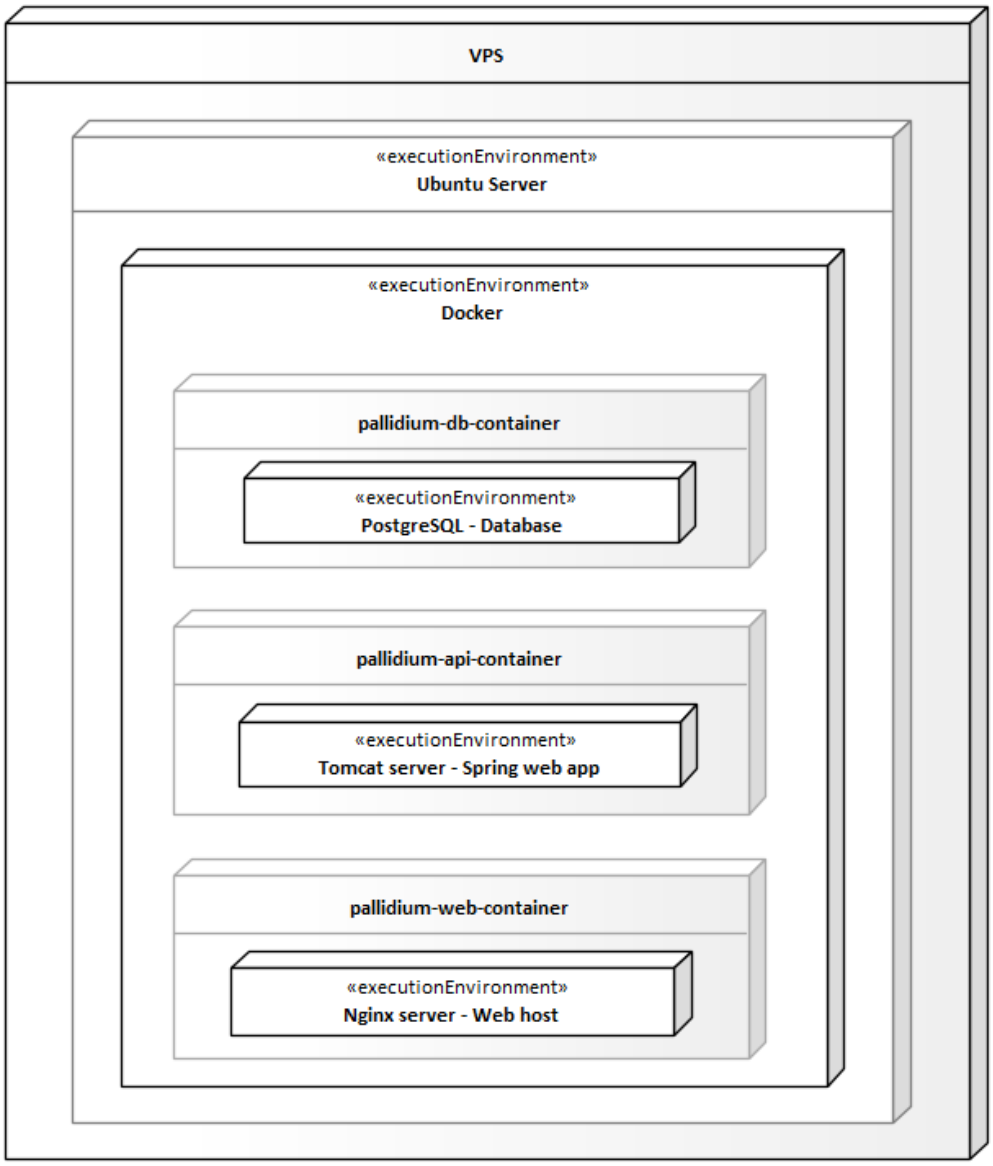



*Obrázek 18: Diagram infrastruktury*

## 2.7 Diagram nasazení

Diagram nasazení nejvíce souvisí s obsahem kapitoly 6. Popisuje rozložení součástí webové aplikace Pallidium na VPS. Hlavní softwarovou komponentou je tak kontejnerový systém Docker běžící na serverovém operačním systému Ubuntu Server. Uvnitř systému Docker jsou k nalezení kontejnery:

- **pallidium-db-container** (databáze),
- **pallidium-api-container** (backend),
- **pallidium-web-container** (frontend).



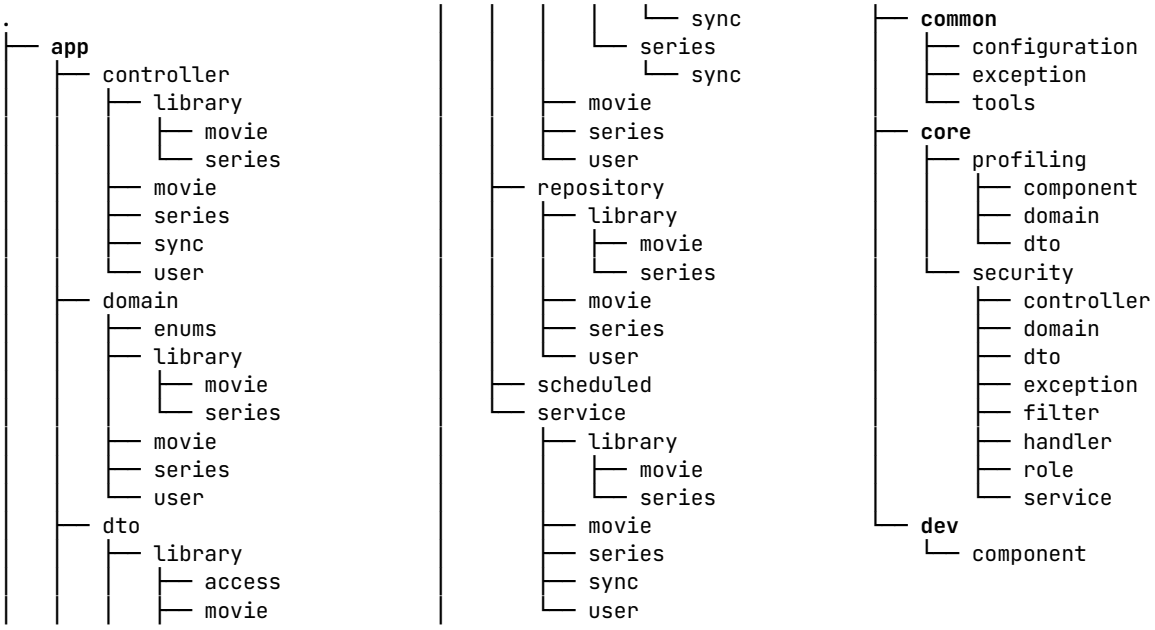

*Obrázek 19: Diagram nasazení*

## 2.8 Balíčkový diagram

Balíčkový diagram popisuje vnitřní strukturu kódu již implementovaného systému. Jde dbán důraz na přehledné členění systému spolupracujících částí.

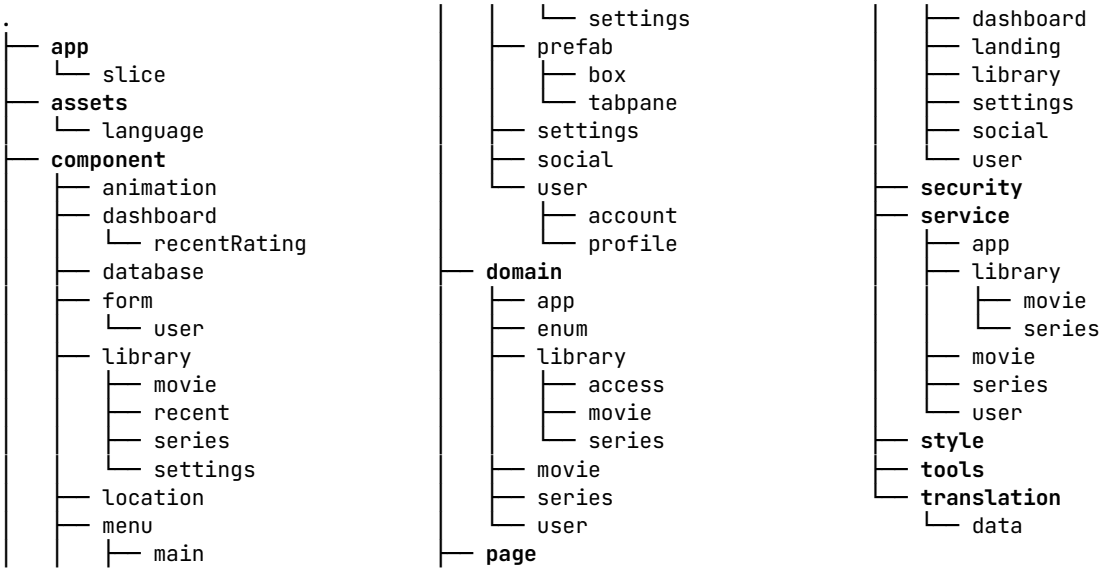
### 2.8.1 Backend

Hlavními částmi BE jsou: **app** – hlavní problematika systému, **common** – nastavení systému, **core** – zabezpečení a profilování, **dev** – testovací a vývojové prvky.




### 2.8.2 Frontend

Hlavními částmi FE jsou: **app** – správa globálních objektů, **assets** – multimédia, **component** – znovupoužitelné bloky UI, **domain** – datové třídy, **page** – jednotlivé stránky UI, **security** – nastavení Axios pro komunikaci s API a aplikací, **service** – obslužné metody komunikace s API, **style** – specifikace globálních stylů, **tools** – obecné nástroje, **translation** – správa lokalizace UI.



# 3 POUŽITÉ TECHNOLOGIE

## 3.1 Backend

*Tabulka 1: Verze použitých technologií BE*

| Název technologie | Číslo verze |
|---|---|
| Java | 19 |
| Spring Boot | 3.1.4 |
| AspectJ | 1.9.21 |
| Gradle | 8.0.2 |
| Swagger a Springdoc OpenAPI | 2.2.0 |
| PostgreSQL | 16.0 |

### 3.1.1 Java

Java je multiplatformní vysokoúrovňový programovací jazyk vyvíjený společností Oracle Corporation. Za autora Javy se považuje vývojář James Gosling, který v roce 1991 sestrojil první verzi tohoto jazyka. Gosling tehdy pracující v již zaniklé společnosti Sun Microsystems vyvinul programovací jazyk Oak. V roce 1995 byl jazyk oficiálně uveden pod názvem Java, který vydržel dodnes. Společnost Oracle Corporation ohlásila 20. dubna 2009 dohodu o zakoupení společnosti Sun Microsystems, která byla 27. ledna 2010 dokončena. Společnost byla prodána za 7,4 miliard dolarů a dokončením proběhl její zánik. [13], [14], [15]

Hlavním motem jazyka Java jsou slova jejího tvůrce „Write Once, Run Anywhere“ znamenající „Napiš jednou, spusť všude“. Java je jedním z nejpřívětivějších jazyků pro začátečníky. Její jednoduchost je způsobena nejen robustností jazyka, ale hlavně díky jejímu stáří. Díky dlouholeté dostupnosti byly sepsány různé knihy a bohaté dokumentace. Dlouhý výskyt také způsobil její integraci do základních výukových plánů, která způsobila dostatek programátorů, a tak bohatý trh plný pracovních pozic. [13]

Java je používána v mnohých zaměřeních, do kterých spadají:

- **Firemní software** – je jedním z nejčastějších míst kde nalezneme Javu, která je i po letech první volbou při vývoji rozsáhlého softwaru a důležité IT infrastruktury.
- **Big data** – neboli „Velká data“ jsou obrovské a komplexní sady dat, které jsou v reálném čase zpracovávány pomocí jazyka Java.
- **Internet věcí (IoT)** – původně „Internet of Things“ je pojem spojený s vývojem programů a aplikací koncových zařízení, kde Java je jednou z prvních možností.
    - Do této sekce spadá i vývoj různých mobilních aplikací nebo vestavěných systémů. [13], [16]

#### 3.1.1.1 Java API

Java API definuje syntaxi a sémantiku programovacího jazyka Java. Zahrnut je jednoduchý slovník a pravidla, která slouží k psaní datových typů a různých algoritmů jako podmínkové bloky if/else, for/while cykly a spousta dalších. Java API je tedy nedílnou softwarovou komponentou zabalenou s balíčkem Java Platform. Zmíněné API jsou již napsané programy, které lze libovolně využít k rozšíření funkcionality aktuálního kódu. Jejich využití se nachází v mnohých problematikách jako jsou výpočty, manipulace s textem či získání a práce s časem. [16]

#### 3.1.1.2 Java JVM

V prvních letech vývoje existovaly dvě kategorie zpracování programovacích jazyků. První kategorií jsou kompilované jazyky psané v přirozeném textu obsahující kompilátor, který se stará o převedení celého kódu do jeho strojové reprezentace. Takový kód je možné spustit přímo na hardwaru. Druhou kategorií jsou interpretované jazyky. U těchto jazyků je každý vysokoúrovňový kód tzv. „za letu“ interpretován na strojový kód. Sepsaný kód je tak spuštěn nehledě na to, co je napsané na dalších řádcích. [16]

Java je prvním jazykem kombinujícím obě výše zmíněné kategorie pomocí Java Virtual Machine (JVM). Všechny kódy napsané v jazyce Java jsou kompilovány do bajtového kódu (bytecode) a jsou spustitelné pouze s pomocí JVM. Následný kód je tak interpretován pro specifický operační systém, kde je JVM nainstalováno. Tím je zaručena jeho konzistence napříč systémy a multiplatformní funkcionalita. [16], [17]

#### 3.1.1.3 Verze, JDK a IDE

Java se dělí do různých edicí podle určeného využití. Oracle tak vyvíjí a publikuje Javu: Java Standard Edition (Java SE), Java Enterprise Edition (Java EE) a Java Micro Edition (Java ME). [16]

JDK je vývojářský balíček jazyka Java, který obsahuje rozsáhlou kolekci knihoven dovolujících kompilaci a spouštění vyvíjené aplikace. Některé z těchto nástrojů jsou využívány při tvorbě vytvářené desktopové aplikace. Přesněji se jedná o nástroje: jdeps, jlink a jpackage.

- **Nástroj jdeps** – je využit k analýze využívaných závislostí napříč aplikací a dovoluje jejich formátovaný výpis.
- **Nástroj jlink** – podporuje složení a optimalizaci vlastního listu modulů do přizpůsobeného runtime image.
- **Nástroj jpackage** – je využíván pro vytvoření instalátoru aplikace za pomocí nástrojů WiX Toolset pro cílový formát .msi. [17], [18], [19], [20]

### 3.1.2 Spring Framework

Spring usiluje o usnadnění postupu při vývoji složitých, rozsáhlých a obtížně spravovatelných podnikových Java aplikací tím, že nabízí framework, který zahrnuje technologie jako:

- **aspektově orientované programování (AOP)**,
- **vkládání závislostí (DI)**,
- **obyčejné objekty Java (POJO)**. [21]

Spring je odlehčený framework, který nabízí nejlepší funkcionality ostatním frameworkům například Hibernate, EJB, JSF, Tapestry a Struts. Na pozadí spravuje a eliminuje obtížnosti spojené s vývojem softwaru reálného času (RT). Dalším obsahem jsou moduly Web MVC, IoC, DAO, AOP, Context a ORM. [21]

Spring také pomáhá ve vytvoření škálovatelných, zabezpečených a robustních webových aplikací. Spring můžeme označit za kolekci různých aplikačních rámců, které společně nabízí spoustu výhod a možností. Spring obsahuje kromě hlavní podpory Java také podporu jazyků Kotlin a Groovy. Spring Framework nabízí základ pro kontrolu jiných na něm založených projektů, jako jsou Spring Boot, Spring Cloud a Spring GraphQL. [21]

Hlavní výhody Spring Frameworku spočívají:

- **Předdefinované šablony** – usnadňující práci s Hibernate, JDBC a JPA.
- **Volné propojení** – zaručuje slabý motýlí efekt na dalších komponentách.
- **Neintruzivní** – Spring nenutí k používání specifických rozhraní.
- **MVC** – zaručuje dobrou organizaci velkých projektů.
- **IoC** – v porovnání s ostatními lépe pracuje s pamětí.
- **Správa transakcí** – umožňuje škálování transakcí od lokálních až na globální.
- **Rychlý vývoj** – díky DI je jednoduché integrovat další rozšíření. [21]

### 3.1.3 Spring Core

Funkcionality aplikačního rámce Spring Core jsou mnohé. Z tohoto důvodu byly vybrány pouze nejdůležitější funkcionality, kterých bylo nejvíce využito při vývoji BE. [21]

#### 3.1.3.1 Vkládání závislostí (DI)

Vkládání závislostí je hlavní funkcionalitou celého Spring Framework. Koncept je založen na návrhovém vzoru IoC. Ten uvolňuje pevné vazby mezi objekty a odstraňuje tak vlastní inicializaci vlastností. Díky tomu není architektura tak úzce svázaná a je více flexibilní. Změny v kódu jsou jednodušší a nevyžadují takový zásah od vývojáře, protože jsou řešeny automaticky. Takto navržený systém má za následek lepší podporu při testování. Automatická správa závislostí je využita definováním konstruktorů nebo po dokončení konstrukce pomocí metody setter. [21], [22]

Se závislostmi se lze rychle vypořádat při použití Java knihovny Lombok, která nabízí různé anotace pro automatizaci opakovaných funkcionalit v kódu. Knihovna tak odstraňuje zbytečný obsah nesouvisející s řešením hlavního problému. Pro DI lze využít anotací @RequiredArgsConstructor nebo @AllArgsConstruktor. Z hlediska doporučení je lepší využít první možnosti, protože vložené závislosti většinou nastavujeme na neměnné pomocí klíčového slova final. Zmíněná praktika je jedním ze základů práce s BE částí projektu. Za zmínku stojí dnes nepříliš používané způsoby vkládání závislostí pomocí @Autowired nad atributem třídy nebo nad metodou setter. [23]

Špatný návrh při používání DI může způsobit cyklické závislosti. Při jejich výskytu Spring ve výchozím nastavení nelze spustit. V konzolovém výstupu pak lze zjistit o jaké závislosti se jedná.

Mezi metody řešení cyklických závislostí řadíme:

- předělání struktury závislostí,
- použití @Autowired nad metodou setter,
- nastavení závislosti na @Lazy v místě vložení,
- pozdní inicializací pomocí @PostConstruct,
- nejhorší metodou je přidání nastavení „spring:main:allow-circular-references: true“. [24], [25]

#### 3.1.3.2 MVC

Spring MVC umožňuje vývojářům vytvářet aplikace využívající architekturu MVC (Model View Controller). Hlavní součástí Spring MVC je třída DispatcherServlet, která spravuje uživatelské požadavky a doručuje je na správný controller. To umožňuje zpracovat požadavek, vytvořit nebo získat doménový objekt a poté doručit výsledky koncovému uživateli. Výsledky jsou před odesláním zabaleny pomocí objektu omezeného pohledu nazývaného DTO. [21]

### 3.1.4 Spring Boot

Spring Boot je nadstavbou nad Spring Frameworkem. Kromě toho, že implementuje veškerou funkcionalitu zmíněného frameworku, poskytuje zabudovaný servlet kontejner (Tomcat). Usnadňuje tak práci vývojářům, kteří nemusí danou funkčnost nastavovat sami. Kontejner sám po spuštění zkonstruuje a nakonfiguruje server, který bude přijímat uživatelskou komunikaci na portu HTTP. Spring Boot je hojně využíván ve firemním prostředí díky nízkým nárokům na znalosti spojené s nastavováním zmíněného serveru. [26]

### 3.1.5 Spring Data JPA

Spring Data JPA je součást rozsáhlejší skupiny Spring Data a usnadňuje implementování repositářů založených na JPA. Umožňuje jednodušší vytváření aplikací s využitím technologií pro přístup k datům. Implementace datové přístupové vrstvy (repository) pro aplikaci může být velmi zdlouhavá. Pro provádění i nejjednodušších dotazů je třeba psát příliš mnoho kódu. Po přidání funkcionalit jako stránkování, auditování a další se může kód zbytečně zkomplikovat. [27]

JPA si klade za cíl významně zlepšit implementaci datových přístupových vrstev tím, že snižuje práci vývojáře na minimum. V definici funkcí repositáře lze využít zabudovaného překládacího jazyka, který funkce pojmenované správnou syntaxí překládá na reálný dotaz do databázového systému. Příkladem může být spojení „findAll“ pro nalezení všech záznamů nebo „findByUserId(long id)“ pro vyhledání uživatele dne jeho unikátního identifikátoru. [27]

### 3.1.6 AspectJ

Aspektově orientované programování (AOP) je výkonné paradigma, který doplňuje tradiční přístup k objektově orientovanému programování (OOP). Umožňuje vývojářům efektivněji řešit různé záležitosti ve svých aplikacích. [28]

Aspekty jako takové vznikly koncem devadesátých let a jako koncept byly vydány v roce 1997 autory Gregor Kiczales, John Lamping a další. Původní práce se nazývala „Aspect-Oriented Programming“. V roce 2001 byla na toto téma publikována kniha a od té doby se začala popularita pouze zvyšovat. [29]

Aspekty mohou řešit úseky projektu spojené s logováním, profilováním, monitorováním, bezpečností nebo správou transakcí. Aspekt je modul, který zaštiťuje specifické chování a znaky napříč vybranými částmi aplikace. Nastavení je velice robustní, a tak lehce aplikovatelné na cílené úlohy. [28]

V rámci AOP rozlišujeme několik hlavních klíčových úseků, do kterých spadá:

- **Joinpoint** – místo v kódu pro vložení požadované logiku za pomoci AOP, zde nejčastěji používané volání metody, inicializace třídy a dalších.
- **Advice** – ukazuje na kód, který je spuštěn v určitém joinpoint, existuje pět typů:
    - **Before** – před spuštěním metody,
    - **After** – po spuštění metody nehledě na výsledek,
    - **After-returning** – po =spěšném spuštění metody,
    - **After-throwing** – po neúspěšném spuštění metody,
    - **Around** – před a po dokončení metody.
- **Pointcut** – je souborem joinpoint, pro které je spouštěna vybraná advice.
- **Aspect** – je kombinací advice a pointcut, určuje následnou logiku, která je do aplikace vložena a místo, kde bude spuštěna.
- **Weaving** – proces vkládání aspektů do aplikace.
- **Target** – cíl, objekt, jehož chování je ovlivněné některým z implementovaných aspektů. [30], [31]

Alternativou použitého AspectJ je Spring AOP, který nabízí pouze základní implementaci. Jeho funkčnost je omezena pouze na Spring komponenty. AspectJ je původní technologií AOP a usiluje o poskytnutí úplně funkčnosti. Rozdílem těchto dvou implementací je způsob vkládání aspektů (weaving) do kódu. SpringAOP vkládá aspekty až při běhu aplikace využitím proxy cílených objektů. Oproti tomu AspectJ nabízí tři typy vkládání: při kompilaci, po kompilaci a při načítání. AspectJ také podporuje více míst na vložení své funkcionality a je až 35krát rychlejší než SpringAOP. [32]

### 3.1.7 Gradle

Gradle je automatizační nástroj určený pro flexibilní sestavení softwaru. Sestavení obsahuje procesy kompilace (compile), spojení (linkage) a balení kódu (packaging). Gradle je populární díky automatizaci sestavení programovacích jazyků Java, Scala, Android, C/C++ a Groovy. Gradle nabízí kromě podpory různých programovacích jazyků i schopnost nasazovat na operačních systémech Windows, Linux a dalších. Gradle usiluje o spojení kladů z Ant a Maven. Dále se poučuje z chyb svých konkurentů a zaměřuje se na pokrytí jejich nedostatků. [33]

Gradle cílí na udržovatelnost, použitelnost, rozšiřitelnost, výkon a flexibilitu. Díky podpoře mnohých nastavení je vysoce přizpůsobitelný, čímž je oblíbený mezi vývojáři. Nabízená rychlost může dosahovat až dvojnásobného výkonu než u Maven. Kvalita uživatelského komfortu a práce je zajištěna pomocí nástrojů IDE podporujících Gradle. [33]

Z hlediska porovnání nástrojů Gradle a Maven jde spíše o preferenci. Systémy se primárně odlišují zápisem a dokáží dosáhnout stejných cílů. Gradle je novější, obsahuje modernější přístupy k řešením problémů a nabízí kratší zápis konfigurace. Maven je starší a obsahuje pokročilejší řešení některých specializovaných problematik. [33]

Gradle nabízí lepší podporu načítání privátních repositářů z GitHub Packages. Celé nastavení tak lze přehledně uchovat na jednom místě v konfiguračním souboru „build.gradle“. Postup v Maven je složitější díky nutnosti přesunu autentizační části do konfiguračního souboru settings.xml v adresáři cache Maven knihoven „~/.m2“. Ukázka konfigurace v kapitole 3.4.1.3. [34], [35]

### 3.1.8 REST

REST je architektura rozhraní navržená pro distribuované prostředí. Pojem byl představen v disertační práci Roye Fieldinga z roku 2000. REST je oproti ostatním technologiím (protokol XML-RPC, SOAP) orientován datově, nikoliv procedurálně. REST tak pouze určuje, jak se k potřebným datům přistupuje. Rozhraní REST se používá pro jednotný a snadný přístup ke zdrojům (data nebo stavy). Každý zdroj má vlastní identifikátor URI a REST definuje čtyři základní metody k jejich zpřístupnění. [36]

**GET** – je základní metodou přístupu ke zdrojům a slouží k jejich získání. Odkazy odeslané touto formou neobsahují žádné tělo, jsou cíleny na identifikátor zdroje a obvykle se při jeho použití využívají query parametry zadané v adrese URI. Tyto dodatečné parametry slouží k přizpůsobení výstupu pomocí třídění, stránkování, počtu záznamů a dalších metod. Obvyklou odpovědí serveru na tuto metodu jsou kódy 200 – nalezeno a 404 – nenalezeno. [36]

**POST** – metoda je využívána pro vytváření nových dat. V URI se nedotazujeme přímo na identifikátor zdroje (zdroj zatím neexistuje), ale na nadřazenou adresu. POST obvykle využívá tělo dotazu k odeslání potřebných dat a při úspěchu je zvykem obdržet návratový kód 201 - vytvořeno. [36]

**PUT** – slouží k úpravě cíleného zdroje. Součástí těla dotazu jsou odesílány nové informace o stavu zdroje potřebné k jeho úpravě. URI se odkazuje přímo pod identifikátorem zdroje a v případě úspěšné úpravy vrací návratový kód 204 – žádný obsah. [36]

**DELETE** – je použit pro odstranění zdroje specifikovaného pomocí cílového identifikátoru v URI. Metoda obvykle vrací kód 200 – ok nebo 201. [36]

#### 3.1.8.1 RESTful

RESTful je pojem označující dodržení všech níže zmíněných zásad návrhu REST architektury. [37]

**Využití jednotného rozhraní (UI)**

Zdroje by měly být jednoznačně identifikovatelné pomocí jediné URL, a pouze prostřednictvím základních metod síťového protokolu, jako je DELETE, PUT a GET s protokolem HTTP, by mělo být možné manipulovat se zdrojem. [37]

**Model klient-server**

Mělo by existovat jasně definované oddělení mezi klientem a serverem. Funkcionalita týkající se uživatelského rozhraní a požadavků je úkolem klienta. Přístup k datům, správu zátěže a zabezpečení má na starosti server. Popsané oddělení funkcionalit umožňuje každému z nich být vyvíjen a vylepšován nezávisle na druhém. [37]

**Bez-stavové operace**

Všechny operace klient-server by měly být bez-stavové, a jakékoliv řízení stavu, které je vyžadováno, by mělo probíhat na straně klienta, nikoliv na straně serveru. [37]

**RESTful cache zdrojů**

Všechny zdroje by měly umožňovat uložení cache, pokud není explicitně uvedeno, že ukládat cache není možné. [37]

**Vrstvený systém**

REST umožňuje architekturu složenou z více vrstev serverů. [37]

**Kód na vyžádání**

Většinou server odešle statické reprezentace zdrojů ve formě XML nebo JSON. Nicméně, když je to nutné, servery mohou odeslat spustitelný kód klientovi. [37]

#### 3.1.8.2 RESTful v implementovaném IS

Z hlediska vysoké náročnosti a specializovanosti takového návrhu nebyly plně dodrženy zásady architektury RESTful. Vytvořený backend obsahuje minimální podporu ukládání cache a není složen z několika serverových části. Hlavní server řeší veškerou funkcionalitu a není samostatně rozdělen na části spravující přístup k databázi, zabezpečení a obsluhu požadavků. Implementovaný backend nedodržuje ani poslední zásadu spustitelného kódu na vyžádání.

### 3.1.9 YAML

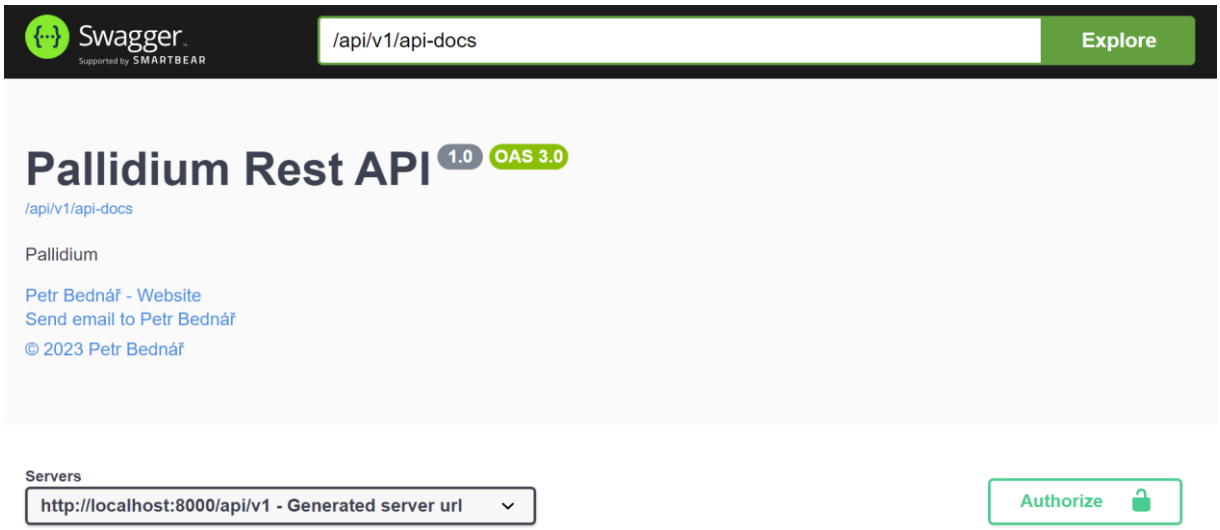
YAML je formát pro ukládání textových dat. Data jsou obvykle ukládána binárním způsobem, který je nejrychlejší pro zpracování. Takto uložená data nejsou dobře čitelná a nelze je tak lehce upravovat bez specializovaných nástrojů. Tento problém byl částečně vyřešen pomocí formátu XML, který podporuje vytváření značek, podle kterých jsou data identifikována. Jako reakce na nepřehlednost XML byl vytvořen jazyk POX. YAML se tak vydává podobnou cestou, kde formát neobsahuje složité konstrukce, ale i tak je schopný vše potřebné vyjádřit. Syntaxe YAML se nejčastěji používá v reprezentaci konfiguračních souborů, zápisů pro wiki systémy a různých nástrojích. [38]

Nejjednodušším prvkem YAML je sekvence. Lze ji přirovnat k prvkům pole, kde každý prvek je na novém řádku a je oddělen spojovníkem „-“. Dalším prvkem jazyka je mapa, která pracuje podobně jako asociativní pole. Mapa je vytvořena kombinací klíče, dvojtečky „:“ a hodnoty. Sekvence a mapy lze libovolně kombinovat. YAML může obsahovat více logických částí neboli dokumentů. Ty lze využít například u specifických konfigurací profilů. Dokumenty oddělujeme třemi spojovníky „---“ za sebou. Dalšími prvky jsou texty, které se specifikují symbolem větší „>“, kde text je rozdělen na odstavce v případě prázdného řádku. Symbol lze také nahradit svislou čárou „|“, která oddělí každé slovo novým řádkem. Komentáře se v jazyce YAML používají pomocí symbolu mřížky „#“. [38]

### 3.1.10 Swagger API

Swagger je jedním z nejoblíbenějších dokumentačních nástrojů REST API využívaným giganty jako Google, Netflix a Microsoft. Swagger slouží primárně k vizualizaci a práci s výstupem dat OpenAPI specifikace. Ta je automaticky nebo manuálně generována a uložena v souborech typu JSON případně YAML. Swagger je tvořen třemi moduly: Swagger Editor, Swagger UI a Swagger Codegen. [39]

V této práci je využito pouze modulu Swagger UI, s jehož pomocí je zobrazována specifikace tříd controller. Swagger UI je přístupný v testovacím prostředí pod adresou „/swagger-ui/index.html“.




*Obrázek 20: Úvodní strana prohlížeče dokumentace OpenAPI ve SwaggerUI*

### 3.1.11 PostgreSQL

PostgreSQL je výkonným open-source databázovým systémem používající relační přístup. První zmínky pochází z 1986, díky tomu má za sebou více než 35 let aktivního vývoje a rozšířenou komunitu. Hledání postupů a řešení tak není příliš velký problém. PostgreSQL nabízí podporu: rozšířených datových typů, integrity dat, konkurence procesů, vysoké rychlosti zpracování, odolnosti, obnovy dat, bezpečnosti, rozšiřitelnosti, internacionalizace a full-textových vyhledávání. Společně s využitím databázových systému se objevují i výrazy jako RDBMS a ACID. Označení databáze nebo databázový systém není úplným označením takové technologie. [40]

Pojem **RDBMS** neboli „systém řízení báze dat“ není uživatelsky příliš přívětivý, a tak se označení „databáze“ osvědčilo více. [41]

**ACID** je akronym slov Atomicity (atomicita), Consistency (konzistence), Isolation (izolace) a Durability (trvanlivost).

- **Atomicity** – znamená provádění celé transakce nedělitelně najednou.
- **Consistency** – je stav, ve kterém musí databáze vždy být.
- **Isolation** – zaručuje vzájemnou neovlivnitelnost operací.
- **Durability** – garantuje okamžité zapsání dat na fyzické médium. [41]

## 3.2 Frontend

*Tabulka 2: Verze použitých technologií FE*

| Název technologie | Číslo verze |
|---|---|
| Node.js | 20.9.0 |
| Vite | 4.2.0 |
| Typescript | 4.9.3 |
| React | 18.2.0 |
| Axios | 1.3.5 |

### 3.2.1 Node.js

Node.js je softwarový systém umožňující spuštění JavaScript kódu mimo webový prohlížeč. Základem Node je engine Chrome V8, který se shoduje s tím v prohlížeči Google Chrome a jiných používající stejný základ. Hlavním účelem Node.js je tvorba serverové části webových aplikací a škálovatelnost. [42]

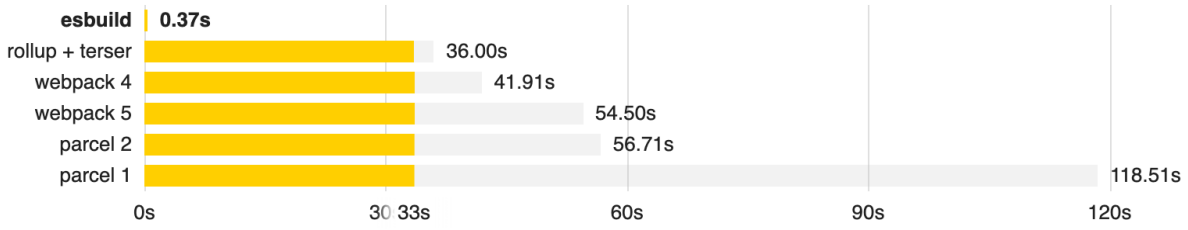
#### 3.2.1.1 NPM

NPM je balíčkový ekosystém a je součástí standardní instalace Node.js. V rámci projektu se stará o správu knihoven (primárně instalace a údržba) a distribuce JavaScript kódu. NPM má k dispozici stovky tisíc různých balíčků a jedním z největších balíčkovacích systémů vůbec. Obsahuje například populární frameworky jako jQuery, React a Angular. [42]

V projektu je použit k instalaci technologií Vite, TypeScript, React, Axios a dalších. Dále je využit k sestavení a zabalení funkcionality FE do uceleného JavaScript/HTML balíčku, který je posléze nasazován na webovém serveru Nginx.

### 3.2.2 Vite

Vite představuje nástroj pro ulehčení práce vývojáře snížením odezvy mezi změnami a jejich efektem. Vývojář tak nemusí několik minut čekat na viditelné změny, a tak rychleji pracovat na řešené problematice. Jedná se o technologii “hot reload” s původem někdy v roce 2013. Jejími předchůdci jsou chronologicky LiveStyle, LiveReload, Grunt, gulp.js, Parcel a rollup.js. V roce 2020 vznikl nástroj ESBuild napsaný v jazyce Go, který urychlil zpracování sestavení až o stonásobek. [43]




*Obrázek 21: Porovnání rychlosti nástrojů sestavení FE [43]*

Instalace Vite je provedena přes NPM nebo Yarn zadáním jednoduchých příkazů. Vite dokáže pracovat s populárními knihovnami jako Vue.js nebo React. Vite dále podporuje psaní kódu v jazyce TypeScript. Vývojový server lze spustit třetím příkazem níže. [43]

*Zdrojový kód 1: Instalace Vite pomocí NPM a Yarn*

```
npm init @vitejs/app
yarn create @vitejs/app
npm run dev
```

V práci byl použit k tvorbě a sestavení FE. Byl vybrán na základě doporučení kolegů a již zmiňované rychlosti. Prvním důvodem jeho rychlosti je použití nativních modulů jazyka JavaScript (ES 6). Tyto moduly dovolují rozdělení kódu do několika souborů, které jsou při změně nahrazeny. Není tak důvod znovu sestavovat celý kód, ale jen postiženou část. Druhým důvodem zvýšené rychlosti je využití mnohonásobně rychlejšího jazyka Go, než je JavaScript. [43]

### 3.2.3 TypeScript

TypeScript je open-source programovací jazyk vydaný společností Microsoft. Jedná se o nadstavbu jazyka JavaScript, která ho rozšiřuje o statické typování, třídy, rozhraní a další věci z OOP. Díky své popularitě byl využit společností Google a integrován přímo do frameworku Angular. TypeScript obsahuje kompilátor typu „transpiler“, který překládá TypeScript kód do ekvivalentní reprezentace v jazyce JavaScript. [44]

JavaScript je založen na dynamickém typovém systému, kde se neřeší, jaký typ nějaká proměnná má. Kód tak může zpočátku vypadat lehce, ale jedná se o jednu z největších nevýhod JavaScript. TypeScript rozšiřuje dynamický typový systém a přidává podporu statického typového systému. Jazyk tak obsahuje striktní typovou kontrolu, ale zároveň zpětnou kompatibilitu s dynamickým systémem pomocí typu „any“. [44]

TypeScript vyžaduje stejně jako Vite instalaci softwaru Node.js. Následná instalace tak probíhá obvykle pomocí balíčkové systému NPM. [44]

### 3.2.4 React.js

React je open-source knihovnou jazyka JavaScript pro tvorbu uživatelských rozhraní původně vyvinutý společností Meta (dříve Facebook). Na vývoji se také podílely komunity samostatných vývojářů a jiné společnosti. React je aktuálně nejoblíbenější a nejžádanější technologií na trhu práce. [45]

Uživatelské rozhraní React je obvykle implementováno ve stylu obsluhy webového API. Prostředí tak nabízí dostupnost dat napříč různými platformami díky jejich persistenci na serveru. React jako takový není framework, ale jedná se pouze o knihovnu. Dle architektury MVC, která je již zmíněna v kapitole 3.1.3.2, by popisoval pouze zobrazovací vrstvu prezentující data uživateli. [45]

React slouží k tvorbě jednostránkových (SPA) aplikací. Stránka tohoto typu je načtená pouze jednou a poté jen reaguje na odpovědi bez načítání stránky. Jedna taková stránka tak obsahuje celou funkčnost aplikace. React je tvořen speciálními znovupoužitelnými komponentami, obslužnými funkcemi a podporuje zápis HTML přímo v kódu JavaScript pomocí JSX. [45]

### 3.2.5 Axios

Axios je JavaScript knihovna podporující vytváření HTTP dotazů z Node.js nebo XMLHttpRequest dotazů v prohlížečích s podporou ES6 Promise API. Cílem knihovny Axios je vylepšení metody odesílání dotazů (fetch) a lepší zpracování chybových hlášení (catch). [46]

Axios nabízí tvorbu speciálních funkcí (interceptors), které nabízí přidání funkcionality před odesláním dotazu nebo jeho zpracováním. Dovoluje tak elegantní řešení odesílání autorizačních hlaviček či jiných nastavení. Dalším vylepšením je také podpora dotazovacích metod HTTP například: GET, POST, PUT a DELETE. [47], [48]

### 3.2.6 MUI

MUI je knihovna UI komponent, které lze lehce přidat do React aplikace. Knihovna nabízí pro použití jednoduché a složité elementy. Každá z těchto elementů dovoluje upravit design přes styly nebo pomocí šablon. Mezi hlavní klady vývoje s MUI patří rychlejší vývoj, konzistence UI, přístupnost a rozšiřitelnost. MUI nabízí několik hlavních kategorií UI komponent určených na zobrazení dat, datovou mřížku, datum/čas, zpětnou vazbu, vstupy, rozvržení, navigaci, povrchy a nástroje. [49]

### 3.2.7 I18next

I18next je populární internacionalizační aplikační rámec pro aplikace napsané v jazyce JavaScript. I18next zjednodušuje proces začlenění podpory pro vícejazyčnost do webových, mobilních a desktopových aplikací tím, že poskytuje funkce pro zpracování překladů, formátování a další aspekty internacionalizace. Umožňuje vývojářům dynamicky spravovat a načítat překlady, což usnadňuje vytváření aplikací přizpůsobitelných pro různé jazyky a regiony. [50]

### 3.2.8 MobX

MobX je knihovna založená na signálech, která zjednodušuje a škáluje správu stavu transparentně aplikováním funkcionálního reaktivního programování. Jejím účelem je vytváření různých objektů, které si vždy udržují konzistentní stav napříč jejími využitími. Zjednodušeně řečeno se jedná o knihovnu, která dovoluje vytvoření globálních useState napříč aplikací pouze vytvořením tzv. „store“ pro správu hodnot a specifikací hodnoty jako „observable“. [51]

## 3.3 Aplikace

*Tabulka 3: Verze použitých technologií desktopové aplikace*

| Název technologie | Číslo verze |
|---|---|
| Java | 19 |
| OpenJFX | 19 |
| SQLite JDBC | 3.43.0.0 |

### 3.3.1 JavaFX / OpenJFX

JavaFX je framework pro tvorbu okenních desktopových aplikací. JavaFX přináší podporu obrázků, videa, hudby, grafů, CSS stylů a dalších technologií. Framework byl původně zamýšlen jako náhrada Swingu, ale firma Oracle se rozhodla vývoj ukončit, a JavaFX byla předána open-source komunitě. Nově je framework přejmenován na OpenJFX a od Java 11 není součástí vývojářského balíčku (JDK). [52]

Pro vytváření uživatelských rozhraní slouží editor Gluon Scene Builder. Jedná se o nástroj tvorby UI pomocí jednouchého principu „drag&drop“ a je volně dostupný ke stažení. Formátem pro ukládání vytvořených stránek je FXML. [52]

### 3.3.2 SQLite

SQLite je open-source knihovna pracující v procesu vyvíjené aplikace. SQLite je tak vestavěný databázový engine a na rozdíl od většiny ostatních SQL databází nemá samostatný serverový proces. SQLite čte a zapisuje přímo do běžných diskových souborů. Úplná SQL databáze s tabulkami, indexy, spouštěči a pohledy je obsažena v jediném diskovém souboru. Formát databázového souboru je přenositelný mezi platformami. K běhu databáze není potřebná žádná konfigurace a nabízí podporu transakčního zpracování požadavků. SQLite je nejvíce nasazovanou databází na světě s více aplikacemi, než dokážeme spočítat a několika významnými projekty. [53]

## 3.4 Systém pro verzování

### 3.4.1 Git

Git je systém verzování sloužící k ukládání projektů a jejich verzí. Jedná se o distribuovaný systém správy verzí, který nabízí přístup k celému kódu a jeho historii napříč podporovanými zařízeními. Git zpracovává většinu prováděných změn lokálně a dovoluje nahrát změny na centrální server s cílem verzování a sdílení. Git je integrován do mnohých IDE jako jsou IntelliJ IDEA a Visual Studio Code. Správu lze provádět také přes aplikaci GitHub Desktop nebo Git Bash. Git je používán více než 87 % všech vývojářů. [54]

#### 3.4.1.1 GitHub

GitHub je platforma pro práci s nástrojem Git a sdílení kódu mezi vývojáři. Jedná se prakticky o takovou sociální síť, kde může každý vidět přehled o aktuálně rozdělaných projektech a aktivitě. GitHub tak nabízí přívětivější a jednodušší prostředí než Git samotný. [54]

#### 3.4.1.2 GitHub Actions

GitHub Actions je služba provozovaná platformou GitHub a umožňuje vytváření skriptovacích workflow pro celé repositáře. GitHub umožňuje do workflow skriptů importovat i funkcionality od jiných vývojářů. GitHub Actions pomáhají zjednodušit správu projektů a umožňují vytváření automatizovaných procesů jako například CI/CD workflow. Actions jsou zpřístupnitelné po registraci v systému GitHub a nastavení příslušně workflow v jazyce YAML. Na osobní účty se však vztahují přívětivé limity, které zamezují zneužívání na korporátní škále. [55]

#### 3.4.1.3 GitHub Packages

GitHub Packages je platforma pro hosting a správu balíčků, kontejnerů a dalších závislostí. Platforma kombinuje úložiště pro zdrojový kód a balíčky na jednom místě pro zajištění centralizovaného přístupu, nastavení přístupových oprávnění a účtování. Přístupová práva mohou být zděděna ze zdrojového repositáře nebo manuálně nastavena pro jednotlivé uživatele a organizace. Veřejné balíčky nejsou nijak omezeny využitím a jsou plně zdarma. Privátní balíčky jsou omezeny pomocí limitů vázaných na účet jejich tvůrce. Díky rozšířené podpoře a dobrému finančnímu modelu je dobrou volbou pro hostování privátních a veřejných balíčků. [56]

K privátním repositářům hostovaným na serverech GitHub Packages lze přistoupit pomocí Gradle konfigurace níže. Obsahem je definování dalšího repositáře včetně přístupových údajů k získání požadovaného přístupu. [34]

*Zdrojový kód 2: Nastavení privátních GitHub balíčků pro Gradle*

```
repositories {
  mavenCentral()
  maven {
    url = uri("https://maven.pkg.github.com/pPetrBednar/tmdb-api-v3")
    credentials {
      username = System.getenv("GH_PACKAGES_USERNAME")
      password = System.getenv("GH_PACKAGES_ACCESS_TOKEN_READONLY")
    }
  }
}
```

Postup získání stejné knihovny z Maven je složitější díky přidání dodatečného nastavení v globálním souboru uživatelské konfigurace settings.xml uloženého ve složce ~/.m2. Složka poskytuje místo k uložení cache stažených balíčků. Vložené osobní údaje nelze přidávat automaticky a je nutné jejich manuální zadání. [35]

*Zdrojový kód 3: Nastavení privátních GitHub balíčků pro Maven, část 1.*

```
<servers>
    <server>
        <id>github</id>
        <username>pPetrBednar</username>
        <password>GitHub přístupový token</password>
    </server>
</servers>
```

Následně pokračuje zápis cílového privátního repositáře GitHub Packages, po jehož zadání je možné závislost stáhnout. [35]

*Zdrojový kód 4: Nastavení privátních GitHub balíčků pro Maven, část 2.*

```
<repositories>
    <repository>
        <id>github</id>
        <url>
            https://maven.pkg.github.com/pPetrBednar/tmdb-api-v3
        </url>
        <releases>
            <enabled>true</enabled>
        </releases>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </repository>
</repositories>
```

## 3.5 Nasazení

### 3.5.1 Docker

Docker je sada produktů a nástrojů typu platforma-jako-služba (PaaS) využívajících virtualizaci na úrovni operačního systému pro dodávání softwaru formou kontejnerů. Docker je open-source software, díky kterému se není nutné omezovat specifickými operačními systémy nebo nástroji. Docker a jeho kontejner můžeme spustit na jakémkoliv operačním systému s jeho podporou. Původ sahá do roku 2010 a v dnešní době se jedná o jeden z nejpoužívanějších softwarů na celém světě. [57]

Problematika softwaru Docker obsahuje pojmy:

- **Obraz (image)** – je šablona uložená v registru, která je určená pouze ke čtení, a je používána k vytvoření Docker kontejnerů.
- **Dockerfile** – je textový soubor popisující návod pro vytvoření požadovaného kontejneru.
- **Kontejner (container)** – je spuštěný obraz běžící za pomoci softwaru Docker. Kontejnery mohou být plně izolovány nebo mohou obsahovat prostupy do ostatních kontejnerů či širší sítě. [57]

#### 3.5.1.1 Docker Compose

Docker Compose je nástroj pro definování a spouštění více kontejnerových aplikací. Jeho využití je klíčem k jednoduché a efektivní práci s vývojem i nasazením. Docker Compose zjednodušuje kontrolu nastavení celého aplikačního zásobníku a umožňuje správu služeb, sítí a svazků v jediném, srozumitelném konfiguračním souboru YAML. Následným zadáním jediného příkazu dojde k vytvoření Docker kontejneru a spuštění všech požadovaných služeb z konfiguračního souboru. [58]

### 3.5.2 Nginx

Nginx je bezplatný open-source webový server. Nginx vytvořil Igor Sysoev v roce 2004 a je používán giganty jako Microsoft, IBM, Google, Meta, X, Apple, Intel a dalšími. Nginx poskytuje mnoho funkcí včetně reverzních proxy serverů, vyrovnávačů zátěže a podpory http cache. Nginx dále podporuje webové sokety a dokáže zprostředkovávat přístup k hostovaným souborům webových stránek. [59]

Server je využit pro nasazení FE části vyvíjené webové aplikace, směrování dotazů na Spring API a zabezpečení komunikace přes SSL.

## 3.6 Bezpečnost

### 3.6.1 OpenVPN

OpenVPN je populární a vysoce zabezpečený tunelovací protokol založený na SSL pro přístup k virtuálním privátním sítím. Projekt založil James Yonan v roce 2001. Oproti jiným protokolům VPN se OpenVPN odlišuje využitím šifrování dat pomocí SSL. SSL je technologie, která zavádí zabezpečená spojení point-to-point nebo site-to-site. Obvyklým využitím je zabezpečení a šifrování bankovních transakcí a citlivých dat. OpenVPN dokáže vytvořit zabezpečený komunikační tunel mezi klientem a cílovým VPN serverem. [60]

Protokol je využit k zabezpečené komunikaci a správě VPS, který obsahuje nasazené verze vyvíjeného systému. Pro navázání spojení je u klienta používána aplikace OpenVPN Connect. OpenVPN server byl na VPS zprovozněn pomocí automatizovaného instalátoru uživatele angristan dostupného v GitHub repositáři openvpn-install. [61]

### 3.6.2 Let's Encrypt

Let's Encrypt je automatizovaná certifikační autorita (CA), kterou provozuje ISRG. CA poskytuje certifikáty typu X.509 pro šifrování TLS. CA umožňuje vytvářet, obnovovat a spravovat SSL/TLS k povolení zabezpečené komunikace přes HTTPS. Každý poskytnutý certifikát je platný po dobu 90 dní a po 60 dnech ho lze obnovit. [62]

CA podporuje tvorbu dvou typů certifikátů:

- **Validační certifikát domény** – poskytuje silné šifrování webových stránek a nastavuje se na specifické názvy domén.
- **Žolíkový certifikát** – podporuje zabezpečení všech podřazených domén pomocí znaku hvězdičky. [62]

CA má nastavené jisté limity sloužící k zamezení zneužívání této bezplatné služby. Samotné tvůrci tvrdí, že jsou limity nastaveny férově. Počet nově vytvářených certifikátů je omezen na 50 pro každou doménu. V jednom certifikátu lze zahrnout maximálně 100 domén. Při neúspěšných ověření certifikátu na webové stránce žadatele umožňuje systém maximálně 5 pokusů za hodinu. Ostatní limitace souvisí s omezením provozu registračního a ověřovacího systému CA. [62]

Let's Encrypt používá ACME protokol ke stanovení, jak mají klienti komunikovat se servery. Pro prvotní ověření domény potřebuje CA kontaktovat klienta, který musí potvrdit CA, že běží na specifikovaném serveru v žádosti o vytvoření certifikátu. Po prvotním navázání komunikace je klient požádán o vytvoření klíčů, které si CA následně ověří. Tyto klíče musí být přístupné pod adresou domény, na kterou je certifikát vytvářen. Pokud je vše správně nastaveno, CA ověří správnost klíčů pod adresou „/.well-known/acme-challenge/“. Při úspěšném ověření CA odešle klientovi certifikát. Certifikát je poté používán při komunikaci pomocí protokolu HTTPS. [62]

Mezi výhody používání certifikátů Let's Encrypt patří bezplatnost, automatizace, jednoduchost, zabezpečení, transparentnost a otevřenost. Díky těmto kladům patří Let's Encrypt k nejvíce používaným certifikačním autoritám na světě s použitím na více než 260 milionech webových stránkách. [62]

# 4 IMPLEMENTACE

Tato kapitola pojednává o implementaci celé webové aplikace a dodatkových změn v desktopové aplikaci. Součástí kapitoly jsou tři hlavní cíle práce, po jejichž splnění je sestrojena finální verze aplikace.

První z hlavních cílů stanoví, že bude implementován webový informační systém se zaměřením na uchování a správu multimediálních metadat. Tedy vytvoření systému, ve kterém mohou uživatelé přidávat a hodnotit obsah filmů i seriálů. První cíl primárně souvisí s obsluhou databázového systému, který ukládá všechna potřebná data k uspokojení požadavků správy obsahu. Rozsahově tvoří první z cílů více než 50% časové náročnosti vývoje.

Druhým cílem je implementována funkcionalita spojená se synchronizací mezi desktopovou a webovou verzí aplikace. Obsahem je úprava již vytvořené aplikace a přidání nové funkcionality dle stanovených požadavků na hodnocení a mechaniky synchronizace. Součástí BE webové aplikace je vytvoření speciálních koncových bodů sloužících k obsluze aplikace. Součástí vyhrazené kapitoly synchronizace je tak podrobné vysvětlení všech termínů a mechanik spojených s tímto systémem.

Posledním z cílů je vytvoření sociální zóny pro uživatele, která poskytne navazování nových spojení s ostatními, a dovolí sdílení privátních knihoven mezi uživateli. Sociální síť je implementována pouze na webové aplikaci.

## 4.1 Databázový systém

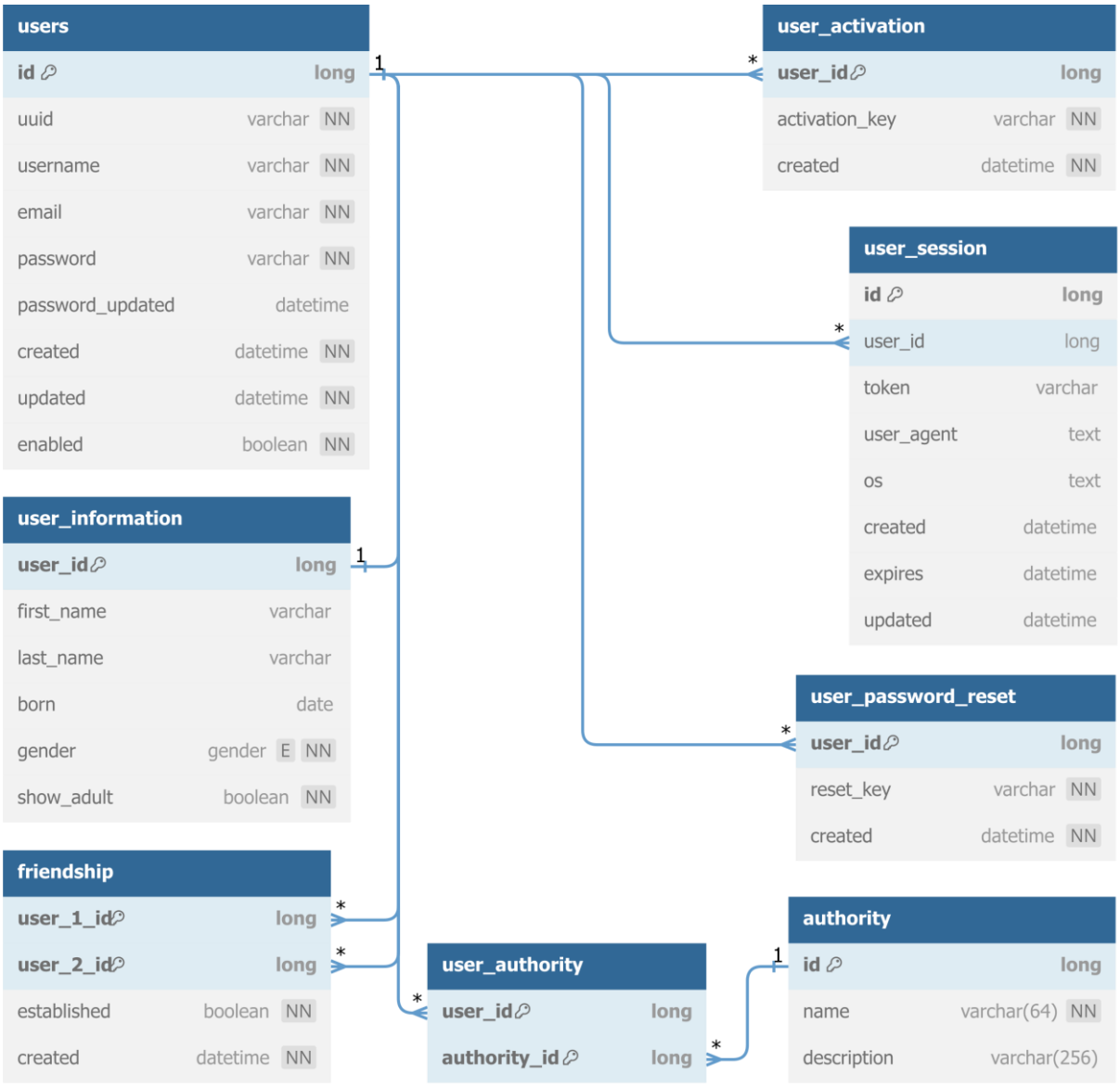
Pro poskytnutí služeb databázového systému bylo využito databáze PostgreSQL. Databáze je kontejnerizována a přes lokální kontejnerovou síť dostupná BE. Více informací o kontejneru databáze je k nalezení v kapitole 6.8.1. K tvorbě tabulek databázového systému bylo využito automatické vytváření doménových entit pomocí JPA repositářů.

Databázový systém je aktuálně složen z více než 35 tabulek. Ukázka celého modelu na jednom modelu by tak nebyla možná, a proto byl model rozdělen do pěti segmentů. V následujících kapitolách jsou představeny modely databázových tabulek popisující segmenty databázového prostoru:

- uživatelů,
- filmové cache,
- filmových knihoven a záznamů,
- seriálové cache,
- seriálových knihoven a záznamů.

### 4.1.1 Segment uživatelského účtu

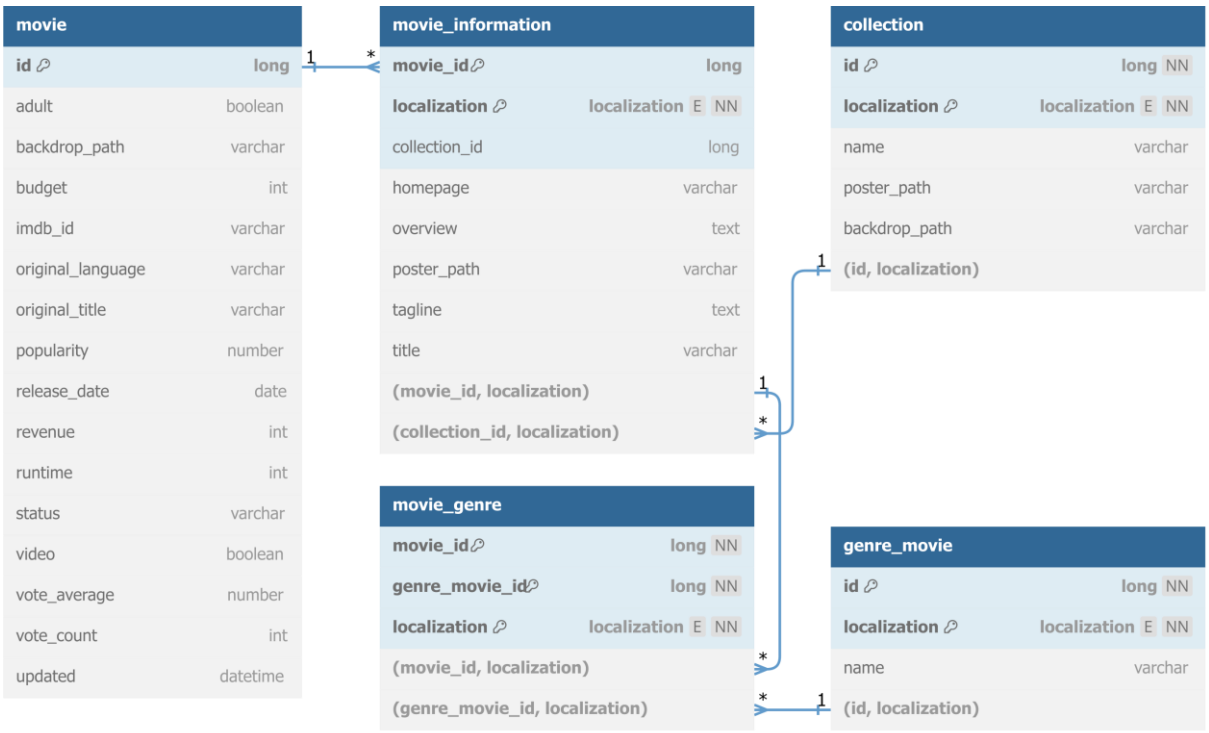
Segment uživatelského účtu obsahuje veškeré osobní informace uživatelů a jejich oprávnění. Součástí jsou tabulky ukládající poslední aktivitu uživatele a jeho navázaná sezení. Tabulky „user_activation“ a „user_password_reset“ slouží k uložení dočasných ověřovacích klíčů uživatelského klíčů nebo autorizace změny hesla. Tento způsob persistence dat v databázi oproti cache byl zvolen z důvodu možné ztráty dat při nasazení nové verze systému. Tabulka „friendship“ obsahuje navázaná spojení mezi spřátelenými účty a z důvodu optimalizace je po navázání takových spojení záznam duplikován, aby byl oboustranně přístupný jedním dotazem.




*Obrázek 22: Databázové schéma uživatelského segmentu*

### 4.1.2 Segment filmové cache

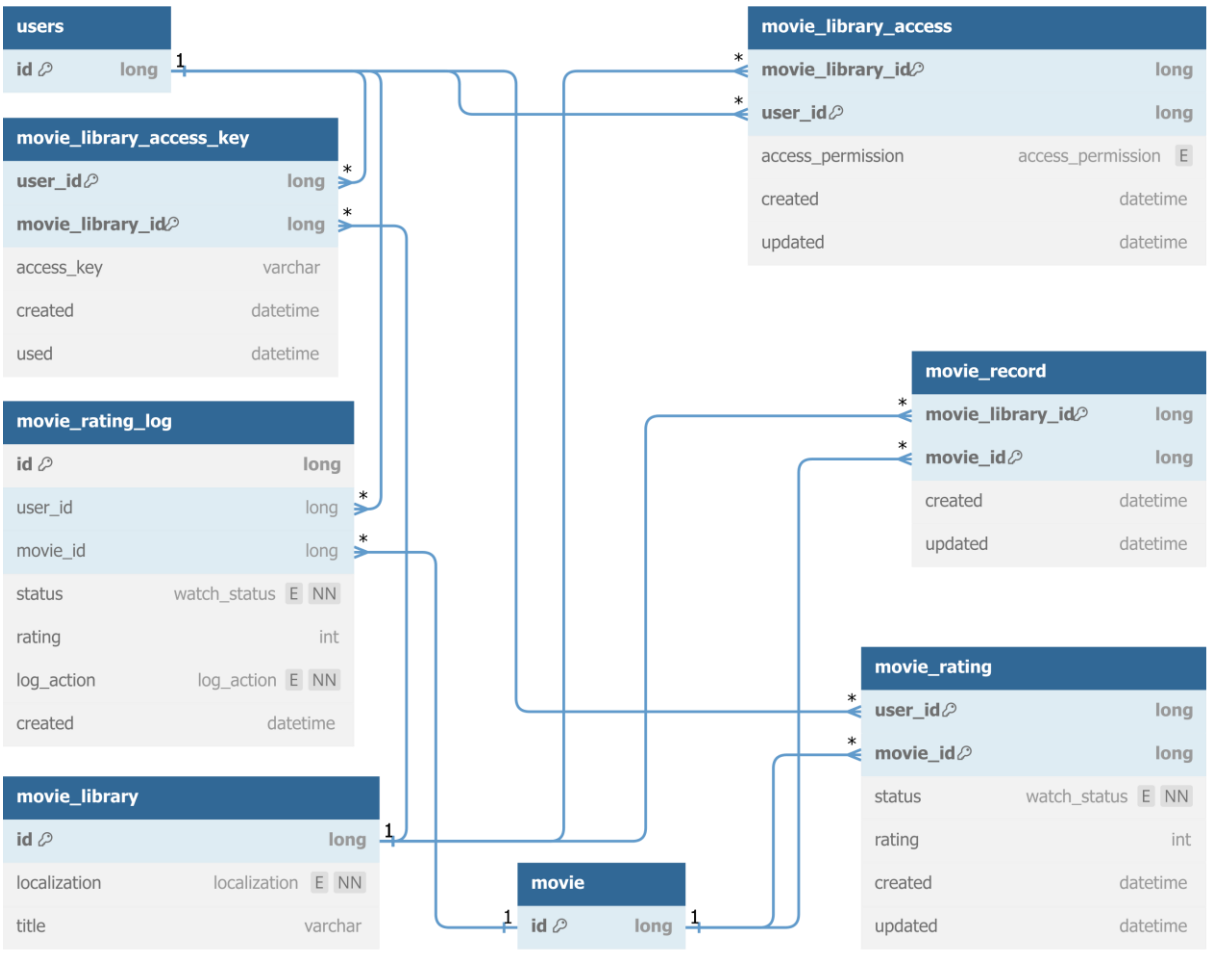
Databáze obsahuje prostor pro uložení filmových metadat z důvodu limitací provozu jejich poskytovatele. Tabulky jsou optimalizovány tak, že hlavní tabulka drží informace, které jsou neměnné napříč lokalizacemi. Dodatečné lokalizované tabulky jsou označeny kompozitním primárním klíčem spojeným z identifikátoru filmu a příslušné lokalizace. Identifikátor filmu je shodný s identifikátorem používaným pro filmy u poskytovatele těchto metadat (TMDB). Tabulky tak podporují přidání jakéhokoliv množství různých lokalizací vybraného záznamu filmu. Při neexistenci lokalizovaných metadat specifického filmu je zvolena za výchozí jazyk angličtina, jelikož má nejvyšší pokrytí záznamů. Tabulka „users“ je pojmenována množným číslem z důvodu vyhrazeného názvu jednotného čísla „user“, které je klíčovým slovem.




*Obrázek 23: Databázové schéma segmentu filmové cache*

### 4.1.3 Segment filmů

Model popisuje provázání tabulek spojených s persistencí dat filmových knihoven, jejich záznamů a uživatelských hodnocení. Z modelu byly pro přehlednost odebrány nepotřebné sloupce tabulek „users“ a „movie“. Každá filmová tabulka je provázaná přes přístupová práva na uživatele. Výhradní právo pro tabulku má uživatel s oprávněním vlastníka. Tabulka sdružuje pouze filmové záznamy. Příslušná filmová hodnocení jsou vázána přímo na uživatele společně s tabulkou jejich změn.




*Obrázek 24: Databázové schéma segmentu filmů*

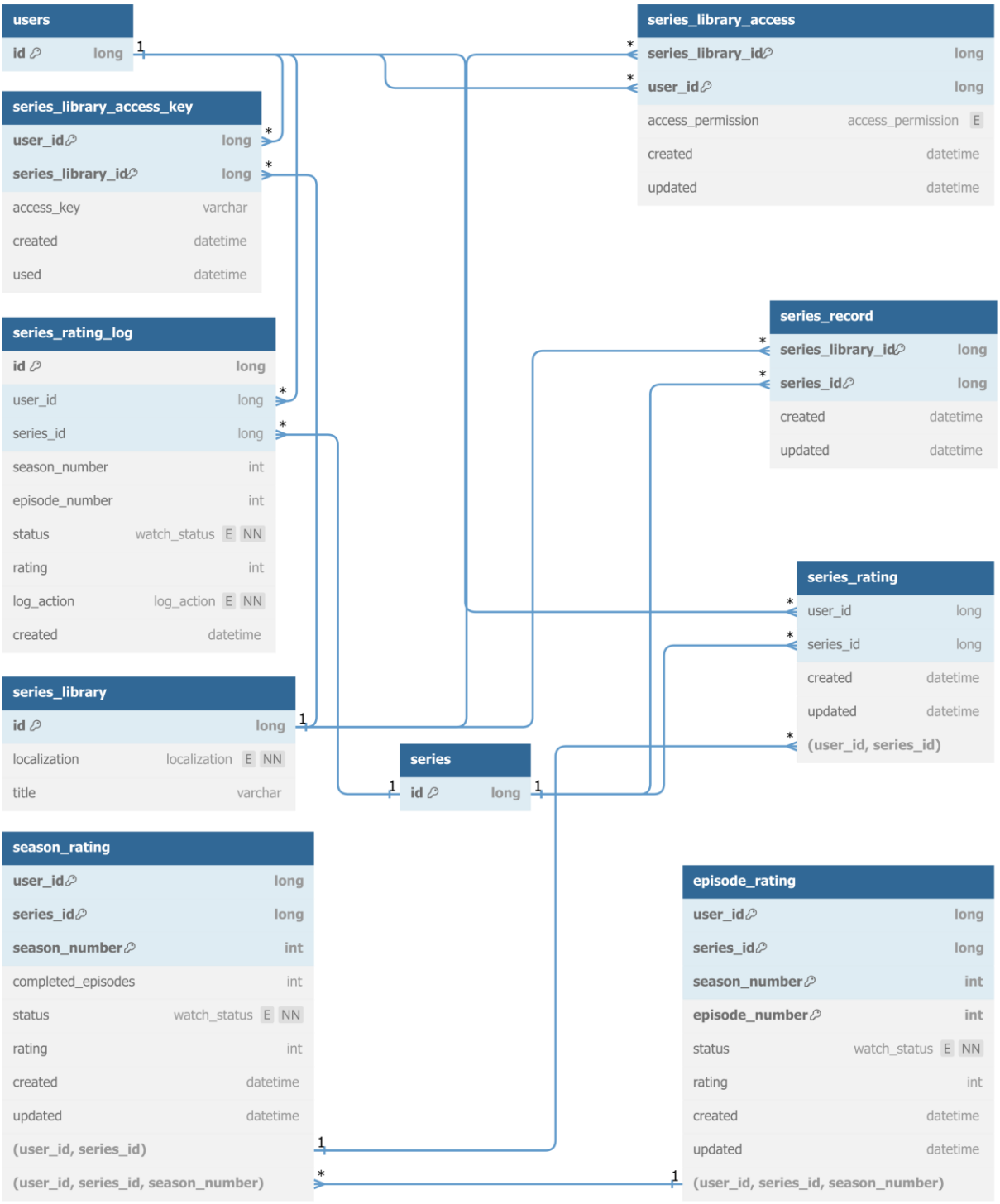
### 4.1.4 Segment seriálové cache

Model popisuje strukturu tabulek sloužících k uložení metadat seriálů. Důvod a praktika rozložení dat je shodná s cache u filmů v kapitole 4.1.2. Rozdílem je požadavek na uložení informací o více záznamech. Díky nutnosti ukládání dat sezón a epizod seriálů je tak model složitější. K veškeré identifikaci záznamů jsou použity kompozitní klíče, kde takový klíč u tabulky epizod obsahuje čtyři sloupce.


*Obrázek 25: Databázové schéma segmentu seriálové cache*

### 4.1.5 Segment seriálů

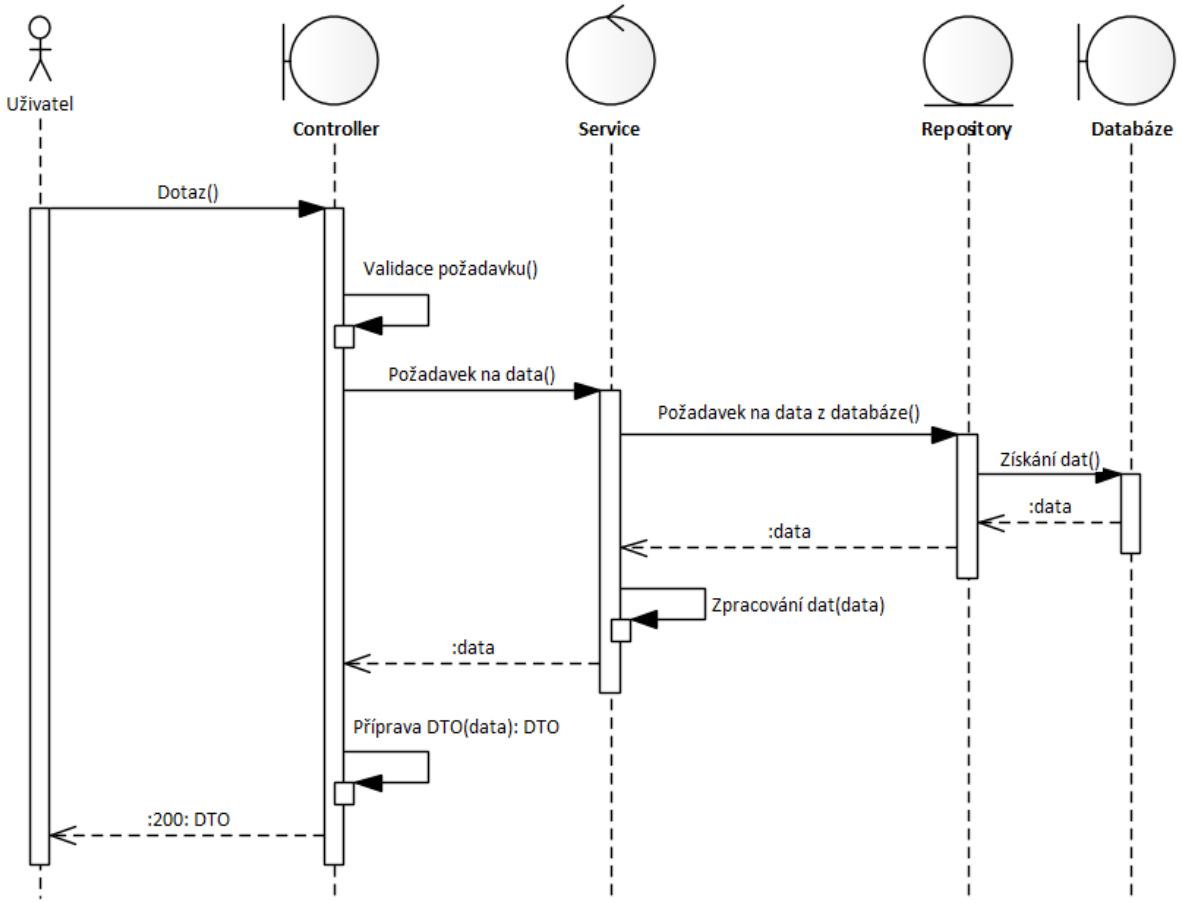
Model ukazuje strukturu provázání tabulek seriálových knihoven, jejich záznamů a uživatelských hodnocení. Popis jednotlivých částí je shodný s popisem u filmů v rámci kapitoly 4.1.3. Rozdílem je vyšší počet tabulek spojených se záznamy knihoven a příslušných uživatelských hodnocení seriálů.




*Obrázek 26: Databázové schéma segmentu seriálů*

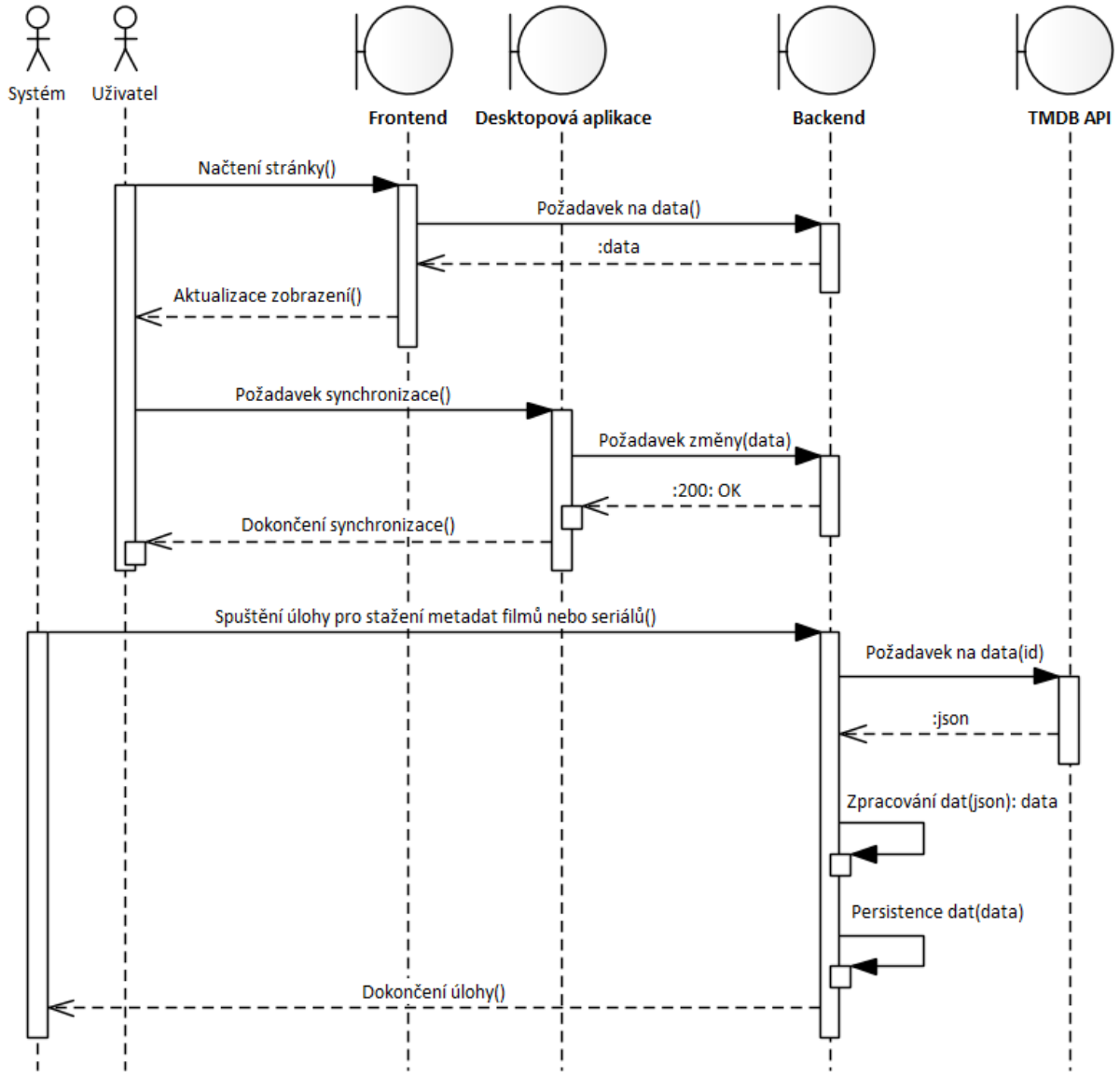
## 4.2 Backend

Model ukazuje hlavní strukturu BE tvořenou různými vrstvami, kterými dotaz prochází. Před zpracováním dotazu v komponentě controller je dotaz ověřen bezpečnostním filtrem. Při úspěšném ověření je dotaz odeslán na příslušný koncový bod komponenty controller. Prvním krokem je obecně validace požadavku. Validace většinou kontroluje přístupová práva k datům nebo správnost zadaného požadavku. Zpracování a získávání dat je primárně řešeno přes komponentu service, jejíž účel tkví v komunikaci s komponentou repository a zpracováním získaných dat. Komponenta repository obsahuje funkce pro komunikaci s databází a je jedinou komponentou, která takové spojení může navázat.




*Obrázek 27: Diagram interakce komponent BE*

Následující model znázorňuje podporovanou komunikaci BE s FE a desktopovou aplikací. Požadavky od FE jsou primárně produkovány procházením webové aplikace. Komunikace z desktopové aplikace je výhradně tvořena dotazy synchronizačního systému záznamů a hodnocení.




*Obrázek 28: Diagram interakce BE s ostatními systémy*

### 4.2.1 Zabezpečení

Zabezpečení celé aplikace je rozděleno do dvou důležitých komponent, které představují celé nastavení bezpečnosti Spring. Tvoří je nastavení „SecurityFilterChain“ a přístupového filtru „RequestFilter“, který se stará o zpracování autentizace uživatele.

#### 4.2.1.1 Security filter chain

Zabezpečení je zpracováváno řetězem různých bezpečnostních filtrů. Součástí hlavního filtru specifikujeme základní nastavení ochrany systému, zabezpečení koncových bodů a přidání jiných filtrů.

*Zdrojový kód 5: Nastavení security filter chain*

```
http
    .cors(withDefaults())
    .csrf(AbstractHttpConfigurer::disable)
    .sessionManagement((conf) ->
        conf.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
    .formLogin(AbstractHttpConfigurer::disable)
    .logout(AbstractHttpConfigurer::disable)
    .exceptionHandling((conf) -> {
        conf.authenticationEntryPoint(authenticationEntryPoint);
        conf.accessDeniedHandler(accessDeniedHandler);
    })
    .authorizeHttpRequests((requests) -> {
        requests
            .requestMatchers(new AntPathRequestMatcher("/login")).permitAll()
            .requestMatchers(new AntPathRequestMatcher("/logout")).authenticated()
            .anyRequest().denyAll()
    })
    .addFilterBefore(
        requestFilter,
        UsernamePasswordAuthenticationFilter.class
    );
```

Použitá nastavení zabezpečení dle reference výše:

- CORS je nastaven na výchozí nastavení, které je následně upřesněno konfigurační komponentou.
- CSRF je zablokováno.
- Není použito žádné cache aktuálního sezení, API je tedy plně bez-stavové.
- Nativní koncové body pro přihlášení a odhlášení jsou zablokovány.
- Jsou přidány vlastní chybové objekty pro sjednocení chybových zpráv i při zablokování přístupu bezpečnostním segmentem Spring Security.
- Nastavení přístupu k jednotlivým koncových bodů.
- Zablokování přístupu k Swagger UI koncovým bodům v produkční verzi.
- Přidání autentizačního filtru, který má za úkol ověřit identitu připojovaných uživatelů na základě jejich přístupového JWT tokenu.

#### 4.2.1.2 Request filter

Filtr dotazů pracuje pouze s ověřovací hlavičkou „Authorization“ a obsaženým JWT tokenem. Pokud je hlavička přítomná v dotazu, je otestována přítomnost JWT tokenu s prefixem „Bearer“ pro přístup z FE nebo s prefixem „Api“ pro přístup z aplikace. Pokud je zjištěna přítomnost takové hodnoty, jsou data zpracována, zkontrolována a připravena na další krok. Poslední částí je ověření správnosti tokenů, které se liší na základě prostředí, ze kterého token

pochází. Validita tokenu znamená zkontrolování existence daného klíče v systému a jestli nebyl odebrán z důvodu expirace nebo zablokování přístupu. U tokenů z FE je navíc monitorováno jejich využití.

### 4.2.2 Controller

Pří úspěšném prostupu zabezpečením následuje zpracování dotazu příslušným koncovým bodem. Každý controller má oddělenou funkčnost a zpracovává jeden úsek aplikace. Controller a jeho koncové body obsahují anotace sloužící jako profilovací identifikátory. Uživatelské koncové body obsahují OpenAPI dokumentaci, která podrobně popisuje význam koncových bodů, vstupně/výstupní hodnoty a návratové kódy. Controller z pravidla neobsahuje přílišné řešení zpracování dotazu a hlavní část zpracování je provedena příslušnou service. Na ukázce níže se nachází obvyklá implementace třídy controller napříč aplikací.

*Zdrojový kód 6: Ukázka zápisu komponenty controller*

```
@Tag(name = "User")
@ControllerId(9535750675138229 87L)
@Source(SourceSystem.WEB)
@RestController
@RequestMapping("/user")
@AllArgsConstructor
public class UserController extends ProfilingControllerAncestor {

    private final UserService userService;

    @EndpointId(5045904804523270 20L)
    @Operation(summary = "Find user by ID")
    @ApiResponses(value = {
        @ApiResponse(
            responseCode = "200",
            description = "Found user",
            content = {@Content(
                        mediaType = "application/json",
                        schema = @Schema(implementation = UserResponse.class))}),
        @ApiResponse(
            responseCode = "400",
            description = "Invalid id supplied",
            content = @Content),
        @ApiResponse(
            responseCode = "404",
            description = "User not found",
            content = @Content)
    })
    @PreAuthorize("#id == principal.id or hasRole('ADMIN')")
    @Transactional
    @GetMapping("/{id}")
    public ResponseEntity<UserResponse> findById(
        @Parameter(description = "ID of user") @PathVariable final Long id
    ) throws ResourceNotFoundException {
        var result = userService.findById(id);
        return ResponseEntity.ok(result.toResponse());
    }
}
```

### 4.2.3 Service

Service tvoří takový můstek mezi koncovými body a funkcemi repositářů pro přístup k databázi. Do service jsou většinou umístěny rozsáhlé funkcionality nebo často opakované procedury. Unikátní části kódu, které se provádějí pouze jednou (např. jednorázové kontroly přístupu nebo zpracování dat pro výstup), jsou většinou ponechávány přímo na místě ve třídě controller. V případě nepřítomnosti požadovaných dat je vrácen návratový kód 404 (nenalezeno) pomocí výjimky „ResourceNotFoundException“. Níže se nachází ukázka obecné funkcionality třídy service.

*Zdrojový kód 7: Ukázka zápisu komponenty service*

```
@Service
@AllArgsConstructor
public class UserService {

    private final UserRepository userRepository;

    public User findById(final Long id) throws ResourceNotFoundException {
        var result = userRepository.findById(id);
        if (result.isEmpty()) throw new ResourceNotFoundException();
        return result.get();
    }
}
```

### 4.2.4 Repository

Třída repository spravuje přístup k databází a přináší podporu pro automatické vytváření dotazů na základě pojmenování funkce. Při nutnosti vyšší přizpůsobitelnosti dotazu podporuje tvorbu nativních query skrze standardní syntaxi SQL nebo pomocí JPQL. Použití JPQL může přinést zajímavé výhody při hromadném načítání dat bez využití pozdního (lazy) stažení dat z databáze. Takto vytvořené dotazy jsou tedy mnohem optimalizovanější než normální přístup. V systému se takových JPQL query používá pro načtení lokalizovaných dat filmu a seriálu společně s jejich hodnocením.

*Zdrojový kód 8: Ukázka zápisu komponenty repository*

```
@Repository
public interface MovieRecordRepository extends JpaRepository<MovieRecord, MovieRecord.Id> {
    @Query("""
            select new <cesta zkrácena>.movie.MovieRecordFetchResult(mr, mra)
            from MovieRecord mr
            join mr.movie.movieInformation mi
            left join MovieRating mra on (
                mr.id.movieId = mra.id.movieId and
                mra.id.userId = :userId)
            where
                mr.id.movieLibraryId = :libraryId and
                mi.id.localization = :localization and
                mr.id.movieId = :movieId
            """)
    Optional<MovieRecordFetchResult> findById(
        final Long userId, final Long libraryId, final Long movieId,
        final Localization localization);
}
```

### 4.2.5 Domain

Domain popisuje persistentní entity ukládané v databázi, se kterými je manipulováno napříč systémem. Pro výstup k uživateli jsou výhradně používány jejich upravené reprezentace ve stylu DTO. Entity jsou označeny buďto číselným primárním klíčem nebo kompozitním klíčem složeným z více hodnot. Takové klíče jsou použity pro realizaci lokalizované cache získaných metadat filmů a seriálů. Pro příklad metadata filmu jsou složena z entity filmu, která obsahuje napříč lokalizací neměnné hodnoty. Ke každé této entitě je připojeno několik entit informací o filmu, které jsou lokalizované a označené pomocí kompozitních klíčů. Kompozitní klíče se používají podobně jako normální klíče, ale jejich správné nastavení je mnohem náročnější. Kód níže ukazuje entitu „MovieInformation“, která je identifikována kompozitním primárním klíčem.

*Zdrojový kód 9: Ukázka zápisu komponenty domain, část 1.*

```
@Data
@Entity
@NoArgsConstructor
@Table(name = "movie_information")
public class MovieInformation {

    @Data
    @Embeddable
    @NoArgsConstructor
    @AllArgsConstructor
    @SuppressWarnings("JpaDataSourceORMInspection")
    public static class Id implements Serializable {

        @Serial
        private static final long serialVersionUID = 1L;

        @Column(name = "movie_id")
        private Long movieId;

        @Column(
                name = "localization",
                nullable = false
        )
        private Localization localization;
    }

    @EmbeddedId
    private final Id id = new Id();

    @ManyToOne
    @MapsId("movieId")
    @JoinColumn(name = "movie_id")
    @EqualsAndHashCode.Exclude
    @ToString.Exclude
    @JsonIgnore
    private Movie movie;
}
```

Kód níže popisuje entitu „Movie“, která udržuje seznam lokalizovaných popisů. Na této straně vazby @OneToMany je využito restrikce CascadeType.ALL k propagaci stavů podřazeným entitám a restrikce orphanRemoval ke snadnému čištění. [63]

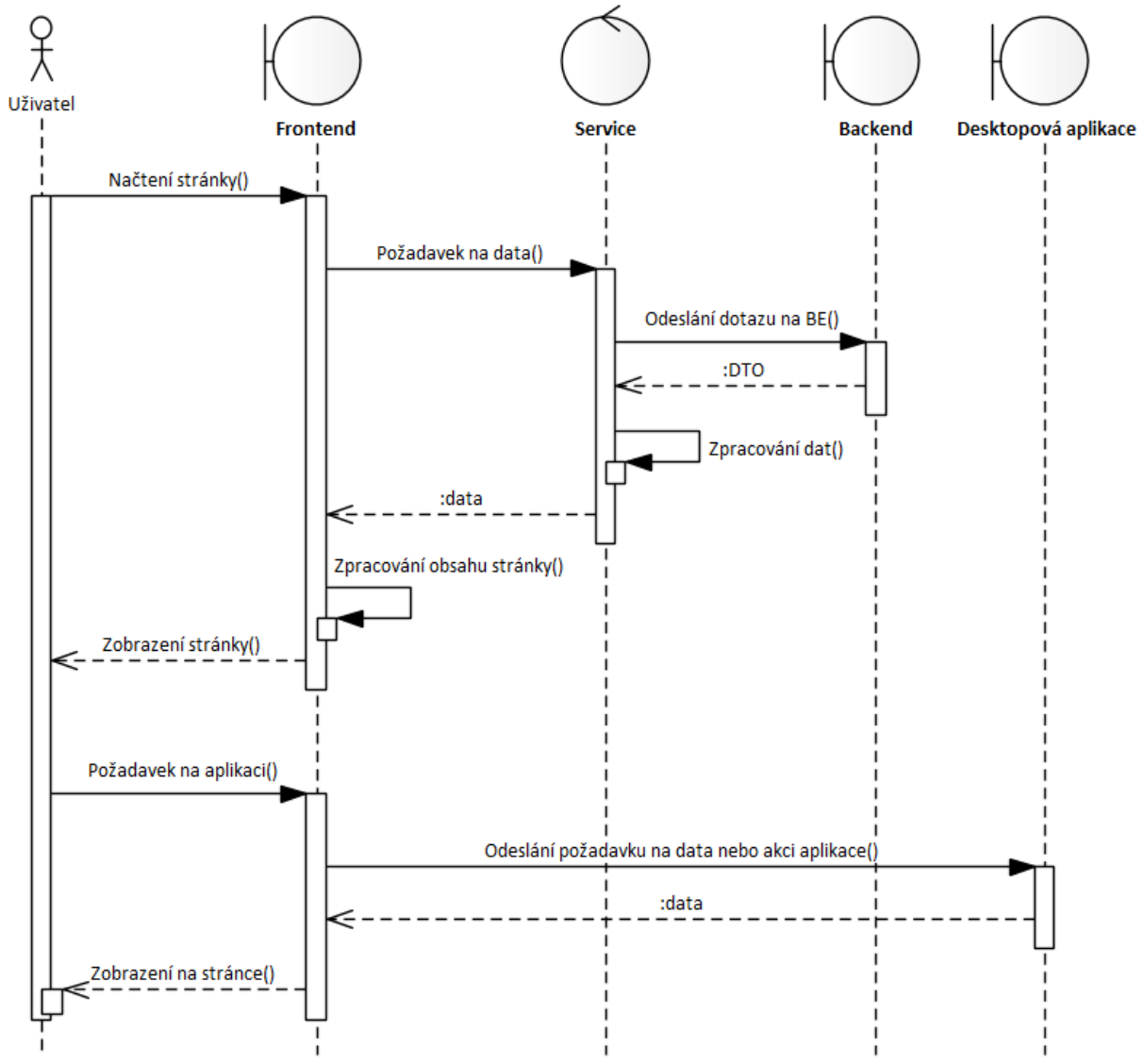
*Zdrojový kód 10: Ukázka zápisu komponenty domain, část 2.*

```
@Data
@Entity
@NoArgsConstructor
@Table(name = "movie")
public class Movie {

    @Id
    @Column(name = "id")
    private Long id;

    @OneToMany(
            mappedBy = "movie", fetch = FetchType.LAZY,
            cascade = CascadeType.ALL, orphanRemoval = true
    )
    @EqualsAndHashCode.Exclude
    @ToString.Exclude
    @JsonIgnore
    private Set<MovieInformation> movieInformation = new HashSet<>();
}
```

## 4.3 Frontend

### 4.3.1 Architektura

Model popisuje hlavní strukturu FE webové aplikace. Požadavek uživatele je scriptem zpracován a pomocí funkcí komponenty service odeslán na backend API. Odezva BE je obdržena ve formě DTO, které je transformováno do známých objektů v FE. Objekty jsou dále zpracovávány napříč různými komponentami FE pro vykreslení UI. Výstupem uživatelského požadavku je aktualizace FE a zobrazení změn. FE obsahuje metody pro komunikaci s desktopovou aplikací. Požadavky na aplikaci mohou být pro získání souborových metadat nebo spuštění vybraného záznamu v rozhraní webové aplikace.




*Obrázek 29: Diagram interakce komponent FE*

Kořenovým souborem FE je „index.html“, který připojuje JavaScript modul „main.tsx“ s veškerou funkčností. Modul se stará o vykreslení celého FE. Součástí je webový směrovač, poskytovatel dat ze „store“, poskytovatel překladů a samotná React aplikace.

*Zdrojový kód 11: Nastavení kořenového scriptu FE*

```
ReactDOM.createRoot(document.getElementById('root') as HTMLElement).render(
    <BrowserRouter>
        <Provider store={store}>
            <I18nextProvider i18n={i18n}>
                <App/>
            </I18nextProvider>
        </Provider>
    </BrowserRouter>
);
```

### 4.3.2 Aplikace

V souboru „App.tsx“ se nachází směrovač pro poskytování překreslení stránky v rámci jednostránkové aplikace. Každá cesta směrovače vede na jednu z podporovaných stránek, které mají postfix „Page.tsx“, a také obsahuje výchozí cestu pro případ neexistence vybrané cesty. Podporované cesty jsou centralizovány v enum URL a jsou uloženy v jejich typické reprezentaci adres React.

*Zdrojový kód 12: Ukázka navigačního směrovače React*

```
<Routes>
    <Route path={URL.LANDING} element={<LandingPage/>}/>
    <Route path={URL.USER} element={<ProfilePage/>}/>
    <Route path="*" element={<BlankPage/>}/>
</Routes>

export enum URL {
    LANDING = "/landing",
    USER = "/user/:userId"
}
```

Pro potřeby tvorby dynamických odkazů byl vytvořen script, který dosazuje požadované parametry do šablony vybrané cesty/adresy. Pro směrování v aplikaci je obvykle použit „useNavigate“.

*Zdrojový kód 13: Funkce pro tvorbu dynamických odkazů*

```
export const URLBuilder = (urlType: URL, ...params: any[]): string => {
    let url = urlType.toString();
    for (let param of params) {
        url = url.replace(new RegExp("(\\/)(:.*?)(\\/|$)"), `$1${param}$3`);
    }
    return url;
}
```

### 4.3.3 Komponenty

Komponenty obvykle pochází z balíčku „component“ a hlavní z nich jsou odkazovány v souborech s postfixem „Page.tsx“. Pro posílání dotazů na backend jsou využívány sborníky funkcí pojmenované s postfixem „Service.tsx“. Struktura funkcí pro volání je standardizovaná napříč FE. Volání takových funkcí je vždy asynchronní a zadané parametry obsahují metodu návratu získaných dat. Návrat může být řešen například funkcí nebo přidáním stavového nastavovače. K odesílání dotazů je využívána knihovna Axios, kde po obdržení odpovědi jsou data zpracována a odeslána návratovou metodou zpět. V případě chyby je funkčnost řešena globálním správcem chyb nebo jsou chyby zpracovány jinak před zpracováním takovým správcem.

*Zdrojový kód 14: Funkce pro komunikaci s BE*

```
async searchUsersByUsername(
    username: string,
    setUsers: React.Dispatch<React.SetStateAction<Array<User>>> |
              ((users: Array<User>) => void)
) {
    Api
        .get<Array<User>>(`/user/search/username/${username}`)
        .then(response => {
            setUsers(response.data);
        })
        .catch(e => {
            exceptionHandler.handleException(e);
        });
}
```

### 4.3.4 Design aplikace

Frontend webové aplikace využívá knihovny MUI k jednoduché organizaci a použití částí UI. Pro stylizaci jsou využity stylové šablony pouze na místech s nejvyšší nutností. Primárně se jedná o úpravy speciálních komponent, kde by byla změna designu přes styly složitá nebo nemožná.

Styly různých komponent byly ponechány přímo v atributech „sx“, které po sestavení přemosťují styly do atributu „style“. Alternativou tohoto přístupu je obvyklá tvorba souborů se styly, kde každý styl je s elementem provázán pomocí tříd nebo identifikátorů. Této varianty nebylo využito z důvodu vysokého množství dynamicky měněných komponent a přílišného navýšení práce. Takto oddělené styly ve svých souborech sice nabízejí jisté klady, jako například lepší čitelnost kódu a znuvupoužitelnost, ale jejich tvorba zvyšuje práci pro vývojáře. Aktuální řešení sice zhoršuje pár věcí, ale přináší rychlý způsob změn v designu aplikace, která je složena z vysokého množství různých komponent.

Samotný design webové aplikace je inspirován stylizací desktopové aplikace, na které si staví, ale zároveň ji rozšiřuje o možná vylepšení. React s MUI jsou mnohem mocnějšími pro tvorbu dynamických UI než Java FX. Jedinou použitou věcí z aplikace byl seznam barev aplikace, který byl rozšířen a upraven pro nové potřeby. Žádný z kódů pro tvorbu UI nebylo možné použít díky přílišnému rozdílu mezi technologiemi. Celý design webové aplikace byl tedy vytvořen od úplného základu s inspirací v desktopové aplikaci. Byl kladen důraz na podobnost obou aplikací z důvodu propojení nejen na BE úrovni, ale i na té FE.

### 4.3.5 Propojení s desktop aplikací

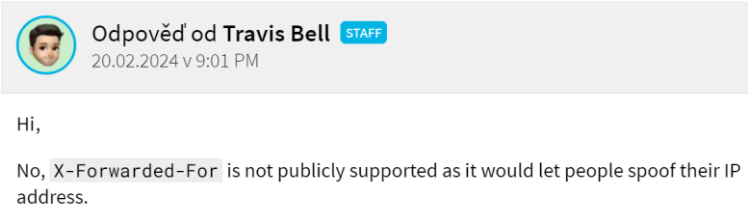
Frontend obsahuje grafický identifikátor, který slouží k signalizaci aktuálního stavu desktopové aplikace. V případě aktivní desktopové aplikace na pozadí počítače je frontend schopný zjistit tuto skutečnost a zobrazit tak informaci o tom, že registruje propojení s aplikací. Propojení je primárně určeno pro budoucí vývoj, který bude zahrnovat postupnou migraci celého systému na webové rozhraní a přesun funkcionality desktopové aplikace pouze do pozadí počítače. Více informací je zmíněno v závěru práce.

## 4.4 Nginx server

Nginx server je jednou z nejdůležitějších částí webové aplikace a jeho správné nastavení je tak kritické pro celý systém. Na správný běh Nginx je také navázána funkčnost desktopové aplikace, ale k tomu později. Vytvořená konfigurace serveru je rozdělená na nastavení globální a serverové.

V následujících kapitolách se budeme věnovat již zmíněnému globálnímu nastavení, kde budou ukázány nejdůležitější parametry pro bezpečný a bezproblémový běh Nginx. Dále jsou rozebraná jednotlivá nastavení serverů, které se starají o přeposílání dotazů a poskytování služeb FE.

V původním návrhu byla tvorba reverzního proxy serveru, který by směroval všechny dotazy z aplikace na TMDB API. Díky tomu by byla odebrána nutnost podpory přístupového klíče a síťová omezení by byla eliminována pomocí hlavičky „X-Forwarded-For“. Na vývoji této funkcionality byl stráven celý pracovní den času, a nakonec musela být plně odebrána. Po komunikaci s podporou TMDB bylo zjištěno, že nepodporují použití této hlavičky z důvodu možného zneužití limitu záměnou IP adres.




*Obrázek 30: Odpověď personálu TMDB v reakci na X-Forwarded-For*

Jedinou výhodou této nevyužité tvorby byl nález při tvorbě reverzních proxy serverů směřujících do venkovní sítě. Pro zprovoznění takového serveru je totiž nutné přidat nastavení „proxy_ssl_server_name on“, které Nginx serveru signalizuje, aby byl použit název cílového externího serveru. [64]

### 4.4.1 Globální nastavení

Globální nastavení určuje základní vlastnosti související s právy běžícího procesu serveru, vyrovnávání zátěže, znakovou sadou, stanovením známých datových typů, logování a SSL zabezpečení. Součástí tohoto nastavení jsou také globální proměnné (například map) používané pro nastavení jednotlivých serverů a samotné připojení ostatních konfiguračních souborů.

Základní kostra pro konfigurační soubor globálního nastavení byla získána z generátoru konfigurací na stránce DigitalOcean. Pro vytvoření byl zvolen přednastavený profil pro jednostránkovou aplikaci. Požadované hodnoty byly pravdivě vyplněny a základní konfigurace tak byla vytvořena. Získané nastavení je původně rozdělené do několika souborů. Z hlediska přehlednosti byly různé soubory spojeny do dvou konfiguračních souborů. První z nich je toto globální nastavení a další je serverové, které bude rozebráno v nadcházejících kapitolách. [65]

V konfiguraci je nastaven specifický uživatel k procesu Nginx z důvodu poskytnutí potřebných oprávnění k manipulaci souborů na disku. Nastavení bylo přidáno pro přístup ke složce určené k uložení ověřovacích klíčů na aktivaci SSL certifikátů. Konfigurace dále vyžaduje přidání souboru s podporovanými datovými typy „mime.types“. Tento soubor byl stažen z GitHub repositáře docker-tomcat7-cluster uživatele djangofan. [66]

Celý globální konfigurační soubor je dostupný v příloze 1. Konfigurační soubor serverů rozebíraných v následujících kapitolách je k nalezení v příloze 2. Za zmínku stojí vytvoření mapy webových adres, které nemají být přesměrovány pomocí automatického přesměrování subdomén. Přesněji se jedná o subdoménu „api“, která je použita pro směrování dotazů na backend webové aplikace. [67]

*Zdrojový kód 15: Mapa výjimek subdomén*

```
map "$host" $match {
    "api.pallidium.cz" 0;
    default 1;
}
```

### 4.4.2 HTTP server

Hlavním účelem tohoto serveru je přesměrování jakéhokoliv dotazu pomocí protokolu HTTP na hlavní stránku webové aplikace pod zabezpečeným protokolem HTTPS. Výjimkou této činnosti je veřejná složka „/.well-known/acme-challange“, na kterou se odkazuje certifikační autorita Let’s Encrypt z důvodu ověření pravosti stránky. Samotná konfigurace tedy obsahuje odposlech na síťovém portu 80, je spuštěna na jakékoliv subdoméně, obsahuje výjimku pro výše zmíněný cíl a jinak přesměrovává veškerou komunikaci na hlavní stránku.

*Zdrojový kód 16: Nastavení HTTP serveru*

```
server {
    listen                  80;
    listen                  [::]:80;
    server_name             .pallidium.cz;

    location ^~ /.well-known/acme-challenge/ {
        root /var/www/_letsencrypt/;
    }

    location / {
        return 301 https://www.pallidium.cz$request_uri;
    }
}
```

### 4.4.3 HTTPS servery pro hlavní portál

Prvním z těchto serverů je server na síťovém portu 443 (HTTP/SSL), který přesměrovává veškeré dotazy mimo hlavní stránku zpět na ní. Toto přesměrování obsahuje podmínku využívající globální proměnnou „$match“ zmíněnou v kapitole 4.4.1. Přesměrování je tedy spuštěno pouze v případě, že zavolaná subdoména není ve specifikovaném seznamu. Úseky kódu v komentářích slouží k nastavení SSL v kapitole 4.4.5.

*Zdrojový kód 17: Nastavení HTTPS serveru pro přesměrování*

```
server {
    listen                  443; # ssl;
    listen                  [::]:443; # ssl;
    http2                   on;
    server_name             .pallidium.cz;

    #;#ssl_certificate /etc/letsencrypt/live/pallidium.cz/fullchain.pem;
    #;#ssl_certificate_key /etc/letsencrypt/live/pallidium.cz/privkey.pem;
    #;#ssl_trusted_certificate /etc/letsencrypt/live/pallidium.cz/chain.pem;

    if ($match) {
        return 301 https://www.pallidium.cz$request_uri;
    }
}
```

Druhý server je pro hlavní stránku webové aplikace. Server obsahuje kromě nastavení serveru a SSL také hlavičky pro zvýšení bezpečnosti, zákaz přístupu k „.well-known“ prostoru, nastavení logování provozu a hlavně nastavení přístupu k souborům webové stránky.

*Zdrojový kód 18: Nastavení hlavního HTTPS serveru*

```
server {
    listen                  443; # ssl;
    listen                  [::]:443; # ssl;
    http2                   on;
    server_name             www.pallidium.cz;

    #;#ssl_certificate         /etc/letsencrypt/live/pallidium.cz/fullchain.pem;
    #;#ssl_certificate_key     /etc/letsencrypt/live/pallidium.cz/privkey.pem;
    #;#ssl_trusted_certificate /etc/letsencrypt/live/pallidium.cz/chain.pem;

    add_header X-XSS-Protection             "1; mode=block" always;
    add_header X-Content-Type-Options       "nosniff" always;
    add_header Referrer-Policy              "no-referrer-when-downgrade" always;
    add_header Content-Security-Policy      "default-src 'self' http: https: ws: wss: data:
blob: 'unsafe-inline'; frame-ancestors 'self';" always;
    add_header Permissions-Policy           "interest-cohort=()" always;
    add_header Strict-Transport-Security    "max-age=31536000; includeSubDomains" always;

    location ~ /\.(?!well-known) {
        deny all;
    }

    location / {
        root                /usr/share/nginx/html;
        index               index.html;
        try_files           $uri $uri/ /index.html =404;
    }
}
```

Doplňkem nastavení serveru je vypnutí logování přístupu k souboru „favicon.ico“, který prohlížeče vyžadují a vyhodí chybu v případě jeho nepřítomnosti, a souboru „robots.txt“, který je využíván k instruování vyhledávacích robotů, jak procházet stránky. [68], [69]

*Zdrojový kód 19: Dodatečné nastavení Nginx serveru*

```
location = /favicon.ico {
    log_not_found off;
    access_log off;
}

location = /robots.txt {
    allow all;
    log_not_found off;
    access_log off;
}
```

### 4.4.4 Reverse proxy pro backend

Frontend webové aplikace musí mít přístup ke BE API a taková komunikace musí být zabezpečená. Situace je vyřešena nastavením Nginx serveru jako reverzní proxy s cílem v lokální síti. Taková proxy zastává funkci prostředníka v komunikaci mezi klientem a cílovým serverem. Proxy server překládá požadavky klienta vůči serveru a přebírá jeho odpověď, kterou následně zasílá zpět klientovi. V tomto případě jsou dotazy pro „api.pallidium.cz“ přeposlány do lokální sítě Docker kontejnerů na statickou adresu backend API. [70]

*Zdrojový kód 20: Nastavení reverzního proxy serveru*

```
server {
    listen                  443; # ssl;
    listen                  [::]:443; # ssl;
    http2                   on;
    server_name             api.pallidium.cz;

    #;#ssl_certificate         /etc/letsencrypt/live/pallidium.cz/fullchain.pem;
    #;#ssl_certificate_key     /etc/letsencrypt/live/pallidium.cz/privkey.pem;
    #;#ssl_trusted_certificate /etc/letsencrypt/live/pallidium.cz/chain.pem;

    access_log /var/log/nginx/access.log combined buffer=512k flush=1m;
    error_log  /var/log/nginx/error.log warn;

    location / {
        proxy_read_timeout                  90;
        proxy_set_header Host               $host;
        proxy_set_header X-Real-IP          $remote_addr;
        proxy_set_header X-Forwarded-For    $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto  $scheme;

        proxy_pass http://172.10.0.3:8000;
    }
}
```

K dotazu přeposlaném od klienta jsou přidány speciální hlavičky, které informují cílový server o klientovi. Takové hlavičky obsahují informace o klientově IP adrese, trasovacích cookies, preferenci cache, autorizaci a dalších. V tomto případě se používají pouze základní hlavičky o klientovi. [71]

### 4.4.5 SSL

Napříč konfigurací serverů se nalézají za-komentované úseky kódu zakazující SSL. Je tomu z důvodu nepřítomnosti SSL klíčů při spuštění Docker kontejneru s Nginx. SSL klíče jsou po spuštění buďto nově vytvořeny nebo nahrány staré. Po dokončení přesunu klíčů jsou části nastavení SSL od-komentovány a Nginx server restartován. Při jeho dalším spuštění je tak aktivováno SSL šifrování komunikace. Změny v konfiguraci ohledně SSL jsou více rozebrány v kapitole 6.8.5.

K odebrání komentářů se používá příkaz „sed“, který prohledá cílový soubor a nalezené speciálně označené komentáře odstraní. K příkazu jsou přidány tři volby: „**-i**“ (provede příkaz na místě přepsáním původního souboru), „**-r**“ (pro identifikaci je použit regex) a „**-z**“ (soubor je čten po řádcích). [72]

*Zdrojový kód 21: Příkaz pro odebrání speciálních komentářů*

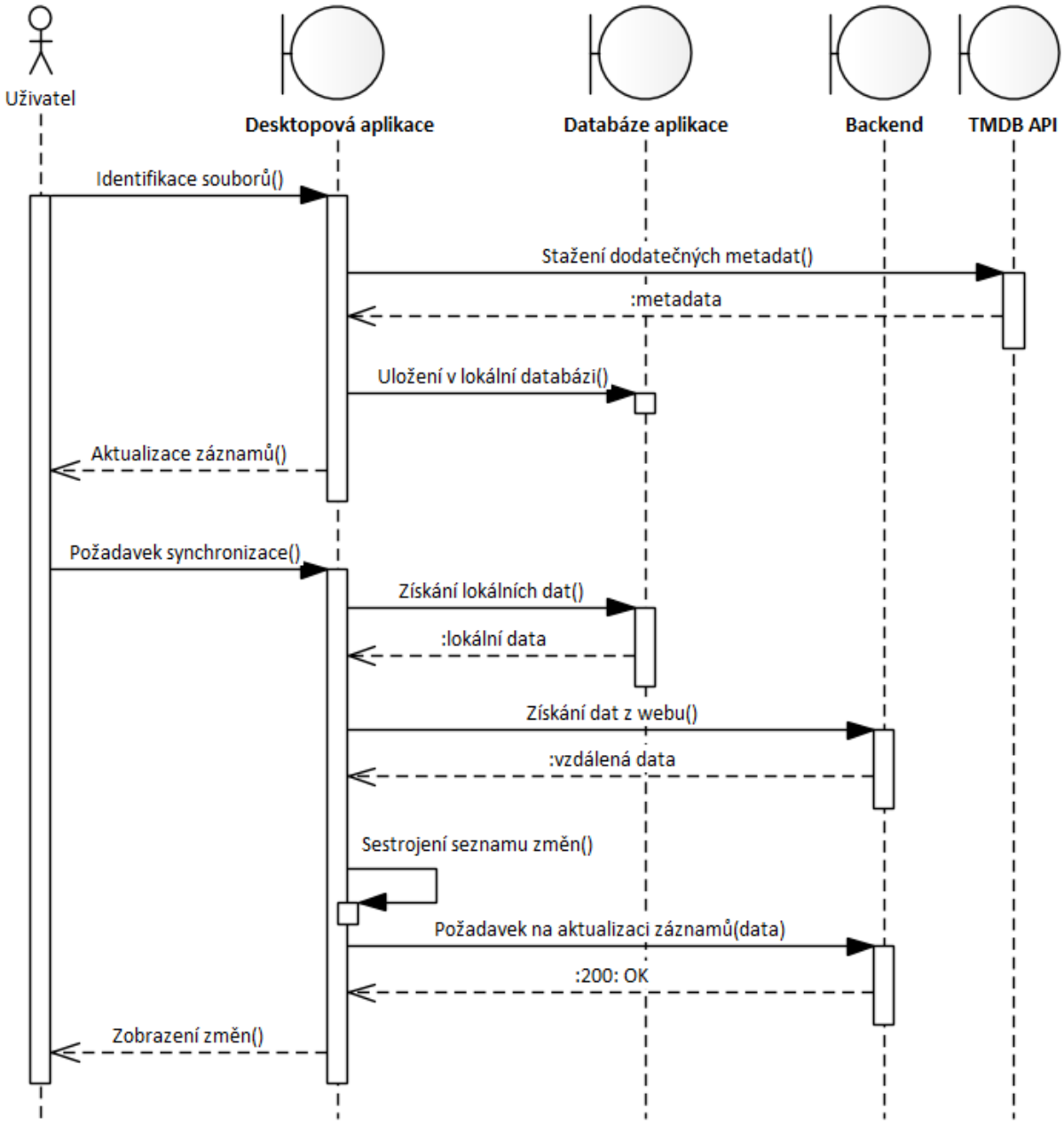
```
sed -i -r -z 's/#?; ?#//g' default.conf
```

## 4.5 Desktopová aplikace

### 4.5.1 Architektura

Model zjednodušeně popisuje činnost upravené aplikace. Úplné vnitřní dělení není nijak zobrazeno, jelikož je vysvětleno v původní práci. Aplikace komunikuje dvěma směry:

- TMDB API pro stahování potřebných multimediálních metadat,
- BE API webové aplikace pro účely synchronizace.




*Obrázek 31: Diagram interakce desktopové aplikace*

### 4.5.2 Implementované změny

Do aplikace byl implementován detailnější systém hodnocení s lepším způsobem ukládání, než bylo původní řešení. Rozhraní nastavení knihoven filmů a seriálů bylo rozšířeno o nastavení synchronizace. Součástí této záložky je zobrazení aktuálního cíle synchronizace a možnost přidání či otestování nového spojení.

Ovládání knihovny bylo rozšířeno o synchronizační funkcionalitu, která má za cíl udržet lokální hodnocení a soubory konzistentní s webovou verzí.

## 4.6 Synchronizace

Pro splnění zadaných požadavků byl implementován systém synchronizace, který je podobný funkcionalitě systému verzování jako Git. V tomto případě nejsou ukládány detailní verze, a není tak podporována jejich obnova. Celý synchronizační systém slouží k aktualizaci hodnocení jak ve webové aplikaci, tak v její desktopové podobě. Systém tak dovoluje vytváření hodnocení bez přístupu k internetu a poté rychlé synchronizování s webovou verzí aplikace.

V rámci kapitoly budou používány přirovnání:

- **Aplikace** – popisující desktopovou verzi aplikace a její **lokální** data.
- **Web** – popisující webovou verzi aplikace a její data na internetu.

Synchronizační systém podporuje dva směry synchronizace:

- z aplikace do webu,
- z webu do aplikace.

### 4.6.1 Změny hodnocení

Jedná se o dodatečnou volbu při synchronizaci jak z aplikace na web, tak z webu do aplikace. Volba specifikuje, co se má stát s hodnoceními v rámci dokončení synchronizace.

Nastavení změn hodnocení:

- **Žádné** – Hodnocení nebude nijak upraveno i při dostupných změnách.
- **Automatická aktualizace** – Hodnocení budou upravena na základě časové známky, kde novější hodnocení přepíše to staré.
- **Přepsání** – Hodnocení budou přepsána nehledě na jejich časové známky.

### 4.6.2 Synchronizace z aplikace do webu

Tímto synchronizačním režimem chápejme přenos informací z desktopové aplikace do webové. Synchronizace tímto režimem začíná stažením celého seznamu záznamů příslušné knihovny. Záznamy jsou podrobeny otestování jejich přítomnosti v desktopové aplikaci a je tak vytříděn seznam požadovaných změn. Dle vybraného režimu synchronizace je s daty ve webové aplikaci naloženo jinak.

Mezi režimy synchronizace patří:

- **aktualizace**,
- **pouze unikátní**,
- **zrcadlení**.

#### 4.6.2.1 Aktualizace

Záznamy získané z webové verze, které mají svůj protějšek v desktopové aplikaci, jsou porovnány, zdali nedošlo k jejich změně. Změnou se rozumí změna pouze v lokální verzi, nikoliv změna vytvořená na webu. Za změnu je považována odlišnost v časové známce hodnocení. Změněné soubory jsou zařazeny do seznamu změn. Lokální záznamy, které nejsou přítomny na webu jsou také přidány do seznamu změn. Záznamy pouze na webu nejsou nijak brány v potaz. Seznam změn je poté odeslán webové aplikaci na zpracování.

#### 4.6.2.2 Pouze unikátní

Režim odebírá kontrolu změn v záznamech a za změnu počítá pouze záznamy přítomné pouze v desktopové aplikaci. Pouze tyto záznamy jsou začazeny do seznamu změn a odeslány do webové aplikace.

#### 4.6.2.3 Zrcadlení

Režim zrcadlení ignoruje veškeré kontroly přítomnosti, jak v desktopové aplikace, tak v té webové. Všechny záznamy desktopové aplikace jsou zařazeny do seznamu změn a odeslány webové aplikace. Režim smaže všechny nepřítomné záznamy oproti seznamu změn a přidá ty, které nebyly přítomny. Již přítomné záznamy jsou aktualizovány.

### 4.6.3 Synchronizace z webu do aplikace

Tento synchronizační režim pracuje opačným způsobem než výše popsaný. Jeho účelem je aktualizace záznamů v desktopové verzi aplikace. Režim pracuje podobně jak již probraný, kde začátkem získá veškeré záznamy knihovny z webové aplikace. Následuje vytřídění požadovaných změn dle vybraného režimu a jejich okamžitá lokální úprava.

Mezi režimy synchronizace patří:

- **aktualizace**,
- **zrcadlení**.

#### 4.6.3.1 Aktualizace a Zrcadlení

Režim nalezne společné záznamy lokální a webové knihovny, nad kterými provede porovnání časů úprav. Záznamy, které byly změněny jsou lokálně upraveny. Celý režim synchronizace do aplikace je omezen na funkčnosti, jelikož nemůže existovat záznam v desktopové aplikaci bez nějaké reprezentace pomocí souboru na disku. Režimy jsou tedy omezeny pouze na úpravy časů přidání záznamů a úprav v hodnoceních. Výchozím nastavením pro aktualizaci je automatická aktualizace hodnocení závislá na jejich časových známkách. Zrcadlení má ve výchozím nastavení přepis všech lokálních hodnocení dle jejich protějšku ve webové aplikaci.

### 4.6.4 Shrnutí

Díky podporovaným režimům synchronizace je možné jednoduše aktualizovat hodnocení záznamů a aktualizovat je napříč webovou aplikací. Před lokálními úpravami je vždy doporučeno stáhnout si nejnovější změny aktualizací do aplikace, která zaručí konzistenci a nezpůsobí nutnost opakovat hodnocení pro již hodnocený záznam.

V tabulkách níže jsou shrnuty všechny režimy synchronizace a záznamy, na které se režim vztahuje. Takto vybrané záznamy jsou aktualizovány

*Tabulka 4: Synchronizace dat z desktopové aplikace do webové aplikace*

| Aplikace → web | Cílená přítomnost souboru | | | |
|---|---|---|---|---|
| Režim synchronizace | Lokální | Společné | Web | Cílený stav záznamu |
| Aktualizace | ✓ | ✓ | | Změněné a nové |
| Pouze unikátní | ✓ | | | Pouze nové |
| Zrcadlení | ✓ | ✓ | ✓ | Všechny |

*Tabulka 5: Synchronizace dat z webové aplikace do desktopové aplikace*

| Web → aplikace | Cílená přítomnost souboru | | | |
|---|---|---|---|---|
| Režim synchronizace | Lokální | Společné | Web | Cílený stav záznamu |
| Aktualizace | | ✓ | | Změněné a nové |
| Zrcadlení | | ✓ | | Všechny |

## 4.7 Sociální síť

Webová aplikace obsahuje prvky pro podporu sociálních aktivit mezi uživateli. Každý registrovaný uživatel má možnost vyhledávat uživatele dle jejich uživatelského jména. V případě zájmu může cílovému uživateli odeslat žádost o přátelství. Taková žádost je následně zobrazeny mezi příchozími žádostmi u příjemce a mezi odeslanými žádostmi u odesílatele. Odesílatel může odeslanou žádost kdykoliv zrušit nebo v případě již propojeného přátelství, takové spojení rozvázat.

### 4.7.1 Sdílení knihoven

Při úspěšném navázání přátelství mají oba zúčastnění mezi sebou možnost sdílení svých knihoven. Sdílení může jeden z uživatelů nastavit v nastavení vybrané knihovny. V kategorii „Přístup“ se nachází ovládací panel pro synchronizaci knihoven a pod ním nastavení správy přístupu ke knihovně. Vlastník knihovny může přiřadit přístupová práva jinému uživateli a umožnit mu tak přístup k datům v knihovně.

Přístupová práva se dělí do tří kategorií, kde zakladatel knihovny má vždy práva vlastníka. Dalším právem je přístup v režimu prohlížejícího, kde uživatel vidí záznamy knihovny a může přidávat vlastní hodnocení, ale nemůže měnit obsah knihovny. Rozšiřujícím právem je režim přispěvatele, který může v knihovně libovolně přidávat nebo odebírat soubory.

Systém sdílení knihoven využijí uživatelé, kteří chtějí sdílet své oblíbené filmy nebo seriály. Oblíbené záznamy jsou umístěny do vhodně pojmenované knihovny a nasdíleny přátelům. Přátelé si posléze mohou přidávat k těmto záznamům svá hodnocení nebo pokud mají práva, tak seznam rozšiřovat.

### 4.7.2 Přátelé

Každý přihlášený uživatel má přístup ke svému seznamu přátel. Výběrem nějakého z přátel může navštívit jeho veřejný profil se statistikami.

### 4.7.3 Uživatelský profil

Uživatelé mají přístup ke svému uživatelskému profilu i profilu jiných uživatelů. Profil obsahuje základní informace o uživateli a jeho aktuální stav aktivity. Pokud byl uživatel aktivní v posledním 5 minutách, je jeho status zelený (aktivní). Při delší nečinnosti se změní status na oranžový (nečinný) a po vypršení aktivní doby se stává červeným (neaktivní). Součástí profilu je přehled o sociální stránce a vyvářených knihovnách. Nedílnou součástí profilu jsou také grafické vizualizace aktuálního stavu filmů a seriálů. Pomocí barevně odděleného grafu je vizualizován postup a poměr dokončených a nedokončených záznamů.

## 4.8 Testování

Detailnímu testování nebyla kladena příliš velká pozornost a testy byly pokryty pouze kontroly oprávnění uživatelů. Jedná se o sekci, která je z hlediska zabezpečení nejzásadnější a při jejím splnění je zamezeno neoprávněným přístupům uživatelů k cizím datům. Díky striktnímu zabezpečení většiny koncových bodů je vytváření testů poněkud ztíženo.

Pro možnosti testování byly vytvořeny usnadňující anotace, které se starají o zasazení testovacího účtu do aktuální relace přihlášení v rámci Spring Security. Při testování je využito anotace @WithMockUser, které jsou dány specifické hodnoty podle typu uživatelského účtu. Tato anotace se stará o simulaci přihlášení uživatele bez nutnosti posílání dotazu na zpracování. Další variantou může být anotace @WithUserDetails, která oproti již zmíněné vyžaduje náročnější přípravu a dodatečné nastavení UserDetailsService. Díky tomu je zvolena jednodušší varianta a jsou vytvořeny dvě anotace. První anotace @WithMockUser_Admin se stará o přihlášení administrátora a druhá @WithMockUser_User přihlášení uživatele. Anotace následně pro každý test simulují přihlášeného uživatele a poskytují tak možnost otestování jednotlivých koncových bodů nejen na funkčnost, ale i jejich zabezpečení. [73]

*Zdrojový kód 22: Anotace s právy uživatele a administrátora*

```
@Retention(RetentionPolicy.RUNTIME)
@WithMockUser(
        username = "user",
        password = "user",
        authorities = {"ROLE_USER"}
)
public @interface WithMockUser_User {
}

@Retention(RetentionPolicy.RUNTIME)
@WithMockUser(
        username = "admin",
        password = "admin",
        authorities = {"ROLE_ADMIN", "ROLE_USER"}
)
public @interface WithMockUser_Admin {
}
```

Pro ukládání je využita databáze H2, která pracuje v paměti procesu, umožňující jednoduché a rychlé otestování korektnosti persistence dat. Pomocí předchůdce testů je před každým testem naplněna databáze základními testovacími daty a po jejich dokončení jsou data vždy odebrána. Každý pokus testu je tedy izolován a testy se nijak neovlivňují. Databáze v paměti dále zaručuje automatické vyčištění po dokončení.

*Zdrojový kód 23: Ukázka přístupových testů*

```
@ActiveProfiles("test")
@SpringBootTest
class UserControllerTest extends ControllerTestAncestor {

    @Test
    @WithMockUser_User
    void givenUser_whenPersonalInfoIsRetrieved_thenReturn200() throws Exception {
        mockMvc.perform(
            get("/user/" + user.getId()
        ).with(authentication(Example.createAuthentication(
            user,
            Example.createPrincipal(user),
            userAuthorities
        )))).andExpect(status().isOk());
    }

    @Test
    @WithMockUser_User
    void givenUser_whenUserInfoOfAnotherUserIsRequested_thenReturn403() throws Exception {
        mockMvc.perform(
            get("/user/" + admin.getId()
        ).with(authentication(Example.createAuthentication(
            user,
            Example.createPrincipal(user),
            userAuthorities
        )))).andExpect(status().isForbidden());
    }
}
```

## 4.9 Profilování

Všechny použité koncové body ve vytvořené webové aplikaci podporují tvorbu statistických dat, které v hlavní řadě souvisejí s rychlostí prováděných dotazů na takový koncový bod. Každý controller obsahuje anotace specifikující jeho unikátní identifikátor napříč systémem. Dále každý koncový bod obsahuje podobný unikátní identifikátor napříč svojí třídou.

*Zdrojový kód 24: Anotace pro profilování*

```
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
public @interface ControllerId {
    long value();
}

@Target({ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
public @interface EndpointId {
    long value();
}
```

Takto označené koncové body jsou pomocí AspectJ vybrány a monitorovány. K zaměření na tyto anotace je využit pointcut.

```
@Pointcut("@within(io.github.ppetrbednar.pallidium.core.profiling.ControllerId) && " +
          "@annotation(io.github.ppetrbednar.pallidium.core.profiling.EndpointId)")
public void profilingEnabledEndpoints() {
}
```

Pointcut je využit pro zacílení na specifické koncové body a je i využit pro označení provozu z webové aplikace nebo synchronizace pomocí desktopové aplikace. Pro účely profilování a tvorby log záznamu API je využit advice @Around. Ten dovoluje zjistit potřebné informace o dotazu před jeho provedením a spustit měření délky trvání provádění.

*Zdrojový kód 25: Ukázka profilování koncových bodů*

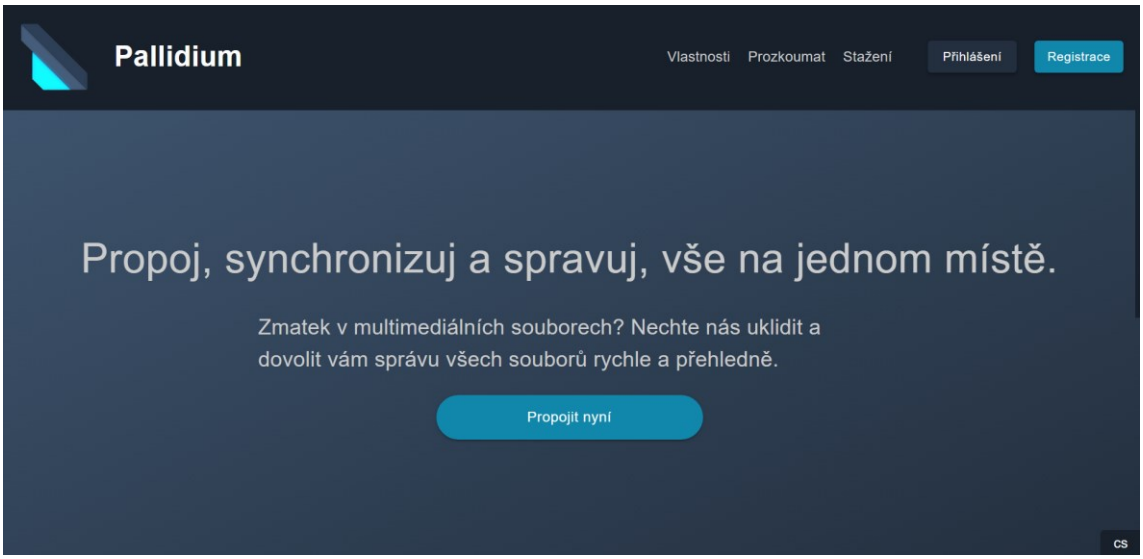
```
@Around("profilingEnabledEndpoints()")
public Object appProfiling(ProceedingJoinPoint joinPoint) throws Throwable {
    ApiLog apiLog = generateApiLog(joinPoint);
    Object returnValue;

    try {
        StopWatch stopWatch = new StopWatch();
        stopWatch.start();
        returnValue = joinPoint.proceed();
        stopWatch.stop();
        apiLog.setExecutionTime(stopWatch.getTotalTimeMillis());
    } catch (Throwable t) {
        throw t;
    } finally {
        log.debug(apiLog.toString());
    }

    return returnValue;
}
```

# 5 POUŽITÍ SYSTÉMU

Kapitola popisuje prostup uživatele celým systémem od založení účtu až po sdílení knihoven. Stránky s obsahem filmů, seriálu, sezón a epizod nejsou ukázány. Výsledný design webové aplikace je velice podobný již vytvořené desktopové aplikaci Pallidium. Podobnost je způsobena primárně dodržením konzistence mezi těmito aplikacemi.

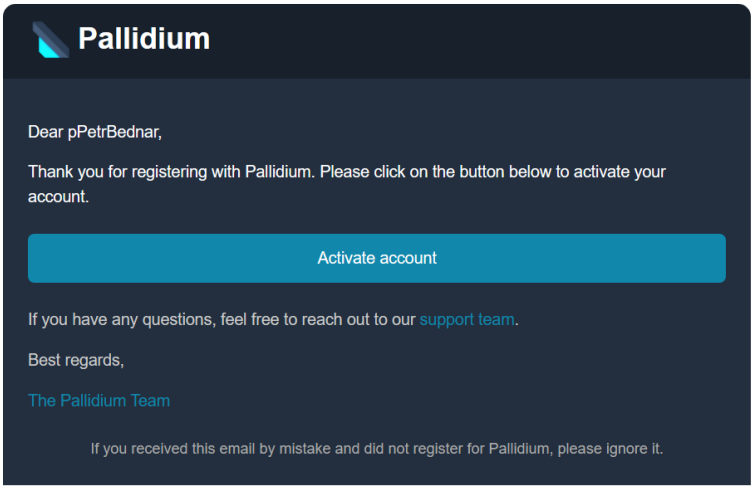
## 5.1 Webová aplikace

První úsek začíná ve webové aplikaci, kde uživatel zatím nemá přístup k žádným datům a přistupuje pouze k základním ovládacím prvkům.



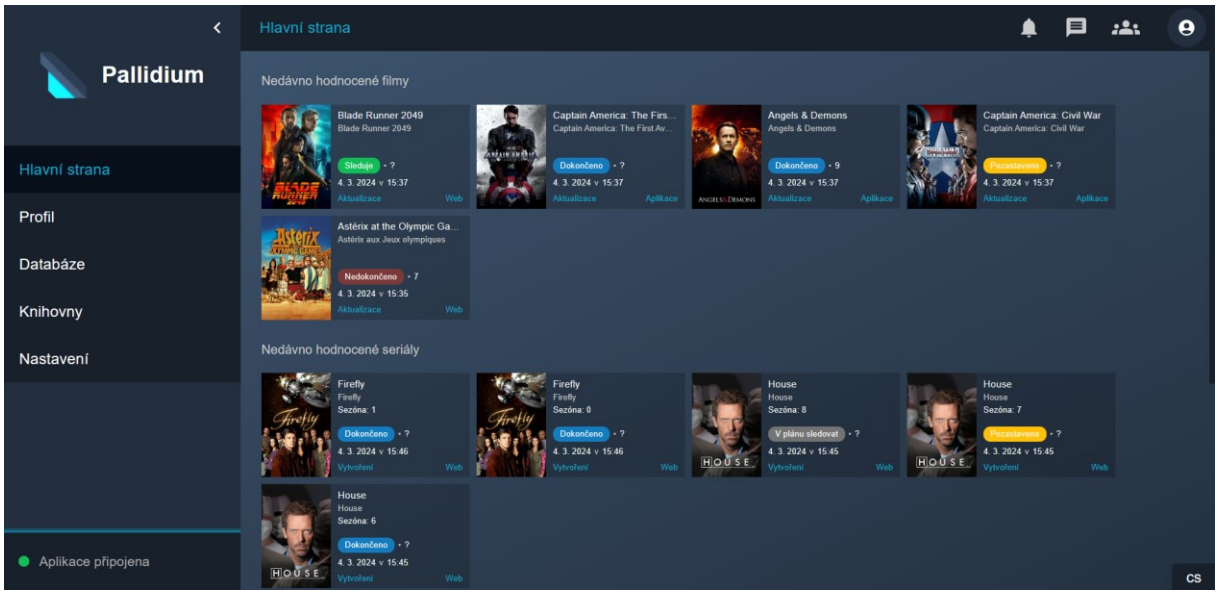

*Obrázek 32: Vstupní stránka*

Uživatel má možnost registrace pomocí svých osobních údajů, kde je vyžadováno unikátní přihlašovací jméno a emailová adresa pro ověření. Po zadání všech údajů a hesel, je odeslán ověřovací email do emailové schránky uživatele. Email slouží k ověření existence uživatele před povolením přístupu do systému a mírně bojuje proti možnému spamu.



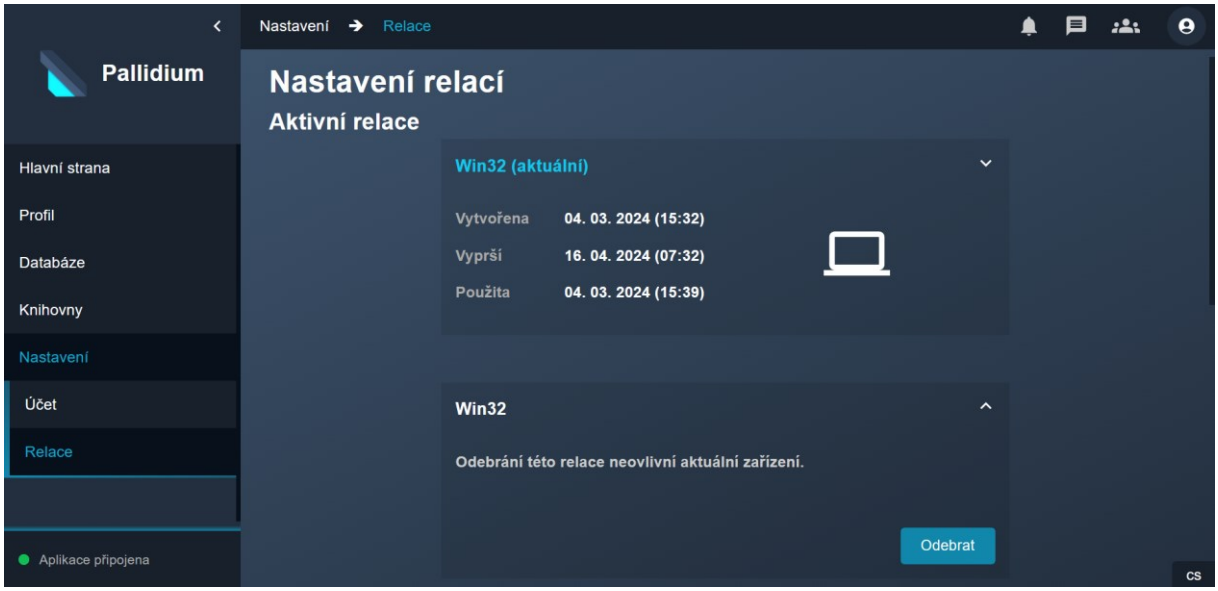

*Obrázek 33: Email pro aktivaci uživatelského účtu*

Po ověření emailové adresy je účet odemknut a je možné jeho přihlášení. V případě, že uživatel zapomněl přihlašovací heslo, je možné požádat o změnu hesla. Díky tomu je odeslán další email, který odkáže na portál systému a pomocí předvyplněného tokenu je možné zvolit nové heslo. Po úspěšném přihlášení je uživatel uvítán hlavní stránkou s přehledem posledních hodnocení. Tuto stránku uvidí při každém svém přihlášení.




*Obrázek 34: Hlavní stránka a přehled nedávných hodnocení*

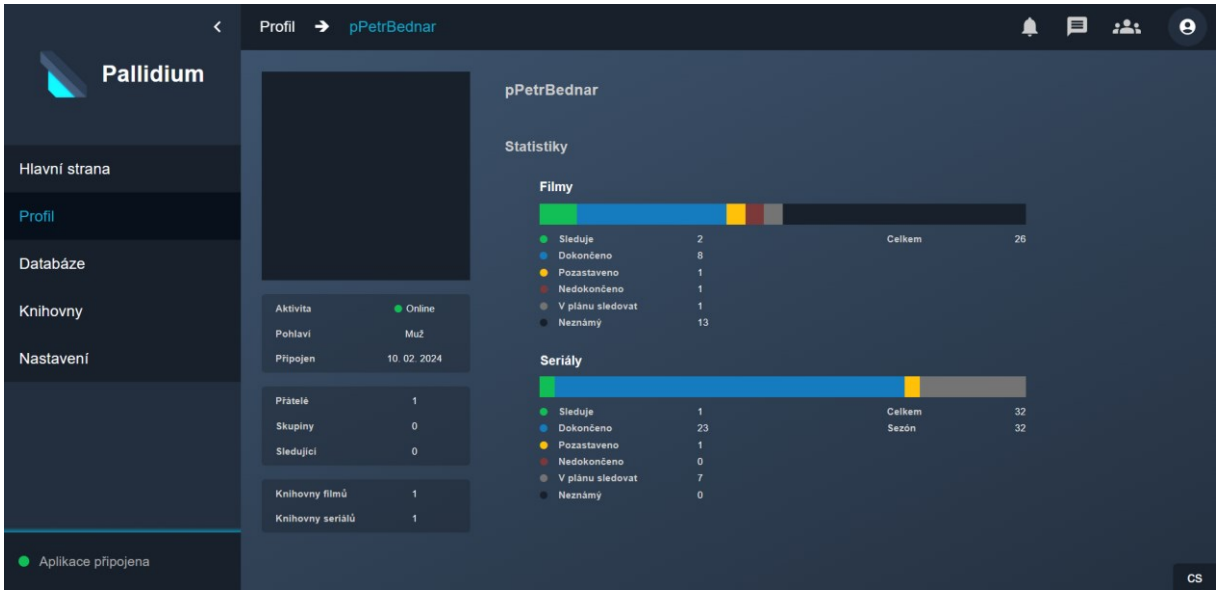
V závislosti na jeho přihlášení bylo vytvořeno nové sezení. Uživatel může spravovat své aktivní sezení na různých zařízeních a zamítnout jim přístup. Uživatel se zamítnutým sezením přichází o možnost manipulace s účtem a musí se znovu přihlásit. Sezení automaticky zaniká při odhlášení.




*Obrázek 35: Správa aktivních relací*

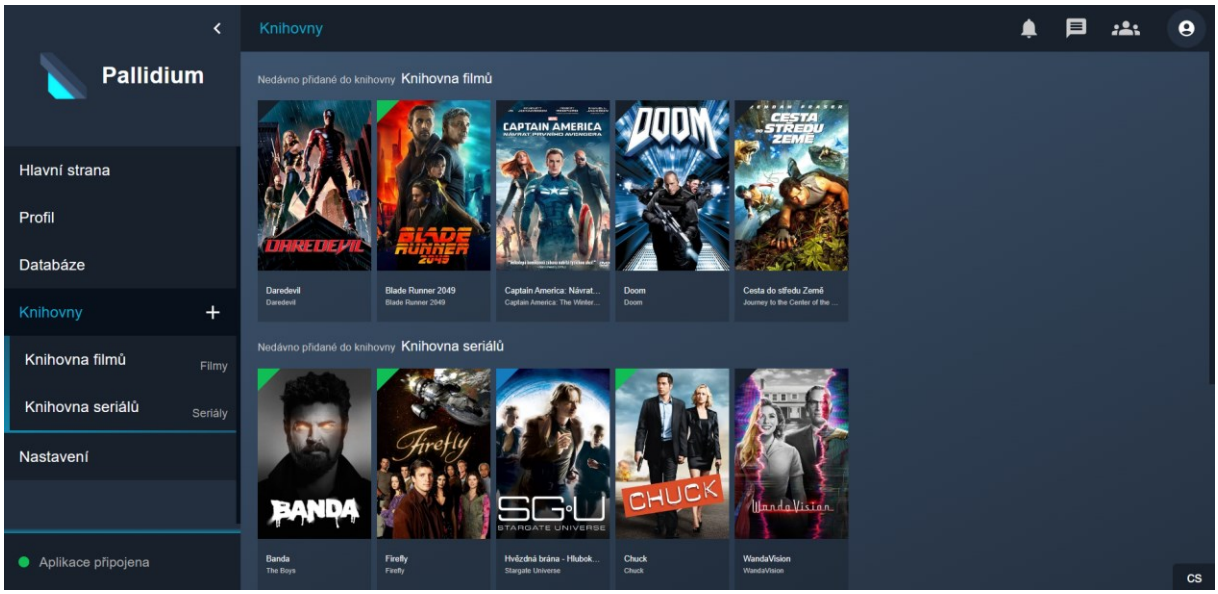
Uživatel má možnost vyhledat a požádat ostatní uživatele o přátelství. Odchozí a příchozí žádosti o přátelství jsou viditelné na stránce správy přátelství. Žádosti mohou být přijaty nebo odmítnuty. Na stejné stránce je viditelný i seznam navázaných spojení s ostatními, který umožňuje rychlý přístup k profilům těchto uživatelů.

Jednou z hlavních částí webové aplikace je správa hodnocení. Uživatel má možnost na stránce „Databáze“ vyhledávat záznamy a hodnotit jejich obsah. Vytvořená hodnocení jsou provázána s uživatelským účtem a jejich přehled uživatel nalezne na svém profilu. Uživatel má možnost označit záznam aktuálním stavem a případným číselným hodnocením. Mezi stavy zobrazení záznamu patří: neznámý, v plánu sledovat, pozastaveno, sleduje, dokončeno, nedokončeno. Stavy hodnocení nabývají hodnot od 0 do 10. Pokud záznam není ohodnocen, tak hodnota není nastavena.



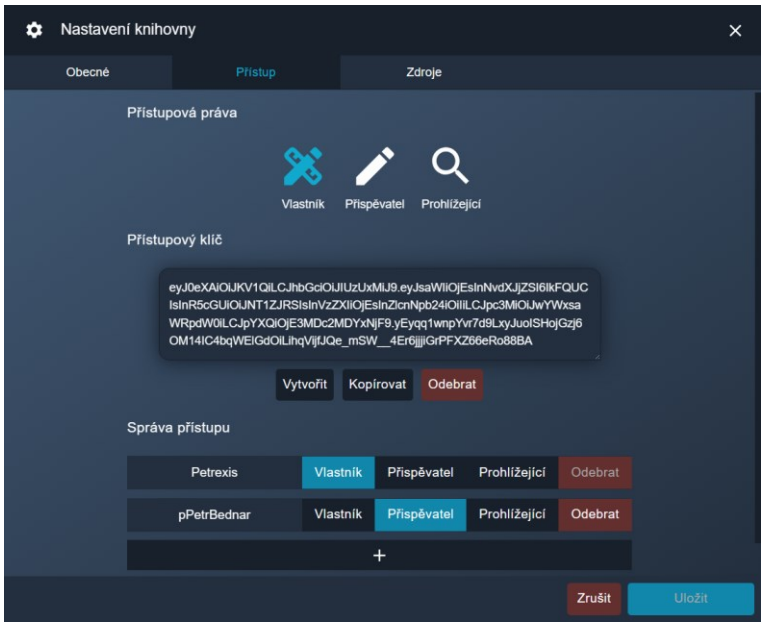

*Obrázek 36: Uživatelský profil*

Záznamy mohou být shlukovány do knihoven, které fungují jako takové skupiny záznamů. Knihovny lze využít například při sdílení doporučení, organizaci shlédnuté tvorby a pro synchronizaci dat s desktopovou aplikací. Knihovny jsou vytvářeny s určitým typem dat, které budou v knihovně vystupovat. Podporované typy knihoven jsou aktuálně filmy a seriály. Po vytvoření lze do knihovny přidávat nové záznamy a přímo v knihovně záznamy označovat nebo hodnotit. Knihovna je ve výchozím nastavení automaticky vázána na lokalizaci stránky. Tuto vlastnost lze vypnout a zvolit si manuálně lokalizaci dat v nastavení knihovny. Předcházející stránkou knihoven je přehled knihoven. V tomto přehledu jsou ukázány poslední změny v záznamech všech knihoven.




*Obrázek 37: Přehled knihoven a posledních úprav*

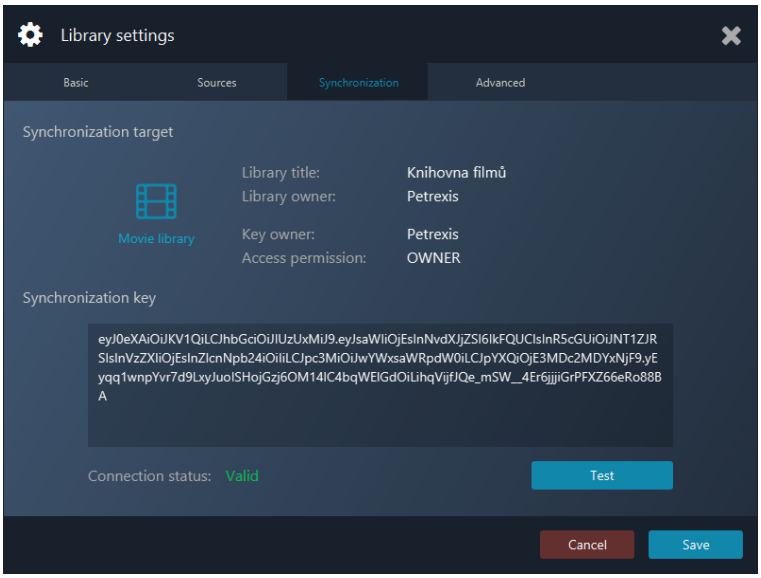
Dalším nastavení knihovny je nastavení sdílení a synchronizace. Uživatel může nasdílet přístup ke své knihovně někomu ze svých přátel. Po nasdílení uvidí cílový přítel knihovnu ve svém výpisu knihoven. Dále se na stejné kartě s nastavením nachází nastavení synchronizace. Synchronizace je nastavena pomocí vytvoření přístupového klíče, který dává držiteli práva měnit záznamy vybrané knihovny a hodnocení na účtu vlastníka klíče. Je tedy nutné, aby byl klíč držen v bezpečí.




*Obrázek 38: Nastavení sdílení a synchronizace knihovny*

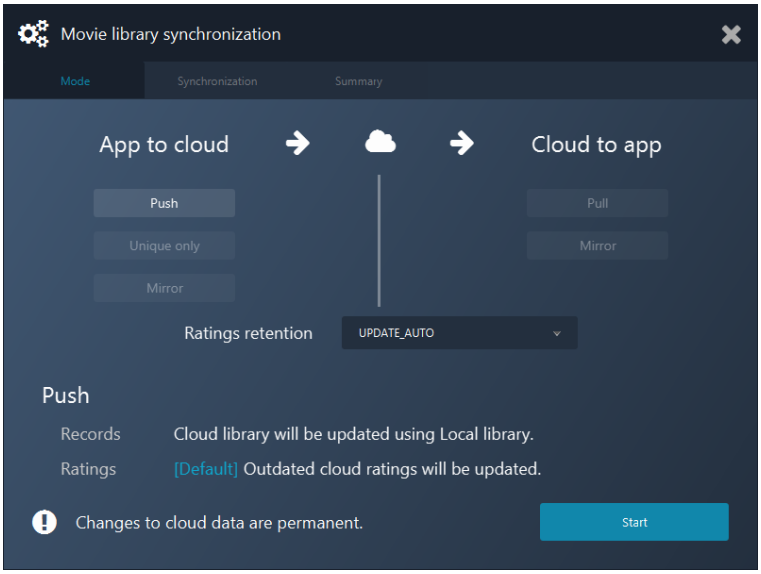
## 5.2 Desktopová aplikace

V předchozích kapitolách došlo k vytvoření knihovny a byl vytvořen přístupový klíč synchronizace. Pro propojení knihovny z desktopové aplikace na knihovnu ve webové aplikaci může uživatel zadat přístupový klíč knihovny do nastavení vybrané knihovny. Po zadání klíče je otestováno spojení s knihovnou a vytvořené spojení je uloženo.




*Obrázek 39: Nastavení synchronizace knihovny*

Díky úspěšně nastavené synchronizaci je v aplikaci odemknuta možnost synchronizace. Synchronizace je více rozebrána v kapitole 4.6. Zjednodušeně lze říci, že uživatel má na výběr synchronizaci dat do webové aplikace nebo možnost aktualizace lokálních hodnocení. Po výběru nějaké metody a případně podkategorie hodnocení, je obsah synchronizován s vybranou knihovnou a účtem uživatele. Aplikované změny jsou ihned k dispozici ve webové aplikaci a odrazí se na profilu uživatele.




*Obrázek 40: Výběr režimu synchronizace*

# 6 INTEGRACE A NASAZENÍ

Kapitola pojednává o finálních krocích vývoje softwaru. Pro začátek jsou probrány pojmy integrace, dodání a nasazení. Následující kapitoly obsahují informace o pořízení potřebné domény a virtuálního privátního serveru. Z hlediska potřeby jednoduchého a robustního nasazení jsou pro distribuci vyvíjeného systému použity Docker kontejnery. K jejich vytvoření jsou potřebné soubory Dockerfile, s jejichž pomocí lze vytvořit Docker image.

V dalších kapitolách jsou ukázány a podrobně popsány vytvořené workflow pro kontinuální integraci (CI), dodávání (CD) a nasazení (CD). Workflow jsou napsány v jazyce YAML a zpracovány pomocí systému GitHub Actions, který nabízí jednoduchý přehled nad jejich průběhem a sledování využití. Jejich spuštění je plně automatizované nebo vyžaduje zahájení od uživatele. V obou případech použitím automatizovaných workflow dochází k obrovské úspoře vývojářského času. Často opakované kroky jsou tak soběstačné a pozornost může být věnována důležitějším částem vývoje softwaru.

## 6.1 Kontinuální integrace (CI)

Kontinuální integrace je praktika automatické a časté integrace změn v projektu. Tato praktika usnadňuje a doporučuje časté přidávání změn na sdílené větvi. Při každé změně je tak automaticky provedeno otestování aktuálního sestavení. Testování je provedeno za pomoci integračních a jednotkových testů. Problémy jsou tak okamžitě odhaleny a je tak zajištěna spolehlivost přidaných změn. [74]

Při vývoji software je obvyklou praktikou přidávat nový obsah na vybranou větev v jeden vybraný den. Nové přídavky jsou sloučeny od několika různých vývojářů, čímž je najednou provedeno velké množství rozsáhlých změn. Tato skutečnost může díky jejich nechtěnému provázání ztížit hledání případných problémů. Primární výhoda průběžné integrace tak spočívá v možnosti rozdělení velkého množství změn na několik částí pro postupné automatizované testování. Možné problémy jsou tak přímo zjištěny a mohou být rovnou opraveny. V ostatních případech jsou změny vráceny nazpět pro opravu a může proběhnout integrace změn od ostatních vývojářů. [74]

## 6.2 Kontinuální dodávání (CD)

Kontinuální dodávání je praktika automatické dodávky sestavených verzí do připravených kontejnerů nebo repositářů. Tato fáze navazuje na již otestované změny v kódu provedené kontinuální integrací. Na konci tohoto procesu je tedy možné okamžitě nasadit změny do produkčního prostředí. [74]

Hlavní výhody této fáze spočívají v aktuálnosti sestavené verze, která je vždy připravena na nasazení. Díky tomu je částečně vyřešen problém špatné komunikace vývojářů a jiných týmů. Vývojářský tým tak může rychleji doručit opravy kritických chyb a celkově zrychlit přidávání nového obsahu. Takové řešení vyměňuje rychlost doručení za spolehlivost. Záleží tedy na prostředí a jestli taková vlastnost chtěná nebo se jedná o bezpečnostní riziko. [75]

Z hlediska reálného využití se jedná o vynikající prvek pro usnadnění práce vývojářského týmu. Pro optimální průběh musí být určeny přísné bezpečnostní praktiky a dostatečně kvalitní proces kontinuální integrace. Mezi praktiky můžeme například zařadit dělení verzí projektu na hlavní a jiné testovací větvě. Pokud jsou do hlavní přidávány pouze otestované funkcionality nejedná se tak o bezpečnostní riziko, které by mohlo často způsobovat problémy. [75]

## 6.3 Kontinuální nasazení (CD)

Kontinuální nasazení je praktika automatizace nasazení finální verze systému do produkčního prostředí. Výstupem této fáze je funkční nová verze, kterou mohou uživatelé systému využívat. [74]

Tato fáze usiluje o vyřešení problémů spojených s přetěžováním operačních týmů. Bez použití této praktiky by každé nasazení produkčního systému muselo být manuálně odstartováno. Díky kontinuálním nasazením lze rychle otestovat malé fragmenty změn. Malé úpravy tak nemusí způsobit rozsáhlé problémy a urychlí se jejich případné opravy. Takto rozložené nasazování může přinést jednodušší aplikaci změn než vydání jedné rozsáhlé verze, ve které se mohou problémy neúmyslně ovlivňovat. [76]

Nevýhoda celého procesu spočívá ve velké závislosti na zpracování fáze kontinuální integrace. Kvalitně vytvořená automatizace testování ovlivní a omezí možnou propagaci chyb do produkčních verzí. První fáze tak odstřihne problém přímo při jeho vytvoření před možnou integrací dále do vydaných verzí. Na druhou stranu to znamená, že kontinuální nasazení vyžaduje velkou počáteční investici do první fáze, která nepřinese žádné okamžité výsledky a případné zpětné vazby. [76]

Aby mohl být systém takto optimalizován automatizací, je nutné být konzistentní napříč jeho verzemi. Toto zahrnuje různá testovací, před-produkční, produkční a další prostředí. Případné nedodržení tohoto požadavku může způsobit problémy spojené s manuálním nastavováním na jeden systém oproti automatickým do ostatních. [76]

## 6.4 Doménové jméno

Doménové jméno neboli zkráceně „doména“ je unikátní adresa na internetu, jejíž primární funkce spočívá v zastoupení složitého číselného nebo alfanumerického kódu IP adresy. Protokoly takových adres se nazývají IPv4 a IPv6. Doménové jméno je velice důležité, jak pro usnadnění jeho zadávání, tak pro vyhledání pomocí internetových vyhledávačů. Mezi prvními výsledky jsou tak zobrazeny webové stránky, které v doménovém jménu či názvu stránky obsahují hledaný výraz. Vhodná volba tedy může vylepšit pravděpodobnost vyhledání požadované stránky a zlepšit tak její návštěvnost. Čím více lidí bude o stránce vědět, tím více lidí se o ní bude zajímat. V závislosti na zvýšené návštěvnosti tak může dojít k navýšení celkového výdělku. [77]

V rámci nasazení produkční verze vytvořeného informačního systému bylo nutné zřídit přístupné doménové jméno. Takový název musí být dostatečně unikátní pro jednoznačnou identifikaci, ale i snadno zapamatovatelný pro podpoření jeho distribuce.

Na českém trhu operují mnozí registrátoři domén. Takový registrátor je subjekt, který je oprávněný přistupovat definovaným způsobem k centrálnímu registru a zadávat do něj požadavky na změnu záznamů. Registrátor tak provádí správu domény pro koncového uživatele. [78]

Centrální registr pro doménové jméno nejvyšší úrovně „.cz“ je spravován organizací CZ.NIC. Jedná se o zájmové sdružení právnických osob založené v roce 1998 předními poskytovateli internetových služeb. Jejich hlavní náplň práce je provozování doménového registru a zabezpečení domény nejvyšší úrovně. CZ.NIC je členem mezinárodních organizací, které sdružují podobné organizace ve světě (CENTR, ccNSO, EURid a další). [79]

### 6.4.1 Registrace doménového jména

Doménové jméno pro vyvíjený systém se nazývá pallidium.cz. Doména byla zakoupena 7. 2. 2024 v administrační sekci portálu Hukot.net. Hukot dovoluje zakoupení domén pod českou doménou nejvyššího stupně „.cz“ za 171 Kč bez DPH dle ceníku z doby zakoupení. Po dokončení objednávky byla doména promptně dodána na registrovaný účet. Následné změny se po chvíli připsaly i do informací o doméně u správce CZ.NIC. Hukot využívá registrátora domén Subreg a zastupuje klienta jako administrativní kontakt. [80], [81]

### 6.4.2 Problém veřejných dat

V rámci registrace domény je registrátor povinen zveřejnit jméno, příjmení a adresu fyzické osoby klienta. Případný nesouhlas se zveřejněním těchto údajů lze změnit v nastavení u správce nevyšší domény. Pro možnost takové změny byla nutnost vytvoření osobního profilu mojeID, pomocí kterého se lze přihlásit do doménového prohlížeče organizace CZ.NIC. Tento účet je také spravován organizací CZ.NIC a k jeho ověření je zapotřebí Czech POINT (Český podací ověřovací informační národní terminál) nebo jiné podporované možnosti ze státních ověřovacích systémů. Změna kontaktu u domény je možná pouze ze strany registrátora. Díky tomu byla kontaktována podpora portálu Hukot, která rychle odpověděla a vystavila žádost ke změně kontaktu domény. Proces byl však přerušen neobdržením emailu na budoucího majitele z důvodu neveřejnosti emailové adresy v profilu mojeID. Pro opětovné odeslání pak již nešlo vyzvat Hukot, ale musel být kontaktován registrátor Subreg, který se postaral o další kroky převodu. Po dokončení procesu tak byla doména přidána pod správu účtu mojeID a zveřejněné údaje tak byly skryty. [82], [83]

### 6.4.3 Nastavení domény

Hlavním účelem vlastnictví doménového jména je možná konfigurace jejich DNS záznamů. DNS se používá k překladům názvu domény na její přidělenou hodnotu IPv4 nebo IPv6 adresy. Záznamy se dělí na různé typy. A záznam převádí textovou adresu na přidanou hodnotu IPv4 a AAAA na hodnotu IPv6. Další důležitý záznam je CNAME, který specifikuje, v jaké doméně se nachází DNS záznamy pro danou subdoménu. Následují MX záznamy, kde se zadávají adresy emailových serverů spojené s vybranou doménou. NS záznamy specifikují adresy jmenných serverů, které jsou pro danou doménu autoritativní, respektive na kterých jsou uloženy její DNS záznamy. Stejně jako u MX záznamů se nastavují primární a sekundární záznamy. Posledním typem jsou TXT záznamy, které nemají přímý vliv na doménu a jsou k užitku validačním technologiím jako jsou DKIM, SPF a DMARC. [84]

*Tabulka 6: Nastavení DNS domény pallidium.cz*

| Název | Typ | Hodnota | Priorita | TTL |
|---|---|---|---|---|
| | A | 46.36.38.223 | | 3600 |
| api | A | 46.36.38.223 | | 3600 |
| www | A | 46.36.38.223 | | 3600 |
| * | CNAME | pallidium.cz | | 86400 |
| | MX | mail.hukot.net | 10 | 86400 |
| | MX | mx2.securitynet.cz | 100 | 86400 |
| | TXT | v=spf1 a mx include:spf.hukot.net ~all | | 86400 |
| | NS | ns1.hukot.cz | | 86400 |
| | NS | ns2.securitynet.cz | | 86400 |
| | NS | ns3.hukot.cz | | 86400 |

U domény je také zapnut a plně nastaven DNSSEC, který zabezpečuje převod adres z DNS serverů. Jelikož je nutné se vždy dotázat DNS serveru na IP adresu, lze takový proces podvrhnout a přesměrovat tak oběť na podvrženou webovou stránku. Na takové stránce se mohou vyskytovat různé typy podvodů, phishing útoků a jiných praktik pro získání osobních údajů. DNSSEC chrání před podvrženými záznamy v DNS serverech pomocí digitálních podpisů. Přístup k DNS záznamům pro danou doménu je tak povolen pouze při znalosti ochranného klíče. Neoprávněná manipulace s DNS záznamem je tedy zamítnuta. [85]

## 6.5 Virtuální privátní server (VPS)

VPS je vyhrazený server, který nepoužívá obvyklý sdílený výkon webhostingů, ale dedikovaný výkon. Sdíleny jsou tak pouze vstupně/výstupní operace a procesory. S VPS je tedy zakoupen zaručený výkon a absolutní kontrola nad systémem. Vyšší znalost všech komponent je tedy nutností pro správné nastavení takového serveru. Slovo „virtuální“ v názvu je hlavní odlišností oproti čistě dedikovaným serverům. V případě dedikovaného je celý server vyhrazen jednomu kupujícímu namísto sdílení hardwaru mezi několika klienty. [86]

Pro účely nasazení produkční verze byla nutnost pořídit takový server. Po důkladném průzkumu možností na českém a globálním trhu byly nalezeny nejvýhodnější ceny na portálu Hukot.net. Nabízený ceník obsahuje různé stupně konfigurací VPS pro nenáročné až velmi vytěžující klienty. Hukot také nabízí rozšíření konektivity z výchozí 100 Mbps na 1 Gbps, možnost fakturačního období 1 rok, režim dedikovaných procesorů a správu serveru. Cena za nejnižší stupeň konfigurace (VPS-L02G) stála v době pořízení (7. 2. 2024) 100 Kč bez DPH za měsíc.

*Tabulka 7: Rozpis podporovaných tarifů VPS poskytovatele Hukot.net [87]*

| **Název** | VPS-L02G | VPS-L04G | VPS-L08G | VPS-L16G | VPS-L32G | VPS-L64G |
|---|---|---|---|---|---|---|
| **SSD** | 20 GB | 40 GB | 80 GB | 120 GB | 200 GB | 400 GB |
| **RAM** | 2 GB | 4 GB | 8 GB | 16 GB | 32 GB | 64 GB |
| **CPU** | 1 | 2 | 4 | 6 | 8 | 10 |

Každá z konfigurací obsahuje neomezení přenosový limit, plný root přístup a garantuje dostupnost SLA 99,8 % pro nejnižší stupeň s rozšířením na SLA 99,9 % v případě ostatních konfigurací. K VPS je také dodávána veřejná IPv4 a IPv6 adresa. [87]

### 6.5.1 Prvotní konfigurace

Po dokončení objednávky byl server promptně dodán a přístupný. Od základu obsahuje pravidla pro nastavení firewallu, která blokují veškerý přístup z vnější. Interní konzole je přístupná přímo z administrace Hukot tímto firewallem není nijak ovlivněna. Prvotní instalace funguje jako každá jiná instalace serverového Linuxu. V tomto případě byla zvolena instalace Ubuntu Server 22.04 LTS (nejnovější verze).

Instalace je podmíněna vložením instalačního ISO disku, který Hukot podporuje pro různé operační systémy. Instalovaný systém lze přeinstalovat jiným z vybrané selekce. V rámci instalace dojde k nastavení základních věcí jako je například uživatelský účet s právy root. Root účet je ve výchozím nastavení deaktivován.

### 6.5.2 Konfigurace SSH

Pro zvýšení ochrany přihlašování byly vytvořeny privátní a veřejné klíče pro přístup přes SSH. K vytvoření klíčů byl použit nástroj PuTTY Key Generator. Klíče byly vytvořeny pomocí šifrovacího algoritmu RSA na bitové šířce 4096 pro dostatečné zabezpečení. Privátní klíč byl uložen na klientském systému ve složce .ssh v domovském adresáři. Veřejný klíč byl pomocí programu WinSCP odeslán do úložiště serveru pro nadcházející nastavení.

SSH je na serveru již přítomné díky instalaci systému a není nutné ho dodatečně instalovat. V adresáři .ssh domovského prostoru je vytvořen soubor pojmenovaný „authorized_keys“, do kterého je vložen text veřejného klíče s případným komentářem. Každý takový záznam na novém řádku specifikuje jedno přístupové oprávnění k uživatelskému účtu. [88]

Pro nastavení SSH byla použita následující konfigurace. Jedná se o kompletaci doporučení z různých webových stránek. Hlavním bodem je vypnutí autentizace za pomoci hesla, která by snížila kvalitu zabezpečení při používání SSH pro připojování. Toho je dosaženo pomocí konfiguračních položek „PasswordAuthentication no“ a „ChallengeResponseAuthentication no“. [89]

*Zdrojový kód 26: Konfigurace SSH na VPS*

```
Include /etc/ssh/sshd_config.d/*.conf
Port 22
ListenAddress 0.0.0.0
Protocol 2
SyslogFacility AUTH
LogLevel INFO
LoginGraceTime 120
PermitRootLogin no
StrictModes yes
RSAAuthentication yes
PubkeyAuthentication yes
PasswordAuthentication no
PermitEmptyPasswords no
ChallengeResponseAuthentication no
KbdInteractiveAuthentication no
UsePAM yes
X11Forwarding yes
X11DisplayOffset 10
PrintMotd no
PrintLastLog yes
TCPKeepAlive yes
AcceptEnv LANG LC_*
Subsystem sftp /usr/lib/openssh/sftp-server
```

Při vypínání ověření hesel je nutné dávat pozor. V případě ztráty privátního klíče by byl přístup na server znemožněn. Dalším dodatkem je kontrola přítomnosti dalších konfiguračních souborů, které by mohli upravovat nastavení. V případě užitého serveru se jednalo o konfigurační soubor „50-cloud-init.conf“, který obsahoval povolení použití hesel.

### 6.5.3 Nastavení firewallu

Hukot dodává ke každému serveru přehledné nastavení firewallu, který v základním nastavení blokuje z důvodu bezpečnosti veškerý přístup k serveru. Typ firewallu byl tedy změněn ze základního na rozšířený a byla nastavena potřebná pravidla. V nastavení blokovaného provozu jsou blokovány všechny protokoly na všech přístupových portech. Těmto pravidlům předchází nastavení povoleného provozu. Zde je povolen provoz pro protokoly HTTP, HTTPS, SSH a OpenVPN spojení.

*Tabulka 8: Nastavení povoleného provozu pro VPS firewall*

| Směr | Protokol | Zdrojový port | Cílový port | Zdrojová IP | Cílová IP |
|---|---|---|---|---|---|
| IN | TCP | 0 | 22 | | 46.36.38.223 |
| IN | TCP | 0 | 80 | | 46.36.38.223 |
| IN | TCP | 0 | 443 | | 46.36.38.223 |
| IN | UDP | 0 | 1194 | | 46.36.38.223 |

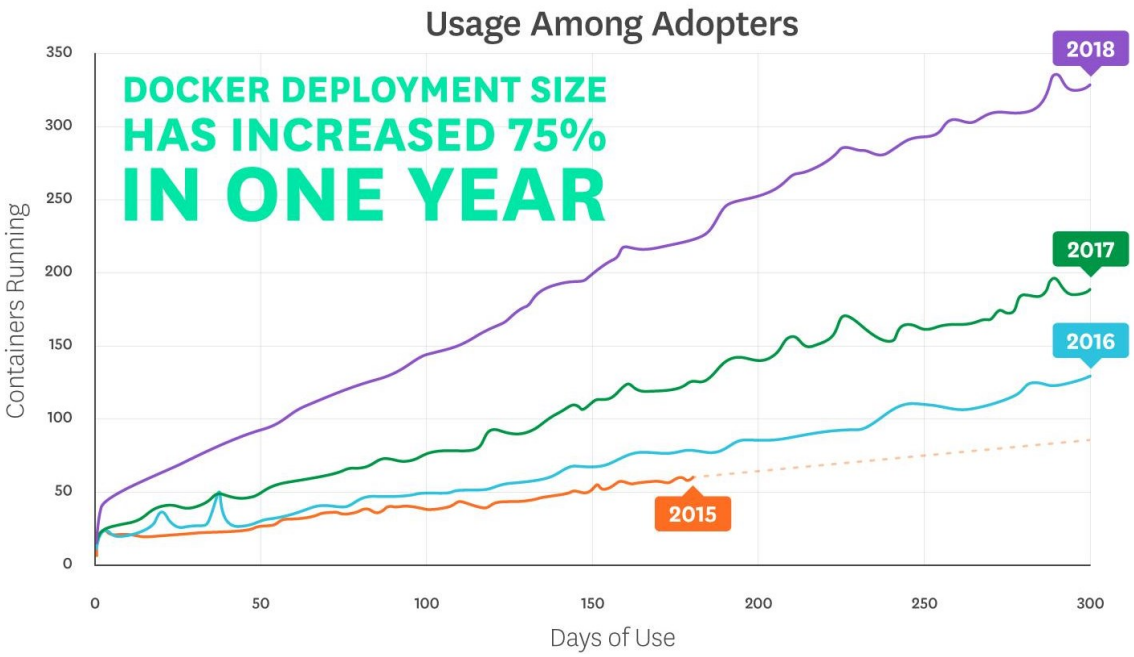
*Tabulka 9: Nastavení zakázaného provozu pro VPS firewall*

| Směr | Protokol | Zdrojový port | Cílový port | Zdrojová IP | Cílová IP |
|---|---|---|---|---|---|
| IN | ICMP | 0 | 0 | | 46.36.38.223 |
| IN | TCP | 0 | 0 | | 46.36.38.223 |
| IN | UDP | 0 | 0 | | 46.36.38.223 |

## 6.6 Docker kontejnery

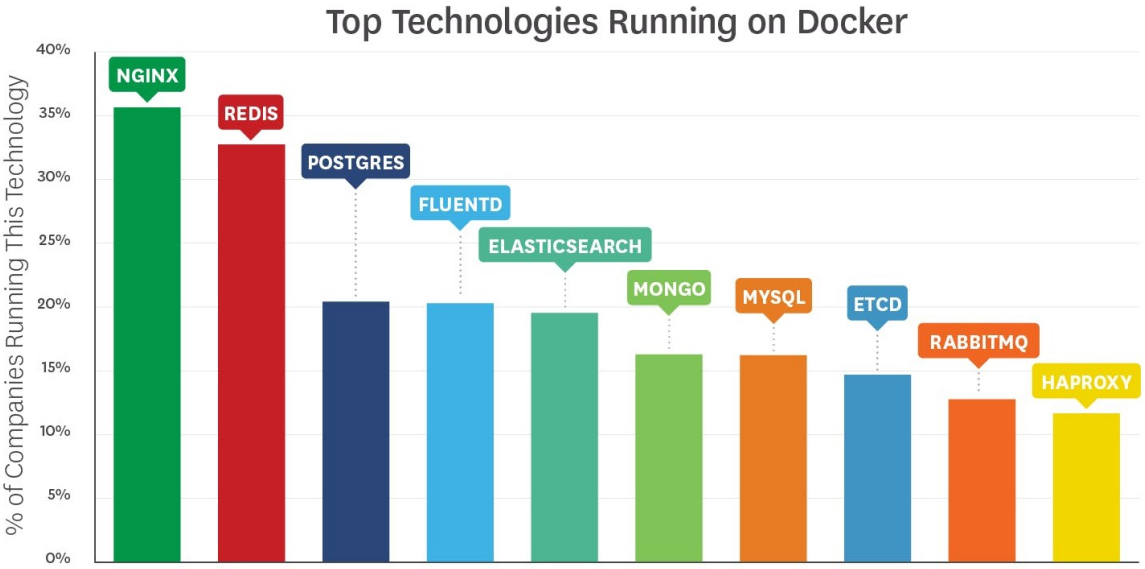
Kontejnery odstraňují nedostatky klasické plné virtualizace. U plné virtualizace dochází k instalaci softwarové komponenty hypervisor. Jedná se o software pro virtualizaci hardwaru do izolovaných virtuálních strojů. Na jedné fyzické jednotce tak lze spustit více operačních systémů a izolovaně provozovat různé aplikace, služby a jiná vývojová prostředí. Díky tomu jsou oblíbené a vysoce používané po celém světě. Společnost Datadog, která se zabývá poskytováním monitorovacích služeb cloudových datacenter, v roce 2018 zveřejnila statistiky užití softwaru Docker. Dle dostupných výsledků jde patrný nárust počtu používaných kontejnerů napříč celou jejich platformou. Lze tedy počítat se stoupajícím tendencí využití i v následujících letech. [90], [91], [92]

V roce 2018 přes 20 % Datadog klientů provozovalo na svých hostitelských systémech Docker. Graf podílu klientských systémů na obrázku 41 ukazuje prakticky lineární nárust v užití. Můžeme tedy usuzovat o vysoké robustnosti zmiňovaného systému a jeho vhodnost pro účely této práce. Inspiraci k využití různých systémů lze čerpat ze statistik provozovaných technologií v Docker kontejnerech. [92]




*Obrázek 41: Graf využití Docker kontejnerů [92]*

Na obrázku 42 můžeme vidět vysoké využití softwarových technologií jako jsou webové servery, databáze a systémy kolekce dat. Nejpoužívanější technologií je Nginx, který nabízí pokročilé řízení zátěže na webových serverech. Díky jeho přístupnosti a účinnosti není překvapivé, že je umístěný tak vysoko. Pro jeho výkon a přívětivost instalace je použit v rámci nasazení FE částí této práce. Další využitou technologií v navrhovaném systému je databázový systém PostgreSQL. Dle výsledků průzkumu se jedná o jednu z předních voleb při výběru databáze. [92]




*Obrázek 42: Graf nejpoužívanějšího software v Docker kontejnerech [92]*

### 6.6.1 Příprava na kontejnerizaci

Prvním krokem k nasazení systému pomocí Docker je tvorba souborů Dockerfile. Docker může automaticky sestavovat obrazy tím, že čte instrukce z Dockerfile. Jedná se o textový dokument, který obsahuje všechny příkazy spouštěné v příkazovém řádku k sestavení obrazu.

Nejpoužívanější instrukce použité v Dockerfile jsou:

- **FROM** – vytvářející novou sekci sestavení ze zadaného obrazu,
- **ARG** – pro využití proměnných sestavení,
- **ENV** – k nastavení systémových proměnných,
- **COPY** – spravující kopírování souborů a složek na určené místo,
- **EXPOSE** – určující port, který bude odhalen ve výsledném kontejneru,
- **RUN** – ke spouštění potřebných příkazů,
- **ENTRYPOINT** a **CMD** – specifikující chování kontejneru po jeho spuštění. [93]

### 6.6.2 Dockerfile pro backend

První z Dockerfile souborů byl vytvořen pro backend. Zkompilovaný a sestavený kód je zabalen do .jar souboru, který je spustitelný na systémech s Java Development Kit (JDK). V tomto případě se jedná o verzi z distribuce OpenJDK ve verzi 21. Konstrukce tak začíná jeho instalací do prostředí pomocí klíčového slova FROM. [94]

*Zdrojový kód 27: Dockerfile pro backend, část 1.*

```
FROM openjdk:21-jdk
```

Následně jsou do prostředí nahrány systémové proměnné, které napomáhají patřičné propagaci nastavení napříč nasazeným systémem. Hodnoty jsou získány z proměnných sestavení za pomoci instrukce ARG a vsazeny pomocí instrukce ENV do prostředí. Hodnoty lze nastavit i napřímo pokud není vyžadováno jejich zabezpečení.

*Zdrojový kód 28: Dockerfile pro backend, část 2.*

```
ARG DB_PASSWORD
ENV DB_PASSWORD ${DB_PASSWORD}
ENV LANG=C.UTF-8
```

Pro zajištění spuštění bylo nutné přesunout .jar soubor do kořene kontejneru. Přesun je zprostředkován instrukcí COPY.

*Zdrojový kód 29: Dockerfile pro backend, část 3.*

```
COPY build/libs/*.jar /pallidium.jar
```

BE server je spuštěn dle konfigurace na síťovém portu 8000. Pro povolení komunikace je potřebné informovat kontejner o jeho povolení. Nastavení je provedeno instrukcí EXPOSE.

*Zdrojový kód 30: Dockerfile pro backend, část 4.*

```
EXPOSE 8000
```

Při každém spuštění kontejneru je nutné automaticky nastartovat BE server. Funkcionalitu nastavíme instrukcí ENTRYPOINT, která je při každém startu zavolána.

*Zdrojový kód 31: Dockerfile pro backend, část 5.*

```
ENTRYPOINT ["java","-jar", "/pallidium.jar"]
```

Celý Dockerfile použitý pro Spring server je k nalezení v příloze 3.

### 6.6.3 Dockerfile pro frontend

Dockerfile FE části není příliš odlišný od toho v minulé kapitole. Díky běhu na jiném systému je zaměněn základní obraz za Nginx.

*Zdrojový kód 32: Dockerfile pro frontend, část 1.*

```
FROM nginx:latest
```

Z hlediska zabezpečení pomocí HTTPS, které je více rozebráno v kapitole 6.8.5, jsou doinstalovány potřebné balíčky pro funkčnost certifikačního nástroje Certbot. Nástroj dokáže posléze vystavit potřebné SSL certifikáty Let's Encrypt pro zabezpečení komunikace s uživateli.

*Zdrojový kód 33: Dockerfile pro frontend, část 2.*

```
RUN apt-get update && \
    apt-get install -y certbot python3-certbot-nginx
```

Pomocí příkazů dole jsou překopírována potřebná nastavení a soubory pro spuštění webové stránky. První z konfiguračních souborů poskytuje obecné nastavení celého serveru, kde následující upřesňuje jednotlivé obslužné servery provozované Nginx. Pro správnou činnost jsou dodány známé typy internetových médií, které byly získány z GitHub repositáře uživatele djangofan nazývaného docker-tomcat7-cluster. [66]

*Zdrojový kód 34: Dockerfile pro frontend, část 3.*

```
COPY nginx.conf /etc/nginx/
COPY nginx-server/nginx.conf /etc/nginx/conf.d/default.conf
COPY mime.types /etc/nginx/
COPY dist /usr/share/nginx/html
```

Předposlední sekcí jsou povoleny síťové porty pro komunikaci na protokolech HTTP (80) a HTTPS (443). Na závěr je nastaven příkaz pro spuštění Nginx při startu kontejneru.

*Zdrojový kód 35: Dockerfile pro frontend, část 4.*

```
EXPOSE 80
EXPOSE 443
ENTRYPOINT ["nginx", "-g", "daemon off;"]
```

Celý Dockerfile použitý pro Nginx server je k nalezení v příloze 4.

## 6.7 Workflow kontinuální integrace a dodávky

Workflow pro GitHub Actions jsou psány v jazyce YAML. Každá workflow začíná podobně, a to nastavením základních parametrů. Přes klíč „name“ je nastaven název workflow. Dále pokračuje specifikace spouštěče. Workflow lze zapínat manuálně pomocí klíčového slova „workflow_dispatch“ nebo automaticky při odeslání nových změn do vybrané GitHub větve.

*Zdrojový kód 36: Specifikace workflow, část 1.*

```
name: CI
on:
  workflow_dispatch:
  push:
    branches:
      - 'main'
```

Následným krokem je specifikace jednotlivých úloh (jobs). Každá úloha má svůj obraz prostředí, název, oprávnění přístupu a jednotlivé kroky prováděné workflow.

*Zdrojový kód 37: Specifikace workflow, část 2.*

```
jobs:
  release:
    name: CI Pipeline
    runs-on: ubuntu-latest
    permissions:
      contents: read
    steps:
      - name: This is individual step of workflow
        run: echo "Hello world!"
```

V obou následujících případech využití workflow je výstupem image Docker kontejneru dodaný do privátního repositáře systému Docker Hub. Osobní účet je bez zakoupení předplatného limitován na vlastnictví pouze jednoho privátního repositáře. Veřejných repositářů se toto omezení nijak netýká. Kvůli limitaci je využito systému značek (tags), kde do vytvořeného repositáře „pallidium“ jsou dodávány, jak image kontejneru pro backend, tak pro frontend. Verze jsou pojmenovány dle příslušného užití na „api.latest“ nebo „web.latest“.

### 6.7.1 Backend

Kroky potřebné k úspěšnému sestavení a vytvoření Docker kontejneru jsou podmíněny specifikací potřebných systémových proměnných. Backend používá ve své funkčnosti knihovnu publikovanou za pomoci platformy GitHub Packages. Jelikož jsou takové knihovny přístupné pouze pomocí privátního přístupového klíče, je nutné tento klíč do prostředí nahrát před pokusem o sestavení.

*Zdrojový kód 38: Workflow CI pro BE, část 1.*

```
env:
  GH_PACKAGES_USERNAME: ${{ secrets.GH_PACKAGES_USERNAME }}
  GH_PACKAGES_ACCESS_TOKEN_READONLY: ${{ secrets.GH_PACKAGES_TOKEN }}
  SYS_ENVIRONMENT: 'prod'
```

První krok workflow spočívá v natažení dat repositáře do virtuálního prostředí. K tomuto a podobným akcím lze využít předem vytvořené funkce různých přispěvatelů. Po získání požadovaného kódu z repositáře je v prostředí připravena Java 19 se zapnutou cache Gradle knihoven. Následný krok sestaví specifikovanou verzi a sestaví spustitelný .jar soubor.

*Zdrojový kód 39: Workflow CI pro BE, část 2.*

```
- name: Checkout code
  uses: actions/checkout@v4

- name: Set up JDK 19 for x64
  uses: actions/setup-java@v2
  with:
        java-version: '19'
        distribution: 'adopt'
        cache: gradle

- name: Build with Gradle
  run: ./gradlew build --no-daemon
```

Posledním krokem je tvorba a publikace Docker kontejneru na platformu Docker Hub. Postup začíná nastavením požadavků k sestavení takového kontejnerového obrazu a přihlášení do Docker Hub. Ve finálním kroku je využit předpřipravený action od společnosti Docker. Funkce sestaví kontejnerový image dle konfigurace v Dockerfile z kapitoly 6.6.2. Výstup je následně odeslán do vybraného repositáře v Docker Hub. Funkci sestavení jsou poskytnuta potřebná data, která budou pomocí specifikace Dockerfile přenastaveny v systémové proměnné. Hodnoty jsou načteny ze zapsaných tajných hodnot v nastavení repositáře. Přístup k nim je podmíněn prefixem „secrets“. [95]

*Zdrojový kód 40: Workflow CI pro BE, část 3.*

```
- name: Build with Gradle
  run: ./gradlew build --no-daemon

- name: Set up Docker Buildx
  uses: docker/setup-buildx-action@v3

- name: Log in to Docker Hub
  uses: docker/login-action@v3
  with:
    username: ${{ secrets.DOCKER_USERNAME }}
    password: ${{ secrets.DOCKER_ACCESS_TOKEN }}

- name: Build and push Docker image
  uses: docker/build-push-action@v5
  with:
    context: .
    push: true
    tags: ppetrbednar/pallidium:api.latest
    build-args: |
      "SYS_ENVIRONMENT=prod"
      "SYS_VERSION=${{ github.event.inputs.version }}"
      "DB_USERNAME=${{ secrets.DB_USERNAME }}"
      "DB_PASSWORD=${{ secrets.DB_PASSWORD }}"
      "AUTH_JWT_SECRET=${{ secrets.AUTH_JWT_SECRET }}"
      "API_JWT_SECRET=${{ secrets.API_JWT_SECRET }}"
      "TMDB_API_TOKEN=${{ secrets.TMDB_API_TOKEN }}"
      "MAIL_USERNAME=${{ secrets.MAIL_USERNAME }}"
      "MAIL_PASSWORD=${{ secrets.MAIL_PASSWORD }}"
```

Celá workflow použitá pro tvorbu image Docker kontejneru Spring serveru je k nalezení v příloze 5.

### 6.7.2 Frontend

Workflow pro integraci a dodání FE je podobné již probranému pro backend. Hlavní odlišnosti se nalézají v použití jiných technologií a postupu sestavení. Workflow začíná obdobně načtením dat z repositáře. V následném kroku je stažen software Node 20 pro možnost sestavení. Vytvoření image pro Docker kontejner se shoduje s již zmíněným, ale neobsahuje žádné dodatečné parametry pro sestavení.

*Zdrojový kód 41: Workflow CI pro FE*

```
- name: Checkout code
  uses: actions/checkout@v4

- name: Set up Node 20
  uses: actions/setup-node@v4
  with:
    node-version: 20
    cache: 'npm'

- name: Install dependencies
  run: npm ci

- name: Build project
  run: npm run build

- name: Set up Docker Buildx
  uses: docker/setup-buildx-action@v3

- name: Log in to Docker Hub
  uses: docker/login-action@v3
  with:
    username: ${{ secrets.DOCKER_USERNAME }}
    password: ${{ secrets.DOCKER_ACCESS_TOKEN }}

- name: Build and push Docker image
  uses: docker/build-push-action@v5
  with:
    context: .
    push: true
    tags: ppetrbednar/pallidium:web.latest
```

Celá workflow použitá pro tvorbu image Docker kontejneru Nginx serveru je k nalezení v příloze 6.

## 6.8 Workflow nasazení na VPS

Nasazení produkční verze není plně automatizováno a nedodržuje tak princip kontinuálního nasazování (CD). Prerekvizitou automatizovaného nasazení je vytvoření Docker Compose konfiguračních souborů, které usnadní nastavování kontejnerů.

Compose je v první řadě složen z verze zápisu, která poskytuje možnosti zpětné kompatibility. Další položkou je název elementu nejvyšší úrovně neboli adresáře, pod kterým budou kontejnery indexovány. Ten je pojmenován podle názvu projektu (pallidium). V případě použití tohoto názvu jako prefixu každého z kontejnerů, není již název v grafickém rozhraní aplikace Docker Desktop zobrazováno. Jde tak o dodatečné zpřehlednění oproti konzolovému výstupu. [96]

Dalším obsahem Compose je zápis potřebných kontejnerů. Každý záznam má vlastní image Docker kontejneru a dodatečné nastavení pro jeho běh. Nejdůležitějšími parametry jsou: název kontejneru, image (repositář odkud má být stažený image kontejneru), metoda restartu při pádu či doběhnutí, mapování vnitřních síťových portů na viditelné vnější porty, nastavení systémových proměnných, hesel a kontejnerové sítě. [97]

### 6.8.1 Docker Compose pro databázi

Nedílnou součástí celého systému je databáze. Její tvorbou současně zavedeme globální nastavení sítě, která dovolí použití lokální komunikace mezi kontejnery. Takto vytvořená síť je využita při komunikaci Spring serveru s databází a směrováním dotazů z Nginx serveru na API Spring serveru.

*Zdrojový kód 42: Nastavení sítě v docker compose*

```
networks:
  network:
    driver: bridge
    ipam:
      config:
        - subnet: 172.10.0.0/16
          gateway: 172.10.0.1
```

Síť je vytvořena mimo specifikaci požadovaných kontejnerů. Součástí nastavení je driver, který určuje chování sítě. Ve výchozím nastavení se jedná o „bridge“. Síť je definována pomocí specifikace IPAM, která dovoluje vytvoření statické lokální sítě. V konfiguraci jsou nastaveny parametry s IP adresou podsítě, jejím rozsahem a definicí výchozí brány. [98], [99]

*Zdrojový kód 43: Docker compose pro databázi*

```
pallidium-db-service:
  container_name: pallidium-db-container
  image: postgres
  restart: unless-stopped
  ports:
    - '4040:5432'
  environment:
    POSTGRES_DB: pallidium
    POSTGRES_USER_FILE: /run/secrets/postgres_user
    POSTGRES_PASSWORD_FILE: /run/secrets/postgres_password
  secrets:
    - postgres_user
    - postgres_password
  networks:
    network:
      ipv4_address: 172.10.0.2
```

V rámci zvýšení zabezpečení a ulehčení změny tajných údajů lze využít načítání těchto soukromých dat ze souborů. Každý údaj je specifikován jedním textovým souborem. Mezi další metody dosazení hodnot spadají autentizační tokeny a certifikáty. [100]

*Zdrojový kód 44: Import hesel v docker compose*

```
secrets:
  postgres_user:
    file: ./secrets/postgres_user.txt
  postgres_password:
    file: ./secrets/postgres_password.txt
```

Celá konfigurace je k dispozici v příloze 7.

### 6.8.2 Docker Compose pro backend

Compose pro kontejner Spring serveru se odlišuje čerpáním Docker image z privátního repositáře. Pro získání dat je tak potřebné přihlášení do uživatelského účtu svázaného s repositářem. Požadovaný Docker image obsahuje nastavení spojené s vlastností „pull_policy“. Hodnota je nastavena na „always“ z důvodu vynucení stažení nového image z repositáře i v případě shody verzí. Je tomu zapotřebí z důvodu nepoužití čísel verzí, ale statického tagu s názvem nasazení a přípony „latest“. [97]

*Zdrojový kód 45: Docker compose pro backend*

```
pallidium-api-service:
  container_name: pallidium-api-container
  image: ppetrbednar/pallidium:api.latest
  pull_policy: always
  restart: always
  ports:
    - '8000:8000'
  networks:
    network:
      ipv4_address: 172.10.0.3
```

Celá konfigurace je k dispozici v příloze 8.

### 6.8.3 Docker Compose pro frontend

Nastavení Compose pro frontend je obdobné jako u BE. Konfigurace se liší jinými cílovými verzemi a mapováním vnějších portů.

*Zdrojový kód 46: Docker compose pro frontend*

```
pallidium-web-service:
  container_name: pallidium-web-container
  image: ppetrbednar/pallidium:web.latest
  pull_policy: always
  restart: always
  ports:
    - '80:80'
    - '443:443'
  networks:
    network:
      ipv4_address: 172.10.0.4
```

Celá konfigurace je k dispozici v příloze 9.

### 6.8.4 Workflow pro backend

Z hlediska automatizace nasazování byla vytvořena manuálně spustitelná workflow zajišťující bezstarostný průběh nasazení Spring serveru pomocí kontejnerového image na VPS. V porovnání s již probranými workflow konfiguracemi se jedná o jednoduchý proces. Nejsložitějším bodem je v tomto úseku odeslání specifických příkazů, které budou provedeny přímo v prostoru VPS. Připojení je řešeno za pomoci SSH klíče, který je přidán jako sekundární přístupový klíč hlavního správce v nastavení autorizovaných klíčů.

*Zdrojový kód 47: Workflow CD pro backend, část 1.*

```
- name: Setup SSH key
  env:
    PRIVATE_KEY: ${{ secrets.SSH_PALLIDIUM_KEY }}
  run: |
    mkdir -p ~/.ssh
    echo "$PRIVATE_KEY" > ~/.ssh/id_rsa
    chmod 600 ~/.ssh/id_rsa
```

Pro úspěšné navázání komunikace s VPS je SSH klíč zkopírován ze schránky privátních dat GitHub repositáře do proměnné prostředí. Tato proměnná je omezena pouze na daný krok běžící úlohy. Proměnná je využita k vytvoření souboru ve složce „.ssh“ s názvem „id_rsa“. Je důležité, aby obsah klíče byl ve formátu „id_rsa“ při použití RSA klíčů. [101]

*Zdrojový kód 48: Workflow CD pro backend, část 2.*

```
- name: Connect, deploy and cleanup on Pallidium server
  run: |
    <příkaz navázání a odeslání příkazů přes SSH> << EOF
      <příkaz přihlášení uživatelského účtu Docker>
      docker-compose -f ./pallidium-api.yaml up -d
      echo y | docker image prune -a
    EOF
```

Z důvodu přehlednosti byly odebrány dva příkazy, které jsou níže prezentovány samostatně. Příkaz „docker-compose“ využívá vytvořeného YAML souboru pro natažení nového kontejneru nebo jeho opětovné nahrání. Poslední příkaz čistí nepoužité image, které byly po každém stažení uloženy, a udržuje tak paměť VPS neobsazenou.

První z příkazů naváže SSH spojení s cílovým serverem a vybraným uživatelem. Celý příkaz je dodán jako textový blok, který je zahájený a ukončený EOF znaky. Obsažené příkazy budou provedeny sekvenčně za sebou a každý počká na dokončení předchozího. [102]

*Zdrojový kód 49: Workflow CD pro backend, část 3.*

```
ssh -o StrictHostKeyChecking=no -i ~/.ssh/id_rsa ${{ secrets.SSH_PALLIDIUM_USERNAME }}@${{
secrets.SSH_PALLIDIUM_HOST }}
```

Druhý příkaz přihlásí poskytnutý uživatelský účet k Docker. Díky tomuto kroku je zajištěna přítomnost účtu a není tak třeba riskovat jeho předčasné vypršení relace.

*Zdrojový kód 50: Workflow CD pro backend, část 4.*

```
echo ${{ secrets.DOCKER_ACCESS_TOKEN_READONLY }} | docker login --username ${{
secrets.DOCKER_USERNAME }} --password-stdin
```

Celá workflow je k dispozici v příloze 10.

### 6.8.5 Workflow pro frontend

FE workflow byly zkomplikované nutností zabezpečit spojení s Nginx serverem. Z tohoto důvodu byl proveden přechod na komunikaci pomocí protokolu HTTPS zabezpečeného pomocí SSL.

Vytváření SSL certifikátů předchází správná konfigurace Nginx. Pro první spuštění v Docker kontejneru je nutné, aby bylo SSL vypnuto. Důvod je takový, že Let's Encrypt má platnost certifikátu pouze 3 měsíce. Ověření certifikátu je prováděno přes webovou doménu, se kterou je provázán. Postup ověření je podrobněji popsán v kapitole 3.6.2. [103]

Z důvodu limitace opětovných vytvoření certifikátů pro stejnou doménu nelze Let's Encrypt certifikáty generovat při každém startu kontejneru. Limit by byl i přes své štědré metriky rychle překročen a nebylo by tak možné web dále provozovat. [104]

Díky výše zmíněným okolnostem byl workflow pro frontend rozdělen na dva.

#### 6.8.5.1 Workflow prvotního nasazení

Tato workflow má za úkol navázat spojení s certifikační autoritou Let's Encrypt, podstoupit tak celý proces generace a ověřování webové domény a vytvořené certifikáty uložit mimo svůj kontejner do složky vyhrazené v paměti VPS. Proces využívá příkazů dle specifikace v dokumentaci Docker pro kopírování do a ven z kontejneru. V ostatních bodech jako je nastavení klíče SSH, připojení k SSH, odeslání příkazů a odebrání klíče je workflow shodná s předchozí.

*Zdrojový kód 51: Workflow CD pro frontend s obnovou SSL*

```
docker exec pallidium-web-container openssl dhparam -out /etc/nginx/dhparam.pem 2048
docker exec pallidium-web-container mkdir -p /var/www/_letsencrypt
docker exec pallidium-web-container chown www-data /var/www/_letsencrypt
docker exec pallidium-web-container certbot certonly --webroot -d pallidium.cz -d
www.pallidium.cz -d api.pallidium.cz --email info@pallidium.cz -w /var/www/_letsencrypt -n -
-agree-tos
rm -rf ~/pallidium-ssl
mkdir -p ~/pallidium-ssl/archive
docker cp pallidium-web-container:/etc/letsencrypt/archive/pallidium.cz ~/pallidium-
ssl/archive
docker cp pallidium-web-container:/etc/nginx/dhparam.pem ~/pallidium-ssl/dhparam.pem
docker exec pallidium-web-container sed -i -r -z 's/#?; ?#//g'
/etc/nginx/conf.d/default.conf
docker exec pallidium-web-container nginx -t
docker exec pallidium-web-container nginx -s reload
```

Jednotlivé příkazy výše jsou oddělené mezerami. V první fázi je vygenerován certifikát a následně jsou zálohovány všechny potřebné hodnoty a systém restartován. Před restartem

jsou pročištěny komentáře zakazující použití SSH zabezpečeného spojení. Celá workflow je k dispozici v příloze PŘÍLOHA 11.

#### 6.8.5.2 Workflow opětovných nasazení

Tato workflow spoléhá na již dokončený proces zjišťování certifikátů a jeho zálohované hodnoty v paměti VPS. Změnou ve funkčnosti oproti předchozí workflow je překopírování již vytvořených certifikátů do prostředí kontejneru, odebrání komentářů blokujících komunikaci přes SSH a restart systému pro načtení přidaných certifikátů.

*Zdrojový kód 52: Workflow CD pro frontend*

```
docker exec pallidium-web-container mkdir -p /etc/letsencrypt/archive
docker cp ~/pallidium-ssl/archive/pallidium.cz pallidium-web-
container:/etc/letsencrypt/archive/pallidium.cz
docker exec pallidium-web-container mkdir -p /etc/letsencrypt/live/pallidium.cz
docker exec pallidium-web-container ln -s /etc/letsencrypt/archive/pallidium.cz/cert1.pem
/etc/letsencrypt/live/pallidium.cz/cert.pem
docker exec pallidium-web-container ln -s /etc/letsencrypt/archive/pallidium.cz/chain1.pem
/etc/letsencrypt/live/pallidium.cz/chain.pem
docker exec pallidium-web-container ln -s
/etc/letsencrypt/archive/pallidium.cz/fullchain1.pem
/etc/letsencrypt/live/pallidium.cz/fullchain.pem
docker exec pallidium-web-container ln -s /etc/letsencrypt/archive/pallidium.cz/privkey1.pem
/etc/letsencrypt/live/pallidium.cz/privkey.pem
docker cp ~/pallidium-ssl/dhparam.pem pallidium-web-container:/etc/nginx/dhparam.pem
docker exec pallidium-web-container sed -i -r -z 's/#?; ?#//g'
/etc/nginx/conf.d/default.conf
docker exec pallidium-web-container nginx -t
docker exec pallidium-web-container nginx -s reload
```

Jednotlivé příkazy jsou odděleny mezerami. Příkazy provedou vytvoření požadovaných složek, překopírují potřebné soubory a sestrojí symbolické vazby mezi soubory. Konfigurace je tedy ekvivalentní jako po čerstvém dokončení procesu získání certifikátů. Celá workflow je k dispozici v příloze 12.

## 6.9 OpenVPN

Pro zajištění přístupu k datům uložených v produkční databázi musel být vytvořen tunel mezi osobním počítačem a vzdáleným serverem. Po neschopnosti manuálně nastavit software OpenVPN na serveru byla objevena jiná možnost pomocí automatizovaného nástroje. Uživatel systému GitHub nesoucí jméno angristan vytvořil za pomoci asi 71 přispěvatelů nástroj pro unixový shell, který je schopen vyřešit nastavení celého OpenVPN. Postup je velice jednoduchý a spočívá ve stažení volně dostupného skriptu, který po spuštění provede uživatele celou instalací. Po nastavení základních parametrů je celý systém automaticky nainstalován. Následně je nabídnuta možnost vytvoření přístupového souboru, pomocí kterého se lze na vzdálený server připojit. Pro připojení je soubor nahrán do desktopové aplikace OpenVPN Connect a propojení s VPS serverem je umožněno. Po připojení se osobní počítač nalezne v nově vytvořené lokální síti serveru s možností přístupu k vystavenému databázovému systému na síťovém portu 4040. Adresa IP je shodná s výchozí bránou vytvořené sítě. [61], [105]

# ZÁVĚR

Cílem diplomové práce byl návrh a implementace webového informačního systému, který je primárně zaměřen k persistenci a správě multimediálních metadat. Při návrhu byly stanoveny požadavky na výsledný systém, které byly s dokončením implementace nad rámec splněny.

Výsledný BE je navržen tak, aby byl lehce rozšiřitelný, zabezpečený a stabilní. Součástí implementace jsou například profilovací nástroje za pomoci aspektů nebo systém správy cache, díky kterému jsou splněny limity nastavené poskytovatelem využitého API. BE je nastaven tak, že zpracovává dotazy od FE a synchronizační dotazy z desktopové aplikace.

Při tvorbě uživatelského FE byl jako základ využit design desktopové aplikace. Původní design byl optimalizován a rozšířen pro účely stanovených požadavků. Díky robustnosti použitého jazyka je výsledný design interaktivnější, něž desktopová aplikace a mnohem rozšiřitelnější. Dotazy jsou zasílány na BE pomocí zabezpečené komunikace SSL. Další vlastností je vestavěná komunikace s desktopovou aplikací pro získávání dodatečných informací a možností spouštět multimédia přímo z webové aplikace.

Desktopová aplikace byla, podle specifikovaných požadavků, úspěšně rozšířena o podporu synchronizace. Aplikace je nyní schopná zálohovat hodnocení a knihovny do systému webové aplikace. Synchronizační nástroj aplikace obsahuje různé režimy pro přizpůsobení synchronizovaných dat s podporou oboustranných změn.

Vytvořená stabilní verze FE a BE byla úspěšně nasazena. Pro účely nasazení byla zakoupena doména pallidium.cz a privátní server. Celý proces vývoje byl zautomatizován pomocí workflow GitHub Actions kde kontejnerizované obrazy aplikací jsou nasazeny v prostředí Docker na VPS. BE je spouštěn za pomoci Tomcat serveru nastaveném díky Spring Boot. FE je spravován software Nginx poskytujícím SSL zabezpečení a směrování komunikace pomocí několika serverů.

V práci byla podrobně probrána problematika spojená s vývojem a nasazením informačních systémů. V prvních kapitolách byl čtenář seznámen s konkurenčními systémy, rozšířením desktopové aplikace, návrhem systému a použitými technologiemi. Implementace pokryla návrh databázového systému, architektury jednotlivých částí systému a hlavní cíle práce. Následující kapitola provedla čtenáře životním cyklem uživatele napříč celým vytvořeným ekosystémem aplikací. Popis byl proveden na reálném scénáři využití.

Každá fáze vývoje aplikace byla důkladně zdokumentována a v této diplomové práci popsána. Výslednou práci mohou využít vývojáři pro inspiraci či jako referenci k návrhu a tvorbě rozsáhlých informačních systémů. Podrobné informace a ukázky nastavení workflow je možné jednoduše použít při tvorbě vlastní automatizace nasazování. Dalším přínosem jsou detailní informace z hlediska nastavení DNS potřebné domény a firewallu VPS. Mezi další zmíněné technologie patří OpenVPN pro vzdálenou správu a nástroj Certbot k tvorbě SSL certifikátů certifikační autority Let's Encrypt.

Budoucnost webové aplikace silně závisí na vyřešení několika závažných problémů. Hlavním problémem jsou finance. Z hlediska měsíčních nákladů na provoz se při dnešních cenách jedná o 138 Kč. Plánem je tedy vytvořit způsob legální a přívětivé monetizace, která by pokryla tyto měsíční náklady. Dalšími cíli jsou vyladění systému, poskytnutí více možností interakce mezi uživateli, propracovanější systém zobrazení statistik a publikace reklamy aplikace. Nehledě na budoucí úspěch/neúspěch se jedná o cenný osobní projekt, který bude podporován i přes jistá úskalí osudu.

Pro desktopovou aplikaci je plán do budoucna nejistý. Při práci na požadovaných rozšířeních byly vytvořeny jisté kroky k migraci od desktopové aplikace. Díky propracovanému a lehce rozšiřitelnému FE webové aplikace je snaha o dodělání funkčnosti správy lokální aplikace skrze webovou stránku. FE již nyní dokáže komunikovat a odesílat dotazy do lokální aplikace, a tak by byl systém rozšířen na celkové ovládání aplikace. Desktopová aplikace by přešla na technologii Spring Boot, která se vyznačuje lepším škálováním, a stal by se z ní nástroj v pozadí. To by zaručilo sjednocení FE na jedné verzi a dovolilo bohatší možnosti pro budoucnost celého projektu.

# POUŽITÁ LITERATURA


[1] HELLERMAN, Jason. *According to a Study 70% of People Would Rather Watch Movies at Home.* Online. In: No Film School, 20. 5. 2020. Dostupné z: https://nofilmschool.com/watch-at-home-data. [cit. 2024-02-27].

[2] NOVÁKOVÁ, Tereza. *Co je rešerše.* Online. In: Ústřední knihovna ČVUT v Praze, 5. 2. 2024. Dostupné z: https://knihovna.cvut.cz/seminare-a-vyuka/jak-psat/co-je-reserse. [cit. 2024-02-27].

[3] BEDNÁŘ, Petr. *Správa multimediálních souborů.* Bakalářská práce. Pardubice: Univerzita Pardubice, Fakulta elektrotechniky a informatiky. 2022. Dostupné z: Digitální knihovna UPCE, https://dk.upce.cz/handle/10195/79547. [cit. 2024-02-27].

[4] BELL, Travis. *Rate Limiting.* Online. In: The Movie Database (TMDB), 4. 2023. Dostupné z: https://developer.themoviedb.org/docs/rate-limiting. [cit. 2024-02-27].

[5] ECONOMIA, A. S. *ČSFD.cz.* Online. In: Aktuálně.cz, 25. 11. 2011, revize 20. 3. 2014. Dostupné z: https://www.aktualne.cz/wiki/kultura/csfdcz-cesko-slovenska-filmova-databaze-csfd/r~i:wiki:2209. [cit. 2024-02-27].

[6] POMO MEDIA GROUP S. R. O. *Česko-Slovenská filmová databáze (ČSFD).* Online. Dostupné z: https://www.csfd.cz. [cit. 2024-02-27].

[7] ECONOMIA, A.S. *Filmová databáze IMDb slaví 30 let. Začala hodnocením očí hereček.* Online. In: Aktuálně.cz, 17. 10. 2020. Dostupné z: https://magazin.aktualne.cz/kultura/film/filmova-databaze-imdb-slavi-30-let-zacala-hodnocenim-oci/r~650a85f4105f11eb8b230cc47ab5f122. [cit. 2024-02-27].

[8] IMDB. *IMDb Essential Metadata for Movies/TV/OTT (API).* Online. In: AWS Marketplace, c2024. Dostupné z: https://aws.amazon.com/marketplace/pp/prodview-wdqq4hg3bcbws. [cit. 2024-02-27].

[9] EDUCATIVE, INC. *Introduction to The Movie Database API.* Online. In: Educative, c2024. Dostupné z: https://www.educative.io/courses/movie-database-api-python/introduction-to-the-movie-database-api. [cit. 2024-02-27].

[10] LIGHTSEY, Bob. *Systems Engineering Fundamentals.* Online. Defense Acquisition University Press Fort Belvoir, 1. 1. 2001. Dostupné z: Defense Technical Information Center, https://apps.dtic.mil/sti/citations/ADA387507. [citováno 2024-02-27].

[11] ŽOLTÁ, Lucie. *Disciplína sběr a analýza požadavků.* Online. In: Lucka Žoltá, 17. 11. 2012. Dostupné z: http://lucie.zolta.cz/index.php/softwarove-inzenyrstvi/150-disciplina-sber-a-analyza-pozadavku. [cit. 2024-02-27].

[12] ARLOW, Jim a NEUSTADT, Ila. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky.* Přeložil Bogdan KISZKA. Brno: Computer Press, 2007. ISBN 978-80-251-1503-9.

[13] JUILLET, Romain. *What you need to know about the Java programming language.* Online. In: Bocasay, 19. 4. 2022. Dostupné z: https://www.bocasay.com/what-you-need-know-about-java-programming-language. [cit. 2024-02-27].

[14] KRČMÁŘ, Petr. *Sun nakonec kupuje firma Oracle: co bude s MySQL?* Online. In: Root.cz, 21. 4. 2009. Dostupné z: https://www.root.cz/clanky/sun-nakonec-kupuje-firma-oracle-co-bude-s-mysql. [cit. 2024-02-27].

[15] THE ASSOCIATED PRESS. *Oracle completes acquisition of Sun Microsystems.* Online. In: Phys.org, 27. 2. 2010. Dostupné z: https://phys.org/news/2010-01-oracle-acquisition-sun-microsystems.html. [cit. 2024-02-27].

[16] AMAZON WEB SERVICES, INC. *What is Java? - Java Programming Language Explained - AWS.* Online. In: Cloud Computing Services – Amazon Web Services (AWS), c2024. Dostupné z: https://aws.amazon.com/what-is/java. [cit. 2024-02-27].

[17] MAMMADOV, Rafael. *What is Java? What is IDE?* Online. In: Medium, 1. 1. 2024. Dostupné z: https://medium.com/@rafaelmammadov/what-is-java-what-is-ide-3b2fa48fb6a1. [cit. 2024-02-27].

[18] ORACLE. *Tools and Commands Reference.* Online. In: Oracle Help Center, c2024. Dostupné z: https://docs.oracle.com/en/java/javase/11/tools/tools-and-command-reference.html. [cit. 2024-02-27].

[19] ORACLE. *The jpackage Command.* Online. In: Oracle, c2024. Dostupné z: https://download.java.net/java/early_access/jdk22/docs/specs/man/jpackage.html. [cit. 2024-02-27].

[20] .NET FOUNDATION. *About.* Online. In: WiX Toolset, c2024. Dostupné z: https://wixtoolset.org/docs/about. [cit. 2024-02-27].

[21] SIMPLILEARN. *What Is Spring Framework and Its Advantages.* Online. In: Simplilearn, 23. 2. 2023. Dostupné z: https://www.simplilearn.com/tutorials/spring-boot-tutorial/what-is-spring-framework-and-its-advantages. [cit. 2024-02-27].

[22] KUNČAR, Petr. *Spring - IoC Kontejner.* Online. In: itnetwork.cz - Učíme národ IT, 5. 9. 2016, revize 18. 1. 2022. Dostupné z: https://www.itnetwork.cz/java/spring-boot/filmova-databaze/spring-ioc-kontejner. [cit. 2024-02-27].

[23] DIAZ, Miguel Angel Perez. *Dependency Injection in Spring: Constructor, Property, or Setter? Discover the Best Option for Your Project!* Online. In: Medium, 5. 4. 2023. Dostupné z: https://medium.com/@miguelangelperezdiaz444/dependency-injection-in-spring-constructor-property-or-setter-which-one-should-i-choose-d38be824c8c1. [cit. 2024-02-27].

[24] KUMAR, Pradeesh. *Circular Dependencies in Spring.* Online. In: Medium, 4. 11. 2020. Dostupné z: https://pradeesh-kumar.medium.com/circular-dependencies-in-spring-9644d2f6eeea. [cit. 2024-02-27].

[25] SPRING CLOUD. *SpringBoot 2.6.x coping strategies after disabling cyclic dependencies by default.* Online. In: Spring Cloud, 23. 2. 2022. Dostupné z: https://www.springcloud.io/post/2022-02/spring-cyclic-dependencies. [cit. 2024-02-27].

[26] STŘECHA, Tomáš. *Lekce 1 - Úvod do Spring Boot frameworku v Javě.* Online. In: itnetwork.cz, 7. 5. 2020, revize 29. 3. 2023. Dostupné z: https://www.itnetwork.cz/java/spring-boot/zaklady/uvod-do-spring-boot-frameworku-pro-javu. [cit. 2024-02-27].

[27] BROADCOM INC. *Spring Data JPA.* Online. In: Spring | Home, c2024. Dostupné z: https://spring.io/projects/spring-data-jpa. [cit. 2024-02-27].

[28] JAVATECHONLINE BLOG. *AOP (Aspect Oriented Programming) in Spring & Spring Boot.* Online. In: Medium, 19. 8. 2023. Dostupné z: https://javatechonline.medium.com/aop-aspect-oriented-programming-in-spring-spring-boot-13cf2b33d835. [cit. 2024-02-27].

[29] KANADE, Vijay. *What Is Aspect-Oriented Programming (AOP)? Meaning, Working, and Frameworks.* Online. In: Spiceworks, 22. 6. 2023. Dostupné z: https://www.spiceworks.com/tech/devops/articles/what-is-aop. [cit. 2024-02-27].

[30] TUTORIALSPOINT. *Spring AOP - Advice Types.* Online. In: Tutorialspoint, c2024. Dostupné z: https://www.tutorialspoint.com/springaop/springaop_advice_types.htm. [cit. 2024-02-27].

[31] SQDW. *Aspektově orientované programování.* Online. In: Rising sun, 5. 4. 2008. Dostupné z: https://sqdw.signaly.cz/0804/aspektove-orientovane-programovani. [cit. 2024-02-27].

[32] BAELDUNG. *Comparing Spring AOP and AspectJ.* Online. In: Baeldung, 8. 1. 2024. Dostupné z: https://www.baeldung.com/spring-aop-vs-aspectj. [cit. 2024-02-27].

[33] GABA, Ishan. *What is Gradle? Why Do We Use Gradle? Explained.* Online. In: Simplilearn, 24. 2. 2023. Dostupné z: https://www.simplilearn.com/tutorials/gradle-tutorial/what-is-gradle. [cit. 2024-02-27].

[34] GRADLE, INC. *Declaring repositories.* Online. In: Gradle User Manual, c2023. Dostupné z: https://docs.gradle.org/current/userguide/declaring_repositories.html. [cit. 2024-02-27].

[35] TOBSEF. *github-plugin-registry-example.* Online. In: GitHub, 18. 10. 2019. Dostupné z: https://github.com/TobseF/github-plugin-registry-example. [cit. 2024-02-27].

[36] MALÝ, Martin. *REST: architektura pro webové API.* Online. In: Zdroják, 3. 8. 2009. Dostupné z: https://zdrojak.cz/clanky/rest-architektura-pro-webove-api. [cit. 2024-02-27].

[37] GILLIS, S. Alexander. *REST API (RESTful API).* Online. In: TechTarget, 9. 2020. Dostupné z: https://www.techtarget.com/searchapparchitecture/definition/RESTful-API. [cit. 2024-02-27].

[38] MALÝ, Martin. *YAML: Serializační formát pro ukládání dat.* Online. In: Zdroják, 4. 12. 2009. Dostupné z: https://zdrojak.cz/clanky/yaml-serializacni-format-pro-ukladani-dat. [cit. 2024-02-27].

[39] ELLIS, Danielle Richardson. *What is Swagger? A Beginner's Guide.* Online. In: HubSpot Blog, 26. 7. 2022. Dostupné z: https://blog.hubspot.com/website/what-is-swagger. [cit. 2024-02-27].

[40] THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. *About.* Online. In: PostgreSQL, c2024. Dostupné z: https://www.postgresql.org/about. [cit. 2024-02-27].

[41] VITA. *Lekce 1 - PostgreSQL - Úvod a příprava prostředí.* Online. In: itnetwork.cz, 29. 7. 2015, revize 14. 12. 2022. Dostupné z: https://www.itnetwork.cz/postgresql/postgresql-uvod-a-priprava-prostredi. [cit. 2024-02-27].

[42] MÁCA, Jindřich. *Lekce 1 - Úvod do Node.js.* Online. In: itnetwork.cz - Učíme národ IT, 18. 4. 2018, revize 3. 4. 2023. Dostupné z: https://www.itnetwork.cz/javascript/nodejs/uvod-do-nodejs. [cit. 2024-02-27].

[43] JAHODA, Bohumil. *Vite – super rychlý dev server / build.* Online. In: Je čas.cz, 1. 3. 2021. Dostupné z: https://jecas.cz/vite. [cit. 2024-02-27].

[44] KVAPIL, Jiří. *Lekce 1 - Úvod do TypeScriptu.* Online. In: itnetwork.cz - Učíme národ IT, 15. 5. 2018, revize 30. 11. 2021. Dostupné z: https://www.itnetwork.cz/javascript/typescript/uvod-do-typescriptu. [cit. 2024-02-27].

[45] MÁCA, Jindřich. *Lekce 1 - Úvod do React.* Online. In: itnetwork.cz - Učíme národ IT, 28. 2. 2019, revize 11. 7. 2023. Dostupné z: https://www.itnetwork.cz/javascript/react/zaklady/uvod-do-react. [cit. 2024-02-27].

[46] KOLLEGGER, Eric. *What is Axios.js and why should I care?* Online. In: Medium, 14. 5. 2018. Dostupné z: https://medium.com/@MinimalGhost/what-is-axios-js-and-why-should-i-care-7eb72b111dc0. [cit. 2024-02-27].

[47] KOMUNITA AXIOS. *Interceptors.* Online. In: Axios, c2024. Dostupné z: https://axios-http.com/docs/interceptors. [cit. 2024-02-27].

[48] KOMUNITA AXIOS. *Axios API.* Online. In: Axios, c2024. Dostupné z: https://axios-http.com/docs/api_intro. [cit. 2024-02-27].

[49] SHARMA, Manik. *What is MUI and what do you need to know about it?* Online. In: The Talent500 Blog, 19. 9. 2022. Dostupné z: https://talent500.co/blog/what-is-mui-and-what-do-you-need-to-know-about-it. [cit. 2024-02-27].

[50] MIRELA. *i18next: what it is, why you should use it.* Online. In: POEditor Blog, 21. 12. 2023, revize 9. 1. 2024. Dostupné z: https://poeditor.com/blog/i18next. [cit. 2024-02-27].

[51] KOMUNITA MOBX. *About MobX.* Online. In: MobX, 27. 11. 2023. Dostupné z: https://mobx.js.org/README.html. [cit. 2024-02-27].

[52] ŠTECHMÜLLER, Petr. *Lekce 1 - Úvod do JavaFX.* Online. In: itnetwork.cz, 13. 3. 2019, revize 4. 10. 2022. Dostupné z: https://www.itnetwork.cz/java/javafx/uvod-do-javafx. [cit. 2024-02-27].

[53] HIPP, Dwayne Richard. *About SQLite.* Online. In: SQLite, 12. 11. 2007, revize 10. 10. 2023. Dostupné z: https://www.sqlite.org/about.html. [cit. 2024-02-27].

[54] CODING SCHOOL. *Co je to git a github.* Online. In: Praha Coding School, 21. 2. 2023. Dostupné z: https://prahacoding.cz/co-je-to-git-github. [cit. 2024-02-27].

[55] TRONÍČEK, Filip. *GitHub Actions aneb CI/CD v cloudu.* Online. In: studuj.digital, 10. 8. 2021. Dostupné z: https://studuj.digital/2021/08/10/github-actions-ci-cd-v-cloudu. [cit. 2024-02-27].

[56] GITHUB, INC. *Introduction to GitHub Packages.* Online. In: GitHub Docs, c2024. Dostupné z: https://docs.github.com/en/packages/learn-github-packages/introduction-to-github-packages. [cit. 2024-02-27].

[57] WEBSUPPORT, S. R. O. *Docker – 1. Představení, instalace a základní operace.* Online. In: Websupport Centrum nápovědy, 20. 9. 2022. Dostupné z: https://www.websupport.cz/podpora/kb/docker-1-predstaveni-instalace-a-zakladni-operace. [cit. 2024-02-27].

[58] DOCKER, INC. *Docker Compose overview.* Online. In: Docker Docs, c2024. Dostupné z: https://docs.docker.com/compose. [cit. 2024-02-27].

[59] KINSTA INC. *What Is Nginx? A Basic Look at What It Is and How It Works.* Online. In: Kinsta, 14. 8. 2023. Dostupné z: https://kinsta.com/knowledgebase/what-is-nginx. [cit. 2024-02-27].

[60] HALLY. *Co je OpenVPN? Je to bezpečné?* Online. In: Soubory, 24. 9. 2020. Dostupné z: https://soubory.info/info/co-je-openvpn-je-to-bezpecne. [cit. 2024-02-27].

[61] ANGRISTAN. *openvpn-install.* Online. In: GitHub, 20. 11. 2023. Dostupné z: https://github.com/angristan/openvpn-install. [cit. 2024-02-27].

[62] S., Shraddha. *What is Let's Encrypt?* Online. In: CloudPanel, 14. 4. 2022. Dostupné z: https://www.cloudpanel.io/blog/what-is-let-s-encrypt. [cit. 2024-02-27].

[63] LEONARD, Anghel. Spring Boot Persistence Best Practices: Optimize Java Persistence Performance in Spring Boot Applications. 1st ed. New York: Apress, 2020. ISBN 978-1-4842-5625-1.

[64] KUENZLER, Claudio. *Nginx reverse proxy error: SSL alert number 40 while SSL handshaking to upstream server (SSL server name).* Online. In: Claudio Kuenzler, 30. 8. 2021. Dostupné z: https://www.claudiokuenzler.com/blog/1120/nginx-reverse-proxy-ssl-alert-number-40-while-ssl-handshaking-upstream. [cit. 2024-02-27].

[65] DIGITALOCEAN, LLC. *NGINXConfig.* Online. In: DigitalOcean, c2024. Dostupné z: https://www.digitalocean.com/community/tools/nginx. [cit. 2024-02-27].

[66] DJANGOFAN. *docker-tomcat7-cluster.* Online. In: GitHub, 11. 3. 2018. Dostupné z: https://github.com/djangofan/docker-tomcat7-cluster/blob/master/nginx/mime.types. [cit. 2024-02-27].

[67] DONG, Leon. *Fast-Setup: Nginx Https Reverse Proxy for Springboot API.* Online. In: Medium, 1. 2. 2023. Dostupné z: https://blackist.medium.com/fast-setup-nginx-https-reverse-proxy-for-springboot-api-32f9ca70f97. [cit. 2024-02-27].

[68] HADZHIEV, Borislav. *GET favicon.ico 404 (Not Found) Error [Solved].* Online. In: bobbyhadz blog, 27. 5. 2023. Dostupné z: https://bobbyhadz.com/blog/get-favicon-ico-404-not-found. [cit. 2024-02-27].

[69] SEOMOZ, INC. *What Is A Robots.txt File? Best Practices For Robot.txt Syntax - Moz.* Online. In: Moz, c2024. Dostupné z: https://moz.com/learn/seo/robotstxt. [cit. 2024-02-27].

[70] AIRA GROUP, S. R. O. *Co je to proxy server?* Online. In: Správa sítě, c2022. Dostupné z: https://www.sprava-site.eu/proxy-server/. [cit. 2024-02-27].

[71] TRAN, Tony. *How To Configure Nginx as a Reverse Proxy on Ubuntu 22.04.* Online. In: DigitalOcean, 16. 9. 2022. Dostupné z: https://www.digitalocean.com/community/tutorials/how-to-configure-nginx-as-a-reverse-proxy-on-ubuntu-22-04. [cit. 2024-02-27].

[72] SOMIC, Sofija. *Linux sed Command: How To Use the Stream Editor.* Online. In: phoenixNAP, 6. 4. 2022. Dostupné z: https://phoenixnap.com/kb/linux-sed. [cit. 2024-02-27].

[73] SHIN, Yeachan. *Testing SecurityContextHolder in Spring Security Tests with @WithMockUser.* Online. In: Medium, 17. 12. 2023. Dostupné z: https://medium.com/@kjavaman12/testing-securitycontextholder-in-spring-security-tests-with-withmockuser-38ce8060088b. [cit. 2024-02-27].

[74] RED HAT, INC. *What is CI/CD?* Online. In: Red Hat, 12. 12. 2023. Dostupné z: https://www.redhat.com/en/topics/devops/what-is-ci-cd. [cit. 2024-02-27].

[75] RED HAT, INC. *What is continuous delivery?* Online. In: Red Hat, 24. 11. 2020. Dostupné z: https://www.redhat.com/en/topics/devops/what-is-continuous-delivery. [cit. 2024-02-27].

[76] RED HAT, INC. *What is deployment automation?* Online. In: Red Hat, 10. 9. 2020. Dostupné z: https://www.redhat.com/en/topics/automation/what-is-deployment-automation. [cit. 2024-02-27].

[77] ACTIVE 24, S. R. O. *Co to je doména?* Online. In: Domény.cz, c2024. Dostupné z: https://domeny.cz/jak-na-to/co-to-je-domena-7. [cit. 2024-02-27].

[78] CZ.NIC, Z. S. P. O. *Jak se stát registrátorem?* Online. In: CZ.NIC, c2024. Dostupné z: https://www.nic.cz/page/309. [cit. 2024-02-27].

[79] CZ.NIC, Z. S. P. O. *Často kladené dotazy.* Online. In: CZ.NIC, c2024. Dostupné z: https://www.nic.cz/page/383/casto-kladene-dotazy. [cit. 2024-02-27].

[80] CZ.NIC, Z. S. P. O. *Vyhledávání v registru (WHOIS).* Online. In: CZ.NIC, c2024. Dostupné z: https://www.nic.cz/whois/domain/pallidium.cz. [cit. 2024-02-27].

[81] SECURITYNET.CZ S. R. O. *Domény.* Online. In: Hukot.net, c2024. Dostupné z: https://www.hukot.net/cs/domeny. [cit. 2024-02-27].

[82] CZ.NIC, Z. S. P. O. *Jak na to.* Online. In: MojeID, c2024. Dostupné z: https://www.mojeid.cz/cs/jak-na-to. [cit. 2024-02-27].

[83] CZ.NIC, Z. S. P. O. *Doménový prohlížeč.* Online. Dostupné z: https://www.domenovyprohlizec.cz. [cit. 2024-02-27].

[84] MŮČKA, Jan. *DNS záznamy: k čemu slouží a jak se v nich vyznat.* Online. In: MasterDC, 17. 8. 2022. Dostupné z: https://www.master.cz/blog/dns-zaznamy-typy. [cit. 2024-02-27].

[85] CZ.NIC, Z. S. P. O. *DNSSEC.* Online. Dostupné z: https://www.dnssec.cz. [cit. 2024-02-27].

[86] KUČERA, Radek. *VPS hosting – co to je a jak funguje.* Online. In: wpmax, 9. 4. 2018, revize 10. 7. 2020. Dostupné z: https://www.wpmax.cz/vps-hosting-co-to-je-a-jak-funguje. [cit. 2024-02-27].

[87] SECURITYNET.CZ S. R. O. *Linux VPS.* Online. In: Hukot.net, c2024. Dostupné z: https://www.hukot.net/cs/linux-vps. [cit. 2024-02-27].

[88] CANONICAL LTD. *OpenSSH Server.* Online. In: Ubuntu Server, 25. 2. 2023. Dostupné z: https://ubuntu.com/server/docs/service-openssh. [cit. 2024-02-27].

[89] KOMUNITA UBUNTU. *SSH/OpenSSH/Configuring.* Online. In: Official Ubuntu Documentation, 24. 8. 2015. Dostupné z: https://help.ubuntu.com/community/SSH/OpenSSH/Configuring. [cit. 2024-02-27].

[90] GRYGAŘÍKOVÁ, Michaela. *Docker, Kubernetes a kontejnery. Jak fungují a proč je chtít.* Online. In: MasterDC, 14. 8. 2019. Dostupné z: https://www.master.cz/blog/docker-kubernetes-kontejnery-jak-funguji-proc-je-chtit. [cit. 2024-02-27].

[91] MASTERDC. *Hypervizor.* Online. In: MasterDC, c2024. Dostupné z: https://www.master.cz/help/slovnik/hypervizor. [cit. 2024-02-27].

[92] DATADOG. *8 surprising facts about real Docker adoption.* Online. In: Datadog, 6. 2018. Dostupné z: https://www.datadoghq.com/docker-adoption. [cit. 2024-02-27].

[93] DOCKER, INC. *Dockerfile reference.* Online. In: Docker Docs, c2024. Dostupné z: https://docs.docker.com/engine/reference/builder. [cit. 2024-02-27].

[94] ORACLE CORPORATION. *JDK 21.* Online. In: OpenJDK, 19. 9. 2023. Dostupné z: https://openjdk.org/projects/jdk/21. [cit. 2024-02-27].

[95] POEHNELT, Justin. *Environment Variables in GitHub Docker build-push-action.* Online. In: DEV Community, 25. 9. 2022. Dostupné z: https://dev.to/jpoehnelt/environment-variables-in-github-docker-build-push-action-23pj. [cit. 2024-02-27].

[96] DOCKER, INC. *Version and name top-level elements.* Online. In: Docker Docs, c2024. Dostupné z: https://docs.docker.com/compose/compose-file/04-version-and-name. [cit. 2024-02-27].

[97] DOCKER, INC. *Services top-level elements.* Online. In: Docker Docs, c2024. Dostupné z: https://docs.docker.com/compose/compose-file/05-services. [cit. 2024-02-27].

[98] DOCKER, INC. *Networks top-level elements.* Online. In: Docker Docs, c2024. Dostupné z: https://docs.docker.com/compose/compose-file/06-networks. [cit. 2024-02-27].

[99] DOCKER, INC. *Network drivers overview.* Online. In: Docker Docs, c2024. Dostupné z: https://docs.docker.com/network/drivers. [cit. 2024-02-27].

[100] DOCKER, INC. *Secrets top-level elements.* Online. In: Docker Docs, c2024. Dostupné z: https://docs.docker.com/compose/compose-file/09-secrets. [cit. 2024-02-27].

[101] GITHUB, INC. *Variables. Online.* In: GitHub Docs, c2024. Dostupné z: https://docs.github.com/en/actions/learn-github-actions/variables. [cit. 2024-02-27].

[102] DEVOPS. *How To Deploy A Git Repository To A Server Using GitHub Actions.* Online. In: Programonaut, 19. 8. 2023. Dostupné z: https://www.programonaut.com/how-to-deploy-a-git-repository-to-a-server-using-github-actions. [cit. 2024-02-27].

[103] LET'S ENCRYPT. *How It Works.* Online. In: Let's Encrypt, 18. 10. 2019. Dostupné z: https://letsencrypt.org/how-it-works. [cit. 2024-02-27].

[104] LET'S ENCRYPT. *Rate Limits.* Online. In: Let's Encrypt, 5. 10. 2021. Dostupné z: https://letsencrypt.org/docs/rate-limits. [cit. 2024-02-27].

[105] OPENVPN, INC. *OpenVPN.* Online. Dostupné z: https://openvpn.net. [cit. 2024-02-27].

# SEZNAM PŘÍLOH

PŘÍLOHA 1: Globální nastavení Nginx ..... 126
PŘÍLOHA 2: Nastavení serverů Nginx ..... 127
PŘÍLOHA 3: Dockerfile backend ..... 129
PŘÍLOHA 4: Dockerfile frontend ..... 130
PŘÍLOHA 5: Workflow CI backend ..... 131
PŘÍLOHA 6: Workflow CI frontend ..... 133
PŘÍLOHA 7: Docker Compose databáze ..... 134
PŘÍLOHA 8: Docker Compose backend ..... 135
PŘÍLOHA 9: Docker Compose frontend ..... 136
PŘÍLOHA 10: Workflow CD backend ..... 137
PŘÍLOHA 11: Workflow CD frontend nové SSL ..... 138
PŘÍLOHA 12: Workflow CD frontend ..... 139

# PŘÍLOHA 1: Globální nastavení Nginx

```
user                        www-data;
pid                         /run/nginx.pid;
worker_processes            1;
worker_rlimit_nofile        65535;

include                     /etc/nginx/modules-enabled/*.conf;

events {
    multi_accept            on;
    worker_connections      65535;
}

http {
    charset                 utf-8;
    sendfile                on;
    tcp_nopush              on;
    tcp_nodelay             on;
    server_tokens           off;
    log_not_found           off;
    types_hash_max_size     2048;
    types_hash_bucket_size  64;
    client_max_body_size    16M;

    include                 mime.types;
    default_type            application/octet-stream;

    access_log              off;
    error_log               /dev/null;

    ssl_session_timeout     1d;
    ssl_session_cache       shared:SSL:10m;
    ssl_session_tickets     off;
    ssl_dhparam             /etc/nginx/dhparam.pem;

    ssl_protocols           TLSv1.2 TLSv1.3;
    ssl_ciphers             ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-
ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-
CHACHA20-POLY1305:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384;

    ssl_stapling            on;
    ssl_stapling_verify     on;
    resolver                1.1.1.1 1.0.0.1 8.8.8.8 8.8.4.4 208.67.222.222 208.67.220.220
valid=60s;
    resolver_timeout        2s;

    map "$host" $match {
        "api.pallidium.cz" 0;
        default 1;
    }

    include                 /etc/nginx/conf.d/*.conf;
}
```

# PŘÍLOHA 2: Nastavení serverů Nginx

```
server {
    listen                  443; # ssl;
    listen                  [::]:443; # ssl;
    http2                   on;
    server_name             www.pallidium.cz;

    #;#ssl_certificate         /etc/letsencrypt/live/pallidium.cz/fullchain.pem;
    #;#ssl_certificate_key     /etc/letsencrypt/live/pallidium.cz/privkey.pem;
    #;#ssl_trusted_certificate /etc/letsencrypt/live/pallidium.cz/chain.pem;

    add_header X-XSS-Protection             "1; mode=block" always;
    add_header X-Content-Type-Options       "nosniff" always;
    add_header Referrer-Policy              "no-referrer-when-downgrade" always;
    add_header Content-Security-Policy      "default-src 'self' http: https: ws: wss: data:
blob: 'unsafe-inline'; frame-ancestors 'self';" always;
    add_header Permissions-Policy           "interest-cohort=()" always;
    add_header Strict-Transport-Security    "max-age=31536000; includeSubDomains" always;

    location ~ /\.(?!well-known) {
        deny all;
    }

    location / {
        root                /usr/share/nginx/html;
        index               index.html;
        try_files           $uri $uri/ /index.html =404;
    }

    access_log              /var/log/nginx/access.log combined buffer=512k flush=1m;
    error_log               /var/log/nginx/error.log warn;

    location = /favicon.ico {
        log_not_found off;
        access_log off;
    }

    location = /robots.txt {
        allow all;
        log_not_found off;
        access_log off;
    }

    gzip                    on;
    gzip_vary               on;
    gzip_proxied            any;
    gzip_comp_level         6;
    gzip_types              text/plain text/css text/xml application/json
application/javascript application/rss+xml application/atom+xml image/svg+xml;
}

server {
    listen                  443; # ssl;
    listen                  [::]:443; # ssl;
    http2                   on;
    server_name             .pallidium.cz;

    #;#ssl_certificate         /etc/letsencrypt/live/pallidium.cz/fullchain.pem;
    #;#ssl_certificate_key     /etc/letsencrypt/live/pallidium.cz/privkey.pem;
    #;#ssl_trusted_certificate /etc/letsencrypt/live/pallidium.cz/chain.pem;

    if ($match) {
        return 301 https://www.pallidium.cz$request_uri;
    }
```

```
}

server {

    listen                  443; # ssl;
    listen                  [::]:443; # ssl;
    http2                   on;
    server_name             api.pallidium.cz;

    #;#ssl_certificate         /etc/letsencrypt/live/pallidium.cz/fullchain.pem;
    #;#ssl_certificate_key     /etc/letsencrypt/live/pallidium.cz/privkey.pem;
    #;#ssl_trusted_certificate /etc/letsencrypt/live/pallidium.cz/chain.pem;

    access_log              /var/log/nginx/access.log combined buffer=512k flush=1m;
    error_log               /var/log/nginx/error.log warn;

    location / {
        proxy_read_timeout                  90;
        proxy_set_header Host               $host;
        proxy_set_header X-Real-IP          $remote_addr;
        proxy_set_header X-Forwarded-For    $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto  $scheme;

        proxy_pass http://172.10.0.3:8000;
    }
}

server {
    listen                  80;
    listen                  [::]:80;
    server_name             .pallidium.cz;

    location ^~ /.well-known/acme-challenge/ {
        root /var/www/_letsencrypt/;
    }

    location / {
        return 301 https://www.pallidium.cz$request_uri;
    }
}
```

# PŘÍLOHA 3: Dockerfile backend

```
FROM openjdk:21-jdk

ARG DB_USERNAME
ENV DB_USERNAME ${DB_USERNAME}

ARG DB_PASSWORD
ENV DB_PASSWORD ${DB_PASSWORD}

ARG AUTH_JWT_SECRET
ENV AUTH_JWT_SECRET ${AUTH_JWT_SECRET}

ARG API_JWT_SECRET
ENV API_JWT_SECRET ${API_JWT_SECRET}

ARG TMDB_API_TOKEN
ENV TMDB_API_TOKEN ${TMDB_API_TOKEN}

ARG MAIL_USERNAME
ENV MAIL_USERNAME ${MAIL_USERNAME}

ARG MAIL_PASSWORD
ENV MAIL_PASSWORD ${MAIL_PASSWORD}

ARG SYS_ENVIRONMENT
ENV SYS_ENVIRONMENT ${SYS_ENVIRONMENT}

ARG SYS_VERSION
ENV SYS_VERSION ${SYS_VERSION}

ENV LANG=C.UTF-8

COPY build/libs/*.jar /pallidium.jar

EXPOSE 8000

ENTRYPOINT ["java","-jar", "/pallidium.jar"]
```

# PŘÍLOHA 4: Dockerfile frontend

```
FROM nginx:latest

RUN apt-get update && \
    apt-get install -y certbot python3-certbot-nginx

COPY nginx.conf /etc/nginx/

COPY mime.types /etc/nginx/

COPY nginx-server/nginx.conf /etc/nginx/conf.d/default.conf

COPY dist /usr/share/nginx/html

EXPOSE 80
EXPOSE 443

ENTRYPOINT ["nginx", "-g", "daemon off;"]
```

# PŘÍLOHA 5: Workflow CI backend

```yaml
name: CI

on:
  workflow_dispatch:
    inputs:
      version:
        description: 'Version number'
        required: true
        default: '0.0.1'
  push:
    branches:
      - 'main'

env:
  GH_PACKAGES_USERNAME: ${{ secrets.GH_PACKAGES_USERNAME }}
  GH_PACKAGES_ACCESS_TOKEN_READONLY: ${{ secrets.GH_PACKAGES_ACCESS_TOKEN_READONLY }}
  SYS_ENVIRONMENT: 'prod'
  SYS_VERSION: '0.0.1' # ${{ github.event.inputs.version }}

jobs:
  release:
    name: CI Pipeline
    runs-on: ubuntu-latest
    permissions:
      contents: read
    steps:
      - name: Specified version
        run: echo "Using version ${{ github.event.inputs.version }}"

      - name: Checkout code
        uses: actions/checkout@v4

      - name: Set up JDK 19 for x64
        uses: actions/setup-java@v2
        with:
          java-version: '19'
          distribution: 'adopt'
          cache: gradle

      - name: Build with Gradle
        run: ./gradlew build --no-daemon

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v3

      - name: Log in to Docker Hub
        uses: docker/login-action@v3
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_ACCESS_TOKEN }}

      - name: Build and push Docker image
        uses: docker/build-push-action@v5
        with:
          context: .
          push: true
          tags: ppetrbednar/pallidium:api.latest
          build-args: |
            "SYS_ENVIRONMENT=prod"
            "SYS_VERSION=${{ github.event.inputs.version }}"
            "DB_USERNAME=${{ secrets.DB_USERNAME }}"
            "DB_PASSWORD=${{ secrets.DB_PASSWORD }}"
            "AUTH_JWT_SECRET=${{ secrets.AUTH_JWT_SECRET }}"
```

```
"API_JWT_SECRET=${{ secrets.API_JWT_SECRET }}"
"TMDB_API_TOKEN=${{ secrets.TMDB_API_TOKEN }}"
"MAIL_USERNAME=${{ secrets.MAIL_USERNAME }}"
"MAIL_PASSWORD=${{ secrets.MAIL_PASSWORD }}"
```

# PŘÍLOHA 6: Workflow CI frontend

```
name: CI

on:
  workflow_dispatch:
  push:
    branches:
      - "main"

jobs:
  release:
    name: CI Pipeline
    runs-on: ubuntu-latest
    permissions:
      contents: read
      packages: write
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Set up Node 20
        uses: actions/setup-node@v4
        with:
          node-version: 20
          cache: 'npm'

      - name: Install dependencies
        run: npm ci

      - name: Build project
        run: npm run build

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v3

      - name: Log in to Docker Hub
        uses: docker/login-action@v3
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_ACCESS_TOKEN }}

      - name: Build and push Docker image
        uses: docker/build-push-action@v5
        with:
          context: .
          push: true
          tags: ppetrbednar/pallidium:web.latest
```

# PŘÍLOHA 7: Docker Compose databáze

```
version: "3.9"
name: pallidium
services:
  pallidium-db-service:
    container_name: pallidium-db-container
    image: postgres
    restart: unless-stopped
    ports:
      - '4040:5432'
    environment:
      POSTGRES_DB: pallidium
      POSTGRES_USER_FILE: /run/secrets/postgres_user
      POSTGRES_PASSWORD_FILE: /run/secrets/postgres_password
    secrets:
      - postgres_user
      - postgres_password
    networks:
      network:
        ipv4_address: 172.10.0.2

secrets:
  postgres_user:
    file: ./secrets/postgres_user.txt
  postgres_password:
    file: ./secrets/postgres_password.txt

volumes:
  db:
    driver: local

networks:
  network:
    driver: bridge
    ipam:
      config:
        - subnet: 172.10.0.0/16
          gateway: 172.10.0.1
```

# PŘÍLOHA 8: Docker Compose backend

```
version: "3.9"
name: pallidium
networks:
  network:
    name: pallidium_network
    external: true
services:
  pallidium-api-service:
    container_name: pallidium-api-container
    image: ppetrbednar/pallidium:api.latest
    pull_policy: always
    restart: always
    ports:
      - '8000:8000'
    networks:
      network:
        ipv4_address: 172.10.0.3
```

# PŘÍLOHA 9: Docker Compose frontend

```
version: "3.9"
name: pallidium
networks:
  network:
    name: pallidium_network
    external: true
services:
  pallidium-web-service:
    container_name: pallidium-web-container
    image: ppetrbednar/pallidium:web.latest
    pull_policy: always
    restart: always
    ports:
      - '80:80'
      - '443:443'
    networks:
      network:
        ipv4_address: 172.10.0.4
```

# PŘÍLOHA 10: Workflow CD backend

```
name: CD

on:
  workflow_dispatch:

jobs:
  deploy:
    name: CD Pipeline
    runs-on: ubuntu-latest
    steps:
      - name: Setup SSH key
        env:
          PRIVATE_KEY: ${{ secrets.SSH_PALLIDIUM_KEY }}
        run: |
          mkdir -p ~/.ssh
          echo "$PRIVATE_KEY" > ~/.ssh/id_rsa
          chmod 600 ~/.ssh/id_rsa

      - name: Connect, deploy and cleanup on Pallidium server
        run: |
          ssh -o StrictHostKeyChecking=no -i ~/.ssh/id_rsa ${{ secrets.SSH_PALLIDIUM_USERNAME
}}@${{ secrets.SSH_PALLIDIUM_HOST }} << EOF
            echo ${{ secrets.DOCKER_ACCESS_TOKEN_READONLY }} | docker login --username ${{
secrets.DOCKER_USERNAME }} --password-stdin
            docker-compose -f ./pallidium-api.yaml up -d
            echo y | docker image prune -a
          EOF

      - name: Cleanup SSH key
        run: rm -rf ~/.ssh
```

# PŘÍLOHA 11: Workflow CD frontend nové SSL

```
name: CD with request for new SSL

on:
  workflow_dispatch:

jobs:
  deploy:
    name: CD Pipeline
    runs-on: ubuntu-latest
    steps:
      - name: Setup SSH key
        env:
          PRIVATE_KEY: ${{ secrets.SSH_PALLIDIUM_KEY }}
        run: |
          mkdir -p ~/.ssh
          echo "$PRIVATE_KEY" > ~/.ssh/id_rsa
          chmod 600 ~/.ssh/id_rsa

      - name: Connect, deploy, cleanup and request new SSL certificate
        run: |
          ssh -o StrictHostKeyChecking=no -i ~/.ssh/id_rsa ${{ secrets.SSH_PALLIDIUM_USERNAME
}}@${{ secrets.SSH_PALLIDIUM_HOST }} << EOF
            echo ${{ secrets.DOCKER_ACCESS_TOKEN_READONLY }} | docker login --username ${{
secrets.DOCKER_USERNAME }} --password-stdin
            docker-compose -f ./pallidium-web.yaml up -d
            echo y | docker image prune -a
            docker exec pallidium-web-container openssl dhparam -out /etc/nginx/dhparam.pem
2048
            docker exec pallidium-web-container mkdir -p /var/www/_letsencrypt
            docker exec pallidium-web-container chown www-data /var/www/_letsencrypt
            docker exec pallidium-web-container certbot certonly --webroot -d pallidium.cz -d
www.pallidium.cz -d api.pallidium.cz --email info@pallidium.cz -w /var/www/_letsencrypt -n --
agree-tos
            rm -rf ~/pallidium-ssl
            mkdir -p ~/pallidium-ssl/archive
            docker cp pallidium-web-container:/etc/letsencrypt/archive/pallidium.cz
~/pallidium-ssl/archive
            docker cp pallidium-web-container:/etc/nginx/dhparam.pem ~/pallidium-
ssl/dhparam.pem
            docker exec pallidium-web-container sed -i -r -z 's/#?; ?#//g'
/etc/nginx/conf.d/default.conf
            docker exec pallidium-web-container nginx -t
            docker exec pallidium-web-container nginx -s reload
          EOF

      - name: Cleanup SSH key
        run: rm -rf ~/.ssh
```

# PŘÍLOHA 12: Workflow CD frontend

```
name: CD

on:
  workflow_dispatch:

jobs:
  deploy:
    name: CD Pipeline
    runs-on: ubuntu-latest
    steps:
      - name: Setup SSH key
        env:
          PRIVATE_KEY: ${{ secrets.SSH_PALLIDIUM_KEY }}
        run: |
          mkdir -p ~/.ssh
          echo "$PRIVATE_KEY" > ~/.ssh/id_rsa
          chmod 600 ~/.ssh/id_rsa

      - name: Connect, deploy, cleanup and reuse current SSL certificate
        run: |
          ssh -o StrictHostKeyChecking=no -i ~/.ssh/id_rsa ${{ secrets.SSH_PALLIDIUM_USERNAME
}}@${{ secrets.SSH_PALLIDIUM_HOST }} << EOF
            echo ${{ secrets.DOCKER_ACCESS_TOKEN_READONLY }} | docker login --username ${{
secrets.DOCKER_USERNAME }} --password-stdin
            docker-compose -f ./pallidium-web.yaml up -d
            echo y | docker image prune -a
            docker exec pallidium-web-container mkdir -p /etc/letsencrypt/archive
            docker cp ~/pallidium-ssl/archive/pallidium.cz pallidium-web-
container:/etc/letsencrypt/archive/pallidium.cz
            docker exec pallidium-web-container mkdir -p /etc/letsencrypt/live/pallidium.cz
            docker exec pallidium-web-container ln -s
/etc/letsencrypt/archive/pallidium.cz/cert1.pem /etc/letsencrypt/live/pallidium.cz/cert.pem
            docker exec pallidium-web-container ln -s
/etc/letsencrypt/archive/pallidium.cz/chain1.pem /etc/letsencrypt/live/pallidium.cz/chain.pem
            docker exec pallidium-web-container ln -s
/etc/letsencrypt/archive/pallidium.cz/fullchain1.pem
/etc/letsencrypt/live/pallidium.cz/fullchain.pem
            docker exec pallidium-web-container ln -s
/etc/letsencrypt/archive/pallidium.cz/privkey1.pem
/etc/letsencrypt/live/pallidium.cz/privkey.pem
            docker cp ~/pallidium-ssl/dhparam.pem pallidium-web-
container:/etc/nginx/dhparam.pem
            docker exec pallidium-web-container sed -i -r -z 's/#?; ?#//g'
/etc/nginx/conf.d/default.conf
            docker exec pallidium-web-container nginx -t
            docker exec pallidium-web-container nginx -s reload
          EOF

      - name: Cleanup SSH key
        run: rm -rf ~/.ssh
```