Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

Pokročilé softwarové inženýrství

Webová aplikace pro zpracování geografických údajů v novoasyrských textech – dokumentace backend

Jakub Šmíd
Václav Honzík
Michal Schwob
Viktorie Pavlíčková

23. května 2022

Obsah

1 Úvod

Tento dokument představuje dokumentaci k backendu (serveru) aplikace vytvořené v rámci předmětu Pokročilé softwarové inženýrství na Katedře informatiky a výpočetní techniky Fakulty aplikovaných věd Západočeské univerzity v Plzni. Cílem bylo vytvořit webovou stránku užitečnou pro výzkumníky klínopisce v dané oblasti. Poskytne informace a nástroj, který výzkumníkům a dalším zájemcům pomůže lépe pochopit historické události Novoasyrské říše.

2 Technologie a architektura

Tato sekce se zabývá technologiemi a architekturou backendu.

2.1 Technologie

Pro backend byl zvolen framework programovacího jazyka Java Spring Boot. Jako nástroj pro build je používán Apache Maven. Jakožto databázový systém byl zvolen PostgreSQL. Pro propojení frontendu a backendu je využíváno specifikace OpenAPI, která popisuje rozhraní REST API (zobrazuje např. dostupné endpointy a operace k daným endpointům). Verze použitých technologií a závislostí viz tabulka 1.

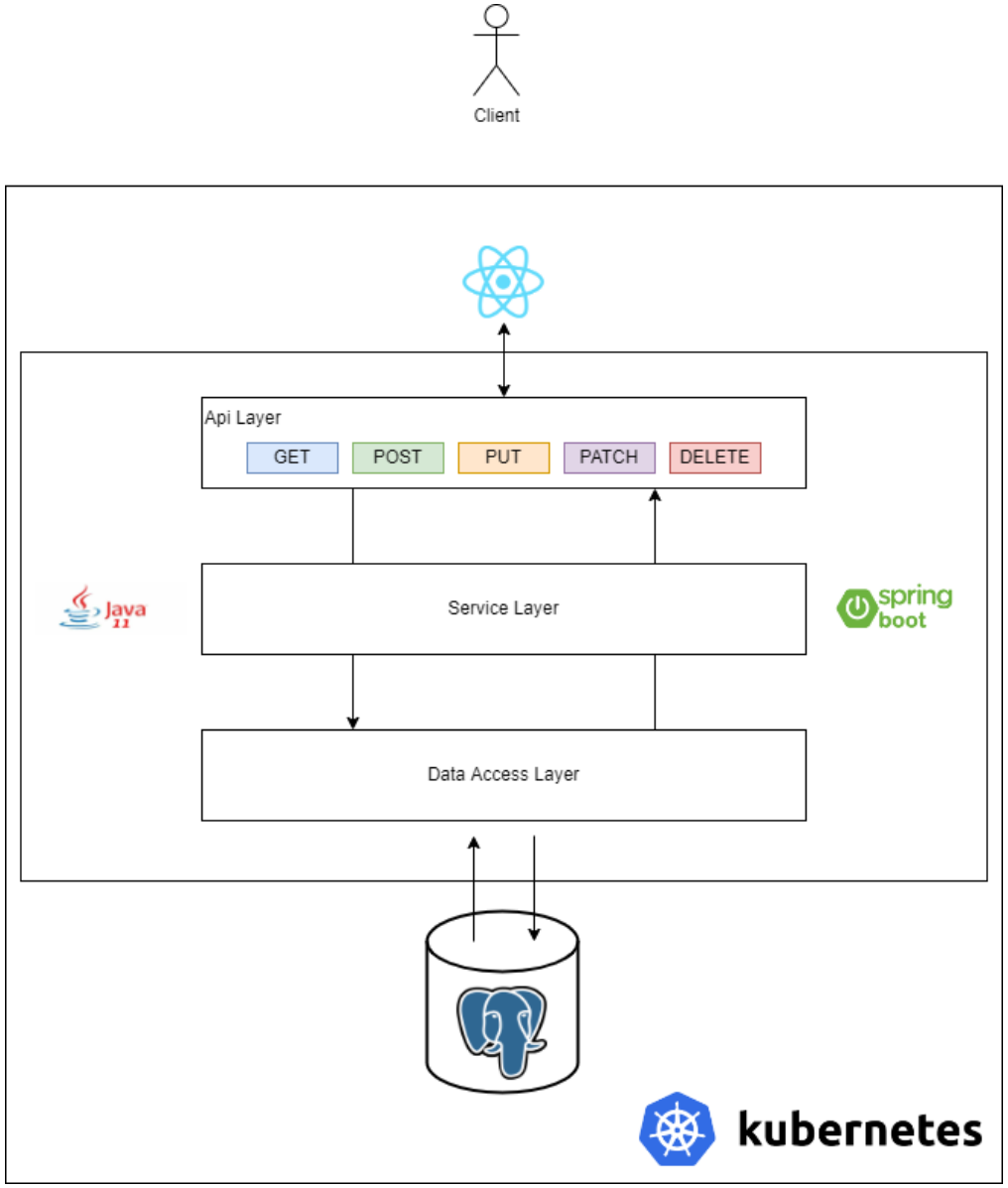
Název	**Verze**
Java	11
Spring Boot Starter Web	2.6.4
Spring Boot Configuration Processor	2.6.4
Spring Boot Starter Test	2.6.4
Spring Boot Starter Data JPA	2.6.4
Spring Boot Starter Validation	2.6.4
Spring Boot Starter Security	2.6.4
Spring Security Test	5.6.2
Lombok	1.18.22
H2 Database Engine	2.1.210
PostgreSQL JDBC Driver	42.3.3
Apache Commons CSV	1.9.0
Java JWT	3.19.1
Guava: Google Core Libraries For Java	31.1-jre
Springdoc OpenAPI UI	1.6.7
PostgreSQL	14.2

Tabulka 1: Verze technologií a závislostí.

2.2 Architektura

Backend aplikace se skládá ze tří hlavních vrstev – datové, servisní a prezentační vrstvy. Každá vrstva má danou vlastní funkcionalitu a může volat nebo být volána jinými vrstvami. Architekturu lze vidět na obrázku 1.

Datová vrstva (`Data Access Layer`) slouží jako spojení k databázi – ukládá, mění, maže a získává data z databáze. Obsahuje tedy hlavně SQL příkazy pro práci s databází.



Obrázek 1: Architektura backendu.

K datové vrstvě má přístup servisní vrstva (`Service Layer`). Tato vrstva obsahuje hlavní logiku aplikace, např. mění stav entit, volá datovou vrstvu pro získání entit z databáze, převádí entity na DTO (data transfer object – objekty, se kterými pracuje prezentační vrstva) a naopak, vrací výsledek prezentační vrstvě.

Prezentační vrstva (`Api Layer`) zpracovává požadavky od frontendu (konkrétně GET, POST, PUT, PATCH a DELETE), volá servisní vrstvu (a tedy

hlavní funkce aplikace), výsledek vrací zpět frontendu. V případě chyby při špatném požadavku vrací chybovou hlášku a příslušný HTTP stavový kód.

Kód je v rámci aplikace rozdělen do balíčků podle funkčnosti. To znamená, že např. vše týkající se uživatele (entita uživatel, DTO pro uživatele, servisní a datová vrstva atd.) je v jednom balíčku `user`.

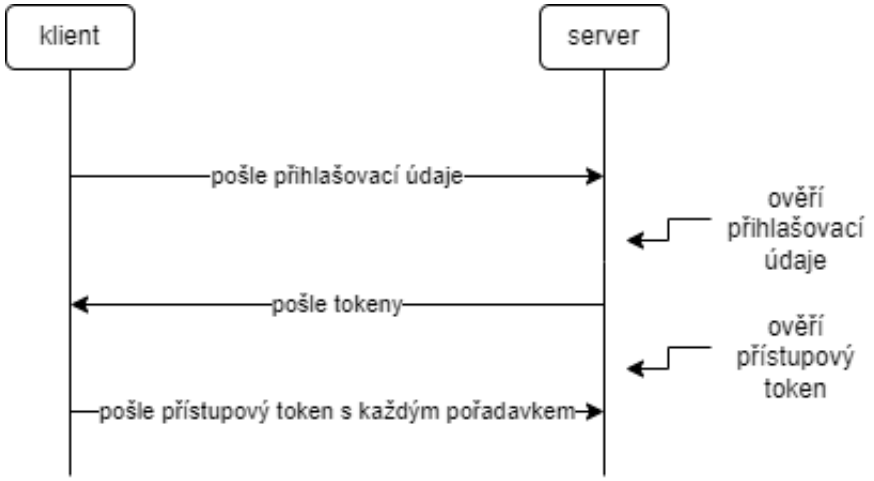
3 Zabezpečení

Nastavení zabezpečení aplikace se nachází v balíčku `security` ve třídě `SecurityConfig.java`. V této třídě se nacházejí nastavení pro endpointy – které jsou volně přístupné, pro které je nutné být přihlášen a pro které je nutné mít určitou roli nebo oprávnění.

3.1 Autentizace a autorizace

Autentizace a autorizace je zajištěna pomocí JSON Web tokenu (JWT), který má v sobě uložené potřebné informace. Těmi jsou uživatelské jméno, práva uživatele a doba platnosti tokenu. Uživatel přístupový token získá při úspěšném přihlášení. Doba platnosti tohoto tokenu je z bezpečnostních důvodů 15 minut. Tento token pak uživatel posílá v hlavičce s každým požadavkem na server (zajištěno frontendem). Aby se uživatel nemusel každých 15 minut znovu přihlašovat, získá při přihlášení ještě jeden token (obnovovací token), který obsahuje pouze uživatelské jméno a který má dobu platnosti 6 měsíců. Pomocí tohoto tokenu může pořádat server o přístupový token obsahující i oprávnění (zajištěno frontendem).

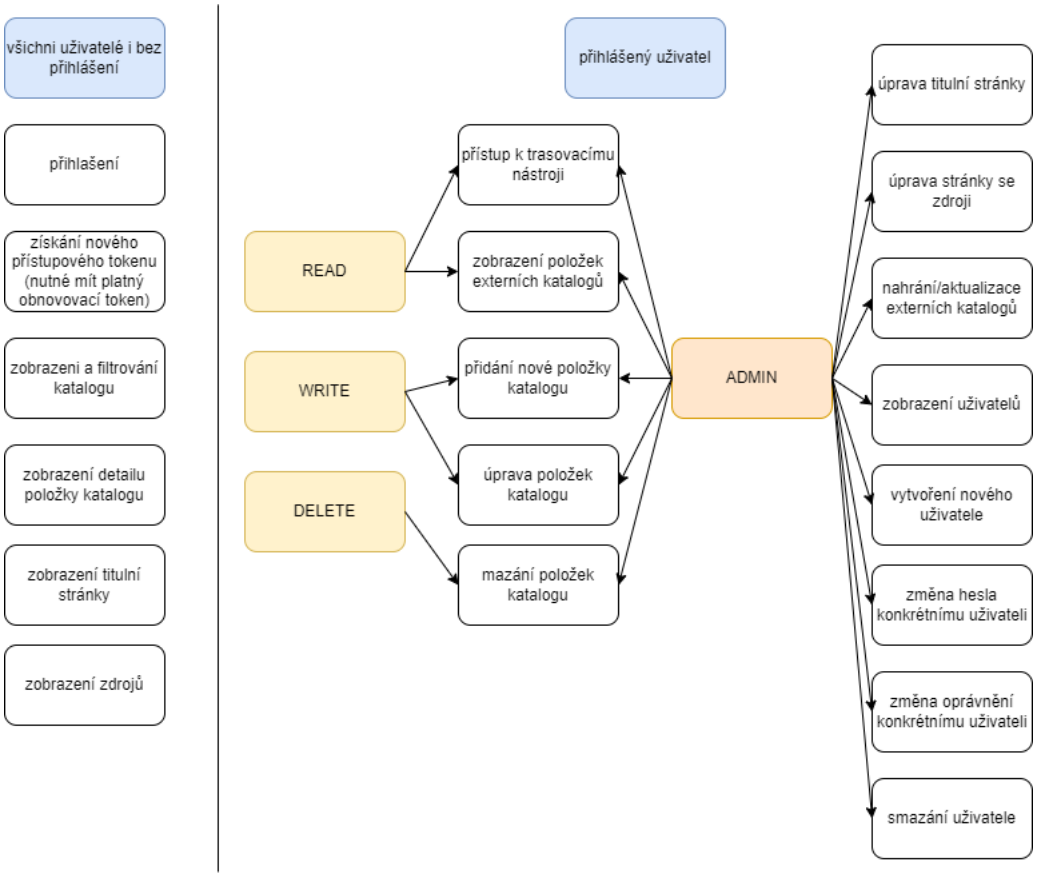
Správu tokenů zajišťují dva filtry v balíčku `security/jwt`, které jsou v nastavení zabezpečení zaregistrovány. Posílání tokenů při přihlášení zajišťuje třída `JwtUsernameAndPasswordAuthenticationFilter.java`. Třída `JwtTokenVerifier.java` poté u každého požadavku ověřuje správnost tokenu, tzn. např. zda nevypršela doba platnosti, a získá z tokenu práva. Pro volně přístupné endpointy (pro nepřihlášené uživatele) se tato kontrola neprovádí. Posílání tokenů mezi klientem a serverem viz obrázek 2.



Obrázek 2: Posílání tokenů mezi klientem a serverem.

3.2 Práva

V aplikaci existují dvě hlavní role – ADMIN a normální uživatel – a tři oprávnění – READ, WRITE a DELETE. Každý uživatel s rolí ADMIN má automaticky také všechny tři oprávnění. Normální uživatel bez jakýchkoli oprávnění má přístup pouze na endpointy jako kterýkoli nepřihlášený uživatel. Přístup do aplikace podle práv ukazuje obrázek 3.



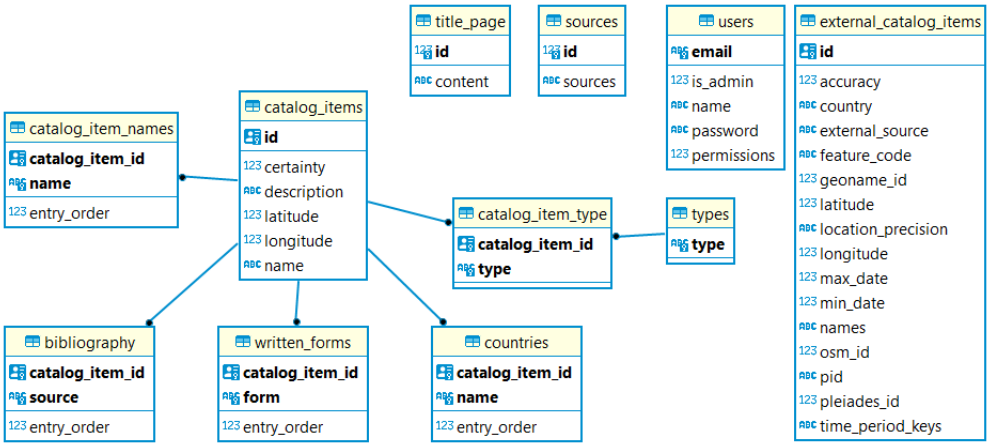
Obrázek 3: Přístup do aplikace podle práv.

Každý přihlášený uživatel si navíc může změnit sám sobě heslo. Podmínka pro heslo je alespoň 8 znaků. Jako uživatelské jméno slouží e-mail (při registraci se kontroluje validita e-mailové adresy regulárním výrazem).

V aplikaci jsou dva uživatelé s rolí ADMIN – normální a SUPER ADMIN. Další ADMIN uživatele nelze přidávat. ADMIN uživatele také nelze mazat a není možné jim upravovat oprávnění. SUPER ADMIN se od normálního ADMINA liší tím, že se nezobrazuje na stránce všech uživatelů, on sám si nemůže měnit heslo a ani jiný ADMIN mu nemůže měnit heslo.

4 Entity

Tato sekce se zabývá entitami. ER model je vidět na obrázku 4.



Obrázek 4: ER model.

4.1 Uživatelé

Tabulka `users` obsahuje vše potřebné pro uložení uživatele – e-mail, jestli je ADMIN, heslo (zahashované), jméno a oprávnění. Oprávnění jsou uložena jako číslo zabírající jeden byte a to tak, že jednotlivé oprávnění reprezentují jeden bit v tomto čísle (READ je binárně 1, WRITE 10 a DELETE 100), viz tabulka 2.

Oprávnění	Číslo	Číslo (binárně)
READ	1	1
WRITE	2	10
READ + WRITE	3	11
DELETE	4	100
READ + DELETE	5	101
WRITE + DELETE	6	110
READ + WRITE + DELETE	7	111

Tabulka 2: Uložení oprávnění do entity reprezentující uživatele.

4.2 Katalog

Katalog je reprezentován tabulkou `catalog_items`. Ta je ve vztahu 1:N s dalšími tabulkami reprezentujícími jména (`catalog_items_names`), bibliografii (`bibliography`), státy (`countries`) a psané formy (`written_forms`). Tyto tabulky mají vždy složený primární klíč, kde část je ID položky katalogu, ke které záznam patří. Navíc mají pořadí pro danou položku katalogu, podle které jsou pak zpětně řazeny pro zobrazení (bylo požadováno jména zobrazovat ve stejném pořadí, v jakém byla ukládána). Katalog je navíc ve vztahu M:N s tabulkou obsahující typy záznamu (`types`).

4.3 Externí katalog

Externí katalog reprezentuje tabulka `external_catalog_items`. Tato tabulka má mnoho položek, které vznikly sloučením všech potřebných položek ze 4 externích zdrojů. Tyto zdroje obsahují vždy alespoň jedno jméno a souřadnice, navíc ale můžou obsahovat i další údaje. Jména v této tabulce jsou uložena na rozdíl od normálního katalogu jako jeden dlouhý řetězec, kde jsou jména oddělena čárkou.

4.4 Další tabulky

Tabulky reprezentující titulní stranu (`title_page`) a zdroje (`sources`) obsahují jen svůj obsah. Pro každou z těchto tabulek existuje vždy jen jeden záznam s ID 1.

5 Další implementační detaily

Tato sekce se zabývá některými dalšími implementačními detaily.

5.1 Prezentační vrstva

Třídy prezentační vrstvy, označené anotací `@Controller`, definují dostupné endpointy. Poté jen volají služby servisní vrstvy, která provede požadovanou akci a případně vrátí výsledek, které pak prezentační vrstva vrátí jako odpověď frontendu. Případné chyby (např. neplatný vstup uživatele) řeší třída `ApiExceptionHandler.java` v balíku `exception`, která vrací frontendu text výjimky a chybový kód.

5.2 Datová vrstva

K přístupu do databáze bylo zvoleno rozhraní `JpaRepository`, které již dodává některé metody, jako např. uložení entity, získání všech entit nebo získání entity podle jména. Z tohoto důvodu nebylo nutné psát mnoho SQL dotazů ručně.

Výjimku tvoří ukládání externího katalogu. Položek externího katalogu je přes 800 000 a jejich ukládání pomocí `JpaRepository` trvalo přes 10 minut. Proto je ukládání externího katalogu řešeno pomocí `PreparedStatement` a `HikariDataSource`. Tato kombinace umožňuje ukládat ve více vláknech po dávkách. Tímto řešením se čas potřebný k uložení záznamů zkrátil pod 1 minutu.

5.3 Filtrování katalogu

Při filtrování katalogu je nejprve nutné převést speciální uživatelské vstupy na odpovídající vstup pro regulární výraz v Javě („*“ pro libovolný počet (včetně 0) jakýchkoli znaků na „.*“ a „?“ jakožto jeden libovolný znak na „.“). Dále se načtou všechny záznamy v katalogu a poté se provede filtrování regulárními výrazy. Filtrovat lze přes jméno, stát, typ a psanou formu, kdy všechny tyto položky nejsou povinné (lze tedy filtrovat např. jen přes jméno nebo zobrazit všechny položky tak, že není použit filtr žádný).

Původně byla filtrace prováděna SQL dotazem, to ovšem vedlo k několika problémům. Za prvé v SQL dotaze pro `JpaRepository` nelze využít závorky nebo znak „|“ pro regulární výraz. To by se dalo odstranit nativním dotazem, kde by ale nešlo načíst rovnou i kolekce, což souvisí s dalším bodem. Dalším problémem je nutnost načítání kolekcí (tabulek ve vazbě s katalogem) rovnou i s položkou katalogu, což při filtrování SQL dotazem vedlo ke zpomalení

dotazu zapříčiněnému kartézským součinem. Načítání kolekcí rovnou tak, aby nedošlo k $N+1$ problému, je řešeno anotací `@Fetch(FetchMode.SUBSELECT)` nad danou kolekcí.

5.4 Externí katalog

Při aktualizaci externího katalogu je nejprve odstraněna celá složka `sources` (pokud existuje, pokud neexistuje tak je vytvořena) a poté odstraněny všechny záznamy tabulky pro externí katalog. Poté jsou stahovány zdroje. Soubory ze zdrojů *Pleiades* a *GeoNames* mají vždy stejnou URL cestu, proto je stáhne aplikace sama. V případě souboru zdroje *Pleiades* se stahuje jeden archiv, který je nutné ještě rozbalit. Souborů ze zdroje *GeoNames* je staženo více – podle států, které byly dány v zadání. Soubor ze zdroje *ANE* je neměnný, proto je uložen trvale ve složce `resources`. Nakonec soubor ze zdroje *CIGS* je nutné dodat současně s požadavkem na aktualizaci externího katalogu.

Uloženy do databáze jsou jen soubory, které se podaří stáhnou a načíst. Pokud se tedy někdy změní formáty stahovaných souborů nebo URL k souborům zdrojů *Pleiades* a *GeoNames*, již je nebude možné ukládat bez změny kódu.

Filtrovat externí katalog lze podle názvu a zdroje (obě povinné položky). V názvu lze použít speciální zástupné znaky podobně jako u filtrování lokálního katalogu. V tomto případě probíhá filtrování přes nativní SQL dotaz. Opět je nutné ve filtru jména nahradit znak „*“ za „.*“ a znak „?“ znakem „.“. Při vstupu do SQL dotazu je jméno doplněno z obou stran částí regulárního výrazu tak, aby celý regulární výraz bral v potaz, že jednotlivé názvy jsou oddělené čárkami.

5.5 Zpracování cesty

Při zpracování textu, ze kterého se chce zobrazit mapa cesty, se nejprve načtou všechny typy a záznamy katalogu z databáze. Typy jsou uloženy do datové struktury `HashSet` pro rychlé ověření, zda obsahuje hledaný řetězec. Z načtených záznamů katalogu je vytvořena hašovací tabulka, kde klíčem je jméno a hodnotou je seznam záznamů katalogu, které mají dané jméno v seznamu všech svých jmen. Tento způsob opět umožňuje rychlé ověření, zda hledaný řetězec odpovídá některému ze jmen záznamů katalogu.

Dále je text rozdělen na tokeny včetně bílých znaků, aby bylo zachováno odřádkování textu. Následně se prochází token po tokenu. Pokud je token bílý znak nebo se skládá pouze z interpunkčních znaků, pouze je přidán do výsledného textu. Jinak je zpracováván dál. Nejprve je z tokenu extrahována část textu, která nezačíná a nekončí interpunkčními znaky (např. „town“

je celý token, ale „(town)“ je jen „town“). U této části se nejprve zkontroluje, zda se shoduje s některým z typů. Pokud ano, text je doplněn tak, aby se v HTML zvýraznil tučně. Pokud ne, zkontroluje se, zda část neodpovídá některému názvu ze záznamu z katalogů. Pokud ano a část není totožná s nalezeným názvem (stejný název se může v textu objevovat vícekrát za sebou, ale stačí zobrazit pouze jednou), pak se projde seznam získaných záznamů z katalogu (tento seznam je také přidán k výsledku). Pokud alespoň jeden záznam nemá chybějící souřadnice, je text doplněn tak, aby měl v HTML zelenou barvu. Pokud mají všechny záznamy chybějící souřadnice, pak je text obarven barvou červenou. Nakonec je k textu vrácena interpunkce, pokud ji původní token obsahoval.

Výsledek je tedy text, který bude v HTML různě zvýrazněn (tučně pro nalezený typ, zeleně pro nalezené položky katalogu se souřadnicemi a červeně pro nalezené položky katalogu bez souřadnic) a pole polí nalezených položek katalogu. Toho pole je seřazené podle toho, jak se položky nacházely v textu. Pole polí proto, že pro jedno jméno může být nalezeno více různých položek katalogu.

5.6 Testování

Pro testování jsou použity Unit testy. Všechny ručně psané SQL dotazy a všechny servisy jsou otestovány. Výjimkou je nahrávání externího katalogu. Pro tento případ by bylo napsání testů složité, takže testování proběhlo pouze ručně.

5.7 Komentáře

Všechny metody, třídy a proměnné tříd jsou komentovány s jednou výjimkou. V případě třídy, která implementuje rozhraní, již nejsou implementované metody komentovány, komentáře jsou pouze v příslušném rozhraní.

6 Závěr

Tato dokumentace vysvětluje architekturu a některé implementační detaily backendu aplikace. Datová a servisní vrstva aplikace je otestována Unit testy. Kód je komentovaný. Aplikace splňuje zadání.