FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Implementace modulu pro import údajů RÚIAN

Martin Schön

FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Implementace modulu pro import údajů RÚIAN

Martin Schön

Citace v seznamu literatury:
SCHÖN, Martin. *Implementace modulu pro import údajů RÚIAN*. Plzeň, 2024. Bakalářská práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky. Vedoucí práce Martin Bíkl, Ing. Petr Přibyl, Ing. Martin Zíma Ph.D.

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2024/2025

Studijní program: Informatika a výpočetní technika
Forma studia: Prezenční
Specializace/kombinace: Informatika (INF18bp)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

Jméno a příjmení: **Martin SCHÖN**
Osobní číslo: **A22B0144P**
Adresa: Rpety 42, Rpety, 26801 Hořovice, Česká republika

Téma práce: Implementace modulu pro import údajů RÚIAN
Téma práce anglicky:
Jazyk práce: Čeština

Související osoby: Ing. Martin Zíma, Ph.D. (Konzultant z univerzity)
Katedra informatiky a výpočetní techniky
Ing. Martin Bíkl (Konzultant mimo univerzitu)
Katedra informatiky a výpočetní techniky
Ing. Petr Přibyl (Vedoucí)
Katedra informatiky a výpočetní techniky

Zásady pro vypracování:

1. Prostudujte datové schéma registru RÚIAN a možnosti získávání dat prostřednictvím datových služeb.
2. Prozkoumejte možnosti konfigurace řešení s přihlédnutím na mapování datových struktur.
3. Navrhněte konfigurační soubor, který bude umožňovat nastavení úrovně přenášených územních objektů a nastavení cílové databáze a cílových struktur.
4. Vytvořte aplikaci, která bude pravidelně synchronizovat veřejnou databázi RÚIAN do databázových struktur podle konfiguračního souboru. Synchronizace bude probíhat buď jako kompletní sada dat nebo přírůstkově. Jako úložiště využijte databázi Oracle, Microsoft SQL Server a PostgreSQL v posledních verzích.
5. Vytvořenou aplikaci ověřte na 3 konfiguračních souborech, zhodnoťte využitelnost daného řešení pro další databázové enginy a otestujte rychlost daného řešení pro kompletní i přírůstkovou sadu dat.

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce.

Podpis studenta: Datum:

Podpis vedoucího práce: Datum:

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného akademického titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Západočeská univerzita v Plzni má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Plzni dne 31. prosince 2024

. . . . . .

Martin Schön

V textu jsou použity názvy produktů, technologií, služeb, aplikací, společností apod., které mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

Abstrakt

Text abstraktu v jazyce práce, tj. zde česky.

Abstract

The abstract text in a secondary language, here in English.

Klíčová slova

Poděkování

Text poděkování.

Obsah

Úvod

Problematika správy uzemní identifikace a prostorových dat a jejich synchronizace mezi různými systémy nabývá na významu s rostoucí digitalizací státní správy a soukromých sektorů. Jedním z klíčových zdrojů těchto dat v České republice je Registr územní identifikace, adres a nemovitostí (RÚIAN), který poskytuje rozsáhlé a aktuální informace o územních objektech, adresách a dalších klíčových entitách. Efektivní využití dat z RÚIAN vyžaduje nejen jejich přístup prostřednictvím datových služeb, ale také robustní řešení pro mapování, konfiguraci a synchronizaci datových struktur.

Cílem této bakalářské práce je analyzovat datové schéma registru RÚIAN a možnosti získávání dat prostřednictvím nabízených datových služeb. Dále bude provedena analýza a návrh konfiguračního řešení, které umožní nastavit úroveň přenášených územních objektů a cílové databázové struktury. V rámci práce bude navržena a implementována aplikace, která umožní pravidelnou synchronizaci dat z veřejné databáze RÚIAN do cílových databázových struktur s podporou databází Oracle, Microsoft SQL Server a PostgreSQL. Aplikace bude schopna provádět synchronizaci kompletních datových sad i přírůstkových změn podle zadané konfigurace.

RÚIAN

RÚIAN je zkratkou pro Registr územní identifikace, adres a nemovitostí. Jedná se o státní informační systém v České republice, který obsahuje informace o adresách, budovách, parcelách a dalších objektech. Systém je spravován Českým úřadem zeměměřickým a katastrálním (ČÚZK). Data jsou využívána v mnoha oblastech, například v urbanistickém plánování, geodézii nebo při správě nemovitostí. Jednotlivé prvky jsou zobrazovány na mapách státního mapového díla a digitální mapě veřejné správy.

Data z RÚIAN jsou veřejně dostupná a lze je získat z webové služby na adrese `https://vdp.cuzk.gov.cz/vdp/ruian`. Lze stahovat data ve formátu XML, která obsahují základní nebo úplné informace o územních prvcích. Mezi tyto prvky patří Stát, VÚSC (Vyšší územní samosprávný celek), ORP (Obec s rozšířenou působností), Obec, Část obce, Ulice, Adresa atd. Data lze vyhledávat, ověřovat a stahovat dle jednotlivých územních prvků, které jsou uloženy v databázi RÚIAN.

2.1 Výměnný formát RÚIAN

Výměnný formát RÚIAN (VFR) je jednou ze služeb, které poskytuje ČÚZK. Tento formát slouží k přenosu dat mezi různými informačními systémy.

Je možné stahovat data podle zadaných formátů: **Standardní**, **Historický** a **Speciální**. Dále je možné si vybrat mezi přírůstkovými daty a úplnou kopií. Přírůstky je možné vyhledávat podle data – od zvoleného dne až do současnosti. Úplná kopie obsahuje všechna data a je možné ji také časově vymezit. Tato data se aktualizují jednou měsíčně.

Každý formát navíc nabízí další parametry, které lze nastavit. Data z VFR jsou ve formátu XML. Každý XML element obsahuje atributy, které nesou informace o dané entitě (tabulce).

- **Standardní** – obsahuje úplná nebo přírůstková data.
  - Časový rozsah: přírůstky od data / úplná kopie
  - Územní prvky: Stát až ZSJ / Obec a podřazené
  - Datová sada: základní / kompletní
  - Výběr údajů: základní údaje / generované hranice, originální hranice, vlajky a znaky
  - Územní omezení: ČR / kraj (VÚSC) / ORP / obec
- **Historický** – obsahuje historická data.
  - Časový rozsah: přírůstky od data / úplná kopie
  - Územní prvky: Stát až ZJS / Obec a podřazené
  - Územní omezení: ČR / kraj (VÚSC) / ORP / obec
- **Speciální** – obsahuje speciální datové sady.
  - Časový rozsah: přírůstky od data / úplná kopie
  - Výběr údajů: číselníky / vazby / vazby a číselníky
  - Kategorie: všechny / geodetické body / nerostné bohatství

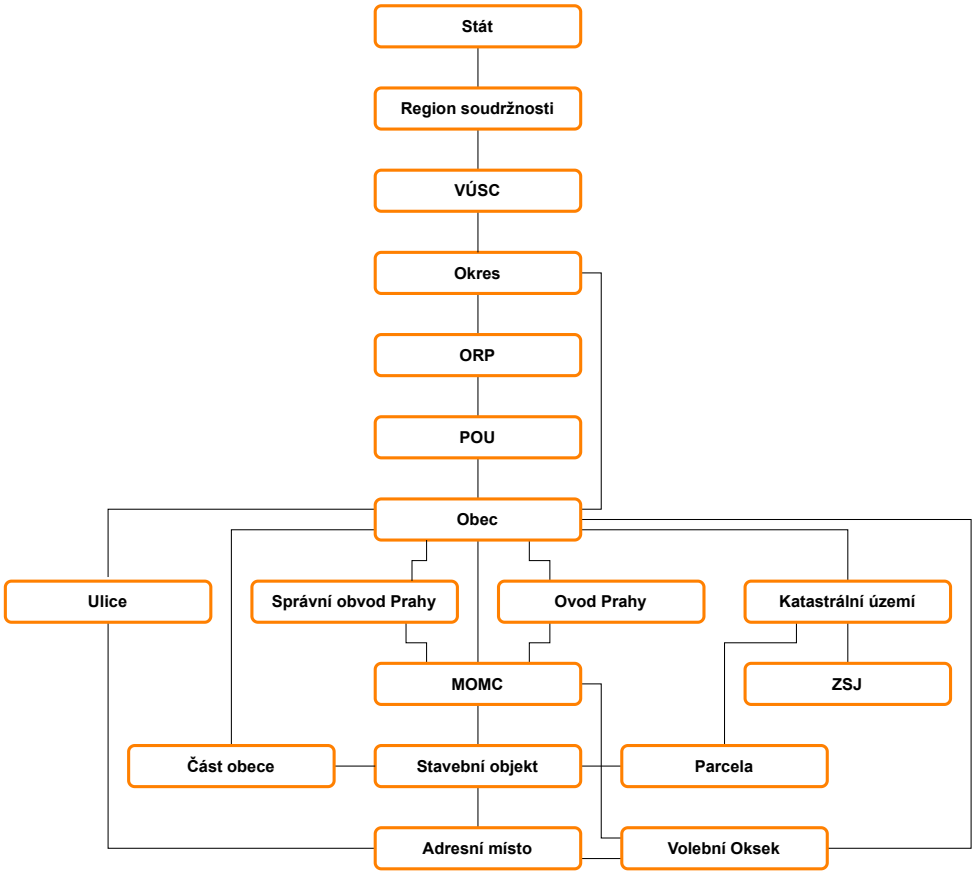
2.2 Datové struktury

Data z RÚIAN jsou rozdělena do několika datových struktur neboli entit. Jak je vidět na obrázku 2.1 [1], každá entita obsahuje specifické informace. Mezi tyto informace patří například název státu, kód státu, geografické souřadnice, datum vzniku a další.

Stát představuje nejvyšší úroveň hierarchie, pod kterou spadají další entity závislé na ní. Příkladem je entita **VÚSC**, která obsahuje informace o vyšších územních samosprávných celcích. Jednotlivé entity na sebe navzájem odkazují pomocí cizích klíčů.

2.3 Formát dat

Data z RÚIAN jsou ve formátu XML. Tento formát je strukturovaný a umožňuje přenášet data mezi různými systémy. Bude proto třeba implementovat parser, který převede data z XML do dále zpracovávaného formátu pro databáze.



Obrázek 2.1: Tabulky RÚIAN

Databázový systémy

Databáze je softwarový nástroj, který slouží k efektivnímu ukládání, organizaci a vyhledávání dat. Díky své strukturované povaze umožňuje správu velkých objemů dat a poskytuje funkcionality pro zajištění konzistence, bezpečnosti a rychlého přístupu k uloženým informacím.

Databáze lze obecně rozdělit do dvou hlavních kategorií: **relační databáze** a **objektové databáze**. Každý z těchto přístupů má své specifické vlastnosti a je vhodný pro odlišné typy aplikací.

Relační databáze

- Data jsou uložena v tabulkách
- Každý řádek tabulky obsahuje jeden záznam
- Každý sloupec tabulky obsahuje jeden atribut
- Vztahy mezi tabulkami jsou definovány klíči
- Využití jazyka SQL

Relační databáze jsou vhodné pro strukturovaná data, která mají pevnou strukturu. Jedná se o nejčastěji používaný typ databáze.

Objektové databáze

- Data jsou uložena jako objekty
- Každý objekt obsahuje atributy a metody
- Vztahy mezi objekty jsou definovány referencemi
- Využití objektově orientovaného jazyka

Objektové databáze jsou vhodné pro nestrukturovaná data, která mají složitou strukturu. Jedná se o novější typ databáze, který je vhodný pro moderní aplikace.

Vzhledem k zadání, kde je přímo specifikováno, které databáze budou použity, není třeba vybírat mezi těmito dvěma typy databází. Všechny tři databáze, které budou popsány, jsou relační databáze. Tyto databáze však mají mezi sebou rozdíly v použití, funkcích a možnostech.

Databáze bude potřebovat některé dodatečné funkce, které jsou nezbytné pro práci s daty. Mezi tyto funkce patří:

- Zpracování geometrických dat
- Podpora JSON
- Podpora vhodných datových typů (čas a datum, čísla, text)

3.1 Microsoft SQL Server

Microsoft SQL Server je relační databázový systém, který vyvinula společnost Microsoft a který se stal jedním z předních nástrojů pro ukládání, správu a analýzu dat. SQL Server je robustní a výkonný systém, který nabízí širokou škálu funkcí a možností přizpůsobení pro různé typy aplikací. Díky své dlouhodobé podpoře a integraci s dalšími produkty společnosti Microsoft, jako je Azure nebo Power BI, je SQL Server oblíbenou volbou pro velké a střední podniky.

Edice SQL Serveru zahrnují Standard, Enterprise a Express, které se liší funkcemi a cenou. Standard Edition je nejčastěji používanou edicí, která obsahuje všechny základní funkce a je vhodná pro většinu aplikací.

SQL Server byl původně navržen výhradně pro Windows, ale od verze SQL Server 2017 je dostupný také pro operační systém Linux. Tato multiplatformní podpora zvyšuje jeho použitelnost v různých IT prostředích. [2]

Pro tuto práci bude využita edice SQL Server 2017 Standard, která je dostupná pro Windows a Linux. Tato edice podporuje vhodné datové typy pro zpracování geometrických dat. Ovšem nepodporuje JSON, který je potřeba pro zpracování dat z RÚIAN. To však není problém, protože JSON je možné uložit jako řetězec.

3.2 PostgreSQL

PostgreSQL je open-source relační databázový systém, který je známý svou spolehlivostí, výkonem a rozšiřitelností. Původně byl vyvinut jako alternativa k proprietárním řešením, jako je SQL Server, a dnes se řadí mezi nejpokročilejší relační databázové systémy na trhu. Díky své otevřené povaze a aktivní komunitě uživatelů a vývojářů se stal oblíbenou volbou nejen mezi malými firmami, ale také ve středních a velkých organizacích.

PostgreSQL je dostupný zdarma a podporuje všechny hlavní operační systémy, včetně Windows, Linux a macOS. Tato multiplatformní dostupnost umožňuje snadnou integraci PostgreSQL do různých vývojových prostředí. [3]

PostgreSQL také podporuje jak formát JSON, tak všechny potřebné datové typy kromě geometrických dat. Pro ukládání geometrických dat je třeba použít *PostGIS*,

což je rozšíření pro PostgreSQL, které přidává podporu pro prostorové a geometrické datové typy.

3.3 Oracle

Oracle Database, vyvinutá společností Oracle Corporation, patří mezi přední relační databázové systémy na světě. Tento systém je známý svou robustností, vysokým výkonem a schopností zvládat kritické podnikové aplikace a rozsáhlé datové sady. Oracle Database je navržena tak, aby poskytovala spolehlivé a efektivní řešení pro ukládání, správu a analýzu dat ve velkých i středních organizacích.

Edice Oracle Database zahrnují Standard Edition, Enterprise Edition a Express Edition, které se liší funkcemi a cenou. Enterprise Edition je nejkomplexnější edicí, která obsahuje všechny pokročilé funkce a je vhodná pro velké podniky s náročnými požadavky.

Oracle Database je kompatibilní s většinou hlavních operačních systémů, včetně Windows, Linux a Unix. Díky tomu může být nasazena v různorodých IT prostředích podle požadavků organizace. [4]

Pro tuto práci bude využita edice Oracle Express Edition, která je dostupná zdarma a je určena pro vývoj a testování. Tato edice podporuje všechny potřebné datové typy a také JSON. Ovšem není zde možnost uložení geometrických dat. Ta jsou obsažena v `SDO_GEOMETRY`, což je rozšíření pro Oracle Database.

3.4 Komunikace s databází

Všechny tři databázové systémy podporují komunikaci pomocí jazyka SQL (Structured Query Language). SQL je standardizovaný jazyk pro práci s relačními databázemi, který umožňuje vytváření, čtení, aktualizaci a mazání dat. Pro komunikaci s databází je tedy třeba vytvořit SQL dotazy, které budou provádět operace nad daty. O tuto komunikaci se stará aplikační vrstva, která zajišťuje připojení k databázi a následné zpracování a odeslání SQL dotazů.

3.5 SQL

SQL neboli Structured Query Language je standardizovaný jazyk pro práci s relačními databázemi. Je to jazyk, který umožňuje vytváření, čtení, aktualizaci a mazání dat v databázi. Vytvářet dotazy, pro ukládání různých dat bude velmi náročné a neefektivní. Ve výsledné aplikaci bude použito ORM (Object-Relational Mapping), které umožňuje práci s daty pomocí objektů. ORM je technika, která mapuje objekty v programovacím jazyce na tabulky v databázi a naopak. Konkrétně bude použito *Hibernate*, což je open-source ORM framework pro jazyk Java.

3.6 JDBC

Jedná se o Java API pro přístup k relačním databázím. JDBC (Java Database Connectivity) poskytuje standardní rozhraní pro připojení k databázím, Bude použito pro připojení k databázi a provádění SQL dotazů. Využívá různé ovladače pro různé databázové systémy, takže je možné připojit se k jakékoli databázi, která podporuje JDBC.

Návrh aplikace

Aplikace je prostředníkem mezi RÚIAN VFR a cílovou databází. Hlavním úkolem je stažení dat z RÚIAN VFR, jejich přečtení, zpracování a následné uložení do cílové databáze. Aplikace bude psána v jazyce Java. Bude třeba zajistit funkce pro zpracování a uložení dat. Dále bude třeba zajisti aby se aplikace věděla co bude dělat a jak se chovat. Pro toto bude třeba vytvořit konfigurační soubor, který bude obsahovat nastavení aplikace.

4.1 Stahování dat

Stahování dat bude zajištěno pomocí knihovny *Apache HttpClient*, která je součástí balíčku *Apache HttpComponents*. Tato knihovna umožňuje snadné a efektivní stahování dat z webových stránek a API. V rámci této aplikace bude použita k stahování dat z RÚIAN VFR. Data budou stažena jako ZIP soubor, který bude následně rozbalen a zpracován. Soubory budou uloženy do dočasného adresáře, který bude po zpracování smazán z důvodu úspory místa na disku.

4.2 Zpracování dat

Zpracování dat se bude skládat z několika částí: přečtení a mapování dat do objektů, které budou následně uloženy do databáze. Pro čtení dat bude třeba parser XML souborů, který přečte data a převede je do objektů. Vzhledem k velikosti dat bude třeba zajistit efektivní zpracování, aby nedocházelo k přetížení paměti a CPU. Pro čtení je možné využít knihovnu *Jackson* nebo *StAX*.

1. **Jackson** – knihovna pro zpracování JSON a XML dat. Umožňuje snadné mapování objektů na JSON a XML a naopak. Je velmi rychlá a efektivní, ale může být složitější na použití.

2. **StAX** – knihovna pro zpracování XML dat. Umožňuje čtení a zápis XML dat pomocí událostí. Je velmi rychlá a efektivní, ale může být složitější na použití.

Následně bude třeba vytvořit objekty pro mapování dat do objektů. Tyto objekty budou mít stejnou strukturu jako data z RÚIAN VFR. Pro mapování bude využita

knihovna *JPA* (Java Persistence API), která umožňuje snadné mapování objektů na databázové tabulky a naopak. Pro tuto technologii je třeba vytvořit databázové entity, které budou mít stejnou strukturu jako v databázi. Dále bude třeba vytvořit repozitáře pro práci s databází (CRUD operace). A nakonec bude třeba vytvořit služby, které budou sloužit pro práci s repozitáři a pro zpracování dat.

Před uložením do databáze bude třeba provést validaci dat, aby nedocházelo k chybám při ukládání. Je třeba zajistit, aby se zabránilo ukládání dat, která jsou neplatná nebo nekompletní. Možné chyby při validaci dat:

- Chybějící primární klíče
- Nevalidní cizí klíče

4.3 Komunikace s databází

Pro ukládání dat je třeba se nejprve připojit k databázi. Pro připojení k databázi bude použita knihovna *JDBC* (Java Database Connectivity), která umožňuje připojení k různým databázím pomocí standardního API. Pro jednotlivé databáze budou použity různé JDBC ovladače, které umožňují připojení k vybraným databázím.

Pro připojení k databázi bude třeba vytvořit konfigurační soubor, který bude obsahovat informace o připojení k databázi.

4.4 Formát konfiguračního souboru

Je třeba zajistit, aby aplikace byla schopna načíst konfigurační soubor a podle něj se správně nakonfigurovat. Je tedy nutné zvolit vhodný formát tohoto souboru. Existuje několik možností, z nichž je třeba vybrat tu nejvhodnější:

- **XML (XSD)** – Extensible Markup Language
- **JSON** – JavaScript Object Notation
- **YAML** – YAML Ain't Markup Language
- **INI** – Initialization File

4.4.1 XML

[5] XML je značkovací jazyk, na rozdíl od JSON a YAML. Jedná se o textový formát, který však není tak snadno čitelný pro člověka jako JSON nebo YAML. Podporuje víceúrovňovou strukturu a komentáře. Dále umožňuje použití schémat, což dovoluje definovat strukturu a typy dat. XML je však zbytečně složité a náročné na čtení i psaní. Navíc je čtení velmi pomalé a náročné na výkon.

4.4.2 JSON

[5] JSON je formát pro výměnu dat, který je snadno čitelný jak pro člověka, tak pro počítač. Jedná se o textový formát, který se často používá k přenášení dat mezi serverem a klientem. Podporuje víceúrovňovou strukturu, ale nepodporuje komentáře. Je však velmi rozšířený a podporovaný většinou programovacích jazyků. Čtení i zápis dat je jednoduchý a rychlý.

4.4.3 YAML

[5] YAML je formát pro serializaci dat, který je čitelností a strukturovaností podobný JSON. Oproti JSON podporuje komentáře. Pro člověka však může být YAML složitější a náročnější na psaní. Rychlost čtení je srovnatelná s JSON, ale zápis je pomalejší.

4.4.4 INI

INI je formát konfiguračních souborů, který je snadno čitelný pro člověka. Podporuje komentáře i víceúrovňovou strukturu. Je však často nejednotný a obtížně se s ním pracuje. Používá se spíše pro jednodušší konfigurační soubory a není vhodný pro složitější aplikace.

4.4.5 Volba formátu

Vzhledem k tomu, že aplikace bude obsahovat mnoho funkcionalit a bude vyžadovat nastavení řady parametrů, je třeba zvolit vhodný formát. Prozatímní verze aplikace si žádá formát, který bude čitelný, snadno rozšiřitelný a dostatečně flexibilní. Proto je nejvhodnější volbou formát **JSON**. JSON je snadno čitelný a zápis je přehledný, podporuje víceúrovňovou strukturu a je široce rozšířený. Přestože nepodporuje komentáře, lze jeho strukturu snadno pochopit a rozšířit o další parametry. Zároveň je JSON podporován většinou programovacích jazyků, což zajišťuje snadnou integraci.

4.5 Obsah konfiguračního souboru

Konfigurační soubor bude obsahovat nastavení aplikace. Jedná se o nastavení databáze, aplikace, plánovače a parametry pro stahování dat. Konkrétně se jedná o:

1. **Databáze** – Nastavení připojení k databázi:
    - Typ databáze – MSSQL, PostgreSQL, Oracle
    - Connection string – připojovací řetězec
    - Uživatelské jméno – přihlašovací jméno do databáze
    - Heslo – heslo do databáze
    - Název databáze – cílová databáze pro ukládání dat
2. **Nastavení plánovače** – Nastavení plánovače, který bude stahovat data:
    - Interval – interval stahování dat
    - Přeskočení – možnost přeskočit naplánované stahování
3. **Parametry pro stahování dat** – Parametry pro stahování dat z webové služby:
    - Seznam krajů – výčet krajů, pro které se budou data stahovat
    - Stahovat geometrii – ANO/NE (např. kvůli absenci geometrických typů v Oracle XE)
    - Velikost commitů – počet záznamů na jeden commit do databáze
    - Konkrétní tabulky, sloupce a způsob práce s vybranými daty

Toto nastavení bude uloženo v konfiguračním souboru, který bude načten při startu aplikace. Bude tedy třeba vytvořit třídu, která zajistí načtení tohoto souboru a zpracování jeho obsahu.

Použité technologie a nástroje

Některé technologie pro tuto práci již byly zmíněny (Java, JDBC, Hibernate atd.) v sekci 4.3. Pro účely této práce budou potřeba ještě další technologie, které umožní tvorbu aplikace.

5.1 REST API

REST API (Representational State Transfer Application Programming Interface) je architektonický styl pro návrh webových služeb, který umožňuje snadnou a efektivní komunikaci mezi klientem a serverem. Tento přístup se stal jedním z nejrozšířenějších způsobů integrace aplikací díky své jednoduchosti, flexibilitě a nezávislosti na platformě. REST API využívá standardní metody protokolu HTTP, jako jsou GET, POST, PUT a DELETE, k provádění různých operací s daty.

REST API poskytuje ideální prostředí pro implementaci přenosu dat díky své flexibilitě a schopnosti pracovat s různými datovými zdroji. V této práci bude kladen důraz na robustnost a spolehlivost API, což zahrnuje zpracování chybových stavů, zabezpečení komunikace (například prostřednictvím HTTPS) a optimalizaci výkonu. [6]

5.2 Spring Framework

Spring Framework je open-source framework pro vývoj aplikací v jazyce Java. V tomto frameworku bude vytvořena aplikace, která bude vše spojovat dohromady. Spring Framework obsahuje mnoho modulů, které usnadňují vývoj aplikací, jako například Spring Boot, Spring Data, Spring Security atd. Pro účely této práce bude využit modul Spring Boot, který umožňuje rychlé vytvoření aplikace s minimální konfigurací. [7]

5.3 Maven

Maven je nástroj pro správu projektů v jazyce Java, který usnadňuje správu závislostí, kompilaci a nasazení aplikací. Jedná se o jednoduchý a efektivní nástroj, který umožňuje správu projektů pomocí XML konfiguračního souboru. Konkrétně se jedná o soubor `pom.xml`, který obsahuje informace o projektu, jako jsou závislosti, pluginy a další nastavení. [8]

5.4 Plánovač

V popisu, co bude obsahovat konfigurační soubor, bylo zmíněno, že bude třeba zvolit plánovač. Plánovač je nástroj, který umožňuje spouštění úloh v pravidelných intervalech. Výhodou plánovače je, že umožňuje automatické spouštění úloh bez nutnosti manuálního zásahu.

Existuje několik možných plánovačů, které lze použít:

- **Cron** – Unix
- **Task Scheduler** – Windows
- **Quartz** – Java
- **Apache Airflow** – Python

Všechny tyto plánovače umožňují spouštění úloh v pravidelných intervalech. Pro účely této práce byl zvolen plánovač Quartz, který je napsán v jazyce Java a je podporován Spring Frameworkem.

Intervaly mohou být nastaveny v cron notaci, což umožňuje velkou flexibilitu při plánování úloh (například každý den ve 3:00, každý týden v pondělí v 8:00, každý měsíc první den ve 12:00 atd.).

5.5 Docker

Docker je open-source platforma pro vývoj, nasazení a provoz aplikací. Umožňuje vytváření kontejnerů, které obsahují všechny potřebné závislosti pro běh aplikace.

Pro každou databázi je třeba vytvořit instanci, která bude obsahovat potřebné tabulky, data a klienta pro komunikaci a práci s databází. Toto je úloha jako stvořená pro Docker, který umožňuje vytvoření kontejneru s databází, který bude obsahovat veškeré potřebné závislosti pro běh aplikace. [9]

5.6 GUI klient pro komunikaci s databází

Klient pro komunikaci s databází je nástroj, který umožňuje připojení k databázi a provádění dotazů. Je třeba vybrat klienta, který bude podporovat náhled do všech databází použitých v této práci.

Možnosti jsou:

- **DBeaver** – open-source nástroj pro správu databází, který podporuje mnoho různých databází.
- **SQL Developer** – nástroj pro správu databází Oracle.
- **Adminer** – open-source nástroj pro správu databází, který může zároveň běžet v Dockeru.

Všechny tyto nástroje umožňují připojení k databázi a provádění dotazů. Pro účely této práce byl zvolen DBeaver, který podporuje širokou škálu databází a je open-source. Zároveň umožňuje export dat z databáze do různých formátů (CSV, Excel, JSON atd.). [10]

5.7 Log4j

Log4j je open-source knihovna pro logování v jazyce Java. Umožňuje snadné a efektivní logování událostí v aplikaci. Je možné nastavit různé úrovně logování (DEBUG, INFO, WARN, ERROR, FATAL) a také různé výstupy (soubor, konzole, databáze atd.). Tato technologie bude používána v průběhu vývoje aplikace pro logování událostí. [11]

Příprava databázových systémů

Každá databáze měla své problémy, ale všechny se nakonec podařilo vyřešit. V následujících částech bude popsáno jak byly jednotlivé databáze implementovány a jaké problémy se objevily.

6.1 PostgreSQL

PostgreSQL se ukázala jako nejjednodušší databáze pro implementaci. Databáze běží na lokálním serveru v Dockeru. Pro vytvoření byl stažen oficiální image z Docker Hubu společně s knihovnou PostGIS, která je potřebná pro práci s geodaty.

```
docker pull postgres:latest
```

Po stažení image byl následně vytvořen a spuštěn kontejner pomocí docker-compose:

```
docker-compose up -d
```

Databáze je inicializována skriptem ‘init.sql‘, který vytvoří potřebné tabulky a indexy. Tento skript byl napsán specificky pro databázi PostgreSQL a pomocí něj je vytvořena databáze se všemi potřebnými tabulkami.

6.2 Microsoft SQL Server

Microsoft SQL Server byla druhá databáze, která byla implementována. Při stahování oficiálního image z Docker Hubu se objevily problémy. Po úspěšném stažení image ‘mssql/server:2017‘ byl kontejner spuštěn stejně jako u PostgreSQL pomocí docker-compose.

Menším rozdílem je způsob spouštění ‘init‘ skriptu, který se nespouští při inicializaci databáze přes ‘entrypoint‘, ale až následně pomocí příkazu ‘sqlcmd‘.

6.3 Oracle Database

Oracle byla poslední databáze, která byla implementována. Při stahování oficiálního image z Docker Hubu došlo k problémům. Když se však image podařilo stáhnout a kontejner spustit, databáze nefungovala správně. Nezbývalo nic jiného než stáhnout Oracle XE a nainstalovat databázi na lokální stroj. Byla stažená verze 21c Express Edition, která je zdarma pro osobní použití. Tato verze byla stažena z oficiálních stránek Oracle `https://www.oracle.com/database/technologies/appdev/xe/quickstart.html`. Konkrétně byla stažena verze pro Windows 64-bit. Po nainstalování byla v databázi vytvořeno schéma **XEPDB1**, které je výchozím schématem pro databázi XE. V tomto schématu byl vytvořen uživatel *ruian_user* s heslem *12345*, který nadále bude přistupovat do databáze. V GUI DBeaver by následně použit inicializační skript *init.sql*, který vytvoří potřebné tabulky.

6.4 Odlišnost ve skriptech pro tvorbu databází

Každá databáze měla vlastní skript pro inicializaci databáze. Hlavní rozdíl byl v syntaxi SQL příkazů.

Tabulka 6.1: Datové typy v různých databázích

	Oracle	**PostgreSQL**	**MSSQL**
Integer	NUMBER	INTEGER	INT
Long	NUMBER(19)	BIGINT	BIGINT
DateTime	DATE	TIMESTAMP	DATETIME
String	VARCHAR2(length)	VARCHAR(length)	NVARCHAR(length)
JSON	JSON	JSONB	NVARCHAR(MAX)
Boolean	NUMBER(1)	BIT	BIT
Geometry	SDO_GEOMETRY	GEOMETRY	GEOMETRY
Binary	BLOB	BYTEA	VARBINARY

Velkým rozdílem byly také cizí klíče (foreign key), které byly v každé databázi definovány odlišně.

- **PostgreSQL**:

```
<column_name> INTEGER REFERENCES <table>(<column_name>)
<column_name> BIGINT REFERENCES <table>(<column_name>)
```

- **MSSQL**:

```
<column_name> INT FOREIGN KEY REFERENCES
    <table>(<column_name>)
<column_name> BIGINT FOREIGN KEY REFERENCES
    <table>(<column_name>)
```

- **Oracle**:

```
<column_name> <column_type>,
CONSTRAINT <constraint_name> FOREIGN KEY (<column_name>)
    REFERENCES <table>(<column_name>)
```

Implementace aplikace

Před začátkem popisu implementace aplikace je vhodné upřesnit, jaký je její účel. Cílem aplikace je stahování dat z API RÚIAN a jejich následné zpracování. Výsledkem bude projekce těchto dat do databáze, která bude sloužit jako zdroj pro další zpracování. Aplikace je napsána v programovacím jazyce Java a využívá framework Spring Boot.

Projekce databáze

Projekce databáze představuje způsob zobrazení dat z jedné databáze do jiné. Podle konfigurace se nastavuje úroveň projekce, na jejímž základě se data zobrazují. Nastavení projekce je dále popsáno v sekci 7.3.5.

7.1 Architektura aplikace

Architektura je rozdělena do několika modulů, z nichž každý má svůj specifický úkol:

- **main** – hlavní modul aplikace, který obsahuje třídu `Main`, jež spouští celou aplikaci.
- **config** – stará se o konfiguraci aplikace a její inicializaci.
- **download** – zajišťuje stahování dat z API RÚIAN a následné zpracování těchto dat.
- **scheduler** – spravuje spouštění úloh a konfiguraci plánovače.
- **utils** – obsahuje pomocné konstanty a další užitečné nástroje.

Všechny závislosti a knihovny jsou spravovány pomocí nástroje `Maven`.

7.2 Inicializace aplikace

Na začátku běhu aplikace se provede její inicializace. Důvodem, proč byl zvolen framework Spring Boot, je schopnost aplikace automaticky načítat všechny komponenty a konfigurace. Konkrétně se jedná o moduly označené anotacemi `@Component`, `@Service`, `@Configuration`, `@Entity`, `@Repository` a `@Bean`.

7.3 Mapování dat

Pro mapování dat mezi databází a aplikací se používá `Hibernate ORM`. `Hibernate` je objektově-relační mapovací framework, který umožňuje práci s databází pomocí objektů. Pro mapování bylz použity **Dto objekty**, které reprezentují jednotlivé tabulky v databázi. Následně je zde Repository interface, které dědí z `JpaRepository`. A nakonec je zde Service třída, která obsahuje logiku pro práci s databází a komunikuje s Repository.

7.3.1 Dto objekty

Dto objekty nebo také **Data Transfer Object** jsou objekty, které slouží k přenášení dat mezi vrstvami aplikace. Pro jejich označení se používá anotace `@Entity`, která určuje, že se jedná o entitu mapovanou na tabulku v databázi. Dále mají anotaci `@Table`, která určuje název tabulky v databázi, a anotaci `@Id`, která určuje primární klíč tabulky. Pro kontrolu je zde anotace `ToString`, která slouží pro převod objektu na řetězec a anotace `@Data` z knihovny `@Lombok`, která generuje gettery a settery pro jednotlivé atributy. Pro každé atributy, které jsou ve výsledné databázi jako JSON je pro sloupec použita anotace `@JdbcTypeCode(SQLType.JSON)`.

7.3.2 Repositáře

Repositáře jsou rozhraní, která dědí z `JpaRepository` a poskytují metody pro práci s databází. Mezi tyto metody patří například `findAll`, `findById`, `save` a `delete`. Repositáře jsou označeny anotací `@Repository`, která určuje, že se jedná o komponentu pro práci s databází.

7.3.3 Service třídy

A nakonec jsou zde Service třídy, které obsahují logiku pro práci s databází a komunikaci s repositáři. Tyto třídy jsou označeny anotací `@Service`, která určuje, že se jedná o komponentu pro práci s databází. Zde se provádí veškeré veškerá logika před uložení do databáze:

- Kontrola zda Dto má primární klíč.
- Kontrola zda jsou cizí klíče validní a existují v databázi.
- Doplnění dat do objektu podle již existujících dat v databázi.
- Úprava dat podle konfigurace.

Po provedení těchto kontrol se objekt uloží do databáze pomocí repositáře.

7.3.4 Připojení k databázi

O připojení k databázi se stará třída `DatabaseSource`, která zajišťuje navázání spojení s databází. Z konfiguračního souboru se načtou potřebné informace pro připojení:

- `type` – typ databáze (např. `postgresql`, `mssql`, `oracle`),
- `url` – adresa databáze (např. `localhost:5432` pro PostgreSQL),
- `dbname` – název databáze (např. `ruian`),
- `username` – uživatelské jméno pro připojení k databázi,
- `password` – heslo pro připojení k databázi.

Na základě těchto informací se vytvoří připojení k databázi, tzv. **DataSource**. Tento zdroj bude dále upravován v jiném modulu. Základem `DataSource` je nastavení základních parametrů připojení. Vytváří se výsledný connection string, který se upravuje podle typu databáze. K URL se připojí název databáze, uživatelské jméno, heslo a v případě MSSQL také certifikát pro zabezpečené připojení.

V dalším modulu se pro `DataSource` nastavují další parametry potřebné pro přístup k databázi a přenos dat. Dále se nastaví dialekt pro správnou syntaxi SQL příkazů podle použité databáze.

7.3.5 Načtení konfigurace

`DatabaseSource` se stará pouze o připojení k databázi, třída `AppConfig` načítá zbytek potřebné konfigurace:

1. **Konfigurace úkolů pro** `Quartz Scheduler` Načítá se čas ve formátu cron pro spouštění přírůstkových dat a nastavení, zda přeskočit inicializaci hlavních územních prvků a krajů.

2. **Seznam krajů s příslušnými kódy** Každý řádek obsahuje kraj a jeho kód. Pokud je v konfiguraci nastaveno přeskočení inicializace krajů nebo je seznam prázdný, tento krok se přeskočí.

3. **Dodatečná nastavení** Například volba pro ignorování geometrických dat (některé databáze nepodporují geometrické typy) a nastavení velikosti jednotlivých commitů, které slouží pro optimalizaci výkonu.

4. **Nastavení zpracování jednotlivých tabulek** Základním parametrem je způsob zpracování tabulek: `all` nebo `selected`.

   - **all** – všechny tabulky budou zpracovány bez ohledu na konfiguraci,
   - **selected** – budou zpracovány pouze tabulky výslovně uvedené v konfiguraci.

5. **Konfigurace jednotlivých tabulek** Pokud je nastaven režim `selected`, zpracovávají se pouze specifikované tabulky. Každá tabulka může mít vlastní nastavení:

   Zdrojový kód 7.1: Konfigurace tabulek

   ```
   "<table_name>": {
       "howToProcess": "all | exclude | include",
       "columns": ["<column_name>", ..., "<column_name>"]
   }
   ```

   Možnosti zpracování:

   - **all** – všechny sloupce budou zpracovány,
   - **exclude** – vybrané sloupce budou ignorovány,
   - **include** – vybrané sloupce budou zpracovány.

6. **Sloupce** u jednotlivé tabulky jsou definovány jako pole řetězců s názvy sloupců ve formátu lowercase bez mezer a speciálních znaků. Sloupce jsou odděleny čárkami. Sloupce nemusí být uvedeny pokud je nastaveno `all` u konkrétní tabulky.

Možné chyby při načítání konfigurace:

- Pokud je nastaveno `exclude` nebo `include`, ale nejsou uvedeny žádné sloupce → neplatná konfigurace,
- Sloupce nebo tabulky, které neexistují v databázi → budou přeskočeny,
- Seznam krajů je povinný atribut, i pokud je prázdný. V takovém případě budou zpracovány pouze základní územní prvky.

7.4 Quartz Scheduler

Pro synchronizaci dat a spouštění úloh v určitých intervalech je použit `Quartz Scheduler`. Ten je přímo integrován do frameworku Spring Boot jako součást jedné z jeho knihoven. Samotný scheduler se spouští při startu aplikace.

7.4.1 Jobs

Joby jsou úlohy, které se spouštějí v určitých intervalech nebo na základě určité události. V našem případě se jedná o 3 úlohy:

- `InitStatAzZsj`
- `InitRegion`
- `Additions`

InitStatAzZsj

Tato úloha se spouští při startu aplikace a je zodpovědná za inicializaci základních územních prvků a krajů. Pokud je v konfiguraci nastaveno přeskočení inicializace těchto prvků, úloha se neprovede. Úloha nejprve stáhne data o základních územních prvcích a krajích z API RÚIAN za poslední měsíc. Následně se soubor zpracuje a data se uloží do databáze.

InitRegion

Tato úloha se spouští jako druhá, bezprostředně po dokončení úlohy `InitStatAzZsj`. Pokud je v konfiguraci nastaveno přeskočení inicializace regionů, tato úloha se přeskočí. Úloha ze zadané konfigurace načte všechny kraje, které byly uvedeny. Každý kraj má přiřazen seznam obcí, které se postupně stahují z API RÚIAN, zpracují a uloží do databáze.

Additions

Tato úloha se spouští na základě nastavení v konfiguračním souboru nebo po dokončení úlohy `InitRegion`. Každý den je na API RÚIAN vytvořen nový soubor s přírůstkovými daty za poslední den. Úloha tento soubor stáhne, zpracuje a uloží do databáze. Poté čeká na další spuštění podle časového nastavení v konfiguraci.

7.4.2 Triggers

Triggers jsou spouštěče, které určují, kdy se má daný `job` spustit. Jak bylo zmíněno výše, úlohy `InitStatAzZsj` a `InitRegion` se spouštějí jednorázově při startu aplikace a navazují na sebe. Úloha `Additions` se naproti tomu spouští pravidelně podle nastavení v konfiguračním souboru. Konkrétně je čas spuštění definován v cron formátu.

Cron formát

Cron formát je způsob, jak určit čas spouštění úloh. Původně byl použit v systémech Unix a stal se široce rozšířeným standardem. Skládá se z následujících částí:

- sekundy (0–59),
- minuta (0–59),
- hodina (0–23),
- den v měsíci (1–31),
- měsíc (1–12 nebo zkratky názvů měsíců),
- den v týdnu (0–6 nebo zkratky názvů dní).

Zkratky měsíců a dnů se uvádějí v angličtině, typicky ve formátu tří písmen (např. `JAN`, `MON`). **Příklady:**

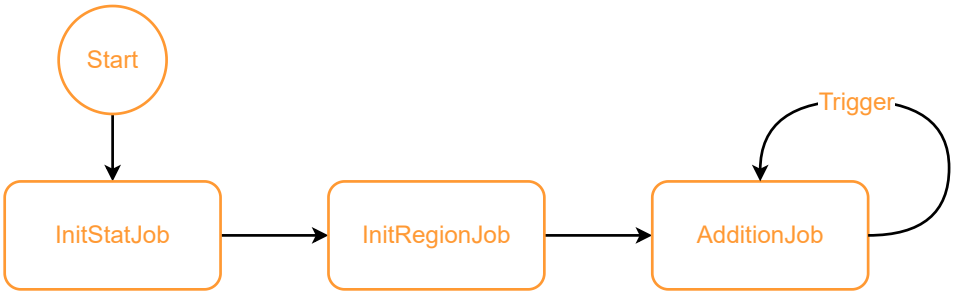
- `0 0 2 * * ?` – úloha se spustí každý den ve 2:00.
- `0 0 0 * * MON` – úloha se spustí každé pondělí o půlnoci.
- `30 14 3 JAN * ?` – úloha se spustí 3. ledna ve 14:30.

7.4.3 Zajištění správného pořadí

Je důležité zajistit, aby se úlohy spouštěly v správném pořadí. Kdyby se spustila úloha `Additions` před úlohou `InitRegion`, mohlo by dojít k problémům s neexistujícími daty. Tomu se předejde vytvořením závislostí mezi joby. Pořadí jobů bylo zvoleno:

Při startu aplikace se spustí job `InitStatAzZsj`, až úspěšně skončí, spustí se job `InitRegion`. Pokud je v konfiguraci nastaveno, že se job přeskočí tak job stále provede a dokončí jen bez zpracování dat. Stejně to platí pro job `InitRegionJob`. Problémem bylo, že job `InitStatAzZsj` a `Additions` se spouštěly vždy při startu aplikace, protože měly trigger pro start. Ovšem job `Additions` se má spustit až po

Obrázek 7.1: Pořadí jobů



úspěšném dokončení jobu `InitRegion`. Tento problém bzl vyřešen pomocí **Semaforu**. Konkrétně byl použit `Semaphore` z balíčku `java.util.concurrent`. Vzhledem k tomu, že Quartz Scheduler má pro každý job vlastní vlákno, je možné použít `Semaphore` pro zajištění, že se job `Additions` může spustit až po úspěšném dokončení jobu `InitRegion`.

Semafor je inicializován na hodnotu 0, což znamená, že job `Additions` se nemůže spustit, dokud není semafor uvolněn. Ten je uvolněn následně na konci jobu `InitRegion`. Jakmile se job `Additions` spustí poprvé, je pak opakovaně spouštěn podle nastavení cron v konfiguračním souboru.

Semafor

Semafor je synchronizační primitivum, které umožňuje řídit přístup k sdíleným prostředkům v multithreadovém prostředí. Semafor může být binární (0 nebo 1) nebo počítací (libovolné celé číslo). Následně má dvě hlavní operace:

- `V()` – uvolnění semaforu, zvýšení hodnoty semaforu o 1.
- `P()` – zablokování semaforu, snížení hodnoty semaforu o 1, pokud je hodnota semaforu 0, vlákno se zablokuje a čeká na uvolnění semaforu.

Tyto dvě operace jsou atomické, což znamená, že jsou prováděny jako jedna operace a nelze je přerušit. V Javě je semafor implementován pomocí třídy `Semaphore` z balíčku `java.util.concurrent`. P a V operace jsou implementovány jako metody `acquire()` a `release()`. Více o semaforech a jeho použití je popsáno v [12].

7.5 Získávání dat z API

Stahování dat z API RÚIAN je realizováno pomocí tříd `VdpClient` a `VdpDownload`, které jsou součástí modulu `download`.

7.5.1 VdpDownload

Třída `VdpDownload` se stará o nastavení HTTP klienta s důvěrou ke všem certifikátům. Využívá se zde knihovna `Apache HttpClient`, která je součástí Spring Boot. V rámci metody `init()` se konfiguruje `SSLContext`, časové limity a případně HTTP proxy. Výsledný klient je uložen do proměnné `client`, která se následně používá pro volání metod `tryGet()` a `trySaveFilter()`. Tyto metody slouží ke stažení dat ze serveru VDP nebo k inicializaci filtru pomocí HTTP požadavku. Před ukončením aplikace je klient uzavřen pomocí metody označené anotací `@PreDestroy`.

7.5.2 VdpClient

`VdpClient` je hlavní komponenta pro komunikaci se službou Veřejného dálkového přístupu (VDP). Využívá celkem tři URL pro přístup k seznamům souborů a další tři URL pro stahování jednotlivých datových listů. Adresy jsou většinou nastaveny v konstantách této třídy, s výjimkou jedné, která si URL generuje dynamicky podle data předchozího dne. Pro samotné stahování dat používá instanci třídy `VdpDownload`. Třída obsahuje především tři metody, které jsou volány z dříve popsaných `jobů`:

- `zpracovatStatAzZsj()`
- `getListLinksObce()`
- `getAdditions()`

Každá z těchto metod nejprve stáhne textový soubor, který obsahuje seznam dostupných datových souborů. Tento seznam se následně parsuje a získané odkazy se využijí ke stažení souborů. Metody `zpracovatStatAzZsj()` a `getAdditions()` stahují pouze jeden konkrétní soubor, zatímco `getListLinksObce()` stahuje všechny soubory dostupné v seznamu.

Z tohoto důvodu byla doplněna metoda `downloadFilesFromLinks()`, která umožňuje stáhnout všechny soubory uvedené v daném seznamu. Získaná data jsou obvykle ve formátu ZIP, jsou automaticky rozbalena a předána jako `InputStream` dalším komponentám prostřednictvím funkčního rozhraní `Consumer`.

Třída rovněž implementuje opakování požadavků v případě selhání, logování a čištění dočasných souborů.

7.6 Zpracování dat

Jak bylo zmíněno výše, data jsou zpracovávána pomocí třídy `VdpClient` a pomocí funkčního rozhraní `Consumer` se předávají dalším komponentám. A to právě kompo-

nentě `VdpParser`, která se stará o parsování XML souboru v předaném `InputStream` dat a následné ukládání do databáze.

Na začátku se zinicializuje `XMLStreamReader`, který se stará o parsování XML souboru. Reader je vytvořen pomocí třídy `XMLInputFactory`, která je součástí knihovny `StAX`.

Původně byl použit `DocumentBuilder`, ale ten byl nahrazen `StAX` parserem. Celé soubory si nejprve načítal do paměti a poté je parsoval, po prvcích (Node). To bylo v případě malých souborů (příklad při zpracování malých obcí). Problém nastal při zpracování velkých souborů (např. obec Praha), Soubor dosahoval velikosti přes 1 GB a bylo potřeba jej zpracovat po částech. Je ale velmi náročné dělit XML bez rozbití struktury souboru. Proto se později přešlo na `StAX` parser, který zpracovává XML po prvcích a nepotřebuje celou strukturu XML. `StAX` parser je mnohem efektivnější a umožňuje zpracovávat velké soubory bez nutnosti načítat je celé do paměti. Zároveň čte pouze události jako je začátek a konec elementu, což je pro zpracování dat dostačující. Stačí tedy nadefinovat jen název potřebných elementů a tyto elementy parsovat. U `DocumentBuilderu` bylo třeba číst všechny Listy a poté je zpracovávat.

7.6.1 Parsing dat

Jak už bylo řečeno výše, data jsou parsována ve třídě `VdpParser` s použitím `XMLStreamReader`. Zároveň ve stejném modulu se nachází i třída `VdpParserConsts`, která obsahuje konstanty pro názvy jednotlivých elementů. Při čtení XML se nejprve čte hlavička souboru. Ta obsahuje nepotřebné informace, které se ignorují. Dále se čtou jednotlivá data počínající elementem `vf:Data`. Nejprve je potřeba ale rozeznat jaký element se čte. `XMLStreamReader` rozeznává několik eventů a u každého eventu se provádí jiná akce a získává jiná informace.

Tabulka 7.1: Eventy XMLStreamReader

Event	Hodnota Event	Důležitá informace	Příklad
START_ELEMENT	1	Název Elementu	<vf:Data>
END_ELEMENT	2	Název Elementu	<\vf:Data>
CHARACTERS	4	Hodnoty	Data

Příkladem pokud se vyskytne event `START_ELEMENT` tak z něj můžu získat název elementu a jeho atributy. Pokud se vyskytne event `CHARACTERS`, tak z něj získám text uvnitř elementu, ale pokud se pokusím získat jméno elementu vyhodí to výjimku.

Cílem je tedy číst data v v cyklu dokud nenarazím na konec elementu `vf:Data`. V průběhu čtení se nachází další důležité elementy, které určují co se zrovna čte

za objekt. Jeden soubor může rozeznávat až 19 různých objektů, které se parsují do různých DTO objektů. Každý jeden objekt podléhá jinému seznamu objektů. Je tedy potřeba rozeznat kdy začíná list a kdy jeden objekt.

Tabulka 7.2: Seznam objektů a jejich názvy

Název	**List Počátek/konec**	**Objekt Počátek/Konec**
Stát	<vf:Staty>	<vf:Stat>
Region soudržnosti	<vf:RegionySourdznosti>	<vf:RegionSourdznosti>
VÚSC	<vf:Vusc>	<vf:Vusc>
Okres	<vf:Okresy>	<vf:Okres>
ORP	<vf:Orp>	<vf:Orp>
POU	<vf:Pou>	<vf:Pou>
Obce	<vf:Obce>	<vf:Obec>
Spravní obvod	<vf:SpravniObvody>	<vf:SpravniObvod>
MOP	<vf:Mop>	<vf:Mop>
MOMC	<vf:Momc>	<vf:Momc>
Část obce	<vf:CastiObci>	<vf:CastiObce>
Katastrální území	<vf:KatastralniUzemi>	<vf:KatastralniUzemi>
Parcela	<vf:Parcely>	<vf:Parcely>
Ulice	<vf:Ulice>	<vf:Ulice>
Stavební objekt	<vf:StavebniObjekty>	<vf:StavebniObjekt>
Adresní místo	<vf:AdresniMista>	<vf:AdresniMisto>
ZSJ	<vf:Zsj>	<vf:Zsj>
VO	<vf:VO>	<vf:VO>
Zaniklý prvek	<vf:ZaniklePrvky>	<vf:ZaniklyPrvek>

Zdrojový kód 7.2: Příklad XML Struktury

```
<vf:Data>
    <vf:Staty>
        <sti:Stat>
            <sti:Kod>...</sti:Kod>
        </sti:Stat>
    </vf:Staty>
<\vf:Data>
```

Každý element je rozdělen na dvě části. Klasifikátor a název elementu. Klasifikátor je určen pro rozlišení jednotlivých elementů do jaké úrovně patří. Podle názvu elementu se určuje jaký objekt se parsuje. V příkladu 7.2 je uveden `vf` klasifikátor (*výměnný formát*), který je nejvyšší úrovní. Každý list objektů také používá tento klasifikátor, ale každý objekt má jiný klasifikátor. Každý objekt má následně atributy, s klasifikátorem objektu. Pokud je atribut cizí klíč používá klasifikátor odkazovaného objektu. Příklad `vf:Staty` a `sti:Stat`. Dále se může vyskytnout i atribut, který je v další tabulka nebo list tabulek. Tyto tabulky používají klasifikátor `com`.

Zdrojový kód 7.3: Příklad Klasifikátorů

```
<vf:Data>
    <vf:Okresy>
        <vf:Okres>
            <oki:Kod>100</oki:Kod>
            <oki:Nazev>...</oki:Nazev>
        <\vf:Okres>
    </vf:Okresy>
    <vf:Obce>
        <vf:Obec>
            <obi:Kod>...</obi:Kod>
            <obi:Nazev>...</obi:Nazev>
            <obi:Okres>
                <oki:Kod>100<oki:Kod>
            </obi:Okres>
            <obi:MluvnickeCharakteristiky>
                <com:Pad2>...</com:Pad2>
                <com:Pad3>...</com:Pad3>
            </obi:MluvnickeCharakteristiky>
        </vf:Obec>
    </vf:Obce>
</vf:Data>
```

7.6.2 Atributy Objektů

V tabulce 6.1 byly zmíněné všechny datové typy a jejich rozdíly mezi databázemi. Podle dokumentace RÚIAN VFR [1] každý atribut objektu má také svůj datový typ. Mezi datovými objektů se nachází:

- **String** – textový řetězec, který může obsahovat písmena, číslice a speciální znaky.
- **Integer** – celé číslo bez desetinné části.
- **Long** – reálné číslo s desetinnou částí.
- **Boolean** – logická hodnota, která může nabývat hodnoty true nebo false.
- **Binární data** – binární data, která budou primárně obsahovat obrázky.
- **DateTime** – datum a čas ve formátu YYYY-MM-DDTHH:MM:SS.
- **Kolekce** – seznam objektů nebo hodnot, které jsou uloženy v jednom atributu.

Datové typy jako `String`, `Integer`, `Boolean` a `Long` jsou standardní datové typy. DateTime je datový typ, který se používá pro ukládání data a času. Co se týče binárních dat, tak ty budou uloženy jako `byte[]`, ale aplikace tyto data zatím nezpracovává. Ovšem zmíněné `Kolekce` jsou označeny všechny atributy, které obsahují více hodnot nebo cizí klíče. Kolekce budou do databáze ukládány jako **JSON**.

JSON Objekty

Proč byly zvoleny JSON objekty? Tabulky dle specifikace nebyly vhodné pro rozdělení na další tabulky a vytváření cizích klíčů. Některé kolekce totiž mohou být 1:1 nebo N:M. Proto byl zvolen JSON formát, který je vhodný pro ukládání takových to dat. Některé kolekce jsou uloženy jako JSON Objekt a některé jako JSON pole (Pole JSON Objektů). Byl proto vytvořen pro každou kolekci vlastní JSON mapovací metoda. Každá tato metoda funguje podobně jako XML parser. Hledá elementy, které jsou uloženy v kolekci a převede je na JSON objekt nebo pole. Všechny atributy, které se mohou vyskytnout json definován jako konstanty v modulu `jsonObjects` s příslušným jménem k objektu.

Kolekce JSON Objekt

Tyto kolekce vždy obsahují pouze jeden JSON Objekt. Proto byl vytvořen pro každou kolekci vlastní metoda, která parsuje danou kolekci. Jedná se o kolekce:

- **Mluvnické Charakteristiky** – metoda `readMCh()`
- **Čísla domovní** – metoda `readCislaDomovni()`
- **Nespravné Údaje** – metoda `readNespravneUdaje()`

Kolekce JSON Pole

Tyto kolekce mohou obsahovat jednu nebo více Objektů. Proto byl vytvořen pro každou kolekci vlastní metoda, která parsuje danou kolekci. Jedná se o kolekce:

- **Bonitované díly / Bonitovaný díl** – metoda `readBonitovaneDily()` a `readBonitovanyDil()`
- **Způsoby ochrany / Způsob ochrany** – metoda `readZpusobyOchrany()` a `readZpusobOchrany()`
- **DetailniTEA** – metoda `readDetailniTeas()` a `readDetailniTea()`

V přírůstkových datech se nachází i další kolekce a to **Nezjištěné údaje**. Ta je je kompletně ignorována a neukládá se do databáze.

Kolekce s cizími klíči

Jak již bylo zmíněno, cizí klíče jsou jako kolekce uloženy Integer nebo Long. Vzhledem k tomu, že se jedná o cizí klíče, které odkazují na jiné objekty, byly vytvořeny metody pro parsování těchto cizích klíčů. Metoda `readFK()` a `readFKLong` dostane jako parametr název elementu a vrací Integer nebo Long. Rozdělení na Integer a Long je z důvodu, že je zde objekt **Parcela**, která má primární klíč jako Long.

Geometrie

Geometrie je uložena speciální kolekce obsahující až 3 možné geometrické údaje. V základní datové sadě se nachází pouze Definiční bod specifikující střed daného objektu na mapě. V rozšířených se pak dodatečně nachází i Generalizované Hranice a Originální Hranice. Je zde i případ Definiční čáry, která je uložena jako `LineString` nebo `MultiLineString`. Tato geometrie se vyskytuje pouze u objektů **Ulice**. O problematiku parsování geometrie se stará třída `GeometryParser` v modulu `geometry`, která parsuje jednotlivé geometrické objekty. V současné práci se řeší pouze Definiční bod, který je uložen jako `Point` nebo `MultiPoint`. Generalizované Hranice a Originální Hranice jsou uloženy jako `Polygon` nebo `MultiPolygon`. Stejně jako u JSON objektů i geometrie se čte podle eventu a atributů v elementu. Názvy geometrických objektů jsou uloženy jako konstanty v modulu `geometryParserConsts` s příslušným jménem k objektu. Výstupem všech metod pro parsování geometrie je `Geometry` objekt z knihovny `JTS`.

7.6.3 Ukládání dat do databáze

Po úspěšném parsování jedné sady objektů (např. **Obce**) nám vznikne list Dto objektů dané sady, které se následně ukládají do databáze. Tento list se následně předává do metody `prepareAndSave()`, která se nachází v příslušné service třídě (např. `ObecService`) 7.3.3. Tato metoda se stará o přípravu dat pro uložení do databáze a o uložení do databáze. Tato metoda se nevykoná pokud je v konfiguraci nastaveno `howToProcess` na `selected` a není v konfiguraci uvedena tabulka daného objektu.

Jak metoda `prepareAndSave()` funguje? Pro každý objekt se provede několik kontrol a úprav.

- **Bezpečnostní kontrola** – slouží k zabránění chyby nebo neplatným datům v databázi.
- **Úprava dat** – slouží k úpravě dat podle konfigurace a podle již existujících dat v databázi.

1. *(Bezpečnostní kontrola)* Pokud objekt neobsahuje primární klíč, kontrola se zastaví a po zpracování všech objektů bude odstraněn ze seznamu.
2. *(Bezpečnostní kontrola)* Ověří zda objekt obsahuje cizí klíč, ověří se jeho platnost a existence v databázi. Pokud cizí klíč neexistuje, kontrola se zastaví a objekt bude po zpracování všech odstraněn ze seznamu.
3. *(Úprava dat)* Když se objekt již nachází v databázi:
   a) Pokud je v konfiguraci u tabulky nastaveno `howToProcess` na `all`: Pokud je atribut u nového objektu `null` a v databázi je `notnull`, tak se použije hodnota z databáze.
   b) Pokud je v konfiguraci u tabulky nastaveno `howToProcess` na `include`: Pokud je atribut u nového objektu atribut `notnull` a v konfiguraci tento atribut není uveden, tak se použije hodnota z databáze, jinak se použije hodnota z nového objektu.
   c) Pokud je v konfiguraci u tabulky nastaveno `howToProcess` na `exclude`: Pokud je atribut u nového objektu atribut `notnull` a v konfiguraci tento atribut je uveden, tak se použije hodnota z databáze, jinak se použije hodnota z nového objektu.
4. *(Úprava dat)* Když se objekt ještě nenachází v databázi:
   a) Pokud je v konfiguraci u tabulky nastaveno `howToProcess` na `all`: Objekt se uloží do databáze tak jak je bez dodatečných úprav.

b) Pokud je v konfiguraci u tabulky nastaveno `howToProcess` na `include`: Objektu se dá hodnota null u atributů, které nejsou uvedeny v konfiguraci.

c) Pokud je v konfiguraci u tabulky nastaveno `howToProcess` na `exclude`: Objektu se dá hodnota null u atributů, které jsou uvedeny v konfiguraci.

Úprava dat má dvě verze podle toho zdali se jedná o nový nebo již existující objekt. Objekt z databáze se vybírá selekcí podle primárního klíče. Do paměti se načte objekt z databáze a podle toho se upraví nový objekt.

Během těchto kontrol se také loguje kolik prvků z listu bylo zpracováno. Jedná se o milníky *25%*, *50%*, *75%* a *100%*,

Po úspěšném zpracování listu objektů se vypíše kolik objektů bylo odstraněno a nebude uloženo do databáze.

Pak přijde na řadu ukládání dat do databáze. To probíhá tak, že si list objektů rozdělí na menší části podle velikosti `commitSize` z konfigurace nebo menší. Tento list se pak následně uloží do databáze pomocí metody `saveAll()`.

7.7 Po zpracování dat

Jakmile jsou data úspěšně přečtena a uložena do databáze, je potřeba provést další úkony. Ve třídě `VdpClient` u metody `unzipContent()` se po úspěšném rozbalení souboru a zpracování souboru v try bloku nachází finally blok který se postará o smazaní dočasného souboru. Tento blok se vykoná i v případě že dojde k výjimce a soubor se neuloží do databáze.

Jakmile se soubor úspěšně vymaže z disku může začít nový job nebo zpracování dalšího souboru v případě že se jedná o job `InitRegionJob`. Dále po zpracování regionů se zpracují přírůstková data (job `AdditionJob`)

Testování

8.1 Rychlostní porovnání databázových systémů

Tento test byl proveden na stejném konfiguračním souboru pro všechny databázové systémy. Testovací konfigurace je nastavena na provedení stažení, zpracování a uložení Základní datové sady Stát až ZSJ. Tato datová sada byla vybrána z důvodu neměnnosti. Data v této datové sadě se mění jen velmi zřídka a je možné je stáhnout. Dále bylo nastavené, že se zpracují všechny tabulky uvedené v již zmíněné datové sadě. Geometrie bude zpracována a velikost jednotlivých commitů bude 2000. Vzhled konfiguračního souboru je uveden v příkladu 8.1 a bude nakonfigurován pro PostgreSQL. Testování probíhalo na stejném stroji, který měl nainstalované všechny databázové systémy.

Nad všemi databázovými systémy probíhaly testy 3×, aby se eliminovaly chyby způsobené jinými procesy na serveru. Každý test byl proveden na prázdné databázi. Měření začalo, když byla data připravena ke čtení z důvodu eliminace stahování dat z internetu. Konkrétně když se vypsala zpráva „Data proccesing started.“ a následně skončilo, když se vypsala zpráva „Data proccesing finished.“. Během tohoto testu byl také vyfiltrováno 1 ZSJ z důvodu neexistence cizího klíče v tabulce Katastrální území. Testování proběhlo s daty `https://vdp.cuzk.gov.cz/vymenny_format/soucasna/20250331_ST_UZSZ.xml.zip`

Tabulka 8.1: Časy testů pro používané databáze

	PostgreSQL	**MS SQL**	**Oracle**
test 1	0:35:59	0:22:46	1:04:19
test 2	0:37:03	0:22:26	1:05:30
test 3	0:35:53	0:21:26	1:04:57
průměr	0:36:18	0:22:13	1:04:55

Jak je vidět v tabulce 8.1, ukázalo se že MS SQL je nejrychlejší databázový systém pro zpracování datové sady. PostgreSQL je o přesně 14 minut pomalejší než MS SQL a Oracle je o přesně 28 minut pomalejší než MS SQL. Je ale možné že Oracle je

pomalejší z důvodu že se jedná o Express verzi, která je omezena na 2GB RAM a 1 CPU. Dále je možné že rychlost byla omezena prostředím. Zatím co PostgreSQL a MS SQL běžely v Docker kontejneru, Oracle běžel přímo na hostitelském systému.

Zdrojový kód 8.1: Konfigurační soubor pro test rychlosti

```
{
    "database": {
      "type": "postgresql",
      "url": "jdbc:postgresql://localhost:5432",
      "dbname": "ruian",
      "username": "postgres",
      "password": "123"
    },
    "quartz": {
      "cron": "0 0 2 * * ?",
      "skipInitialRunStat": false,
      "skipInitialRunRegion": true
    },
    "vuscCodes": {},
    "additionalOptions": {
      "includeGeometry": true,
      "commitSize": 2000
    },
    "dataToProcess": {
      "howToProcess": "all"
    }
  }
```

Budoucí rozšíření a úpravy

Na této aplikaci je stále co přidávat, vylepšovat a upravit. Proto se v následujících sekcích pokusím shrnout, co by se dalo do budoucna vylepšit a přidat.

GUI pro aplikaci

Jediný uživatelský pohled na aplikaci je v současnosti zajištěn pomocí logu, který je generován v textovém formátu. Jedná se pouze o výpis informací o prováděných úkonech, které aplikace dělá. Bylo by dobré přidat uživatelské rozhraní, které by bylo schopné zobrazit informace o prováděných úkonech v reálném čase.

GUI pro nastavení konfigurace

Při psaní konfigurace uživatelem je možné že dojde k chybě. V současnosti je možné že uživatel udělá chybu a aplikace se se nespustí. Bylo bz užitečné vytvořit nějakou další aplikaci nebo nějaké GUI, při spuštění aplikace, ve kterém bude možné nastavit konfiguraci aplikace. Tímto způsobem by bylo možné uživateli pomoci s nastavením aplikace a zabránit chybám, které by mohly nastat při špatném nastavení aplikace.

Optimalizace

Aplikace je v současnosti napsána tak, že se snaží aby byla schopná zpracovat jakákoliv data z RÚIANu. Ovšem rychlostně to není úplně ideální. Hlavním kamenem úrazu je především kontrola a porovnání nových dat s těmi, které jsou už v databázi. Bylo by dobré nějak zefektivnit tento proces, aby se obecně zrychlil celý proces zpracování dat.

Refektoring kódu

Tenhle oddíl vzchází trochu z předchozího. Aplikace je napsána tak aby byla schopná zpracovat jakákoliv data z RÚIANu. Bylo bz proto dobré udělat refaktoring kódu, aby se zjednodušil a zefektivnil celý proces zpracování dat. Service třídy, Dto by se určitě upravit pomocí nějakého generického rozhraní. Využití abstraktních tříd a rozhraní by mělo celý kód velmi zjednodušit.

Zpracování zbylých dat

V současnosti aplikace zpracovává pouze základní datové sady. To znamená zpracování zbývajících geometrických dat, které jsou v RÚIANu k dispozici. Dále zakomponovat obrázky (binární data), které se u některých objektů vyskytují. Databáze je v současnosti připravena na zpracování těchto dat, ale aplikace nemá implementovanou logiku pro jejich zpracování.

Podpora pro dalších databázových systémů

Rozšíření aplikace o další databázové systémy by bylo mohla býti samozřejmostí. V současnosti je aplikace napsána tak, že je schopná pracovat pouze s PostgreSQL, MS SQL a Oracle databázemi. Bylo by dobré přidat podporu pro další databázové systémy.

Závěr

Hlavním cílem této bakalářské práce bylo vytvořit aplikaci, která by byla schopná stahovat data podle konfigurace z RÚIANu a ukládat je do databáze. V rámci práce jsem navrhl prvotní verzi aplikace, která splňuje tento cíl, implementoval jsem ji a otestoval jsem ji na vzorových datech z RÚIANu pro všechny zadané databázové systémy.

Vytvořená aplikace je první verze aplikace se jménem **Ruian_Puller**, která má za úkol stahovat data z RÚIANu a ukládat je do databáze. Podporované databáze jsou PostgreSQL, MySQL a Oracle. Aplikace je napsaná v programovacím jazyce Java a využívá framework Spring Boot. Je možné použití plánování úloh pomocí Quartz Scheduleru. Aplikace se nastavuje podle konfiguračního souboru, který je ve formátu JSON. Nastavuje se připojení k databázi, jaké prvky se mají stahovat, či ignorovat a jak často se stahování má provádět. Aplikace je naspaná jako základ pro budoucí rozšíření a je tedy zřetelné, že se dá hodně upravit a vylepšit viz kapitola 9.

Seznam použitých zkratek

API Application Programming Interface.

DTO Data Transfer Object.

INI Initialization File.

JDBC Java Database Connectivity.

JSON JavaScript Object Notation.

JPA Java Persistence API.

JTS Java Topology Suite.

MOMC Městský obvod / městská část u statutárně členěných měst.

Mop Obvod Praha.

ORM Object-Relational Mapping.

ORP Obce s rozšířenou působností.

POU Pracovní území obce.

RÚIAN Registr územní identifikace adres a nemovitostí.

SQL Structured Query Language.

VO Volební okrsek.

VFR Veřejná funkční registrace.

VÚSC Vymezené územní samosprávné celky.

XML Extensible Markup Language.

YAML YAML Ain't Markup Language.

ZSJ Základní sídelní jednotka.

Bibliografie

1. CZBAP-127. *VFR*. 2023. Dostupné také z: https://cuzk.gov.cz/ruian/Poskytovani-udaju-ISUI-RUIAN-VDP/Vymenny-format-RUIAN-(VFR)/DL058RR2-v5-0-Struktura-a-popis-VFR_final.aspx.
2. MICROSOFT. Microsoft SQL Server. *Microsoft Documentation*. 2025. Dostupné také z: https://docs.microsoft.com/en-us/sql/sql-server/.
3. GROUP, PostgreSQL Global Development. PostgreSQL. *PostgreSQL Documentation*. 2025. Dostupné také z: https://www.postgresql.org/docs/.
4. ORACLE. Oracle Database. *Oracle Documentation*. 2025. Dostupné také z: https://docs.oracle.com/en/database/.
5. COMMUNITY, Cisco. *XML vs JSON vs YAML*. 2022. Dostupné také z: https://community.cisco.com/t5/devnet-general-knowledge-base/xml-vs-json-vs-yaml/ta-p/4729758.
6. FIELDING, Roy T. *Architectural Styles and the Design of Network-based Software Architectures*. 2000. Dostupné také z: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.
7. SPRING. *Spring Framework Documentation*. 2025. Dostupné také z: https://spring.io/projects/spring-framework.
8. FOUNDATION, Apache Software. *What is Maven?* 2025. Dostupné také z: https://maven.apache.org/what-is-maven.html.
9. DOCKER. *What is Docker?* 2025. Dostupné také z: https://www.docker.com/what-docker.
10. DBEAVER. *DBeaver Documentation*. 2023. Dostupné také z: https://dbeaver.com/docs/dbeaver/.
11. FOUNDATION, Apache Software. *Apache Log4j*. 2025. Dostupné také z: https://logging.apache.org/log4j/2.12.x/.
12. PEŠIČKA, Ladislav. *06. Mutexy, monitory*. 2024. Dostupné také z: https://portal.zcu.cz/CoursewarePortlets2/DownloadDokumentu?id=16897.

Seznam obrázků

Seznam tabulek

Obsah

Instalační příručka příručka

Uživatelská příručka

Struktura přiloženého zip souboru